A Multifactor Empirical Framework for Explaining Power and Energy Variation in Managed Runtime Systems: The Java Runtime Environment Case

VOLUME II OF II

Interaction Effects, Model Development and Synthesis



Jonathan Lambert

In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer Science
Maynooth University
Maynooth, Co. Kildare, Ireland

Supervisors: Professor Rosemary Monahan and Dr Kevin Casey
Heads of Department:
Dr Joseph Timoney (until 2024)
Dr Aidan Mooney (from 2024)
Internal Examiner: Dr Stephen Brown
External Examiner: Professor David Gregg, Trinity College Dublin

This thesis has been prepared in accordance with the PhD regulations of Maynooth University and is subject to copyright. For more information see PhD Regulations (December 2022).

TABLE OF CONTENTS

VOLUME II: Interaction Effects, Model Development and Synthesis

List of Figures **v**

List of Tables **xiv**

Preface — Volume II **xxvi**

6 Research Question 2: Interaction Effects **1**

6.1 How this Chapter Addresses RQ2 2

6.2 OpenJDK JRE Version and Workload Interaction Effects 3

6.2.1 Power Use 4

Descriptive Analysis 5

Inferential Statistical Analysis 9

Summary 10

6.2.2 Energy Consumption 12

Descriptive Analysis 13

Inferential Statistical Analysis 17

Summary 19

6.2.3 Summary 22

6.3 GNU GCC version and workload interaction effects 25

6.3.1 Power Use 26

Descriptive Analysis 26

Inferential Statistical Analysis 31

Summary 34

6.3.2 Energy Consumption 36

Descriptive Analysis 37

Inferential Statistical Analysis 42

Summary 44

6.3.3 Summary 46

6.4 Garbage collector variant and workload interaction effects 48

6.4.1 Power Use 50

Descriptive Analysis 50

Inferential Statistical Analysis 56
Summary 59
6.4.2 Energy Consumption 61
Descriptive Analysis 62
Inferential Statistical Analysis 67
Summary 71
6.4.3 Summary 73
6.5 How this Chapter Answers RQ2 75

7 Research Question 3: Model Development 79
7.1 How this Chapter Addresses RQ3 80
7.2 Power Use Model 81
7.2.1 Theory-Driven Model Specification 82
7.2.2 Hierarchical Regression Results 83
7.2.3 Regression Diagnostics 87
Assessment of Linearity 87
Assessment of Residual Distribution 89
Assessment of Influential Observations 91
Assessment of Homoskedasticity 92
Assessment of Multicollinearity 93
Sensitivity Analysis 94
Bootstrap Resampling Analysis 97
7.2.4 Summary 99
7.3 Power Use Model: Interpretive Execution 102
7.3.1 Hierarchical Regression Results 102
7.3.2 Regression Diagnostics 107
Sensitivity Analysis 109
Bootstrap Validation 110
7.3.3 Summary 113
7.4 Power Use Model: Tiered Execution 115
7.4.1 Hierarchical Regression Results 116
7.4.2 Regression Diagnostics 122
Sensitivity Analysis 124
Bootstrap Validation 126
7.4.3 Summary 129
7.5 Power Use Cross-Model Comparative Analysis 133
7.5.1 Comparative Variance Decomposition Analysis 134
7.5.2 Observed-versus-Fitted Model Correspondence 136
7.5.3 Comparative Coefficient Restructuring Analysis 138

7.6 Energy Consumption Model . . . 140
7.6.1 Theory-Driven Model Specification . . . 142
7.6.2 Hierarchical Regression Results . . . 143
7.6.3 Regression Diagnostics . . . 149
Assessment of Linearity . . . 149
Assessment of Residual Distribution . . . 151
Assessment of Influential Observations . . . 153
Assessment of Homoskedasticity . . . 155
Assessment of Multicollinearity . . . 157
7.6.4 Sensitivity Analysis . . . 158
7.6.5 Bootstrap Validation . . . 161
7.6.6 Summary . . . 164
7.7 Energy Consumption Model: Interpretive Execution . . . 167
7.7.1 Hierarchical Regression Results . . . 168
7.7.2 Regression Diagnostics . . . 173
Sensitivity Analysis . . . 174
Bootstrap Validation . . . 177
7.7.3 Summary . . . 180
7.8 Energy Consumption Model: Tiered Execution . . . 182
7.8.1 Hierarchical Regression Results . . . 183
7.8.2 Regression Diagnostics . . . 189
Sensitivity Analysis . . . 190
Bootstrap Validation . . . 194
7.8.3 Summary . . . 197
7.9 Energy Consumption Cross-Model Comparative Analysis . . . 199
7.9.1 Comparative Variance Decomposition Analysis . . . 200
7.9.2 Observed-versus-Fitted Model Correspondence . . . 202
7.9.3 Comparative Coefficient Restructuring Analysis . . . 204
7.10 How this Chapter Answers RQ3 . . . 207

8 Discussion **211**
8.1 Revisiting the Research Questions . . . 212
8.2 Runtime Configuration Effects Are Real but Highly Uneven . . . 214
8.3 Workload Category Condition Runtime Relationships . . . 221
8.4 Execution Mode Restructures Software–Hardware Relationships . . . 227
8.5 Microarchitectural Behaviour Reveals Hidden Runtime Characteristics 233
8.6 Interpreting the OpenJDK 13 Behaviour . . . 239

8.7 Hierarchical Modelling Reveals Context-Dependent Relationships . . . 247
8.8 Distinct Explanatory Structures Governing Power Use and Energy Consumption . . . 252
8.9 Implications for the Analysis of Managed Runtime Systems . . . 257
8.10 Practical Implications . . . 261
8.11 Limitations . . . 266
8.12 Future Research . . . 272

9 Conclusion **281**
9.1 Conclusions in Relation to the Research Questions . . . 282
9.2 Overall Contribution of the Research . . . 283
9.3 Concluding Perspective . . . 285

A Statistical Analysis Code **287**

A Raspberry Pi 4 Model B Configuration File **289**

References — Volume II **292**

List of Figures

1.1 Structural overview of the dissertation showing the sequential progression from foundational theory and experimental methodology through empirical analysis addressing research questions, culminating in discussion and overall research conclusions. 32

2.1 Schematic representation of the JRE/JVM runtime architecture, distinguishing components internal to the runtime environment from external application artefacts and native libraries. The figure illustrates the principal JVM subsystems, including class loading, runtime data areas, the execution engine, and the Java Native Interface (JNI), together with their interactions with external class files and native method libraries. 38

2.2 Thread Structures of the Garbage Collector Variants 57

4.1 Theoretical Framework showing the high-level relationship between primary parameter groups and their collective influence on modelled outcomes. 129

4.2 Conceptual framework illustrating the key architectural, virtual machine, compiler, memory management, and workload parameters considered in the modelling of system power use and energy consumption, along with associated analytical methods. 131

4.3 Experimental workflow illustrating the construction of JRE JVM instances, benchmark execution, collection of power use, energy consumption, and microarchitectural measurements, and the subsequent statistical analyses used to address RQ1, RQ2, and RQ3. . 135

4.4 Raspberry Pi4 Model B board peripheral device layout. Detailing position of Broadcom BCM2711 processor, RAM, USB Type-C power supply, and network ports. Source: [1] 143
4.5 Jetperch LLC JS110 Joulescope DC Analyzer. Source: [2] 145
4.6 Jetperch LLC JS110 Joulescope depicting the rear-panel binding posts. Source: [2] 145
4.7 TP-Link TL-SG116 16-Port Gigabit Ethernet Switch Used to Interconnect the Experimental Cluster Nodes. Source: [3] 146
4.8 Schematic of the experimental measurement apparatus showing the Joulescope positioned in-line between the mains power supply and the Raspberry Pi 4 Model B under test. The master Raspberry Pi records measurement data from the Joulescope, while the TP-Link TL-SG116 switch provides network connectivity between the experimental cluster nodes. 151
4.9 Representative snapshot of the final analytical dataset comprising runtime configuration factors, workload descriptors, power use and energy consumption observations, and microarchitectural performance measures. Each observation represents a unique JRE JVM instance and benchmark workload used in the subsequent analyses. The full dataset is available through the accompanying research data repository [4]. 191
4.10 Overview of the statistical analysis framework employed to address RQ1 and RQ2, and to support the empirical modelling activities associated with RQ3. 197

5.1 Power Use Distributions: Box-and-whisker plots depicting OpenJDK JRE power use distributions for all pooled measurements (a), and grouped according to tiered (HS) and interpretive (INT) execution modes (b). 236

5.2 Energy Consumption Distributions: Box-and-whisker plots depicting OpenJDK JRE energy consumption across all measurements pooled (a), and grouped according to tiered (HS) and interpretive (INT) execution modes (b). 242
5.3 Interpretive-to-Tiered Execution Ratios: Box-and-whisker plots depicting power use ratios for interpretive execution relative to tiered execution across all JRE JVM version (a), energy consumption ratios for interpretive execution relative to tiered execution (b), and a scatterplot illustrating the association between the corresponding power use and energy consumption ratios (c). 247
5.4 Power Use Distributions: Box-and-whisker plots depicting JRE JVM power use distributions across GNU GCC versions for all pooled measurements (a), and grouped according to tiered (HS) and interpretive (INT) execution modes (b). 256
5.5 Energy Consumption Distributions: Box-and-whisker plots depicting JRE JVM energy consumption distributions across GNU GCC versions for all pooled measurements (a), and grouped according to tiered (HS) and interpretive (INT) execution modes (b). 262
5.6 Interpretive-to-Tiered Execution Ratios: Box-and-whisker plots depicting the ratio of power use for interpretive execution relative to tiered execution across all GCC versions (a), the ratio of energy consumption for interpretive execution relative to tiered execution (b). 268
5.7 Power Use Distributions: Box-and-whisker plots depicting OpenJDK JRE average power use performance across garbage collector implementations, all measurements pooled (a), and clustered based on tiered (HS) and interpretive (INT) modes of execution (b). . . 276

5.8 Energy Consumption Distributions: Box-and-whisker plots depicting OpenJDK JRE energy consumption across garbage collector implementations, all measurements pooled (a), and clustered based on tiered (HS) and interpretive (INT) modes of execution (b). . . 282
5.9 Interpretive-to-Tiered Execution Ratios: Box-and-whisker plots depicting the ratio of power use for interpretive execution relative to tiered execution across all garbage collector variants (a), the ratio of energy consumption for interpretive execution relative to tiered execution (b). 287
5.10 Power Use Distributions: Box-and-whisker plots depicting OpenJDK JRE average power use across workload categories, all measurements pooled (a), and grouped based on tiered (HS) and interpretive (INT) modes of execution (b). 295
5.11 Energy Consumption Distributions: Box-and-whisker plots depicting OpenJDK JRE energy consumption across workload categories, all measurements pooled (a), and grouped based on tiered (HS) and interpretive (INT) modes of execution (b). 301
5.12 Interpretive-to-Tiered Execution Ratios: Box-and-whisker plots depicting the ratio of power use for interpretive execution relative to tiered execution (a), the ratio of energy consumption for interpretive execution relative to tiered execution (b). 307
5.13 Branch misprediction rate distributions across execution mode categories (a), and JRE version clustered by execution mode (b). . 316
5.14 Branch Misprediction and Power Use Relationship: Scatterplot depicting the relationship between observed branch misprediction rates and measured JRE JVM power use across all pooled observations. 321

5.15 Branch Misprediction and Energy Consumption Relationship: Scatterplot depicting the relationship between observed branch misprediction rates and measured JRE JVM energy consumption across all pooled observations. 322
5.16 L1 Data Cache Load Miss Rate distributions across execution mode categories (a), and JRE version clustered by execution mode (b). . 326
5.17 L1 Data Cache Load Miss Rate and Power Use Relationship: Scatterplot depicting the relationship between observed L1 data cache load miss rates and measured JRE JVM power use across all pooled observations. 330
5.18 L1 Data Cache Load Miss Rate and Energy Consumption Relationship: Scatterplot depicting the relationship between observed L1 data cache miss rates and measured JRE JVM energy consumption across all pooled observations. 332
5.19 L2 Data Cache Load Miss Rate distributions across execution mode categories (a), and JRE version clustered by execution mode (b). . 336
5.20 L2 Data Cache Load Miss Rate and Power Use Relationship: Scatterplot depicting the relationship between observed L2 data cache load miss rates and measured JRE JVM power use across all pooled observations. 340
5.21 L2 Data Cache Load Miss Rate and Energy Consumption Relationship: Scatterplot depicting the relationship between observed L2 data cache load miss rates and measured JRE JVM energy consumption across all pooled observations. 342
5.22 Page Fault Rate and Power Use Relationship: Scatterplot depicting the relationship between observed page fault rates and measured JRE JVM power use across all pooled observations. 349

5.23 Page Fault Rate and Energy Consumption Relationship: Scatterplot depicting the relationship between observed page fault rates and measured JRE JVM energy consumption across all pooled observations. 352

6.1 Power Use Distributions: Box-and-whisker plots detailing power use distributions, panelled by workload category, and clustered within each workload category by OpenJDK JRE version. 5

6.2 Power Use Distributions: Box-and-whisker plots depicting power use distributions for each workload category, clustered within by the OpenJDK JRE version. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT. 6

6.3 Energy Consumption Distributions: Box-and-whisker plots detailing energy consumption distributions, panelled by workload category, and clustered within each workload category by OpenJDK JRE version. 13

6.4 Energy Consumption Distributions: Box-and-whisker plots depicting power use distributions for each workload category, clustered within by the OpenJDK JRE version. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT. 14

6.5 Power Use Distributions: Box-and-whisker plots detailing power use distributions, panelled by workload category, and clustered within each workload category by GNU GCC version. 27

6.6 Power Use Distributions: Box-and-whisker plots depicting power use distributions for each workload category, clustered within by the GNU GCC version. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT. 28

6.7 Energy Consumption Distributions: Box-and-whisker plots detailing energy consumption distributions, panelled by workload category, and clustered within each workload category by GNU GCC version. 37
6.8 Energy Consumption Distributions: Box-and-whisker plots depicting energy consumption distributions for each workload category, clustered by GNU GCC version. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT. . . . 38
6.9 Power Use Distributions: Box-and-whisker plots detailing power use distributions, panelled by workload category, and clustered within each workload category by Garbage Collector variant. . . . 50
6.10 Power Use Distributions: Box-and-whisker plots depicting power use distributions for each workload category, clustered within by the Garbage Collector variant. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT. . . . 51
6.11 Energy Consumption Distributions: Box-and-whisker plots detailing energy consumption distributions, panelled by workload category, and clustered within each workload category by garbage collection variant. 62
6.12 Energy Consumption Distributions: Box-and-whisker plots depicting energy consumption distributions for each workload category, clustered by the garbage collector used for heap management. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT. 63

7.1 Linearity Diagnostics (Pooled Power Use Model): Residuals versus fitted values plot used to assess the adequacy of the linear regression specification. The majority of observations remained broadly centred around zero, although modest systematic residual structure was observed together with two observations exhibiting substantially elevated positive residuals. These observations indicate the presence of heterogeneous execution contexts not fully captured within the aggregated pooled model specification. 88

7.2 Residual Distribution Diagnostics (Pooled Power Use Model): Histogram depicting the distribution of regression residuals associated with the pooled power use model (a), and normal Q–Q plot assessing conformity of residuals to the theoretical normal distribution (b). 90

7.3 Residual Distribution Diagnostics (Interpretive Power Use Model): Histogram depicting the distribution of regression residuals associated with the interpretive power use model. 108

7.4 Residual Distribution Diagnostics (Tiered Power Use Model): Histogram depicting the distribution of regression residuals associated with the tiered execution power use model. 123

7.5 Observed-versus-fitted correspondence for the final power use regression models: (a) pooled observations, (b) interpretive execution, and (c) tiered execution. The dashed identity line represents perfect correspondence between observed and fitted mean power use. . . . 136

7.6 Residual-versus-fitted plots comparing linearity diagnostics for the preliminary untransformed energy consumption model and the final log-transformed energy consumption model. 149

7.7 Residual distribution diagnostics for the final pooled JRE JVM energy consumption regression model using the log-transformed dependent variable $\ln(energyWh)$. 152

7.8 Residual-versus-fitted plot for the final pooled JRE JVM energy consumption regression model using the log-transformed dependent variable ln(*energyWh*). 156
7.9 Residual Distribution Diagnostics (Interpretive Energy Consumption Model): Histogram of regression residuals (a) and corresponding normal Q-Q plot (b) for the interpretive execution energy consumption model. 173
7.10 Residual Distribution Diagnostics (Tiered Energy Consumption Model): Histogram of regression residuals (a) and corresponding normal Q-Q plot (b) for the tiered execution energy consumption model. 189
7.11 Observed-versus-fitted correspondence for the final log-transformed energy consumption regression models: (a) pooled observations, (b) interpretive execution, and (c) tiered execution. The dashed identity line represents perfect correspondence between observed and fitted ln(energyWh) values. 202

List of Tables

1.1 Dissertation Contributions and Supporting Empirical Evidence . . 28

2.1 Selected JDK Enhancement Proposals by OpenJDK Version . . . 65

4.1 Summary of Experimental Variables 133

4.2 Summary of the common experimental configuration employed throughout the study 137

4.3 Key GNU GCC configuration flags used during toolchain construction and their associated purposes. 156

4.4 Key OpenJDK configuration flags used during JRE JVM construction and their associated purposes. 158

4.5 Garbage collector availability across the OpenJDK source versions examined in this study. The CMS+ParNew garbage collector (CPGC) was deprecated and excluded from OpenJDK versions subsequent to OpenJDK 9. 161

4.6 Distribution of valid JRE/JVM instances by OpenJDK source version and execution mode. The reduced number of JDK 9 interpretive instances reflects the absence of one garbage collector variant, while the reduction in configurations for OpenJDK versions 10–14 reflects the exclusion of the CMS+ParNew garbage collector (CPGC). 162

4.7 The 24 application workloads composing the Renaissance Benchmark Suite, together with a brief description of each workload. Source: [5]. 166

4.8 Workload Category (WC) factor showing the grouping of Renaissance benchmark workloads into six workload categories. 167
4.9 JRE JVM runtime configuration flags used within the experimental framework. 170
4.10 Key JRE JVM runtime flags used during benchmark execution and their associated purposes. 172
4.11 Microarchitectural and operating system performance measures collected using the Linux `perf stat -d` framework. 177
4.12 Key elements of the benchmark execution procedure and their associated purposes. 180
4.13 Representative HotSpot tiered-compilation transitions observed during verification execution of the Renaissance `db-shootout` workload. 181
4.14 Distribution of JRE/JVM instances and benchmark-condition observations included in the final analytical dataset. 187
4.15 A description of all categorical dataset variables, detailing each factor's name, dataset identifier, underlying type, a short description of the variable, and the values associated with each factor level. . 193
4.16 A description of all numerical dataset variables, detailing each factor's name, dataset identifier, underlying type, and a short description of the variable. 194
4.17 Summary of the analytical dataset. 195

5.1 Power Distributions: Descriptive statistics associated with each of the six OpenJDK JRE JVMs, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with power use. 236

5.2 Energy Distributions: Descriptive statistics associated with each of the six OpenJDK JRE JVMs, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with energy consumption. 243

5.3 Power and Energy Ratios: Descriptive statistics associated with the ratio of interpretive execution relative to tiered execution. . . 247

5.4 Power Distributions: Descriptive statistics associated with each of the four GNU GCC versions, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with power use. 257

5.5 Energy Distributions: Descriptive statistics associated with each of the four GNU GCC versions, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with energy consumption. 264

5.6 Power Energy Ratios: Descriptive statistics associated with each of the four GNU GCC versions, detailing the mean (**M**), the median ($\mathbf{M}_d$), the standard deviation (**SD**), and the interquartile range (**IQR**) associated with the ratio of interpretive power use relative to tiered execution, and the ratio of interpretive energy consumption relative to tiered execution. 268

5.7 Power Distributions: Descriptive statistics associated with each of the five garbage collector variants, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with power use. 277

5.8 Energy Distributions: Descriptive statistics associated with each of the five garbage collector variants, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with power use. 283

5.9 Power Energy Ratios: Descriptive statistics associated with each garbage collector variant, detailing the mean (**M**), the median ($\mathbf{M}_d$), the standard deviation (**SD**), and the interquartile range (**IQR**) associated with the ratio of interpretive power use relative to tiered execution, and the ratio of interpretive energy consumption relative to tiered execution. 287

5.10 Power Distributions: Descriptive statistics associated with each of the six workload categories, reported for all pooled observations and grouped by execution mode, detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and sample size (**n**) associated with power use. 296

5.11 Energy Distributions: Descriptive statistics associated with each of the six workload categories, reported for all pooled observations and grouped by execution mode, detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and sample size (**n**) associated with power use. 302

5.12 Power Energy Ratios: Descriptive statistics associated with each of the six workload categories, detailing the mean (**M**), the median ($\mathbf{M}_d$), the standard deviation (**SD**), and the interquartile range (**IQR**) associated with the ratio of interpretive power use relative to tiered execution, and the ratio of interpretive energy consumption relative to tiered execution. 307

5.13 Branch Misprediction Distributions: Descriptive statistics associated with each of the six OpenJDK JRE JVMs, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with branch misprediction rates. 317

5.14 L1 Data Cache Load Miss Rate Distributions: Descriptive statistics associated with each of the six OpenJDK JRE JVMs, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with L1 data cache load miss rates. 327

5.15 L2 Data Load Miss Rate Distributions: Descriptive statistics associated with each of the six OpenJDK JRE JVMs, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with L2 data cache load miss rates. 337

5.16 Page Fault Distributions: Descriptive statistics associated with each of the six OpenJDK JRE JVMs, reported for all pooled observations, and grouped by execution mode. Detailing the mean (**M**), median ($\mathbf{M}_d$), standard deviation (**SD**), interquartile range (**IQR**), and the sample size (**n**) associated with page fault rates. 346

6.1 Power Use Distributions (Pooled Observations): Descriptive statistics summarising power use observations across workload categories and OpenJDK JRE versions when observations are pooled across both interpretive and tiered execution modes. 7

6.2 Power Use Distributions (Tiered Execution): Descriptive statistics summarising power use observations across workload categories and OpenJDK JRE versions under tiered execution. 8
6.3 Power Use Distributions (Interpretive Execution): Descriptive statistics summarising power use observations across workload categories and OpenJDK JRE versions under interpretive execution. 9
6.4 Energy Consumption Distributions (Pooled Observations): Descriptive statistics summarising energy consumption observations across workload categories and OpenJDK JRE versions when observations are pooled across both interpretive and tiered execution modes. . 15
6.5 Energy Consumption Distributions (Tiered Execution): Descriptive statistics summarising energy consumption observations across workload categories and OpenJDK JRE versions under tiered execution. 17
6.6 Energy Consumption Distributions (Interpretive Execution): Descriptive statistics summarising energy consumption observations across workload categories and OpenJDK JRE versions under interpretive execution. 17
6.7 Power Use Distributions (Pooled Execution (W)): Descriptive statistics summarising power use observations across workload categories and GNU GCC versions under pooled execution. 29
6.8 Power Use Distributions (Tiered Execution (W)): Descriptive statistics summarising power use observations across workload categories and GNU GCC versions under tiered execution. 30
6.9 Power Use Distributions (Interpretive Execution (W)): Descriptive statistics summarising power use observations across workload categories and GNU GCC versions under interpretive execution. . 31

6.10 Energy Consumption Distributions (Pooled Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and GNU GCC versions under pooled execution. 39
6.11 Energy Consumption Distributions (Tiered Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and GNU GCC versions under tiered execution. 40
6.12 Energy Consumption Distributions (Interpretive Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and GNU GCC versions under interpretive execution. 41
6.13 Power Use Distributions (Pooled Execution (W)): Descriptive statistics summarising power use observations across workload categories and Garbage Collector variants under pooled execution (w). . . . 52
6.14 Power Use Distributions (Tiered Execution (W)): Descriptive statistics summarising power use observations across workload categories and Garbage Collector variants under tiered execution (w). 53
6.15 Power Use Distributions (Interpretive Execution (W)): Descriptive statistics summarising power use observations across workload categories and Garbage Collector variants under interpretive execution (w). 54
6.16 Energy Consumption Distributions (Pooled Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and Garbage Collector variants under pooled execution. 64

6.17 Energy Consumption Distributions (Tiered Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and Garbage Collector variant under tiered execution. 65
6.18 Energy Consumption Distributions (Interpretive Execution (W)): Descriptive statistics summarising energy consumption observations across workload categories and Garbage Collector variants under interpretive execution. 66

7.1 Hierarchical regression model development for JRE JVM pooled power use. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block. 84
7.2 Hierarchical regression coefficients for pooled JRE JVM power use. Standard errors correspond to HC3 heteroskedasticity-consistent robust variance estimation. 86
7.3 Cross-classification of influential observations identified through Cook's distance analysis. Percentages are calculated relative to all influential observations ($n = 96$). 92
7.4 Adjusted Generalised Variance Inflation Factor statistics for predictors included within the pooled power use regression model. . . 94
7.5 Final pooled power use regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. Coefficients are reported in watts. The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$. 96
7.6 Stratified bootstrap coefficient stability analysis for the pooled power use regression model. 98

7.7 Hierarchical regression model development for JRE JVM interpretive execution power use. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block. 103
7.8 Hierarchical regression coefficients for interpretive execution JRE JVM power use. Standard errors correspond to HC3 heteroskedasticity-consistent robust variance estimation. 105
7.9 Final interpretive power use regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$. 110
7.10 Stratified bootstrap coefficient stability analysis for the interpretive execution power use regression model. 111
7.11 Hierarchical regression model development for JRE JVM tiered execution power use. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block. 118
7.12 Hierarchical regression coefficients for tiered execution JRE JVM power use. Standard errors correspond to HC3 heteroskedasticity-consistent robust variance estimation. 119
7.13 Final tiered execution power use regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$. Dependent variable is $\ln(energyWh)$. 125
7.14 Stratified bootstrap coefficient stability analysis for the tiered execution power use regression model. 127

7.15 Comparative variance contributions across hierarchical power use regression models 134
7.16 Comparative coefficient restructuring across final power use regression models 138
7.17 Hierarchical regression model development for JRE JVM pooled energy consumption. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block. 144
7.18 Hierarchical regression coefficients for pooled JRE JVM energy consumption ($\ln(energyWh)$). Standard errors correspond to HC3 heteroskedasticity-consistent robust variance estimation. 145
7.19 Cross-classification of influential observations identified through Cook's distance analysis, detailing the number of influential observations belonging to each workload category. Percentages are calculated relative to all influential observations ($n = 130$). 154
7.20 Multicollinearity diagnostics for the final pooled JRE JVM energy consumption regression model. Adjusted GVIF values are reported to permit comparison across predictors with differing degrees of freedom. 158
7.21 Final pooled energy consumption regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$. Dependent variable is $\ln(energyWh)$. 160
7.22 Stratified bootstrap coefficient stability analysis for the pooled energy consumption regression model. 162

7.23 Hierarchical regression model development for JRE JVM interpretive execution energy consumption. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block. . . . 169
7.24 Hierarchical regression coefficient development for interpretive execution JRE JVM energy consumption (HC3 robust standard errors) 171
7.25 Final interpretive execution energy consumption regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. Coefficients are reported on the log-transformed energy consumption scale ($\ln(energyWh)$). The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$. 175
7.26 Stratified bootstrap coefficient stability analysis for the interpretive execution energy consumption regression model. 178
7.27 Hierarchical regression model development for JRE JVM tiered execution energy consumption. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block. 184
7.28 Hierarchical regression coefficient development for tiered execution JRE JVM energy consumption ($\ln(energyWh)$). HC3 heteroskedasticity-consistent robust standard errors are reported for all coefficient estimates. 186

7.29 Final tiered execution energy consumption regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. Coefficients are reported on the log-transformed energy consumption scale ($\ln(energyWh)$). The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$. 191
7.30 Distribution of influential observations identified within the tiered execution energy consumption sensitivity analysis. Influential observations were identified using Cook's distance threshold $4/n$. Percentages in the final row indicate the proportion of all influential observations attributable to each workload category. 193
7.31 Stratified bootstrap coefficient stability analysis for the tiered execution energy consumption regression model. 195
7.32 Comparative variance contributions across hierarchical energy consumption regression models 200
7.33 Comparative coefficient restructuring across final energy consumption regression models 205

Volume II Preface

This dissertation is presented in two volumes as a single, continuous body of research. The division is intended solely to facilitate presentation and navigation and does not separate the research into independent studies. Chapter, section, figure, table, equation, and reference numbering therefore remain continuous across both volumes, while each volume retains independent page numbering.

Volume II: Interaction Effects, Model Development and Synthesis continues directly from the foundations, methodology, and main-effects analysis presented in Volume I. Chapter 6 addresses Research Question RQ2 by examining whether the effects of runtime configuration factors on JRE/JVM power use and energy consumption are conditioned by workload category, with analyses further decomposed by interpretive and tiered execution. Chapter 7 addresses Research Question RQ3 through hierarchical regression modelling of power use and energy consumption using pooled and execution-mode-specific model specifications, supported by diagnostic, sensitivity, bootstrap, and cross-model comparative analyses. Chapter 8 synthesises the findings across all three research questions and relates them to the contributions to knowledge and existing literature, while Chapter 9 presents the overall conclusions. The volume concludes with the supporting appendices.

The theoretical background, related literature, experimental methodology, dataset construction, and RQ1 main-effects analyses underpinning these chapters are presented in **Volume I: Foundations, Methodology and Main Effects**. The two volumes are complementary, with Volume II extending Volume I through interaction analysis, multivariable modelling, interpretation, and synthesis. The Table of Contents covers both volumes, each with references.

Chapter 6

Research Question 2: Interaction Effects

This chapter addresses **Research Question 2 (RQ2)**, which examines whether the effects associated with runtime configuration factors on observed JRE/JVM power use and energy consumption are dependent upon workload category. Whereas the preceding chapter examined the independent effects associated with JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, workload category, and microarchitectural performance event measures, the present chapter evaluates whether these relationships remain stable across differing computational workload categories.

The motivation for examining interaction effects arises from the observation that JRE/JVM execution is inherently workload-sensitive. Different workload categories place differing demands on computation, memory allocation, object lifecycle management, and execution flow. Consequently, system-level factors, including JRE implementation, compiler toolchain configuration, and garbage collector variant, may not exert uniform effects across all forms of execution. Instead, their influence on power use and energy consumption may depend upon the characteristics of the workload being executed.

To systematically evaluate these effects, the chapter adopts a staged analytical approach. We first examine the interaction between OpenJDK JRE source version and workload category §6.2, followed by the interaction between GNU GCC build

compiler version and workload category §6.3, and then the interaction between garbage collector variant and workload category §6.4. These analyses assess whether relationships previously observed in the main effects chapter remain stable across differing workload category levels.

In addition, JVM execution mode is treated throughout this chapter as a cross-sectional analytical dimension. Accordingly, all interaction effects are decomposed according to interpretive and tiered execution modes to facilitate direct comparison of execution strategies under differing runtime conditions.

Across all analyses, we first present descriptive assessments of the underlying distributions, followed by inferential statistical analyses examining differences between distributions and interaction effects. Given the distributional characteristics of the observed measures, non-parametric statistical procedures are employed where appropriate. Results are reported for both power use and energy consumption and are presented at both the pooled level and decomposed according to interpretive and tiered execution modes to provide a detailed understanding of JRE JVM execution characteristics under differing runtime conditions.

6.1 How this Chapter Addresses RQ2

This chapter addresses Research Question 2 (RQ2), which asks:

> ***RQ2****: To what extent are the effects associated with JRE source version, GNU GCC build toolchain compiler version, JVM execution mode, and garbage collector variant on JRE JVM power use and energy consumption dependent upon workload category?*

RQ2 is addressed through a structured inferential statistical analysis in which the effects associated with OpenJDK JRE source version, GNU GCC build toolchain compiler version, and garbage collector variant are examined in combination with workload category to determine whether workload-dependent interaction effects emerge across the observed power use and energy consumption outcomes.

Specifically, this chapter evaluates whether previously observed differences in JRE JVM power use and energy consumption remain stable across workload categories, or whether workload category moderates those relationships. Throughout these analyses, JVM execution mode is treated as a cross-sectional analytical dimension, with interaction effects decomposed according to interpretive and tiered execution to assess whether observed relationships differ between execution strategies.

Each section therefore contributes evidence toward answering RQ2 by determining whether the effects associated with a given runtime configuration remain consistent across workload classes, the magnitude and direction of those effects, and whether interaction patterns differ between interpretive and tiered execution contexts. Collectively, the analyses establish the extent to which workload category moderates the effects associated with runtime configuration variables, while identifying the workload conditions under which these effects differ most substantially.

6.2 OpenJDK JRE Version and Workload Interaction Effects

This section contributes to answering **RQ2** by evaluating whether the effects associated with OpenJDK JRE source version on JRE JVM power use and energy consumption remain stable across differing workload categories. Specifically, the analyses assess whether workload category moderates the relationship between OpenJDK JRE source version and both power use and energy consumption, thereby determining whether the magnitude or direction of JRE version effects depend upon the computational characteristics of the workload being executed.

This section principally addresses sub-research question **SRQ2.1**, which asks:

> **SRQ2.1:** ***To what extent do the effects of JRE source version on JRE JVM power use and energy consumption differ across workload categories?***

Because all analyses are additionally decomposed according to interpretive and tiered execution modes, this section also contributes evidence toward understanding the extent to which execution mode influences these workload-dependent relationships, in particular:

> **SRQ2.4:** ***To what extent do differences between interpretive and tiered execution affect JRE JVM power use and energy consumption across differing workload categories?***

To address these questions, the analyses evaluate whether previously observed differences associated with OpenJDK JRE source version remain consistent across workload categories or whether workload category alters the magnitude and statistical significance of those effects. The analyses first examine the interaction between OpenJDK JRE source version and workload category in relation to power use, followed by an equivalent analysis of energy consumption. Throughout the section, results are considered across the full set of observations and are further decomposed according to interpretive and tiered execution modes to determine whether observed interaction effects remain consistent across differing execution strategies.

Collectively, these analyses establish the extent to which workload category moderates the relationship between OpenJDK JRE source version and both JRE JVM power use and energy consumption under differing runtime conditions.

6.2.1 Power Use

This subsection principally addresses sub-research question **SRQ2.1a**, which examines the extent to which the effects of JRE source version on JRE JVM power differ across workload categories:

> **SRQ2.1a:** ***To what extent do the effects of JRE source version on JRE JVM power use differ across workload categories?***

Because all analyses are additionally decomposed according to execution mode, this subsection also contributes evidence toward sub-research question **SRQ2.4a**, which examines the extent to which differences between interpretive and tiered execution moderate these workload-dependent relationships:

> **SRQ2.4a:** ***To what extent do differences between interpretive and tiered execution affect JRE JVM power use across differing workload categories?***

Descriptive Analysis

Figures 6.1 and 6.2, together with Tables 6.1, 6.2, and 6.3, summarise the distributional characteristics associated with power use across workload categories and OpenJDK JRE versions, reported for pooled observations and further decomposed according to tiered and interpretive execution.

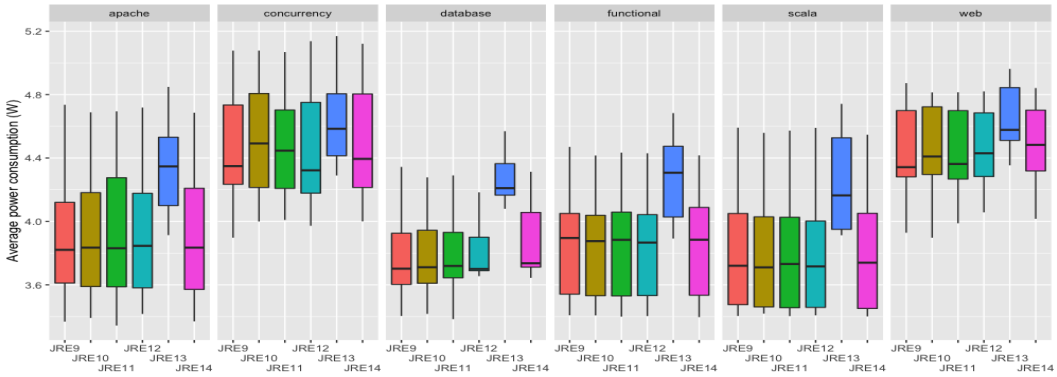


Figure 6.1: Power Use Distributions: Box-and-whisker plots detailing power use distributions, panelled by workload category, and clustered within each workload category by OpenJDK JRE version.

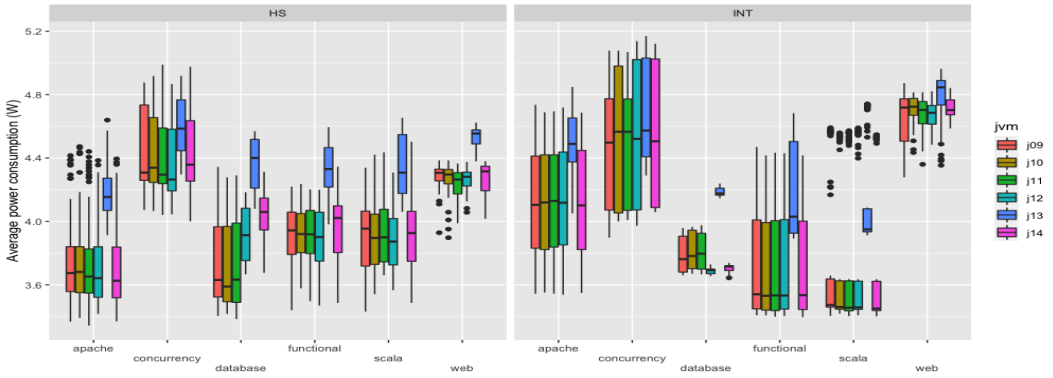


Figure 6.2: Power Use Distributions: Box-and-whisker plots depicting power use distributions for each workload category, clustered within by the OpenJDK JRE version. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT.

Across all workload category levels, OpenJDK JRE version 13 is consistently associated with elevated power use relative to the remaining JRE versions. However, the magnitude of this increase varies substantially across workload categories, indicating that the effect associated with JRE source version is not uniform across differing workload category levels. Across pooled observations, Figure 6.1 and Table 6.1, the largest increases are observed within the `apache`, `database`, `functional`, and `scala` workload categories, where median power use increases by approximately $0.43W$ to $0.52W$ relative to the corresponding OpenJDK JRE version 9 to 12 baselines. In contrast, comparatively smaller increases are observed within the `concurrency` and `web` workload categories.

Considering differences between execution modes, decomposing these observations reveals additional workload-dependent variation. Under tiered execution, the largest increases associated with OpenJDK JRE version 13 are observed within the `database`, `apache`, `functional`, and `scala` workload categories, with the `database` category exhibiting the largest overall increase of approximately $0.70W$, and all other increases ranging between $0.40W$ and $0.50W$. Under interpretive execution, a broadly similar pattern is observed, although the magnitude of these increases is generally reduced, with the largest increases occurring within the `functional`, `scala`, `database`, and `apache` workload categories, where median power use increases range from approximately $0.35W$ to $0.50W$.

Table 6.1: Power Use Distributions (Pooled Observations): Descriptive statistics summarising power use observations across workload categories and OpenJDK JRE versions when observations are pooled across both interpretive and tiered execution modes.

Pooled Observations (W)						Pooled Observations (W)					
Category	**JRE**	**M**	**M_d**	**SD**	**IQR**	**Category**	**JRE**	**M**	**M_d**	**SD**	**IQR**
apache	j09	3.89	3.82	0.34	0.51	functional	j09	3.84	3.90	0.27	0.51
	j10	3.93	3.83	0.36	0.59		j10	3.83	3.88	0.29	0.51
	j11	3.92	3.83	0.37	0.69		j11	3.83	3.88	0.29	0.53
	j12	3.91	3.85	0.37	0.60		j12	3.82	3.87	0.28	0.51
	j13	4.34	4.35	0.24	0.43		j13	4.26	4.31	0.24	0.45
	j14	3.92	3.83	0.38	0.64		j14	3.84	3.88	0.30	0.55
concurrency	j09	4.44	4.35	0.32	0.50	scala	j09	3.82	3.72	0.33	0.58
	j10	4.49	4.49	0.35	0.59		j10	3.80	3.71	0.35	0.57
	j11	4.46	4.45	0.33	0.49		j11	3.81	3.73	0.35	0.57
	j12	4.45	4.32	0.35	0.57		j12	3.80	3.72	0.35	0.54
	j13	4.64	4.58	0.25	0.39		j13	4.23	4.16	0.28	0.58
	j14	4.49	4.39	0.35	0.59		j14	3.82	3.74	0.37	0.60
database	j09	3.77	3.70	0.23	0.32	web	j09	4.44	4.34	0.23	0.42
	j10	3.77	3.71	0.22	0.33		j10	4.48	4.41	0.25	0.43
	j11	3.77	3.72	0.23	0.29		j11	4.45	4.36	0.24	0.43
	j12	3.81	3.70	0.18	0.21		j12	4.47	4.43	0.23	0.40
	j13	4.27	4.21	0.15	0.20		j13	4.65	4.58	0.19	0.33
	j14	3.87	3.74	0.22	0.34		j14	4.49	4.48	0.24	0.38

Several workload-specific interaction effects are also evident. The `concurrency` workload category exhibits relatively narrow variation in median power use across all OpenJDK JRE versions under both execution modes, indicating comparatively limited sensitivity to variation in JRE source version. Most notably, under tiered execution OpenJDK JRE version 13 is associated with a moderate increase in median power use, whereas under interpretive execution this increase is almost entirely absent. In contrast, the `database` workload category exhibits substantially greater sensitivity to variation in OpenJDK JRE version, particularly under tiered execution, where the magnitude of the OpenJDK JRE version 13 increase is substantially amplified relative to interpretive execution.

Differences in distributional spread further reinforce these workload-dependent effects. Interpretive execution is associated with substantially wider interquartile ranges within the `apache`, `concurrency`, and `functional` workload categories, indicating greater variability in power use observations relative to tiered execution. Conversely, narrower interquartile ranges are observed within the `database` and `scala` workload categories under interpretive execution.

Taken together, these descriptive results provide initial evidence that workload category moderates the relationship between OpenJDK JRE source version and power use, with the magnitude of the OpenJDK JRE version effect differing substantially across workload categories. Furthermore, the observed differences between interpretive and tiered execution indicate that execution mode further conditions the magnitude and consistency of these workload-dependent effects, demonstrating that the relationship between OpenJDK JRE source version and power use is jointly dependent upon both workload category and execution mode.

Table 6.2: Power Use Distributions (Tiered Execution): Descriptive statistics summarising power use observations across workload categories and OpenJDK JRE versions under tiered execution.

Tiered Execution (W)						**Tiered Execution (W)**					
Category	**JRE**	**M**	**M_d**	**SD**	**IQR**	**Category**	**JRE**	**M**	**M_d**	**SD**	**IQR**
apache	j09	3.73	3.67	0.23	0.28	functional	j09	3.93	3.94	0.16	0.27
	j10	3.74	3.68	0.25	0.29		j10	3.93	3.92	0.17	0.25
	j11	3.72	3.65	0.25	0.28		j11	3.92	3.92	0.18	0.27
	j12	3.71	3.64	0.23	0.32		j12	3.90	3.90	0.18	0.31
	j13	4.19	4.15	0.16	0.20		j13	4.33	4.33	0.15	0.25
	j14	3.71	3.63	0.25	0.32		j14	3.96	4.02	0.20	0.30
concurrency	j09	4.41	4.31	0.25	0.48	scala	j09	3.92	3.95	0.21	0.35
	j10	4.44	4.34	0.26	0.41		j10	3.91	3.90	0.21	0.32
	j11	4.41	4.30	0.26	0.35		j11	3.93	3.90	0.20	0.33
	j12	4.37	4.26	0.26	0.39		j12	3.90	3.87	0.20	0.29
	j13	4.60	4.59	0.18	0.32		j13	4.35	4.31	0.19	0.37
	j14	4.43	4.36	0.27	0.38		j14	3.95	3.93	0.24	0.32
database	j09	3.76	3.63	0.30	0.44	web	j09	4.28	4.31	0.09	0.07
	j10	3.73	3.59	0.29	0.48		j10	4.26	4.30	0.12	0.09
	j11	3.73	3.63	0.30	0.50		j11	4.24	4.26	0.10	0.13
	j12	3.93	3.91	0.19	0.33		j12	4.26	4.28	0.08	0.09
	j13	4.36	4.40	0.18	0.31		j13	4.53	4.56	0.07	0.09
	j14	4.04	4.06	0.20	0.20		j14	4.26	4.32	0.10	0.15

Table 6.3: Power Use Distributions (Interpretive Execution): Descriptive statistics summarising power use observations across workload categories and OpenJDK JRE versions under interpretive execution.

Interpretive Execution (W)						Interpretive Execution (W)					
Category	**JRE**	**M**	**M_d**	**SD**	**IQR**	**Category**	**JRE**	**M**	**M_d**	**SD**	**IQR**
apache	j09	4.11	4.10	0.35	0.58	functional	j09	3.74	3.54	0.34	0.56
	j10	4.12	4.12	0.36	0.60		j10	3.73	3.53	0.34	0.55
	j11	4.12	4.13	0.36	0.58		j11	3.73	3.53	0.34	0.57
	j12	4.12	4.12	0.36	0.59		j12	3.73	3.53	0.34	0.56
	j13	4.49	4.49	0.22	0.28		j13	4.19	4.03	0.29	0.58
	j14	4.13	4.10	0.37	0.63		j14	3.73	3.54	0.34	0.56
concurrency	j09	4.49	4.50	0.40	0.70	scala	j09	3.69	3.47	0.41	0.18
	j10	4.55	4.57	0.41	0.93		j10	3.70	3.46	0.43	0.18
	j11	4.52	4.57	0.38	0.70		j11	3.69	3.46	0.43	0.19
	j12	4.54	4.52	0.42	0.95		j12	3.70	3.46	0.43	0.18
	j13	4.68	4.57	0.30	0.62		j13	4.11	3.95	0.30	0.15
	j14	4.54	4.51	0.40	0.94		j14	3.69	3.45	0.43	0.19
database	j09	3.79	3.76	0.11	0.23	web	j09	4.64	4.72	0.18	0.27
	j10	3.81	3.78	0.12	0.24		j10	4.69	4.72	0.11	0.11
	j11	3.81	3.80	0.12	0.23		j11	4.67	4.70	0.12	0.14
	j12	3.69	3.69	0.02	0.03		j12	4.67	4.69	0.09	0.12
	j13	4.19	4.18	0.03	0.04		j13	4.76	4.85	0.20	0.16
	j14	3.70	3.72	0.03	0.03		j14	4.71	4.70	0.07	0.09

Inferential Statistical Analysis

The descriptive analyses identified apparent differences in power use across OpenJDK JRE versions within each workload category, while also indicating variation in the magnitude of these effects when decomposed according to execution mode. To determine whether these observed differences reflect statistically significant variation, inferential statistical analyses were conducted. We first assess the distributional characteristics associated with the observed power use measurements. Given the resulting departures from normality, non-parametric statistical procedures are subsequently employed to evaluate differences between OpenJDK JRE versions within workload categories, differences between execution modes, and the extent to which any observed effects are attributable to individual JRE versions.

Assessment of Distributional Assumptions:

Distributional normality was assessed using the Shapiro–Wilk test. The results indicated statistically significant departures from normality across all power use distributions examined, with the exception of the `database` workload category

executing on OpenJDK JRE version 12, which did not significantly deviate from normality ($W = 0.965$, $p = .067$). Given the overall distributional characteristics observed, non-parametric statistical procedures were employed throughout the analysis.

OpenJDK JRE Version Effects Within Workload Categories:

Kruskal–Wallis H tests indicated statistically significant differences in median power use within each workload category across OpenJDK JRE versions, with all workload-specific comparisons returning statistically significant results (all $p < .001$). Pairwise comparisons demonstrated that these differences were predominantly associated with comparisons involving OpenJDK JRE version 13, which exhibited significantly greater power use relative to all remaining JRE versions across all six workload categories (all $p < .001$).

Execution Mode Effects:

Across both execution modes, statistically significant differences between OpenJDK JRE versions remain consistently evident across workload category levels. In particular, the elevated power use associated with OpenJDK JRE version 13 remains present across all workload categories under both tiered and interpretive execution, indicating that the relationship between JRE source version and power use remains broadly stable when decomposed according to execution mode.

Although this overall pattern remains consistent across execution modes, the magnitude of the increase associated with OpenJDK JRE version 13 appears less pronounced within the `concurrency` workload category under interpretive execution.

Summary

The descriptive analyses demonstrated that the relationship between OpenJDK JRE source version and JRE JVM power use differed across workload category

levels. OpenJDK JRE version 13 was consistently associated with elevated power use across all workload categories, but the magnitude of this increase was not uniform. Across pooled observations, the largest increases were observed within the `apache`, `database`, `functional`, and `scala` workload categories, while comparatively smaller increases were observed within the `concurrency` and `web` categories. These findings indicate that workload category moderated the relationship between OpenJDK JRE source version and power use.

Decomposing the analysis according to execution mode provided further evidence of workload-dependent variation. Under tiered execution, the OpenJDK JRE version 13 increase was most pronounced within the `database` workload category, while under interpretive execution the largest increases were observed within the `functional`, `scala`, `database`, and `apache` workload categories. The `concurrency` workload category exhibited the clearest execution-mode-dependent difference, with a moderate OpenJDK JRE version 13 increase under tiered execution but an almost absent increase under interpretive execution. These findings indicate that execution mode further conditioned the workload-dependent relationship between JRE source version and power use.

Inferential statistical analyses supported these observations. Statistically significant differences in median power use were observed across OpenJDK JRE versions within all workload categories. Pairwise comparisons indicated that these differences were predominantly attributable to OpenJDK JRE version 13, which exhibited significantly greater power use relative to the remaining JRE versions across all workload category levels. This pattern remained evident when decomposed by execution mode, although the magnitude of the OpenJDK JRE version 13 effect varied by both workload category and execution mode.

In relation to **SRQ2.1a**, the findings indicate that the effects of OpenJDK JRE source version on JRE JVM power use differ substantially across workload category levels. Although OpenJDK JRE version 13 consistently exhibited elevated power use, the magnitude of this effect varied between workload categories, demonstrating

that workload category moderates the relationship between JRE source version and observed power use.

In relation to **SRQ2.4a**, the findings indicate that differences between interpretive and tiered execution further influence these workload-dependent relationships. Execution mode affected both the magnitude and consistency of the OpenJDK JRE version effect across workload category levels, indicating that the workload-dependent relationship between JRE source version and power use is further conditioned by execution mode.

Subsection Research Question Findings

SRQ2.1a: The effects of OpenJDK JRE source version on JRE JVM power use differed substantially across workload category levels. OpenJDK JRE version 13 consistently exhibited elevated power use, but the magnitude of this increase varied by workload category, indicating that workload category moderated the relationship between JRE source version and observed power use.

SRQ2.4a: Differences between interpretive and tiered execution further conditioned these workload-dependent effects. The magnitude and consistency of the OpenJDK JRE version 13 power use increase varied between execution modes, indicating that execution mode influenced how JRE source version effects manifested across workload category levels.

6.2.2 Energy Consumption

This subsection principally addresses sub-research question **SRQ2.1b**, which examines the extent to which the effects of JRE source version on JRE JVM energy consumption differ across workload category levels:

> **SRQ2.1b:** ***To what extent do the effects of JRE source version on JRE JVM energy consumption differ across***

workload categories?

Because all analyses are additionally decomposed according to execution mode, this subsection also contributes evidence toward sub-research question **SRQ2.4b**, which examines the extent to which differences between interpretive and tiered execution moderate these workload-dependent relationships:

> **SRQ2.4b:** ***To what extent do differences between interpretive and tiered execution affect JRE JVM energy consumption across differing workload categories?***

Descriptive Analysis

Figures 6.3 and 6.4, together with Tables 6.4, 6.5, and 6.6, summarise the distributional characteristics associated with energy consumption across workload categories and OpenJDK JRE versions, reported for pooled observations and further decomposed according to tiered and interpretive execution.

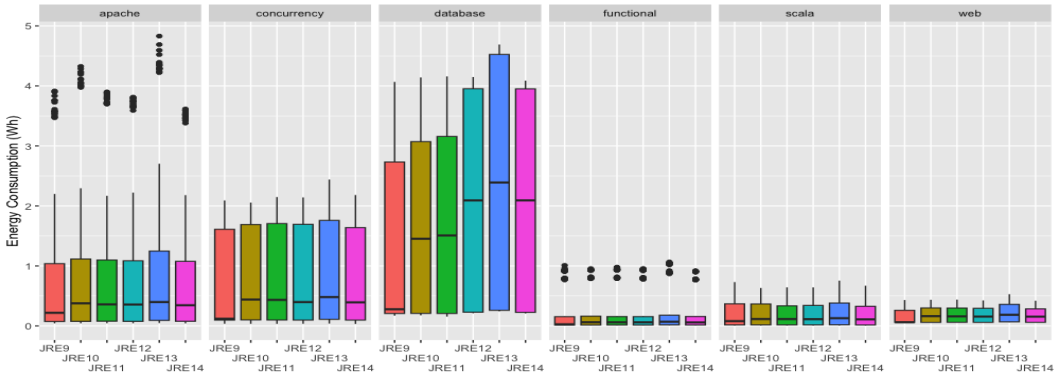


Figure 6.3: Energy Consumption Distributions: Box-and-whisker plots detailing energy consumption distributions, panelled by workload category, and clustered within each workload category by OpenJDK JRE version.

Across all workload category levels, substantial differences in energy consumption are evident, with the `database` workload category consistently exhibiting substantially greater energy consumption relative to all other workload categories. Across pooled observations, Figure 6.3 and Table 6.4 indicate that OpenJDK JRE version 13 is generally associated with elevated energy consumption relative to the

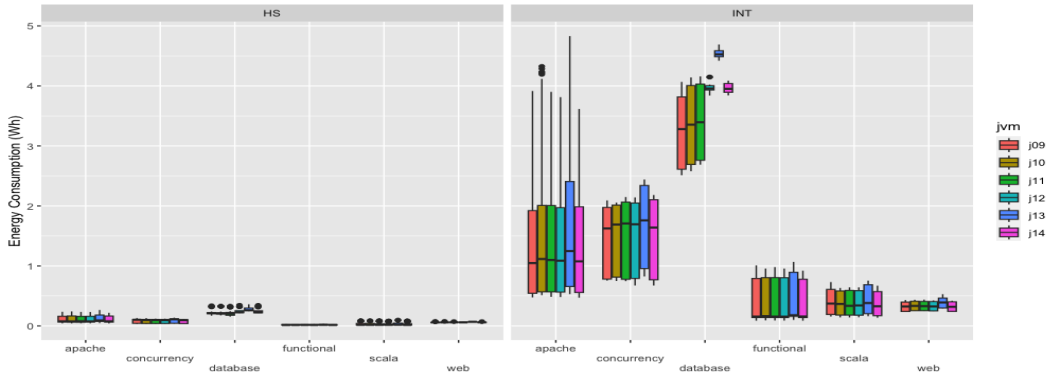


Figure 6.4: Energy Consumption Distributions: Box-and-whisker plots depicting power use distributions for each workload category, clustered within by the OpenJDK JRE version. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT.

remaining JRE versions. However, unlike the corresponding power use results, the magnitude of these differences varies considerably across workload category levels.

The largest increases associated with OpenJDK JRE version 13 are observed within the `database`, `apache`, and `concurrency` workload categories. The strongest effect is observed within the `database` workload category, where median energy consumption increases substantially relative to all preceding OpenJDK JRE versions, continuing the broader pattern of elevated energy consumption previously observed for database-oriented workloads. In contrast, comparatively smaller differences are observed within the `functional`, `scala`, and `web` workload categories, where energy consumption remains relatively stable across OpenJDK JRE versions.

Substantial differences in distributional spread are also evident across workload category levels. The `database`, `apache`, and `concurrency` workload categories exhibit substantially larger interquartile ranges and greater overall dispersion, indicating considerably greater variability in energy consumption observations relative to the remaining workload categories. Conversely, the `functional`, `scala`, and `web` workload categories exhibit comparatively compressed distributions and substantially reduced variability.

Table 6.4: Energy Consumption Distributions (Pooled Observations): Descriptive statistics summarising energy consumption observations across workload categories and OpenJDK JRE versions when observations are pooled across both interpretive and tiered execution modes.

Pooled Observations (Wh)						Pooled Observations (Wh)					
Category	**JRE**	**M**	**M_d**	**SD**	**IQR**	**Category**	**JRE**	**M**	**M_d**	**SD**	**IQR**
apache	j09	0.69	0.22	0.94	0.96	functional	j09	0.20	0.03	0.31	0.14
	j10	0.81	0.38	1.08	1.04		j10	0.22	0.06	0.33	0.15
	j11	0.78	0.36	1.01	1.02		j11	0.22	0.06	0.33	0.14
	j12	0.77	0.36	1.00	1.01		j12	0.22	0.06	0.33	0.14
	j13	0.91	0.40	1.19	1.16		j13	0.25	0.07	0.37	0.16
	j14	0.76	0.34	0.95	1.00		j14	0.22	0.06	0.32	0.14
concurrency	j09	0.71	0.12	0.78	1.51	scala	j09	0.19	0.08	0.23	0.35
	j10	0.80	0.44	0.81	1.59		j10	0.20	0.12	0.22	0.35
	j11	0.80	0.43	0.83	1.61		j11	0.20	0.11	0.22	0.32
	j12	0.80	0.40	0.82	1.60		j12	0.20	0.11	0.22	0.32
	j13	0.88	0.48	0.90	1.65		j13	0.23	0.13	0.26	0.36
	j14	0.80	0.39	0.83	1.54		j14	0.20	0.11	0.22	0.31
database	j09	1.56	0.28	1.57	2.52	web	j09	0.18	0.06	0.14	0.20
	j10	1.78	1.45	1.65	2.86		j10	0.20	0.16	0.15	0.24
	j11	1.81	1.51	1.67	2.95		j11	0.20	0.16	0.15	0.24
	j12	2.11	2.09	1.89	3.72		j12	0.19	0.16	0.15	0.23
	j13	2.41	2.39	2.16	4.26		j13	0.23	0.19	0.17	0.29
	j14	2.11	2.09	1.89	3.73		j14	0.19	0.15	0.14	0.22

Considering differences between execution modes, decomposition of the energy consumption observations reveals a substantially stronger execution-mode effect than was observed for power use. Under tiered execution, energy consumption remains comparatively compressed across all workload category levels, with only modest differences between OpenJDK JRE versions. The largest OpenJDK JRE version 13 increase under tiered execution is observed within the `database` workload category, where median energy consumption increases by approximately $0.05Wh$ relative to the corresponding OpenJDK JRE version 9 to 12 baseline. Smaller increases of approximately $0.01Wh$ are observed within the `apache`, `concurrency`, and `web` workload categories, while little or no median increase is observed within the `functional` and `scala` categories.

Under interpretive execution, the magnitude of the OpenJDK JRE version 13 effect is considerably larger, indicating stronger workload-dependent variation than was observed under tiered execution. The strongest increase is observed within the `database` workload category, where median energy consumption increases by approximately $1.04Wh$ relative to the corresponding OpenJDK JRE version

9 to 12 baseline. More moderate increases are observed within the `apache` and `concurrency` workload categories, where median energy consumption increases by approximately $0.16Wh$ and $0.08Wh$, respectively. Comparatively smaller increases are observed within the `web`, `scala`, and `functional` workload categories.

These results indicate that the workload-dependent effects associated with OpenJDK JRE source version are substantially more pronounced under interpretive execution than under tiered execution. In particular, the `database` workload category exhibits the clearest interaction between workload category levels, OpenJDK JRE version, and execution mode, with OpenJDK JRE version 13 associated with only a modest increase under tiered execution but a substantially larger increase under interpretive execution. This pattern indicates that execution mode further conditions the relationship between JRE source version and energy consumption, with the magnitude of the effect differing substantially across workload category levels.

Differences in distributional spread further reinforce these observations. Under tiered execution, interquartile ranges are generally narrow across most workload categories, indicating relatively stable energy consumption distributions. Under interpretive execution, substantially wider interquartile ranges are observed within the `apache`, `concurrency`, `functional`, and `database` workload categories, indicating greater variability in energy consumption observations. These wider distributions reflect the substantially greater runtime cost associated with interpretive execution and indicate that execution mode influences both the magnitude and variability of workload-dependent energy consumption effects.

Table 6.5: Energy Consumption Distributions (Tiered Execution): Descriptive statistics summarising energy consumption observations across workload categories and OpenJDK JRE versions under tiered execution.

Tiered Observations (Wh)						Tiered Observations (Wh)					
Category	**JRE**	**M**	**M_d**	**SD**	**IQR**	**Category**	**JRE**	**M**	**M_d**	**SD**	**IQR**
apache	j09	0.11	0.08	0.06	0.10	functional	j09	0.02	0.02	0.01	0.01
	j10	0.11	0.08	0.06	0.10		j10	0.02	0.02	0.01	0.01
	j11	0.11	0.08	0.06	0.10		j11	0.02	0.02	0.01	0.01
	j12	0.11	0.08	0.06	0.10		j12	0.02	0.02	0.01	0.01
	j13	0.13	0.09	0.07	0.11		j13	0.02	0.02	0.01	0.01
	j14	0.11	0.08	0.06	0.10		j14	0.02	0.02	0.01	0.01
concurrency	j09	0.08	0.10	0.03	0.07	scala	j09	0.03	0.02	0.03	0.03
	j10	0.08	0.10	0.03	0.07		j10	0.03	0.02	0.03	0.03
	j11	0.08	0.10	0.03	0.07		j11	0.03	0.02	0.03	0.02
	j12	0.08	0.10	0.03	0.07		j12	0.03	0.02	0.03	0.02
	j13	0.09	0.11	0.04	0.08		j13	0.04	0.02	0.03	0.03
	j14	0.08	0.10	0.03	0.07		j14	0.03	0.02	0.03	0.02
database	j09	0.22	0.21	0.04	0.02	web	j09	0.06	0.06	0.00	0.00
	j10	0.22	0.21	0.04	0.02		j10	0.06	0.06	0.00	0.00
	j11	0.21	0.20	0.05	0.05		j11	0.06	0.06	0.00	0.00
	j12	0.25	0.23	0.05	0.03		j12	0.06	0.06	0.00	0.01
	j13	0.28	0.26	0.04	0.04		j13	0.07	0.07	0.00	0.01
	j14	0.25	0.23	0.05	0.03		j14	0.06	0.06	0.00	0.01

Table 6.6: Energy Consumption Distributions (Interpretive Execution): Descriptive statistics summarising energy consumption observations across workload categories and OpenJDK JRE versions under interpretive execution.

Interpretive Observations (Wh)						Interpretive Observations (Wh)					
Category	**JRE**	**M**	**M_d**	**SD**	**IQR**	**Category**	**JRE**	**M**	**M_d**	**SD**	**IQR**
apache	j09	1.41	1.05	1.03	1.38	functional	j09	0.42	0.16	0.36	0.65
	j10	1.51	1.12	1.17	1.44		j10	0.43	0.16	0.36	0.66
	j11	1.45	1.10	1.07	1.44		j11	0.43	0.16	0.37	0.66
	j12	1.44	1.09	1.05	1.41		j12	0.42	0.16	0.37	0.66
	j13	1.70	1.25	1.26	1.75		j13	0.48	0.18	0.41	0.73
	j14	1.41	1.08	0.99	1.43		j14	0.41	0.16	0.35	0.64
concurrency	j09	1.48	1.62	0.53	1.19	scala	j09	0.40	0.37	0.21	0.42
	j10	1.52	1.69	0.53	1.20		j10	0.38	0.37	0.19	0.40
	j11	1.53	1.71	0.56	1.29		j11	0.37	0.33	0.20	0.41
	j12	1.52	1.69	0.56	1.26		j12	0.37	0.34	0.20	0.41
	j13	1.68	1.76	0.61	1.39		j13	0.43	0.38	0.23	0.48
	j14	1.52	1.64	0.58	1.34		j14	0.37	0.33	0.20	0.40
database	j09	3.24	3.28	0.63	1.21	web	j09	0.32	0.32	0.08	0.16
	j10	3.35	3.35	0.69	1.32		j10	0.33	0.33	0.08	0.16
	j11	3.40	3.39	0.64	1.27		j11	0.33	0.33	0.08	0.15
	j12	3.97	3.96	0.08	0.07		j12	0.33	0.32	0.08	0.16
	j13	4.54	4.53	0.08	0.10		j13	0.39	0.39	0.09	0.17
	j14	3.96	3.95	0.08	0.15		j14	0.32	0.32	0.08	0.16

Inferential Statistical Analysis

Assessment of Distributional Assumptions:

Distributional normality was assessed using the Shapiro–Wilk test. The results indicated statistically significant departures from normality across all energy

consumption distributions examined, with all workload category and OpenJDK JRE version combinations returning statistically significant results (all $p < .001$). Given the overall distributional characteristics observed, non-parametric statistical procedures were employed throughout the analysis.

OpenJDK JRE Version Effects Within Workload Categories:

Kruskal–Wallis H tests indicated that differences in median energy consumption across OpenJDK JRE versions were not uniformly observed across workload category levels. Statistically significant differences were identified within the `apache` ($p = .037$), `database` ($p < .001$), and `web` ($p < .001$) workload categories. In contrast, no statistically significant differences were observed within the `concurrency` ($p = .215$), `functional` ($p = .062$), or `scala` ($p = .172$) workload categories.

Pairwise comparisons indicated that these statistically significant differences were predominantly attributable to comparisons involving OpenJDK JRE version 13. Within the `apache` workload category, only the comparison between OpenJDK JRE version 9 and OpenJDK JRE version 13 remained statistically significant following Bonferroni correction ($p = .014$). The strongest workload-dependent effects were observed within the `database` workload category, where OpenJDK JRE version 13 exhibited significantly greater energy consumption relative to OpenJDK JRE versions 9, 10, and 11 (all $p \leq .005$). Similar patterns were observed within the `web` workload category, where OpenJDK JRE version 13 exhibited significantly greater energy consumption relative to OpenJDK JRE versions 9, 10, 11, and 12 (all $p \leq .016$).

Collectively, these findings indicate that the effects associated with OpenJDK JRE source version on energy consumption vary substantially across workload category levels, with statistically significant differences concentrated primarily within the `database` and `web` workload categories. This suggests that workload-specific computational characteristics influence the extent to which OpenJDK JRE source version affects overall JRE JVM energy consumption.

Execution Mode Effects:

Decomposing the analyses according to execution mode revealed additional workload-dependent differences in the relationship between OpenJDK JRE source version and energy consumption. Under tiered execution, statistically significant differences in median energy consumption across OpenJDK JRE versions were observed within the `apache` ($p = .009$), `concurrency` ($p = .008$), `database` ($p < .001$), `functional` ($p = .032$), and `web` ($p < .001$) workload categories. The only exception was the `scala` workload category, which did not exhibit statistically significant differences ($p = .053$).

Under interpretive execution, statistically significant differences were similarly observed within the `apache` ($p = .006$), `database` ($p < .001$), `functional` ($p = .003$), `scala` ($p < .001$), and `web` ($p < .001$) workload categories. In contrast, the `concurrency` workload category did not exhibit statistically significant differences under interpretive execution ($p = .068$).

Across both execution modes, the strongest statistical effects were consistently observed within the `database` and `web` workload categories, indicating that the influence of OpenJDK JRE source version on energy consumption was particularly pronounced for these workload categories. Importantly, several workload categories that did not exhibit statistically significant differences under pooled analysis became statistically significant once decomposed according to execution mode. This indicates that differences between interpretive and tiered execution influence the manifestation of OpenJDK JRE source version effects, revealing additional interaction structure that is obscured when observations are considered in aggregate.

Summary

The descriptive analyses demonstrated that the relationship between OpenJDK JRE source version and JRE JVM energy consumption differed across workload category levels. Across pooled observations, OpenJDK JRE version 13 was gener-

ally associated with elevated energy consumption relative to the remaining JRE versions, but the magnitude of this effect varied considerably between workload categories. The strongest increases were observed within the `database`, `apache`, and `concurrency` workload categories, while comparatively smaller differences were observed within the `functional`, `scala`, and `web` categories. These findings indicate that the effect associated with OpenJDK JRE source version on energy consumption was not uniform across workload category levels.

Decomposing the analysis according to execution mode provided further evidence of workload-dependent variation. Under tiered execution, energy consumption remained comparatively compressed across workload category levels, with only modest differences between OpenJDK JRE versions. In contrast, under interpretive execution, the magnitude of the OpenJDK JRE version 13 effect was substantially larger and more strongly differentiated by workload category. The clearest example was observed within the `database` workload category, where the OpenJDK JRE version 13 increase was modest under tiered execution but substantially larger under interpretive execution. These findings indicate that execution mode further conditioned the workload-dependent relationship between JRE source version and energy consumption.

Inferential statistical analyses supported these observations, while also demonstrating that JRE version effects on energy consumption were less uniform than those observed for power use. Kruskal–Wallis H tests identified statistically significant differences in median energy consumption across OpenJDK JRE versions within the `apache`, `database`, and `web` workload categories, but not within the `concurrency`, `functional`, or `scala` categories when observations were pooled across execution modes. Pairwise comparisons indicated that statistically significant differences were predominantly attributable to comparisons involving OpenJDK JRE version 13, particularly within the `database` and `web` workload categories.

When decomposed according to execution mode, additional statistically signif-

icant relationships were revealed. Under tiered execution, statistically significant differences across OpenJDK JRE versions were observed within all workload categories except `scala`. Under interpretive execution, statistically significant differences were observed within all workload categories except `concurrency`. This indicates that pooling observations across execution modes obscured some workload-specific relationships, and that execution mode influenced how JRE source version effects manifested in relation to energy consumption.

In relation to **SRQ2.1b**, the findings indicate that the effects of OpenJDK JRE source version on JRE JVM energy consumption differ substantially across workload category levels. Statistically significant JRE version effects were concentrated within specific workload categories, particularly `database` and `web`, indicating that workload category moderated the relationship between JRE source version and observed energy consumption.

In relation to **SRQ2.4b**, the findings indicate that differences between interpretive and tiered execution further influenced these workload-dependent relationships. Execution-mode decomposition revealed statistically significant JRE version effects that were not apparent under pooled analysis, indicating that execution mode conditioned both the magnitude and statistical detectability of workload-dependent JRE source version effects on energy consumption.

Subsection Research Question Findings

SRQ2.1b: The effects of OpenJDK JRE source version on JRE JVM energy consumption differed across workload categories. Statistically significant differences were concentrated within specific workload categories, particularly `database` and `web`, indicating that workload category moderated the relationship between JRE source version and observed energy consumption.

SRQ2.4b: Differences between interpretive and tiered execution further conditioned these workload-dependent effects. Execution-mode decomposition revealed additional statistically significant JRE version effects that were obscured under pooled analysis, indicating that execution mode influenced both the magnitude and statistical detectability of workload-dependent energy consumption effects.

6.2.3 Summary

This section examined whether the effects associated with OpenJDK JRE source version on JRE JVM power use and energy consumption differed across workload category levels. Across both outcome variables, the results demonstrate that workload category moderated the relationship between OpenJDK JRE source version and observed power use and energy consumption. However, the nature of this moderation differed between power use and energy consumption.

For power use, statistically significant differences across OpenJDK JRE versions were observed within all workload category levels. These differences were predominantly attributable to OpenJDK JRE version 13, which consistently exhibited elevated power use relative to the remaining JRE versions. However, the magnitude of this increase varied substantially between workload categories, with larger increases observed within the `apache`, `database`, `functional`, and `scala` workload categories, and smaller increases observed within the `concurrency` and

`web` categories. This indicates that the OpenJDK JRE version 13 power use effect was systematic, but not uniform across workload category levels.

For energy consumption, the relationship between OpenJDK JRE source version and workload category was more selective. Statistically significant differences across OpenJDK JRE versions were concentrated within the `apache`, `database`, and `web` workload categories when observations were pooled across execution modes. Pairwise comparisons indicated that these differences were again predominantly associated with OpenJDK JRE version 13, particularly within the `database` and `web` workload categories. In contrast to power use, several workload categories did not exhibit statistically significant JRE version effects under pooled analysis, indicating that energy consumption effects were more strongly conditioned by workload category.

Execution-mode decomposition provided additional evidence that interpretive and tiered execution further conditioned these workload-dependent relationships. For power use, OpenJDK JRE version 13 remained elevated under both execution modes, although the magnitude of the effect differed across workload categories. The clearest execution-mode-dependent difference was observed within the `concurrency` workload category, where the OpenJDK JRE version 13 increase was evident under tiered execution but almost absent under interpretive execution. For energy consumption, execution-mode decomposition revealed additional statistically significant JRE version effects that were obscured under pooled analysis, indicating that execution mode influenced both the magnitude and statistical detectability of workload-dependent JRE version effects.

In relation to **SRQ2.1**, the findings indicate that the effects of OpenJDK JRE source version on both JRE JVM power use and energy consumption differed across workload category levels. Workload category therefore moderated the relationship between JRE source version and both outcome variables. This moderation was broad and systematic for power use, while for energy consumption it was more concentrated within specific workload categories, particularly `database` and `web`.

In relation to **SRQ2.4**, the findings indicate that differences between interpretive and tiered execution further conditioned these workload-dependent relationships. Execution mode influenced the magnitude, consistency, and statistical detectability of OpenJDK JRE version effects, particularly for energy consumption. These results demonstrate that the interaction between OpenJDK JRE source version and workload category cannot be fully understood without also considering execution mode.

Section Research Question Findings

SRQ2.1: The effects of OpenJDK JRE source version on JRE JVM power use and energy consumption differed across workload category levels. For power use, OpenJDK JRE version 13 consistently exhibited elevated values across all workload categories, although the magnitude of this effect varied by workload category levels. For energy consumption, statistically significant JRE version effects were more selectively concentrated within specific workload categories, particularly `database` and `web`. These findings indicate that workload category moderated the relationship between OpenJDK JRE source version and both outcome variables.

SRQ2.4: Differences between interpretive and tiered execution further conditioned the workload-dependent effects associated with OpenJDK JRE source version. Execution mode influenced the magnitude and consistency of JRE version effects for power use, and revealed additional statistically significant energy consumption effects that were obscured under pooled analysis. These findings indicate that execution mode constitutes an additional conditioning factor in the relationship between JRE source version, workload category, power use, and energy consumption.

6.3 GNU GCC version and workload interaction effects

This section contributes to answering **RQ2** by evaluating whether the effects associated with GNU GCC build toolchain compiler source version on JRE JVM power use and energy consumption remain stable across differing workload categories. Specifically, the analyses assess whether workload category moderates the relationship between GNU GCC build toolchain compiler source version and both power use and energy consumption, thereby determining whether the magnitude or direction of compiler version effects depend upon the computational characteristics of the workload being executed.

This section principally addresses sub-research question **SRQ2.2**, which asks:

> **SRQ2.2:** ***To what extent do the effects of GNU GCC build toolchain compiler source version on JRE JVM power use and energy consumption differ across workload categories?***

Because all analyses are additionally decomposed according to interpretive and tiered execution modes, this section also contributes evidence toward understanding the extent to which execution mode influences these workload-dependent relationships, in particular:

> **SRQ2.4:** ***To what extent do differences between interpretive and tiered execution affect JRE JVM power use and energy consumption across differing workload categories?***

To address these questions, the analyses evaluate whether previously observed differences associated with GNU GCC build toolchain compiler source version remain consistent across workload categories or whether workload category alters the magnitude and statistical significance of those effects. The analyses first examine the interaction between GNU GCC build toolchain compiler source version and workload category in relation to power use, followed by an equivalent

analysis of energy consumption. Throughout the section, results are considered across the full set of observations and are further decomposed according to interpretive and tiered execution modes to determine whether observed interaction effects remain consistent across differing execution strategies.

Collectively, these analyses establish the extent to which workload category moderates the relationship between GNU GCC build toolchain compiler source version and both JRE JVM power use and energy consumption under differing runtime conditions.

6.3.1 Power Use

This subsection principally addresses sub-research question **SRQ2.2a**, which examines the extent to which the effects of GNU GCC build toolchain compiler source version on JRE JVM power use differ across workload categories:

> **SRQ2.2a:** ***To what extent do the effects of GNU GCC build toolchain compiler source version on JRE JVM power use differ across workload categories?***

Because all analyses are additionally decomposed according to execution mode, this subsection also contributes evidence toward sub-research question **SRQ2.4a**, which examines the extent to which differences between interpretive and tiered execution moderate these workload-dependent relationships:

> **SRQ2.4a:** ***To what extent do differences between interpretive and tiered execution affect JRE JVM power use across differing workload categories?***

Descriptive Analysis

Figures 6.5 and 6.6, together with Table 6.7, summarise the distributional characteristics associated with power use across workload categories and GNU GCC build

toolchain compiler versions, reported for pooled observations and subsequently decomposed according to tiered and interpretive execution.

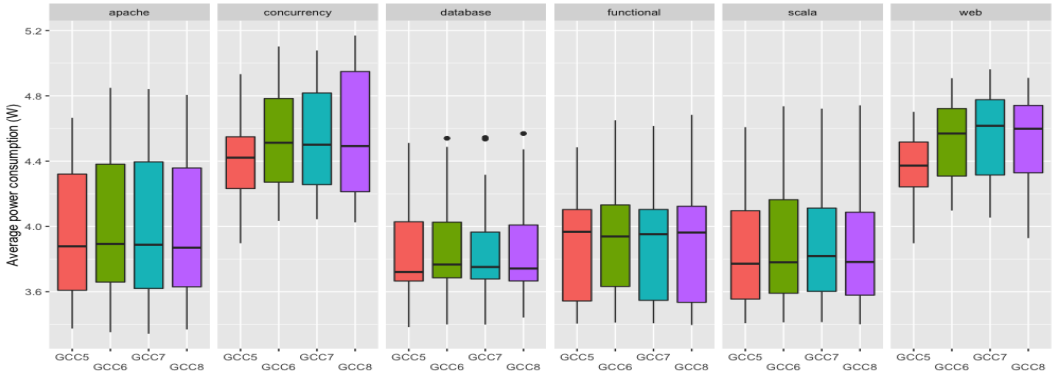


Figure 6.5: Power Use Distributions: Box-and-whisker plots detailing power use distributions, panelled by workload category, and clustered within each workload category by GNU GCC version.

Across all workload category levels, substantial differences in power use are evident, with the `concurrency` and `web` workload categories consistently exhibiting higher power use, while the `database`, `functional`, and `scala` workload categories generally exhibit comparatively lower power use distributions, as illustrated in Figure 6.5 and Table 6.7. In contrast to the previously observed OpenJDK JRE source version effects, variation associated with GNU GCC build toolchain compiler version appears comparatively limited within individual workload categories, suggesting a substantially weaker overall relationship between GNU GCC version and JRE JVM power use.

Examining differences between GNU GCC versions more directly, median power use remains highly consistent across compiler versions within most workload categories. The largest differences are observed within the `web` and `concurrency` workload categories, where median power use under GNU GCC version 5 remains lower than the remaining compiler versions. Within the `web` workload category, median power use increases from approximately $4.37W$ under GNU GCC version 5 to approximately $4.57W$ to $4.62W$ under GNU GCC versions 6 through 8. Similarly, within the `concurrency` workload category, median power use increases modestly from approximately $4.42W$ under GNU GCC version 5 to approximately

4.49W to 4.51W under the remaining compiler versions.

Comparatively smaller differences are observed within the `apache`, `database`, `functional`, and `scala` workload categories, where median power use remains highly consistent across all GNU GCC versions. Within these workload categories, observed differences generally remain below approximately 0.10W, indicating comparatively weak sensitivity to variation in GNU GCC build toolchain compiler version.

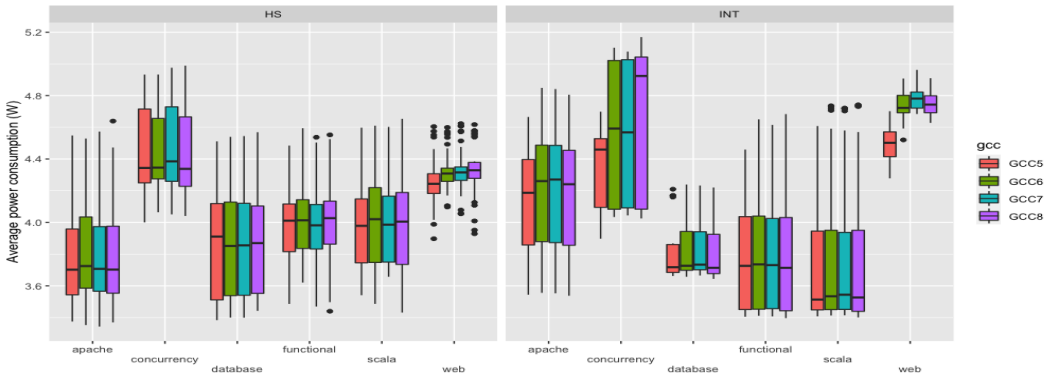


Figure 6.6: Power Use Distributions: Box-and-whisker plots depicting power use distributions for each workload category, clustered within by the GNU GCC version. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT.

Differences in distributional spread are also workload dependent. The `apache`, `concurrency`, and `scala` workload categories exhibit comparatively wider interquartile ranges and greater overall dispersion, indicating greater variability in power use observations across repeated executions. In contrast, the `database` and `web` workload categories exhibit comparatively narrower distributions, suggesting more stable power use behaviour across compiler configurations.

Taken together, these descriptive findings indicate that workload category exerts only limited moderation on the relationship between GNU GCC build toolchain compiler version and JRE JVM power use. Although modest workload-dependent differences are observable, particularly within the `web` and `concurrency` workload categories, overall variation across GNU GCC versions remains comparatively small, suggesting that GNU GCC build toolchain compiler version exerts only a relatively weak influence on JRE JVM power use across differing workload

category levels.

Table 6.7: Power Use Distributions (Pooled Execution (W)): Descriptive statistics summarising power use observations across workload categories and GNU GCC versions under pooled execution.

Pooled Execution (W)						Pooled Execution (W)					
Category	**GCC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GCC**	**M**	**M_d**	**SD**	**IQR**
apache	GCC5	3.95	3.88	0.35	0.71	functional	GCC5	3.89	3.97	0.31	0.56
	GCC6	4.00	3.89	0.39	0.72		GCC6	3.90	3.94	0.32	0.50
	GCC7	4.00	3.89	0.39	0.78		GCC7	3.90	3.95	0.32	0.56
	GCC8	3.98	3.87	0.39	0.73		GCC8	3.91	3.96	0.33	0.59
concurrency	GCC5	4.40	4.42	0.25	0.32	scala	GCC5	3.86	3.77	0.36	0.54
	GCC6	4.52	4.51	0.33	0.51		GCC6	3.89	3.78	0.38	0.57
	GCC7	4.52	4.50	0.34	0.56		GCC7	3.88	3.82	0.37	0.51
	GCC8	4.54	4.49	0.38	0.74		GCC8	3.88	3.78	0.38	0.51
database	GCC5	3.82	3.72	0.27	0.36	web	GCC5	4.37	4.37	0.17	0.27
	GCC6	3.85	3.77	0.26	0.34		GCC6	4.52	4.57	0.23	0.41
	GCC7	3.85	3.75	0.26	0.29		GCC7	4.55	4.62	0.25	0.46
	GCC8	3.84	3.74	0.26	0.34		GCC8	4.53	4.60	0.25	0.41

When examined separately according to execution mode, decomposition of the power use observations presented in Figure 6.6, together with Tables 6.8 and 6.9, reveals that the relationship between GNU GCC build toolchain compiler version and JRE JVM power use differs modestly between tiered and interpretive execution. Under tiered execution, power use remains highly stable across GNU GCC versions within all workload categories, with only negligible differences observed between compiler versions. Across most workload category levels, median power use varies by less than approximately $0.05W$, indicating that GNU GCC build toolchain compiler version exerts only minimal influence on power use under tiered execution.

The greatest differences under tiered execution are observed within the `web` workload category, where median power use increases modestly from approximately $4.24W$ under GNU GCC version 5 to approximately $4.31W$ to $4.33W$ under GNU GCC versions 6 through 8. Comparatively small differences are also observed within the `concurrency` workload category across all compiler versions. Within the remaining workload categories, including `apache`, `functional`, `scala`, and `database`, power use distributions remain highly compressed and exhibit little meaningful variation across GNU GCC versions.

Under interpretive execution, the magnitude of the GNU GCC version effect

becomes somewhat more pronounced within specific workload categories. The strongest differences are observed within the `web` and `concurrency` workload categories, where GNU GCC version 5 is consistently associated with lower power use relative to the remaining compiler versions. Within the `web` workload category, median power use increases from approximately $4.50W$ under GNU GCC version 5 to approximately $4.72W$ to $4.78W$ under GNU GCC versions 6 through 8. Similarly, within the `concurrency` workload category, median power use increases from approximately $4.46W$ under GNU GCC version 5 to approximately $4.57W$ to $4.92W$ under the remaining compiler versions.

Table 6.8: Power Use Distributions (Tiered Execution (W)): Descriptive statistics summarising power use observations across workload categories and GNU GCC versions under tiered execution.

Tiered Execution (W)						**Tiered Execution (W)**					
Category	**GCC**	**M**	**M$_d$**	**SD**	**IQR**	**Category**	**GCC**	**M**	**M$_d$**	**SD**	**IQR**
apache	GCC5	3.78	3.70	0.29	0.42	functional	GCC5	3.98	4.01	0.23	0.30
	GCC6	3.81	3.72	0.28	0.45		GCC6	4.00	4.01	0.23	0.31
	GCC7	3.80	3.71	0.30	0.41		GCC7	3.99	3.98	0.23	0.28
	GCC8	3.79	3.70	0.28	0.42		GCC8	4.00	4.03	0.23	0.27
concurrency	GCC5	4.44	4.34	0.26	0.47	scala	GCC5	3.98	3.98	0.26	0.40
	GCC6	4.45	4.35	0.24	0.38		GCC6	4.00	4.02	0.27	0.47
	GCC7	4.46	4.39	0.27	0.47		GCC7	3.99	3.99	0.25	0.42
	GCC8	4.42	4.34	0.26	0.44		GCC8	3.99	4.01	0.27	0.45
database	GCC5	3.85	3.91	0.35	0.61	web	GCC5	4.25	4.24	0.14	0.12
	GCC6	3.87	3.85	0.33	0.59		GCC6	4.32	4.31	0.11	0.08
	GCC7	3.85	3.86	0.33	0.58		GCC7	4.32	4.32	0.13	0.09
	GCC8	3.86	3.87	0.32	0.55		GCC8	4.32	4.33	0.15	0.10

In contrast, the `apache`, `functional`, `scala`, and `database` workload categories exhibit only limited variation across GNU GCC versions under interpretive execution. Median power use differences within these workload category levels generally remain small, indicating comparatively weak sensitivity to variation in compiler version.

Differences in distributional spread further reinforce these observations. Under tiered execution, distributions remain comparatively narrow across most workload categories, indicating relatively stable power use behaviour across compiler configurations. Under interpretive execution, wider interquartile ranges are particularly evident within the `apache`, `concurrency`, **functional**, and `scala` workload cate-

gories, indicating greater variability in power use observations relative to tiered execution.

Taken together, these descriptive findings indicate that workload category exerts only limited moderating influence on the relationship between GNU GCC build toolchain compiler version and JRE JVM power use. Although modest workload-dependent differences are observable, particularly within the `web` and `concurrency` workload categories, overall variation across GNU GCC versions remains comparatively small. Differences between interpretive and tiered execution introduce some additional variation, but do not substantially alter the overall pattern of comparatively weak workload-dependent GNU GCC effects.

Table 6.9: Power Use Distributions (Interpretive Execution (W)): Descriptive statistics summarising power use observations across workload categories and GNU GCC versions under interpretive execution.

Interpretive Execution (W)						Interpretive Execution (W)					
Category	**GCC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GCC**	**M**	**M_d**	**SD**	**IQR**
apache	GCC5	4.13	4.19	0.32	0.54	functional	GCC5	3.79	3.73	0.35	0.59
	GCC6	4.20	4.26	0.38	0.61		GCC6	3.81	3.74	0.37	0.59
	GCC7	4.20	4.27	0.38	0.61		GCC7	3.82	3.73	0.38	0.57
	GCC8	4.18	4.24	0.38	0.60		GCC8	3.80	3.71	0.38	0.59
concurrency	GCC5	4.36	4.46	0.24	0.43	scala	GCC5	3.74	3.51	0.40	0.50
	GCC6	4.59	4.59	0.39	0.94		GCC6	3.77	3.53	0.44	0.50
	GCC7	4.59	4.57	0.39	0.94		GCC7	3.77	3.54	0.44	0.49
	GCC8	4.66	4.92	0.44	0.96		GCC8	3.76	3.53	0.44	0.51
database	GCC5	3.79	3.72	0.16	0.18	web	GCC5	4.50	4.50	0.10	0.16
	GCC6	3.84	3.73	0.17	0.24		GCC6	4.74	4.72	0.08	0.11
	GCC7	3.84	3.73	0.17	0.24		GCC7	4.78	4.78	0.07	0.10
	GCC8	3.82	3.71	0.17	0.25		GCC8	4.75	4.74	0.07	0.11

Inferential Statistical Analysis

The descriptive analyses identified modest differences in power use across GNU GCC build toolchain compiler versions within individual workload categories, while also indicating some variation in the magnitude of these effects when decomposed according to execution mode. To determine whether these observed differences reflect statistically significant variation, inferential statistical analyses were conducted. We first assess the distributional characteristics associated with the observed power use measurements. Given the resulting departures from normality, non-parametric statistical procedures are subsequently employed to

evaluate differences between GNU GCC versions within workload categories, differences between execution modes, and the extent to which any observed effects vary across workload category levels.

Assessment of Distributional Assumptions:

Distributional normality was assessed using the Shapiro–Wilk test. The results indicated statistically significant departures from normality across the majority of power use distributions examined. Of the twenty-four workload category and GNU GCC build toolchain compiler version combinations assessed, twenty-two exhibited statistically significant departures from normality ($p < .05$). The only exceptions were the `web` workload category executing under GNU GCC version 5 ($W = 0.979$, $p = .115$) and, to a lesser extent, the `concurrency` workload category under GNU GCC version 5, which exhibited only a comparatively modest deviation from normality ($W = 0.979$, $p = .021$). Given the overall distributional characteristics observed, non-parametric statistical procedures were employed throughout the analysis.

GNU GCC Version Effects Within Workload Categories:

Kruskal–Wallis H tests indicated that statistically significant differences in median power use across GNU GCC build toolchain compiler versions were not uniformly observed across workload category levels. Statistically significant differences were identified within the `concurrency` ($p = .010$) and `web` ($p < .001$) workload categories. In contrast, no statistically significant differences were observed within the `apache` ($p = .259$), `database` ($p = .677$), `functional` ($p = .973$), or `scala` ($p = .776$) workload categories.

Pairwise comparisons indicated that the statistically significant differences observed within both the `concurrency` and `web` workload categories were predominantly attributable to comparisons involving GNU GCC version 5. Within the `concurrency` workload category, GNU GCC version 5 exhibited significantly lower power use relative to GNU GCC versions 6, 7, and 8 (all $p < .05$). Similarly,

within the `web` workload category, GNU GCC version 5 exhibited significantly lower power use relative to GNU GCC versions 6, 7, and 8 (all $p < .001$). No statistically significant differences were observed between GNU GCC versions 6, 7, and 8 within either workload category.

Collectively, these findings indicate that workload category moderately influenced the relationship between GNU GCC build toolchain compiler version and JRE JVM power use. However, unlike the stronger workload-dependent effects previously observed for OpenJDK JRE source version, statistically significant GNU GCC version effects were limited to only two workload categories and were primarily attributable to the comparatively lower power use associated with GNU GCC version 5.

Execution Mode Effects:

Decomposition of the analyses according to execution mode revealed additional differences in the extent to which workload category moderated the relationship between GNU GCC build toolchain compiler version and JRE JVM power use.

Under tiered execution, statistically significant differences in median power use across GNU GCC versions were observed only within the `web` workload category ($p = .003$). Pairwise comparisons indicated that these differences were attributable exclusively to GNU GCC version 5, which exhibited significantly lower power use relative to GNU GCC versions 6 ($p = .021$), 7 ($p = .012$), and 8 ($p = .020$). No statistically significant differences were observed within the `apache` ($p = .710$), `concurrency` ($p = .762$), `database` ($p = .958$), `functional` ($p = .888$), or `scala` ($p = .877$) workload categories.

When observations were restricted to interpretive execution, statistically significant differences in median power use across GNU GCC versions were observed within the `apache` ($p = .018$), `concurrency` ($p < .001$), and `web` ($p < .001$) workload categories. Pairwise comparisons again indicated that these differences were predominantly associated with GNU GCC version 5. Within the `apache`

workload category, GNU GCC version 5 exhibited significantly lower power use relative to GNU GCC version 6 ($p = .033$). Within the `concurrency` workload category, GNU GCC version 5 exhibited significantly lower power use relative to GNU GCC versions 6 ($p < .001$), 7 ($p = .002$), and 8 ($p < .001$). Similarly, within the `web` workload category, GNU GCC version 5 exhibited significantly lower power use relative to GNU GCC versions 6, 7, and 8 (all $p < .001$).

Collectively, these findings indicate that workload-dependent GNU GCC version effects were not uniformly distributed across execution modes. Under tiered execution, statistically significant compiler-related differences were highly limited, whereas under interpretive execution statistically significant differences emerged across a greater number of workload categories. Across both execution modes, statistically significant differences were consistently driven by GNU GCC version 5, which repeatedly exhibited lower power use relative to later compiler versions. These findings indicate that workload category moderated the relationship between GNU GCC build toolchain compiler version and JRE JVM power use, although these workload-dependent effects remained comparatively limited and were concentrated within specific workload categories.

Summary

The descriptive analyses indicated that workload category exerted only limited moderating influence on the relationship between GNU GCC build toolchain compiler version and JRE JVM power use. Across pooled observations, substantial differences in power use remained evident between workload categories, with the `concurrency` and `web` workload categories consistently exhibiting higher power use than the remaining workload categories. However, within individual workload categories, power use remained comparatively stable across GNU GCC versions, with only modest variation observed overall. The most notable differences were concentrated within the `web` and `concurrency` workload categories, where GNU GCC version 5 consistently exhibited lower power use relative to GNU GCC

versions 6 through 8.

Inferential statistical analyses largely supported these descriptive observations. Across pooled observations, statistically significant GNU GCC version effects were observed only within the `concurrency` and `web` workload categories, while no statistically significant differences were identified within the `apache`, `database`, `functional`, or `scala` workload categories. Pairwise comparisons demonstrated that these differences were consistently attributable to GNU GCC version 5, which repeatedly exhibited significantly lower power use relative to later compiler versions.

Decomposition according to execution mode revealed additional variation in the extent to which workload category moderated these relationships. Under tiered execution, statistically significant GNU GCC version effects were highly limited and observed only within the `web` workload category. In contrast, under interpretive execution, statistically significant differences emerged within the `apache`, `concurrency`, and `web` workload categories. Across both execution modes, observed differences were again predominantly attributable to GNU GCC version 5, indicating that later GNU GCC versions were generally associated with modestly higher power use under specific workload conditions.

In relation to **SRQ2.2a**, these findings indicate that workload category moderately conditioned the relationship between GNU GCC build toolchain compiler version and JRE JVM power use, although these workload-dependent effects remained comparatively limited and were concentrated within a small subset of workload categories.

In relation to **SRQ2.4a**, the findings indicate that differences between interpretive and tiered execution influenced the magnitude of these workload-dependent relationships, with compiler-related differences becoming more pronounced under interpretive execution than under tiered execution.

Subsection Research Question Findings

SRQ2.2a: Workload category moderately conditioned the relationship between GNU GCC build toolchain compiler source version and JRE JVM power use. However, statistically significant workload-dependent differences were limited to a small number of workload categories, principally `concurrency` and `web`, with GNU GCC version 5 consistently associated with lower power use relative to later compiler versions.

SRQ2.4a: Differences between interpretive and tiered execution influenced the magnitude of workload-dependent GNU GCC version effects on JRE JVM power use. Compiler-related differences were highly limited under tiered execution, but became more pronounced under interpretive execution, indicating that execution mode conditioned the extent to which workload category moderated GNU GCC-related power use differences.

6.3.2 Energy Consumption

This subsection principally addresses sub-research question **SRQ2.2b**, which examines the extent to which the effects of GNU GCC build toolchain compiler source version on JRE JVM energy consumption differ across workload category levels:

> **SRQ2.2b:** ***To what extent do the effects of GNU GCC build toolchain compiler source version on JRE JVM energy consumption differ across workload categories?***

Because all analyses are additionally decomposed according to execution mode, this subsection also contributes evidence toward sub-research question **SRQ2.4b**, which examines the extent to which differences between interpretive and tiered execution moderate these workload-dependent relationships:

SRQ2.4b: ***To what extent do differences between interpretive and tiered execution affect JRE JVM energy consumption across differing workload categories?***

Descriptive Analysis

Figures 6.7 and 6.8, together with Table 6.10, summarise the distributional characteristics associated with JRE JVM energy consumption across workload categories and GNU GCC build toolchain compiler source versions, reported for pooled observations and subsequently decomposed according to tiered and interpretive execution.

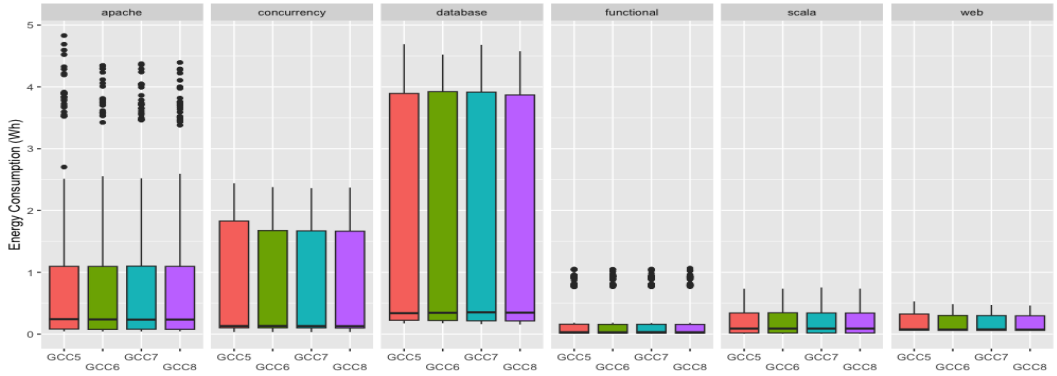


Figure 6.7: Energy Consumption Distributions: Box-and-whisker plots detailing energy consumption distributions, panelled by workload category, and clustered within each workload category by GNU GCC version.

Across all workload category levels, substantial differences in energy consumption are evident, with the `apache`, `concurrency`, and `database` workload categories consistently exhibiting substantially greater energy consumption than the remaining workload categories. In contrast, the `functional`, `scala`, and `web` workload categories exhibit comparatively low overall energy consumption distributions. As illustrated in Figure 6.7 and Table 6.10, differences in energy consumption appear to vary substantially across workload category levels, whereas comparatively little variation is observable across GNU GCC build toolchain compiler versions within individual workload categories.

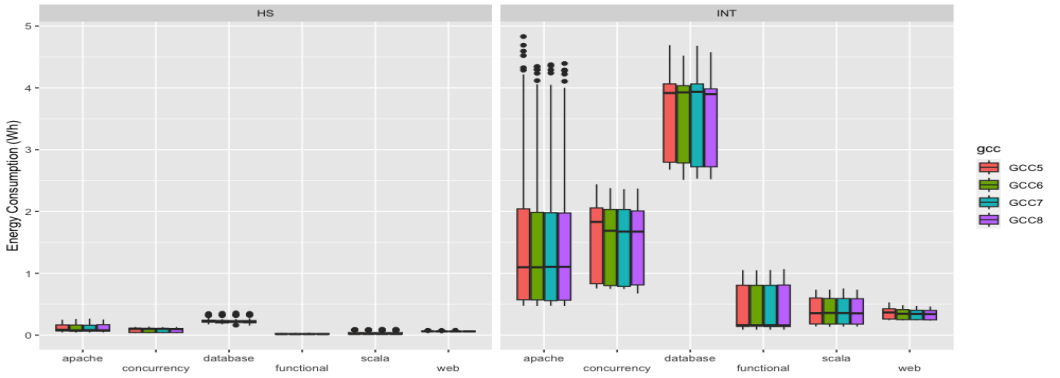


Figure 6.8: Energy Consumption Distributions: Box-and-whisker plots depicting energy consumption distributions for each workload category, clustered by GNU GCC version. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT.

Examining differences between GNU GCC versions more directly, median energy consumption remains highly consistent across compiler versions within all workload categories. Within the `database` workload category, median energy consumption remains stable between approximately $0.34Wh$ and $0.35Wh$ across all GNU GCC versions. Similarly, within the `apache` workload category, median energy consumption remains tightly clustered between approximately $0.23Wh$ and $0.24Wh$, while the `concurrency` workload category exhibits identical median energy consumption of approximately $0.13Wh$ across all compiler versions.

Workload categories associated with comparatively lower overall energy consumption exhibit similarly limited variation across GNU GCC versions. Within the `functional` workload category, median energy consumption remains constant at approximately $0.03Wh$ across all compiler versions, while both the `scala` and `web` workload categories exhibit only negligible variation, with median differences remaining close to zero across all GNU GCC build configurations.

Differences in distributional spread are strongly workload dependent. The `database`, `apache`, and `concurrency` workload categories exhibit substantially wider interquartile ranges and considerably greater overall dispersion, indicating substantially greater variability in observed energy consumption across repeated executions. In contrast, the `functional`, `scala`, and `web` workload categories exhibit comparatively compressed distributions, indicating greater consistency in

observed energy consumption across compiler configurations.

Taken together, these descriptive findings indicate that workload category is associated with substantial variation in JRE JVM energy consumption, while differences associated with GNU GCC build toolchain compiler version remain comparatively limited across workload category levels. Although minor variation is observable across individual compiler versions, the magnitude of these differences remains comparatively small, indicating a high degree of relative stability in JRE JVM energy consumption across GNU GCC build toolchain compiler versions irrespective of workload category.

Table 6.10: Energy Consumption Distributions (Pooled Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and GNU GCC versions under pooled execution.

Pooled Execution (Wh)						Pooled Execution (Wh)					
Category	**GCC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GCC**	**M**	**M_d**	**SD**	**IQR**
apache	GCC5	0.80	0.24	1.07	1.01	functional	GCC5	0.22	0.03	0.33	0.14
	GCC6	0.78	0.24	1.03	1.02		GCC6	0.22	0.03	0.33	0.14
	GCC7	0.78	0.23	1.02	1.02		GCC7	0.22	0.03	0.33	0.14
	GCC8	0.78	0.24	1.01	1.02		GCC8	0.22	0.03	0.33	0.14
concurrency	GCC5	0.83	0.13	0.86	1.73	scala	GCC5	0.21	0.09	0.23	0.32
	GCC6	0.79	0.13	0.82	1.57		GCC6	0.21	0.09	0.23	0.33
	GCC7	0.79	0.13	0.82	1.57		GCC7	0.21	0.09	0.23	0.32
	GCC8	0.78	0.13	0.82	1.57		GCC8	0.20	0.09	0.23	0.32
database	GCC5	1.90	0.34	1.78	3.67	web	GCC5	0.20	0.08	0.16	0.26
	GCC6	1.87	0.34	1.76	3.70		GCC6	0.20	0.07	0.15	0.24
	GCC7	1.87	0.35	1.77	3.70		GCC7	0.19	0.08	0.15	0.24
	GCC8	1.85	0.35	1.75	3.66		GCC8	0.19	0.07	0.15	0.24

When examined separately according to execution mode, decomposition of the energy consumption observations presented in Figure 6.8, together with Tables 6.8 and 6.9, reveals that the relationship between GNU GCC build toolchain compiler version and JRE JVM energy consumption remains highly consistent across both tiered and interpretive execution.

Under tiered execution, energy consumption remains highly stable across GNU GCC versions within all workload categories, with only negligible differences observed between compiler versions. Across all workload category levels, median energy consumption varies only minimally across compiler versions, indicating that GNU GCC build toolchain compiler version exerts little observable influence

on JRE JVM energy consumption under tiered execution.

The greatest overall energy consumption under tiered execution is consistently observed within the `database` workload category, where median energy consumption remains tightly clustered between approximately $0.21Wh$ and $0.22Wh$ across all GNU GCC versions. Similarly, the `apache` and `concurrency` workload categories exhibit only negligible variation, with median energy consumption remaining highly consistent across all compiler versions. The `functional`, `scala`, and `web` workload categories exhibit the lowest overall energy consumption distributions, with median differences across GNU GCC versions remaining effectively unchanged.

Table 6.11: Energy Consumption Distributions (Tiered Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and GNU GCC versions under tiered execution.

Tiered Execution (Wh)						**Tiered Execution (Wh)**					
Category	**GCC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GCC**	**M**	**M_d**	**SD**	**IQR**
apache	GCC5	0.12	0.09	0.06	0.10	functional	GCC5	0.02	0.02	0.01	0.01
	GCC6	0.11	0.08	0.06	0.10		GCC6	0.02	0.02	0.01	0.01
	GCC7	0.11	0.08	0.06	0.10		GCC7	0.02	0.02	0.01	0.01
	GCC8	0.11	0.08	0.06	0.11		GCC8	0.02	0.02	0.01	0.01
concurrency	GCC5	0.09	0.10	0.03	0.07	scala	GCC5	0.03	0.02	0.03	0.03
	GCC6	0.08	0.10	0.03	0.07		GCC6	0.03	0.02	0.03	0.03
	GCC7	0.08	0.10	0.03	0.07		GCC7	0.03	0.02	0.03	0.02
	GCC8	0.08	0.10	0.03	0.06		GCC8	0.03	0.02	0.03	0.02
database	GCC5	0.23	0.22	0.05	0.03	web	GCC5	0.06	0.06	0.01	0.01
	GCC6	0.23	0.22	0.05	0.04		GCC6	0.06	0.06	0.00	0.01
	GCC7	0.23	0.22	0.05	0.03		GCC7	0.06	0.06	0.00	0.01
	GCC8	0.23	0.21	0.05	0.03		GCC8	0.06	0.06	0.01	0.01

Under interpretive execution, substantially greater energy consumption is observed across all workload categories relative to tiered execution, although the overall pattern across GNU GCC versions remains similarly stable. The strongest energy consumption is again observed within the `database` workload category, where median energy consumption remains highly consistent between approximately $3.90Wh$ and $3.94Wh$ across all compiler versions. Similarly, the `apache` and `concurrency` workload categories continue to exhibit comparatively high energy consumption distributions, while the `functional`, `scala`, and `web` workload categories remain associated with comparatively lower overall energy

consumption.

Differences in distributional spread further reinforce these observations. Under tiered execution, energy consumption distributions remain comparatively compressed across all workload categories, indicating relatively consistent observed energy consumption across compiler versions. Under interpretive execution, substantially wider interquartile ranges are evident across all workload categories, particularly within the `database`, `apache`, and `concurrency` workload categories, indicating substantially greater variability in observed energy consumption relative to tiered execution.

Taken together, these descriptive findings indicate that decomposition according to execution mode introduces substantial overall differences in energy consumption magnitude, with interpretive execution consistently associated with substantially greater energy consumption across all workload category levels. However, despite these substantial execution mode differences, JRE JVM energy consumption remains highly stable across GNU GCC build toolchain compiler versions, suggesting that differences between interpretive and tiered execution do not materially alter the overall pattern of comparatively limited GNU GCC-associated variation across workload category levels.

Table 6.12: Energy Consumption Distributions (Interpretive Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and GNU GCC versions under interpretive execution.

Interpretive Execution (Wh)						**Interpretive Execution (Wh)**					
Category	**GCC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GCC**	**M**	**M_d**	**SD**	**IQR**
apache	GCC5	1.52	1.10	1.15	1.47	functional	GCC5	0.43	0.16	0.37	0.67
	GCC6	1.48	1.10	1.09	1.42		GCC6	0.43	0.16	0.37	0.66
	GCC7	1.47	1.10	1.08	1.42		GCC7	0.43	0.16	0.37	0.67
	GCC8	1.47	1.10	1.08	1.41		GCC8	0.43	0.16	0.37	0.67
concurrency	GCC5	1.60	1.83	0.58	1.23	scala	GCC5	0.39	0.35	0.21	0.42
	GCC6	1.53	1.69	0.55	1.23		GCC6	0.39	0.36	0.21	0.41
	GCC7	1.53	1.67	0.56	1.25		GCC7	0.39	0.36	0.20	0.41
	GCC8	1.51	1.67	0.57	1.20		GCC8	0.38	0.35	0.20	0.41
database	GCC5	3.65	3.92	0.67	1.27	web	GCC5	0.35	0.37	0.09	0.17
	GCC6	3.60	3.93	0.68	1.25		GCC6	0.34	0.34	0.08	0.16
	GCC7	3.61	3.94	0.70	1.34		GCC7	0.33	0.34	0.08	0.15
	GCC8	3.57	3.90	0.69	1.26		GCC8	0.33	0.34	0.08	0.15

Inferential Statistical Analysis

The descriptive analyses identified substantial differences in energy consumption across workload category levels, while indicating comparatively limited variation across GNU GCC build toolchain compiler versions within individual workload categories. Additional decomposition according to execution mode further suggested that interpretive and tiered execution may influence the magnitude of these observed differences. To determine whether these observed differences reflect statistically significant variation, inferential statistical analyses were conducted. We first assess the distributional characteristics associated with the observed energy consumption measurements. Given the resulting departures from normality, non-parametric statistical procedures are subsequently employed to evaluate differences between GNU GCC versions within workload categories, differences between execution modes, and the extent to which any observed effects vary across workload categories.

Assessment of Distributional Assumptions:

Distributional normality was assessed using the Shapiro–Wilk test. The results indicated statistically significant departures from normality across all energy consumption distributions examined. All twenty-four workload category and GNU GCC build toolchain compiler version combinations exhibited statistically significant departures from normality ($p < .05$), with test statistics ranging from $W = 0.628$ to $W = 0.801$.

GNU GCC Version Effects Within Workload Categories:

Kruskal–Wallis H tests indicated that statistically significant differences in median energy consumption across GNU GCC build toolchain compiler versions were not observed within any workload category levels. No statistically significant differences were identified within the `apache` ($p = .947$), `concurrency` ($p = .726$), `database` ($p = .882$), `functional` ($p = .997$), `scala` ($p = .977$), or `web` ($p = .271$) workload categories.

Across all workload category levels, median energy consumption remained statistically comparable across GNU GCC versions, indicating that variation in compiler source version was not associated with systematic differences in JRE JVM energy consumption within any individual workload category. These inferential findings are consistent with the earlier descriptive analyses, which indicated only negligible variation in energy consumption across GNU GCC build toolchain compiler versions regardless of workload category level.

Collectively, these findings indicate that workload-dependent GNU GCC version effects were not observed for JRE JVM energy consumption. Unlike the modest workload-specific differences previously observed for power use, energy consumption remained statistically stable across GNU GCC build toolchain compiler versions within all workload category levels, providing no evidence that workload category materially moderates the relationship between GNU GCC build toolchain compiler version and JRE JVM energy consumption.

Execution Mode Effects:

Decomposition of the analyses according to execution mode revealed additional differences in the extent to which workload category influenced the relationship between GNU GCC build toolchain compiler version and JRE JVM energy consumption.

Under tiered execution, Kruskal–Wallis H tests indicated that statistically significant differences in median energy consumption across GNU GCC build toolchain compiler versions were not observed within any workload category level. No statistically significant differences were identified within the `apache` ($p = .688$), `concurrency` ($p = .796$), `database` ($p = .756$), `functional` ($p = .990$), `scala` ($p = .961$), or `web` ($p = .125$) workload categories. These findings indicate that GNU GCC build toolchain compiler version was not associated with statistically significant variation in JRE JVM energy consumption under tiered execution.

When observations were restricted to interpretive execution, statistically significant differences in median energy consumption across GNU GCC versions were

similarly absent across most workload category levels. No statistically significant differences were observed within the `apache` ($p = .945$), `concurrency` ($p = .157$), `database` ($p = .654$), `functional` ($p = .978$), or `scala` ($p = .911$) workload categories. A statistically significant omnibus difference was observed within the `web` workload category ($p = .015$). However, subsequent pairwise comparisons failed to identify statistically significant differences between any individual GNU GCC version comparisons following Bonferroni correction (all $p > .05$), indicating an absence of robust pairwise differences between compiler versions.

Collectively, these findings indicate that workload-dependent GNU GCC version effects were largely absent when analyses were decomposed according to execution mode. Under tiered execution, no statistically significant compiler-related differences were observed across any workload category. Under interpretive execution, only a single workload category exhibited a statistically significant omnibus effect, although no individual pairwise compiler comparisons reached statistical significance following multiple-comparison adjustment. Taken together, these findings provide no evidence that workload category moderates the relationship between GNU GCC build toolchain compiler version and JRE JVM energy consumption. Overall, GNU GCC version effects remained statistically stable across workload category levels irrespective of execution mode.

Summary

The descriptive analysis indicated that JRE JVM energy consumption differed substantially across workload category levels, but remained highly stable across GNU GCC build toolchain compiler versions within each workload category. The `database`, `apache`, and `concurrency` workload categories exhibited the highest overall energy consumption distributions, while the `functional`, `scala`, and `web` workload categories exhibited comparatively lower energy consumption. However, within each workload category, median energy consumption remained highly consistent across GNU GCC versions, indicating little descriptive evidence that

compiler source version substantially affected energy consumption across workload category levels.

When decomposed according to execution mode, interpretive execution was consistently associated with substantially greater energy consumption than tiered execution across all workload category levels. Nevertheless, this execution-mode difference did not materially alter the relationship between GNU GCC build toolchain compiler version and energy consumption. Under tiered execution, energy consumption remained highly stable across GNU GCC versions within all workload categories. Under interpretive execution, the same general pattern was observed, with only limited evidence of compiler-related variation.

Inferential analysis supported these descriptive findings. Shapiro–Wilk tests indicated statistically significant departures from normality across all twenty-four workload category and GNU GCC version combinations, justifying the use of non-parametric procedures. Kruskal–Wallis tests indicated that statistically significant differences in median energy consumption across GNU GCC versions were not observed within any workload category when observations were pooled across execution modes. Similarly, under tiered execution, no statistically significant GNU GCC version effects were observed within any workload category levels. Under interpretive execution, only the `web` workload category exhibited a statistically significant omnibus effect; however, subsequent pairwise comparisons failed to identify statistically significant differences between any individual GNU GCC version comparisons following Bonferroni correction.

n relation to **SRQ2.2b**, these findings indicate that the effects of GNU GCC build toolchain compiler version on JRE/JVM energy consumption do not differ meaningfully across workload categories. Although workload category itself is associated with substantial differences in energy consumption, the effects associated with GNU GCC build toolchain compiler version remained statistically stable across workload categories.

In relation to **SRQ2.4b**, the findings indicate that differences between in-

terpretive and tiered execution strongly affected the overall magnitude of JRE JVM energy consumption, with interpretive execution consistently associated with substantially greater energy consumption. However, execution mode did not provide evidence of a meaningful change in the relationship between GNU GCC version and energy consumption across workload category levels.

Subsection Research Question Findings

SRQ2.2b: The effects of GNU GCC build toolchain compiler source version on JRE JVM energy consumption did not differ meaningfully across workload categories. Energy consumption varied substantially by workload category, but remained statistically stable across GNU GCC versions within workload categories.

SRQ2.4b: Differences between interpretive and tiered execution substantially affected the overall magnitude of JRE JVM energy consumption, with interpretive execution consistently associated with greater energy consumption. However, execution mode did not provide evidence that the GNU GCC version–energy consumption relationship differed across workload category levels.

6.3.3 Summary

This section examined whether the effects associated with GNU GCC build toolchain compiler source version on JRE JVM power use and energy consumption differed across workload category levels, thereby addressing **SRQ2.2**. Because all analyses were additionally decomposed according to tiered and interpretive execution, the section also contributed evidence toward **SRQ2.4**, which concerns the extent to which execution mode affects these workload-dependent relationships.

For power use, the descriptive and inferential analyses indicated that workload category moderately conditioned the relationship between GNU GCC build toolchain compiler version and JRE JVM power use. Across pooled observations,

statistically significant GNU GCC version effects were observed only within the `concurrency` and `web` workload categories. These differences were primarily attributable to GNU GCC version 5, which exhibited lower power use relative to later compiler versions. When decomposed according to execution mode, these compiler-related differences were highly limited under tiered execution but became more pronounced under interpretive execution, where statistically significant differences emerged within the `apache`, `concurrency`, and `web` workload categories.

For energy consumption, a different pattern was observed. Although energy consumption differed substantially across workload category levels, GNU GCC version effects remained statistically stable across workload categories. No statistically significant differences in median energy consumption across GNU GCC versions were observed within any workload category level when observations were pooled across execution modes. Similarly, under tiered execution, no statistically significant GNU GCC version effects were observed. Under interpretive execution, only the `web` workload category exhibited a statistically significant omnibus effect, but no individual pairwise compiler comparisons remained statistically significant following Bonferroni correction.

In relation to **SRQ2.2**, these findings indicate that the effects of GNU GCC build toolchain compiler source version differed across workload category levels for JRE JVM power use, but not for JRE JVM energy consumption. Workload category therefore provided evidence of limited moderation of GNU GCC-related power use differences, while providing no substantive evidence of moderation for energy consumption.

In relation to **SRQ2.4**, the findings indicate that execution mode influenced the magnitude of workload-dependent GNU GCC effects for power use, with compiler-related differences becoming more pronounced under interpretive execution. For energy consumption, execution mode strongly affected the overall magnitude of energy consumption, with interpretive execution consistently associated with substantially greater energy consumption; however, it did not materially alter the

relationship between GNU GCC version and workload category.

Section Research Question Findings

SRQ2.2: The effects of GNU GCC build toolchain compiler source version differed across workload category levels for JRE JVM power use, but not for JRE JVM energy consumption. Workload-dependent power use differences were limited and concentrated primarily within the `concurrency` and `web` workload categories, with GNU GCC version 5 generally associated with lower power use. In contrast, energy consumption remained statistically stable across GNU GCC versions within workload categories.

SRQ2.4: Differences between interpretive and tiered execution affected the magnitude of workload-dependent GNU GCC version effects for JRE JVM power use, with compiler-related differences becoming more pronounced under interpretive execution. For energy consumption, interpretive execution substantially increased overall energy consumption, but did not provide evidence that the GNU GCC version–energy consumption relationship differed across workload category levels.

6.4 Garbage collector variant and workload interaction effects

This section contributes to answering **RQ2** by evaluating whether the effects associated with garbage collector variant on JRE JVM power use and energy consumption remain stable across differing workload categories. Specifically, the analyses assess whether workload category moderates the relationship between garbage collector variant and both power use and energy consumption, thereby determining whether the magnitude or direction of garbage collector effects depend

upon the computational characteristics of the workload being executed.

This section principally addresses sub-research question **SRQ2.3**, which asks:

> **SRQ2.3:** ***To what extent do the effects of garbage collector variant on JRE JVM power use and energy consumption differ across workload categories?***

Because all analyses are additionally decomposed according to interpretive and tiered execution modes, this section also contributes evidence toward understanding the extent to which execution mode influences these workload-dependent relationships, in particular:

> **SRQ2.4:** ***To what extent do differences between interpretive and tiered execution affect JRE JVM power use and energy consumption across differing workload categories?***

To address these questions, the analyses evaluate whether previously observed differences associated with garbage collector variant remain consistent across workload categories or whether workload category alters the magnitude and statistical significance of those effects. The analyses first examine the interaction between garbage collector variant and workload category in relation to power use, followed by an equivalent analysis of energy consumption. Throughout the section, results are considered across the full set of observations and are further decomposed according to interpretive and tiered execution modes to determine whether observed interaction effects remain consistent across differing execution strategies.

Collectively, these analyses establish the extent to which workload category moderates the relationship between garbage collector variant and both JRE JVM power use and energy consumption under differing runtime conditions.

6.4.1 Power Use

This subsection principally addresses sub-research question **SRQ2.3a**, which examines the extent to which the effects of garbage collector variant on JRE JVM power use differ across workload category levels:

> **SRQ2.3a:** ***To what extent do the effects of garbage collector variant on JRE JVM power use differ across workload categories?***

Because all analyses are additionally decomposed according to execution mode, this subsection also contributes evidence toward sub-research question **SRQ2.4a**, which examines the extent to which differences between interpretive and tiered execution moderate these workload-dependent relationships:

> **SRQ2.4a:** ***To what extent do differences between interpretive and tiered execution affect JRE JVM power use across differing workload categories?***

Descriptive Analysis

Figures 6.9 and 6.10, together with Table 6.13, summarise the distributional characteristics associated with power use across workload categories and garbage collector variants, reported for pooled observations and subsequently decomposed according to tiered and interpretive execution.

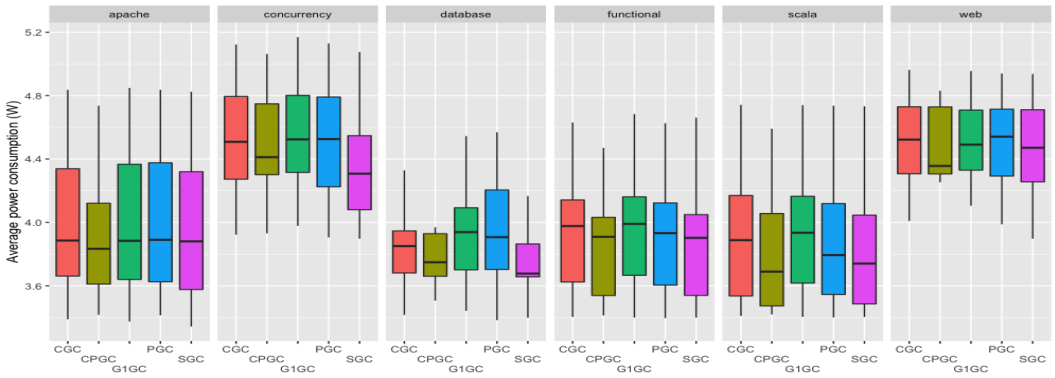


Figure 6.9: Power Use Distributions: Box-and-whisker plots detailing power use distributions, panelled by workload category, and clustered within each workload category by Garbage Collector variant.

Across all workload category levels, substantial differences in power use are evident, with the `concurrency` and `web` workload categories consistently exhibiting comparatively greater power use, while the `database`, `functional`, and `scala` workload categories generally exhibit comparatively lower power use distributions, as illustrated in Figure 6.9 and Table 6.13. Although workload category remains the dominant source of variation, observable differences associated with garbage collector variant appear somewhat more pronounced than those previously observed for GNU GCC build toolchain compiler version, suggesting that garbage collector selection may exert a comparatively greater influence on JRE JVM power use under certain workload conditions.

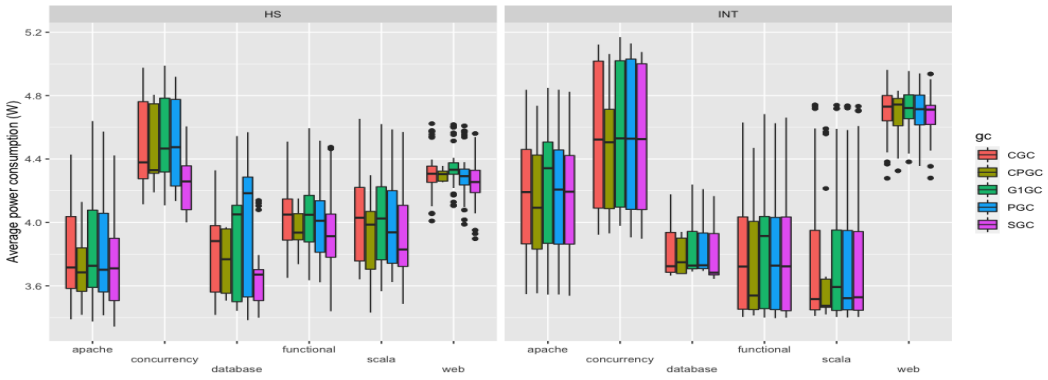


Figure 6.10: Power Use Distributions: Box-and-whisker plots depicting power use distributions for each workload category, clustered within by the Garbage Collector variant. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT.

Examining differences between garbage collector variants more directly, median power use remains broadly consistent across garbage collector variants within most workload categories, although several workload-specific differences are observable. The strongest differences are evident within the `concurrency`, `database`, and `functional` workload categories, where the `SGC` and `CPGC` garbage collector variants consistently exhibit comparatively lower power use relative to the remaining collector variants. Within the `database` workload category, median power use varies from approximately $3.68W$ under `SGC` to approximately $3.94W$ under `G1GC`. Similarly, within the `concurrency` workload category, median power use increases from approximately $4.31W$ under `SGC` to approximately $4.53W$ under

PGC, while within the functional workload category median power use varies from approximately 3.90W under SGC to approximately 3.99W under G1GC.

Comparatively smaller differences are observed within the apache, scala, and web workload categories, where median power use remains relatively stable across garbage collector variants. Within these workload categories, observed differences generally remain modest, indicating comparatively limited sensitivity to variation in garbage collector variant.

Table 6.13: Power Use Distributions (Pooled Execution (W)): Descriptive statistics summarising power use observations across workload categories and Garbage Collector variants under pooled execution (w).

Pooled Execution (W)						Pooled Execution (W)					
Category	**GC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GC**	**M**	**M_d**	**SD**	**IQR**
apache	CGC	3.99	3.89	0.37	0.68	functional	CGC	3.92	3.98	0.32	0.52
	CPGC	3.91	3.83	0.35	0.51		CPGC	3.85	3.91	0.28	0.49
	G1GC	4.00	3.88	0.39	0.73		G1GC	3.93	3.99	0.32	0.50
	PGC	4.00	3.89	0.39	0.75		PGC	3.90	3.93	0.32	0.52
	SGC	3.95	3.88	0.38	0.74		SGC	3.86	3.90	0.32	0.51
concurrency	CGC	4.52	4.51	0.33	0.52	scala	CGC	3.90	3.89	0.38	0.63
	CPGC	4.48	4.41	0.32	0.45		CPGC	3.81	3.69	0.35	0.58
	G1GC	4.54	4.52	0.32	0.49		G1GC	3.92	3.93	0.38	0.55
	PGC	4.53	4.53	0.33	0.57		PGC	3.88	3.79	0.38	0.57
	SGC	4.39	4.31	0.33	0.47		SGC	3.84	3.74	0.36	0.56
database	CGC	3.83	3.85	0.22	0.27	web	CGC	4.51	4.52	0.24	0.42
	CPGC	3.77	3.75	0.17	0.27		CPGC	4.48	4.36	0.23	0.42
	G1GC	3.89	3.94	0.29	0.39		G1GC	4.51	4.49	0.22	0.38
	PGC	3.93	3.91	0.31	0.50		PGC	4.50	4.54	0.24	0.42
	SGC	3.73	3.68	0.20	0.21		SGC	4.47	4.47	0.25	0.45

Differences in distributional spread are also workload dependent. The apache, scala, functional, and concurrency workload categories exhibit comparatively wider interquartile ranges and greater overall dispersion, indicating greater variability in power use observations across repeated executions. In contrast, the database workload category exhibits comparatively narrower distributions across all garbage collector variants, suggesting comparatively greater stability in observed power use. The web workload category remains comparatively stable overall, although moderate variation is observable across individual garbage collector variants.

Taken together, these descriptive findings indicate that workload category exerts a modest moderating influence on the relationship between garbage collector variant and JRE JVM power use. Although workload category remains the princi-

pal determinant of overall power use variation, observable workload-dependent differences between garbage collector variants are more pronounced than those previously observed for GNU GCC build toolchain compiler version. In particular, the `SGC` and `CPGC` configurations repeatedly exhibit comparatively lower power use within several workload categories, suggesting that garbage collector variant may exert a modest but non-uniform influence on JRE JVM power use across differing workload category levels.

Table 6.14: Power Use Distributions (Tiered Execution (W)): Descriptive statistics summarising power use observations across workload categories and Garbage Collector variants under tiered execution (w).

Tiered Execution (W)						**Tiered Execution (W)**					
Category	**GC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GC**	**M**	**M_d**	**SD**	**IQR**
apache	CGC	3.80	3.72	0.26	0.45	functional	CGC	4.04	4.05	0.21	0.26
	CPGC	3.72	3.68	0.20	0.27		CPGC	3.96	3.94	0.13	0.17
	G1GC	3.84	3.73	0.31	0.49		G1GC	4.03	4.05	0.23	0.29
	PGC	3.82	3.70	0.32	0.50		PGC	3.99	4.01	0.24	0.32
	SGC	3.73	3.71	0.25	0.39		SGC	3.92	3.91	0.23	0.27
concurrency	CGC	4.49	4.38	0.25	0.49	scala	CGC	4.03	4.03	0.26	0.46
	CPGC	4.47	4.33	0.24	0.44		CPGC	3.92	3.99	0.24	0.36
	G1GC	4.53	4.47	0.25	0.47		G1GC	4.03	4.02	0.26	0.46
	PGC	4.50	4.47	0.25	0.55		PGC	3.99	3.94	0.26	0.46
	SGC	4.25	4.26	0.15	0.28		SGC	3.92	3.83	0.25	0.39
database	CGC	3.83	3.88	0.26	0.42	web	CGC	4.31	4.31	0.14	0.10
	CPGC	3.75	3.77	0.22	0.41		CPGC	4.30	4.30	0.04	0.06
	G1GC	3.93	4.05	0.35	0.61		G1GC	4.35	4.33	0.12	0.07
	PGC	4.03	4.18	0.39	0.76		PGC	4.30	4.29	0.14	0.10
	SGC	3.66	3.67	0.20	0.20		SGC	4.26	4.25	0.14	0.14

When examined separately according to execution mode, decomposition of the power use observations presented in Figure 6.10, together with Tables 6.14 and 6.15, reveals that the relationship between garbage collector variant and JRE JVM power use differs modestly between tiered and interpretive execution. In contrast to the previously observed GNU GCC interaction effects, differences associated with garbage collector selection appear somewhat more pronounced under tiered execution, while becoming comparatively compressed under interpretive execution.

Table 6.15: Power Use Distributions (Interpretive Execution (W)): Descriptive statistics summarising power use observations across workload categories and Garbage Collector variants under interpretive execution (w).

Interpretive Execution (W)						Interpretive Execution (W)					
Category	**GCC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GCC**	**M**	**M_d**	**SD**	**IQR**
apache	CGC	4.18	4.19	0.37	0.60	functional	CGC	3.81	3.72	0.37	0.58
	CPGC	4.11	4.09	0.36	0.59		CPGC	3.74	3.54	0.35	0.56
	G1GC	4.20	4.34	0.37	0.64		G1GC	3.82	3.91	0.37	0.58
	PGC	4.18	4.21	0.37	0.59		PGC	3.81	3.73	0.37	0.58
	SGC	4.17	4.19	0.36	0.56		SGC	3.81	3.72	0.38	0.59
concurrency	CGC	4.56	4.52	0.39	0.93	scala	CGC	3.77	3.52	0.43	0.50
	CPGC	4.50	4.51	0.40	0.63		CPGC	3.70	3.47	0.42	0.18
	G1GC	4.56	4.53	0.38	0.92		G1GC	3.78	3.59	0.44	0.51
	PGC	4.56	4.53	0.40	0.95		PGC	3.76	3.52	0.43	0.50
	SGC	4.54	4.53	0.39	0.92		SGC	3.76	3.53	0.43	0.50
database	CGC	3.82	3.72	0.17	0.25	web	CGC	4.70	4.73	0.15	0.16
	CPGC	3.79	3.75	0.12	0.22		CPGC	4.66	4.74	0.19	0.17
	G1GC	3.84	3.73	0.19	0.24		G1GC	4.71	4.72	0.13	0.15
	PGC	3.83	3.73	0.16	0.22		PGC	4.69	4.71	0.14	0.19
	SGC	3.80	3.68	0.17	0.26		SGC	4.67	4.71	0.13	0.12

Under tiered execution, observable differences in power use across garbage collector variants emerge within several workload categories. The strongest differences are evident within the `database`, `concurrency`, `functional`, and `scala` workload categories. Within the `database` workload category, median power use varies substantially from approximately $3.67W$ under `SGC` to approximately $4.18W$ under `PGC`, representing the largest workload-specific difference observed. Similarly, within the `concurrency` workload category, median power use ranges from approximately $4.26W$ under `SGC` to approximately $4.47W$ under both `G1GC` and `PGC`. Within both the `functional` and `scala` workload categories, the `CPGC` and `SGC` collectors again exhibit comparatively lower median power use relative to the remaining collector variants.

Comparatively smaller differences are observed within the `apache` and `web` workload categories under tiered execution. Although some variation remains visible across collector configurations, overall median differences remain comparatively modest, indicating relatively weaker sensitivity to garbage collector selection within these workload category levels.

Under interpretive execution, the magnitude of garbage collector-related differences becomes noticeably reduced across most workload categories. Median power

use remains comparatively stable across collector variants within the `concurrency`, `database`, `scala`, and `web` workload categories, indicating substantially weaker collector-specific variation relative to that observed under tiered execution. Within these workload category levels, median differences between garbage collector variants remain comparatively small overall.

The most visible interpretive execution differences occur within the `apache` workloads and `functional` workload categories. Within the `apache` workload category, median power use ranges from approximately $4.09W$ under `CPGC` to approximately $4.34W$ under `G1GC`. Similarly, within the `functional` workload category, median power use varies from approximately $3.54W$ under `CPGC` to approximately $3.91W$ under `G1GC`. Outside these isolated workload categories, garbage collector-related variation remains comparatively limited under interpretive execution.

Differences in distributional spread further reinforce these observations. Under tiered execution, comparatively wider interquartile ranges are evident within the `apache`, `scala`, `database`, and `concurrency` workload categories, indicating greater variability in observed power use across garbage collector variants. Under interpretive execution, distributions remain comparatively similar across collector variants within most workload categories, suggesting greater overall stability in power use behaviour despite differences in garbage collector selection.

Taken together, these descriptive findings indicate that workload category exerts a modest moderating influence on the relationship between garbage collector variant and JRE JVM power use. Unlike the previously observed GNU GCC interaction effects, garbage collector-related differences appear somewhat more pronounced within several workload categories, particularly under tiered execution. However, workload category remains the dominant determinant of overall variation, and the magnitude of garbage collector-related differences remains comparatively moderate rather than substantial across differing runtime conditions.

Inferential Statistical Analysis

Assessment of Distributional Assumptions:

Distributional normality was assessed using the Shapiro–Wilk test. The results indicated statistically significant departures from normality across the majority of power use distributions examined. Of the thirty workload category and garbage collector variant combinations assessed, twenty-eight exhibited statistically significant departures from normality ($p < .05$). The only exceptions were the `concurrency` workload category executing under the `CPGC` garbage collector, which did not exhibit evidence of a statistically significant deviation from normality ($W = 0.955$, $p = .342$), and the `database` workload category executing under the `CGC` collector, which similarly remained statistically consistent with normality ($W = 0.969$, $p = .070$). Given the overall distributional characteristics observed, non-parametric statistical procedures were employed throughout the analysis.

Garbage Collector Effects Within Workload Categories:

Kruskal–Wallis H tests were conducted to determine whether statistically significant differences in median power use were observed across garbage collector variants within individual workload category levels. The results indicated that statistically significant differences in median power use were not uniformly distributed across workload categories. Statistically significant differences were identified within the `concurrency` ($p < .001$) and `database` ($p < .001$) workload categories. In contrast, no statistically significant differences were observed within the `apache` ($p = .219$), `functional` ($p = .066$), `scala` ($p = .142$), or `web` ($p = .565$) workload categories.

Pairwise comparisons indicated that the statistically significant differences observed within both the `concurrency` and `database` workload categories were primarily attributable to the comparatively lower power use associated with the `SGC` garbage collector variant. Within the `concurrency` workload category, `SGC` exhibited significantly lower power use relative to `CGC` ($p = .003$), `G1GC` ($p < .001$),

and PGC ($p = .004$). No statistically significant differences were observed between the remaining garbage collector variants.

Similarly, within the database workload category, SGC again exhibited significantly lower power use relative to CGC ($p = .022$), G1GC ($p = .001$), and PGC ($p < .001$). The strongest difference was observed between SGC and PGC, while no statistically significant differences were observed between CPGC and the remaining garbage collector variants following multiple-comparison adjustment.

Collectively, these findings indicate that workload category moderately influenced the relationship between garbage collector variant and JRE JVM power use. In contrast to the comparatively weaker workload-dependent effects previously observed for GNU GCC build toolchain compiler version, garbage collector-related differences appear more pronounced, although statistically significant effects remain concentrated within a limited subset of workload categories. Across both statistically significant workload category levels, the SGC garbage collector consistently exhibited comparatively lower power use, suggesting that garbage collector selection exerts a modest but non-uniform workload-dependent influence on JRE JVM power use under specific runtime conditions.

Execution Mode Effects:

Decomposition of the analyses according to execution mode revealed substantial differences in the extent to which workload category moderated the relationship between garbage collector variant and JRE JVM power use.

Under tiered execution, Kruskal–Wallis H tests indicated statistically significant differences in median power use across garbage collector variants within all workload category levels, including apache ($p = .016$), concurrency ($p < .001$), database ($p < .001$), functional ($p < .001$), scala ($p = .002$), and web ($p = .004$). Pairwise comparisons indicated that these differences were predominantly attributable to comparisons involving the SGC garbage collector variant. Within the apache workload category, SGC exhibited significantly lower power use relative to G1GC ($p = .016$). Within the concurrency workload category, SGC

exhibited significantly lower power use relative to `CGC` ($p < .001$), `CPGC` ($p = .018$), `G1GC` ($p < .001$), and `PGC` ($p < .001$). Within the `database` workload category, `SGC` exhibited significantly lower power use relative to `G1GC` ($p = .045$) and `PGC` ($p = .003$). Within the `functional` workload category, `SGC` exhibited significantly lower power use relative to `CGC` ($p < .001$) and `G1GC` ($p = .002$). Within the `scala` workload category, `SGC` exhibited significantly lower power use relative to `CGC` ($p = .010$) and `G1GC` ($p = .004$). Finally, within the `web` workload category, `SGC` exhibited significantly lower power use relative to `G1GC` ($p = .003$).

These findings indicate that under tiered execution, garbage collector-related differences in power use became substantially more pronounced than observed within the pooled analyses. Across all workload category levels, statistically significant differences emerged, with the majority of pairwise differences consistently attributable to comparatively lower power use associated with the `SGC` garbage collector variant. This pattern suggests that tiered execution substantially strengthens the extent to which workload category conditions the relationship between garbage collector selection and JRE JVM power use.

When observations were restricted to interpretive execution, statistically significant differences in median power use across garbage collector variants were largely absent across workload category levels. No statistically significant differences were observed within the `apache` ($p = .710$), `concurrency` ($p = .858$), `functional` ($p = .985$), `scala` ($p = .919$), or `web` ($p = .672$) workload categories. A statistically significant omnibus difference was observed only within the `database` workload category ($p = .031$). However, subsequent pairwise comparisons failed to identify statistically significant differences between any individual garbage collector comparisons following Bonferroni correction (all $p > .05$), indicating an absence of robust pairwise differences between specific garbage collector variants.

Collectively, these findings indicate that workload-dependent garbage collector effects on JRE JVM power use were strongly conditioned by execution mode. Under tiered execution, statistically significant garbage collector effects emerged across

all workload category levels, with the majority of pairwise differences consistently attributable to comparatively lower power use associated with the `SGC` garbage collector variant. In contrast, under interpretive execution, statistically significant differences were largely absent, with only a single workload category exhibiting a statistically significant omnibus effect that was not supported by subsequent pairwise comparisons. Taken together, these findings indicate that differences between interpretive and tiered execution substantially influenced the extent to which workload category moderated the relationship between garbage collector variant and JRE JVM power use, with garbage collector-related effects becoming markedly more pronounced under tiered execution.

Summary

The descriptive analysis indicated that workload category exerted a modest moderating influence on the relationship between garbage collector variant and JRE JVM power use. Across pooled observations, substantial differences in power use remained evident between workload categories, with the `concurrency` and `web` workload categories generally exhibiting greater power use than the remaining workload category levels. Within individual workload categories, observable differences across garbage collector variants were also present, particularly within the `concurrency`, `database`, and `functional` workload categories. The `SGC` and `CPGC` configurations repeatedly exhibited comparatively lower power use relative to the remaining collector variants.

When decomposed according to execution mode, descriptive differences associated with garbage collector variant were more pronounced under tiered execution than under interpretive execution. Under tiered execution, garbage collector-related differences were visible across several workload categories, with the largest differences observed within the `database`, `concurrency`, `functional`, and `scala` workload categories. Under interpretive execution, differences across garbage collector variants were noticeably compressed across most workload categories,

indicating weaker collector-specific variation in observed power use.

Inferential analysis supported these descriptive findings. Shapiro–Wilk tests indicated statistically significant departures from normality across twenty-eight of the thirty workload category and garbage collector combinations, justifying the use of non-parametric statistical procedures. Across pooled observations, Kruskal–Wallis tests identified statistically significant garbage collector effects within the `concurrency` and `database` workload categories only. Pairwise comparisons indicated that these differences were primarily attributable to the comparatively lower power use associated with `SGC`, which exhibited significantly lower power use relative to several other collector variants within both workload categories.

Execution-mode decomposition revealed a stronger pattern. Under tiered execution, statistically significant differences in median power use across garbage collector variants were observed within all workload category levels. Pairwise comparisons again indicated that these differences were predominantly attributable to `SGC`, which repeatedly exhibited significantly lower power use relative to `CGC`, `G1GC`, and/or `PGC`, depending on workload category. In contrast, under interpretive execution, statistically significant garbage collector effects were largely absent. Only the `database` workload category exhibited a statistically significant omnibus effect, and this was not supported by statistically significant pairwise comparisons following Bonferroni correction.

In relation to **SRQ2.3a**, these findings indicate that the effects of garbage collector variant on JRE JVM power use differed across workload categories. However, these workload-dependent effects were not uniformly distributed across all workloads. They were most evident within the `concurrency` and `database` workload categories in the pooled analysis and became more widespread under tiered execution.

In relation to **SRQ2.4a**, the findings indicate that differences between interpretive and tiered execution substantially affected the extent to which workload category moderated the relationship between garbage collector variant and JRE

JVM power use. Garbage collector-related differences were strongest and most consistently statistically significant under tiered execution, whereas under interpretive execution these effects were largely absent.

Subsection Research Question Findings

SRQ2.3a: The effects of garbage collector variant on JRE JVM power use differed across workload category levels. Workload-dependent effects were most evident within the `concurrency` and `database` workload categories under pooled execution and became more widespread under tiered execution. Across these effects, `SGC` was repeatedly associated with lower power use relative to several other collector variants.

SRQ2.4a: Differences between interpretive and tiered execution substantially affected workload-dependent garbage collector effects on JRE JVM power use. Under tiered execution, statistically significant garbage collector effects were observed across all workload category levels, whereas under interpretive execution these effects were largely absent or not supported by pairwise comparisons.

6.4.2 Energy Consumption

This subsection principally addresses sub-research question **SRQ2.3b**, which examines the extent to which the effects of garbage collector variant on JRE JVM energy consumption differ across workload category levels:

> **SRQ2.3b:** ***To what extent do the effects of garbage collector variant on JRE JVM energy consumption differ across workload categories?***

Because all analyses are additionally decomposed according to execution mode, this subsection also contributes evidence toward sub-research question **SRQ2.4b**,

which examines the extent to which differences between interpretive and tiered execution moderate these workload-dependent relationships:

> **SRQ2.4b:** ***To what extent do differences between interpretive and tiered execution affect JRE JVM energy consumption across differing workload categories?***

Descriptive Analysis

Figures 6.11 and 6.12, together with Table 6.16, summarise the distributional characteristics associated with energy consumption across workload categories and garbage collector variants, reported for pooled observations and subsequently decomposed according to tiered and interpretive execution.

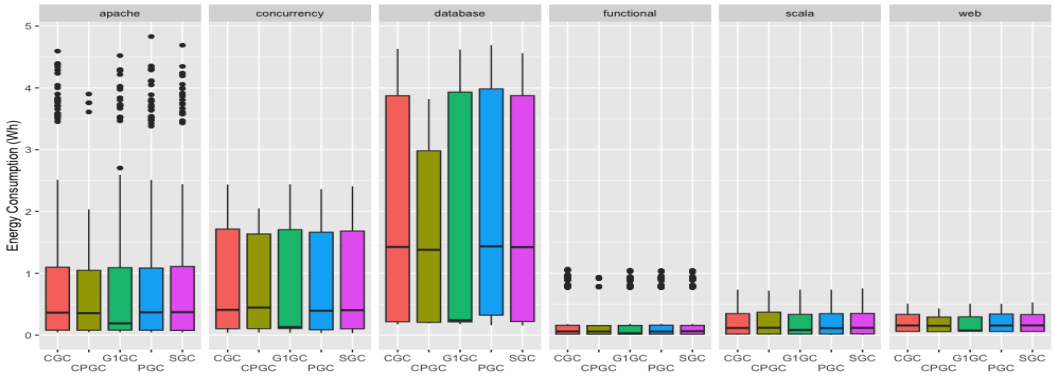


Figure 6.11: Energy Consumption Distributions: Box-and-whisker plots detailing energy consumption distributions, panelled by workload category, and clustered within each workload category by garbage collection variant.

Across all workload category levels, substantial differences in energy consumption are evident between workload categories, with the `database`, `apache`, and `concurrency` workload categories consistently exhibiting comparatively greater energy consumption, while the `functional`, `scala`, and `web` workload categories generally exhibit comparatively lower energy consumption distributions, as illustrated in Figure 6.11 and Table 6.16. In contrast to the modest garbage collector-related differences previously observed for power use, variation associated with garbage collector variant appears comparatively limited when pooled

across execution modes, suggesting that workload category remains the dominant determinant of overall energy consumption.

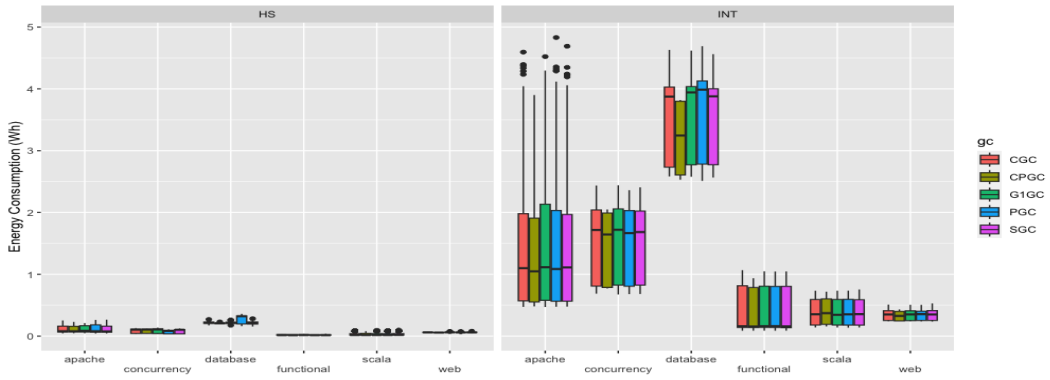


Figure 6.12: Energy Consumption Distributions: Box-and-whisker plots depicting energy consumption distributions for each workload category, clustered by the garbage collector used for heap management. Results presented for both tiered execution, panel HS, and interpretive execution, panel INT.

Examining differences between garbage collector variants more directly, several systematic differences in median energy consumption are observable across workload categories. In contrast to the comparatively modest collector-related differences previously observed for power use, the descriptive statistics indicate that the `G1GC` garbage collector frequently exhibits substantially lower median energy consumption relative to the remaining collector configurations across multiple workload category levels.

The strongest differences are evident within the `database`, `apache`, and `concurrency` workload categories. Within the `database` workload category, median energy consumption decreases substantially from approximately $1.38Wh$ to $1.44Wh$ across the `CGC`, `CPGC`, `PGC`, and `SGC` collector configurations to approximately $0.24Wh$ under `G1GC`. Similarly, within the `apache` workload category, median energy consumption decreases from approximately $0.35Wh$ to $0.37Wh$ across most collector configurations to approximately $0.19Wh$ under `G1GC`. Within the `concurrency` workload category, the same pattern is observed, with median energy consumption decreasing from approximately $0.39Wh$ to $0.45Wh$ across the majority of collector configurations to approximately $0.13Wh$ under `G1GC`.

Although comparatively smaller absolute differences are observed within the `functional`, `scala`, and `web` workload categories, the same general pattern remains evident, with `G1GC` consistently exhibiting lower median energy consumption relative to the remaining garbage collector variants.

Table 6.16: Energy Consumption Distributions (Pooled Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and Garbage Collector variants under pooled execution.

Pooled Execution (Wh)						**Pooled Execution (Wh)**					
Category	**GC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GC**	**M**	**M_d**	**SD**	**IQR**
apache	CGC	0.80	0.36	1.04	1.02	functional	CGC	0.23	0.06	0.34	0.14
	CPGC	0.77	0.35	1.00	0.97		CPGC	0.22	0.06	0.33	0.14
	G1GC	0.75	0.19	1.03	1.01		G1GC	0.21	0.03	0.32	0.14
	PGC	0.80	0.37	1.04	1.01		PGC	0.22	0.06	0.33	0.15
	SGC	0.79	0.37	1.03	1.04		SGC	0.22	0.06	0.33	0.14
concurrency	CGC	0.82	0.41	0.84	1.61	scala	CGC	0.21	0.12	0.23	0.33
	CPGC	0.79	0.45	0.82	1.53		CPGC	0.21	0.12	0.24	0.35
	G1GC	0.76	0.13	0.83	1.60		G1GC	0.19	0.08	0.22	0.32
	PGC	0.80	0.39	0.83	1.58		PGC	0.21	0.11	0.23	0.33
	SGC	0.81	0.40	0.83	1.58		SGC	0.21	0.12	0.23	0.33
database	CGC	1.91	1.43	1.77	3.66	web	CGC	0.20	0.16	0.15	0.27
	CPGC	1.71	1.38	1.61	2.78		CPGC	0.19	0.15	0.15	0.24
	G1GC	1.74	0.24	1.79	3.72		G1GC	0.19	0.07	0.15	0.23
	PGC	1.97	1.44	1.78	3.66		PGC	0.20	0.15	0.15	0.28
	SGC	1.90	1.42	1.76	3.65		SGC	0.20	0.16	0.15	0.27

Differences in distributional spread are strongly workload dependent. The `database`, `apache`, and `concurrency` workload categories exhibit substantially wider interquartile ranges and greater overall dispersion across all garbage collector variants, indicating considerable variability in energy consumption observations across repeated executions. In contrast, the `functional`, `scala`, and `web` workload categories exhibit comparatively narrower distributions, suggesting comparatively greater stability in energy consumption behaviour irrespective of garbage collector selection.

Taken together, these descriptive findings indicate that workload category exerts only limited moderating influence on the relationship between garbage collector variant and JRE JVM energy consumption. Although modest workload-dependent variation between collector configurations is observable within specific workload categories, overall energy consumption patterns remain principally determined by workload category rather than garbage collector variant, suggesting

comparatively weak interaction effects under pooled execution.

Table 6.17: Energy Consumption Distributions (Tiered Execution (Wh)): Descriptive statistics summarising energy consumption observations across workload categories and Garbage Collector variant under tiered execution.

Tiered Execution (Wh)						Tiered Execution (Wh)					
Category	**GC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GC**	**M**	**M_d**	**SD**	**IQR**
apache	CGC	0.11	0.08	0.06	0.09	functional	CGC	0.02	0.02	0.01	0.01
	CPGC	0.11	0.08	0.06	0.09		CPGC	0.02	0.01	0.01	0.01
	G1GC	0.11	0.09	0.05	0.10		G1GC	0.02	0.02	0.01	0.01
	PGC	0.12	0.08	0.07	0.12		PGC	0.02	0.02	0.01	0.01
	SGC	0.11	0.08	0.06	0.10		SGC	0.02	0.02	0.01	0.01
concurrency	CGC	0.09	0.10	0.03	0.07	scala	CGC	0.03	0.02	0.03	0.03
	CPGC	0.09	0.11	0.03	0.07		CPGC	0.03	0.02	0.03	0.03
	G1GC	0.09	0.11	0.04	0.08		G1GC	0.03	0.02	0.03	0.03
	PGC	0.07	0.09	0.03	0.06		PGC	0.03	0.02	0.03	0.02
	SGC	0.08	0.10	0.03	0.07		SGC	0.03	0.02	0.03	0.02
database	CGC	0.22	0.22	0.02	0.02	web	CGC	0.06	0.06	0.00	0.01
	CPGC	0.21	0.21	0.01	0.01		CPGC	0.06	0.06	0.00	0.00
	G1GC	0.21	0.21	0.02	0.02		G1GC	0.06	0.06	0.01	0.01
	PGC	0.28	0.32	0.07	0.13		PGC	0.06	0.06	0.00	0.01
	SGC	0.22	0.22	0.03	0.03		SGC	0.06	0.06	0.01	0.01

When examined separately according to execution mode, decomposition of the energy consumption observations presented in Figure 6.12, together with Tables 6.17 and 6.18, reveals that the relationship between garbage collector variant and JRE JVM energy consumption differs only modestly between garbage collector variants once execution mode is considered explicitly. In contrast to the patterns initially observed under pooled execution, much of the apparent variation associated with garbage collector selection becomes substantially attenuated following decomposition according to execution mode.

Under tiered execution, energy consumption remains highly stable across garbage collector variants within most workload categories. Median energy consumption differences remain negligible within the `apache`, `functional`, `scala`, and `web` workload categories, indicating that garbage collector selection exerts little meaningful influence on energy consumption under tiered execution within these workload category levels. The strongest differences are observed within the `database` workload category, where median energy consumption varies from approximately $0.21Wh$ under both `CPGC` and `G1GC` to approximately $0.32Wh$ under `PGC`. Comparatively modest differences are also observable within the `concurrency`

workload category, where median energy consumption ranges from approximately $0.09Wh$ under `PGC` to approximately $0.11Wh$ under both `CPGC` and `G1GC`.

Decomposing based on interpretive execution, garbage collector-related differences become even less pronounced. Across the `apache`, `functional`, `scala`, and `web` workload categories, median energy consumption remains highly consistent across all garbage collector variants, with only negligible differences observable between collector variants. Similarly, within the `concurrency` workload category, median energy consumption remains tightly clustered, varying only modestly between approximately $1.64Wh$ under `CPGC` and approximately $1.72Wh$ under both `CGC` and `G1GC`.

The largest interpretive execution differences are again observed within the `database` workload category, although variation remains comparatively moderate overall. Within this workload category, median energy consumption ranges from approximately $3.25Wh$ under `CPGC` to approximately $3.99Wh$ under `PGC`, while the remaining collector configurations exhibit broadly similar energy consumption behaviour.

Table 6.18: Energy Consumption Distributions (Interpretive Execution (W)): Descriptive statistics summarising energy consumption observations across workload categories and Garbage Collector variants under interpretive execution.

Interpretive Execution (Wh)						**Interpretive Execution (Wh)**					
Category	**GC**	**M**	**M_d**	**SD**	**IQR**	**Category**	**GC**	**M**	**M_d**	**SD**	**IQR**
apache	CGC	1.49	1.10	1.10	1.41	functional	CGC	0.43	0.16	0.37	0.67
	CPGC	1.42	1.05	1.07	1.36		CPGC	0.42	0.16	0.37	0.65
	G1GC	1.51	1.11	1.11	1.56		G1GC	0.43	0.16	0.37	0.67
	PGC	1.48	1.09	1.10	1.47		PGC	0.43	0.16	0.37	0.66
	SGC	1.48	1.11	1.09	1.40		SGC	0.43	0.16	0.37	0.67
concurrency	CGC	1.55	1.72	0.57	1.23	scala	CGC	0.39	0.35	0.20	0.41
	CPGC	1.50	1.64	0.55	1.21		CPGC	0.40	0.37	0.21	0.41
	G1GC	1.56	1.72	0.58	1.23		G1GC	0.39	0.34	0.21	0.41
	PGC	1.53	1.67	0.56	1.22		PGC	0.39	0.35	0.21	0.42
	SGC	1.54	1.68	0.56	1.20		SGC	0.39	0.36	0.21	0.41
database	CGC	3.60	3.87	0.67	1.30	web	CGC	0.34	0.35	0.09	0.16
	CPGC	3.21	3.25	0.64	1.19		CPGC	0.33	0.33	0.09	0.15
	G1GC	3.70	3.94	0.67	1.27		G1GC	0.34	0.35	0.09	0.16
	PGC	3.66	3.99	0.72	1.34		PGC	0.34	0.36	0.08	0.15
	SGC	3.58	3.88	0.67	1.23		SGC	0.34	0.35	0.09	0.16

Differences in distributional spread further reinforce these observations. Under tiered execution, distributions remain comparatively narrow across most workload

categories, indicating relatively stable energy consumption behaviour irrespective of garbage collector selection. Under interpretive execution, substantially wider interquartile ranges emerge across all workload categories, reflecting the considerably greater overall energy consumption associated with interpretive execution. However, despite this increased overall dispersion, variation attributable specifically to garbage collector selection remains comparatively limited.

Taken together, these descriptive findings indicate that workload category exerts only limited moderating influence on the relationship between garbage collector variant and JRE JVM energy consumption. Although some workload-dependent variation is observable, particularly within the `database` workload category, the pronounced collector-specific differences initially observed under pooled execution become substantially reduced once execution mode is considered explicitly. This pattern suggests that differences between interpretive and tiered execution remain the principal determinant of energy consumption variation, while garbage collector-related effects remain comparatively weak across differing runtime conditions.

Inferential Statistical Analysis

Assessment of Distributional Assumptions:

Distributional normality was assessed using the Shapiro–Wilk test. The results indicated statistically significant departures from normality across all energy consumption distributions examined. All thirty workload category and garbage collector combinations exhibited statistically significant deviations from normality ($p < .05$), indicating a consistent absence of normally distributed energy consumption observations across all workload category levels and garbage collector variants. Given the overall distributional characteristics observed, non-parametric statistical procedures were employed throughout the analysis.

Garbage Collector Effects Within Workload Categories:

Kruskal–Wallis H tests were conducted to determine whether statistically

significant differences in median energy consumption were observed across garbage collector variants within individual workload category levels. In contrast to the power use analyses presented previously, the results indicated no statistically significant differences in median energy consumption across garbage collector variants within any workload category.

No statistically significant differences were observed within the `apache` ($p = .928$), `concurrency` ($p = .219$), `database` ($p = .160$), `functional` ($p = .963$), `scala` ($p = .928$), or `web` ($p = .771$) workload categories. Because no statistically significant omnibus effects were observed, no pairwise post-hoc comparisons were required.

These findings indicate that workload category did not meaningfully influence the relationship between garbage collector variant and JRE JVM energy consumption when observations were pooled across execution modes. Although modest descriptive differences were observable between garbage collector variants within certain workload categories, these differences were not sufficiently systematic to produce statistically significant variation. In contrast to the workload-dependent garbage collector effects previously observed for power use, garbage collector selection does not appear to exert a statistically meaningful workload-dependent influence on overall JRE JVM energy consumption under pooled execution conditions.

Execution Mode Effects:

Decomposition of the analyses according to execution mode revealed substantial differences in the extent to which workload category moderated the relationship between garbage collector variant and JRE JVM energy consumption.

Under tiered execution, Kruskal–Wallis H tests indicated that statistically significant differences in median energy consumption were observed within several workload category levels, including `concurrency` ($p < .001$), `database` ($p = .005$), and `web` ($p = .004$). In contrast, no statistically significant differences were observed within the `apache` ($p = .889$), `functional` ($p = .084$), or `scala` ($p =$

.601) workload categories.

Pairwise comparisons indicated that these statistically significant differences were not uniformly distributed across garbage collector variants. Within the `concurrency` workload category, the `PGC` garbage collector exhibited significantly lower energy consumption relative to `CGC` ($p < .001$), `CPGC` ($p = .036$), and `G1GC` ($p < .001$). In addition, `SGC` exhibited significantly lower energy consumption relative to `G1GC` ($p = .004$) and `PGC` ($p < .001$). Within the `database` workload category, `PGC` exhibited significantly greater energy consumption relative to `CGC` ($p = .035$), `G1GC` ($p = .036$), and `SGC` ($p = .010$). Finally, within the `web` workload category, the only statistically significant pairwise difference was observed between `CPGC` and `G1GC`, with `CPGC` exhibiting comparatively lower energy consumption ($p = .002$).

These findings differ substantially from the pooled analyses, where no statistically significant garbage collector effects were observed across any workload category. The emergence of statistically significant differences under tiered execution indicates that garbage collector-related differences in energy consumption become more observable once interpretive execution effects are isolated. This suggests that workload category moderates the relationship between garbage collector variant and JRE JVM energy consumption under tiered execution, although these effects remain concentrated within a limited subset of workload categories and do not reflect a uniform pattern across all garbage collector variants.

When observations were restricted to interpretive execution, statistically significant differences in median energy consumption across garbage collector variants were largely absent across workload category levels. No statistically significant differences were observed within the `apache` ($p = .887$), `concurrency` ($p = .903$), `functional` ($p = .936$), `scala` ($p = .978$), or `web` ($p = .946$) workload categories. A statistically significant omnibus difference was observed only within the `database` workload category ($p = .029$).

Subsequent pairwise comparisons indicated that the statistically significant

difference observed within the `database` workload category was attributable solely to a comparison between the `CPGC` and `G1GC` garbage collector variants. Specifically, `CPGC` exhibited significantly lower energy consumption relative to `G1GC` ($p = .031$). No statistically significant differences were observed between any remaining garbage collector comparisons following Bonferroni correction.

This pattern indicates that garbage collector-related differences in energy consumption under interpretive execution were substantially weaker than those observed under tiered execution. Across five of the six workload categories, energy consumption remained statistically indistinguishable across garbage collector variants, suggesting that the additional execution overhead associated with interpretive execution substantially reduced the visibility of garbage collector-related differences. The `database` workload category represented the only exception, indicating that memory-intensive workloads may remain comparatively sensitive to garbage collector variant even under interpretive execution.

Collectively, these findings indicate that workload-dependent garbage collector effects on JRE JVM energy consumption were strongly conditioned by execution mode. Under pooled execution, no statistically significant garbage collector effects were observed across any workload category. Under tiered execution, statistically significant garbage collector effects emerged within multiple workload categories, although the specific pairwise differences varied substantially across collector configurations and did not reveal a single consistently superior garbage collector. Under interpretive execution, statistically significant differences were largely absent, with only the `database` workload category exhibiting limited collector-specific variation. Taken together, these findings indicate that differences between interpretive and tiered execution substantially influenced the extent to which workload category moderated the relationship between garbage collector variant and JRE JVM energy consumption, with garbage collector-related effects becoming considerably more visible under tiered execution.

Summary

The descriptive analyses indicated that workload category remained the principal determinant of overall variation in JRE JVM energy consumption, with the `database`, `apache`, and `concurrency` workload categories consistently exhibiting substantially greater energy consumption relative to the `functional`, `scala`, and `web` workload categories. Although pooled descriptive statistics initially suggested several systematic differences between garbage collector variants, most notably the comparatively lower median energy consumption repeatedly observed under the `G1GC` collector, decomposition according to execution mode demonstrated that much of this apparent variation became substantially attenuated once tiered and interpretive execution were examined separately.

Inferential statistical analyses supported these observations. When observations were pooled across execution modes, no statistically significant differences in median energy consumption were observed between garbage collector variants within any workload category. These findings indicate that garbage collector selection did not exert a statistically significant workload-dependent influence on JRE JVM energy consumption under pooled execution conditions, despite several modest descriptive differences observed across individual workload categories.

Decomposition according to execution mode revealed that these relationships were strongly conditioned by differences between interpretive and tiered execution. Under tiered execution, statistically significant differences in median energy consumption emerged within the `concurrency`, `database`, and `web` workload categories, indicating that garbage collector-related energy consumption differences became more observable once interpretive execution overheads were removed. However, the observed pairwise differences varied substantially across collector configurations and did not reveal a single consistently superior garbage collector variant across all workload category levels.

Under interpretive execution, statistically significant differences were largely absent across workload categories, with only the `database` workload category

exhibiting a statistically significant omnibus effect. Subsequent pairwise comparisons indicated that this isolated effect was attributable solely to lower energy consumption observed under `CPGC` relative to `G1GC`. Across all remaining workload category levels, energy consumption remained statistically indistinguishable across garbage collector variants, indicating that interpretive execution substantially reduced the visibility of garbage collector-related differences.

In relation to **SRQ2.3b**, the findings indicate that garbage collector variant exerted only limited workload-dependent effects on JRE JVM energy consumption. Although some workload-specific differences emerged under isolated execution conditions, garbage collector selection did not produce statistically significant differences when observations were pooled across execution modes, indicating comparatively weak overall interaction effects.

In relation to **SRQ2.4b**, the findings indicate that differences between interpretive and tiered execution substantially moderated these workload-dependent relationships. Garbage collector-related energy consumption effects became considerably more visible under tiered execution, whereas interpretive execution largely suppressed collector-specific differences across workload category levels. This indicates that execution mode remained a substantially stronger determinant of overall energy consumption variation than garbage collector selection itself.

Subsection Research Question Findings

SRQ2.3b: Garbage collector variant exerted only limited workload-dependent effects on JRE JVM energy consumption. No statistically significant garbage collector effects were observed when observations were pooled across execution modes, indicating comparatively weak overall interaction effects between garbage collector variant and workload category.

SRQ2.4b: Differences between interpretive and tiered execution substantially influenced workload-dependent garbage collector effects on JRE JVM energy consumption. Garbage collector-related effects became more visible under tiered execution, whereas interpretive execution largely suppressed collector-specific differences across workload category levels, confirming execution mode as the stronger determinant of overall energy consumption variation.

6.4.3 Summary

This section examined whether the effects associated with garbage collector variant on JRE JVM power use and energy consumption differed across workload category levels, thereby addressing **SRQ2.3**. Because all analyses were additionally decomposed according to tiered and interpretive execution, the section also contributed evidence toward **SRQ2.4**, which concerns the extent to which execution mode affects these workload-dependent relationships.

For power use, the descriptive and inferential analyses indicated that workload category moderated the relationship between garbage collector variant and JRE JVM power use. Across pooled observations, statistically significant garbage collector effects were observed within the `concurrency` and `database` workload categories, with pairwise comparisons indicating that these effects were primarily attributable to the comparatively lower power use associated with `SGC`. When decomposed according to execution mode, garbage collector-related power use

differences became substantially more pronounced under tiered execution, where statistically significant effects were observed across all workload category levels. In contrast, under interpretive execution, garbage collector-related differences were largely absent and were not supported by robust pairwise differences.

For energy consumption, the relationship was weaker. When observations were pooled across execution modes, no statistically significant garbage collector effects were observed within any workload category, indicating no substantive evidence that workload category moderated the relationship between garbage collector variant and JRE JVM energy consumption under pooled execution conditions. However, decomposition according to execution mode revealed that garbage collector-related energy consumption differences became more visible under tiered execution, with statistically significant effects observed within the `concurrency`, `database`, and `web` workload categories. Under interpretive execution, statistically significant effects were largely absent, with only the `database` workload category exhibiting a limited pairwise difference between `CPGC` and `G1GC`.

In relation to **SRQ2.3**, these findings indicate that the effects of garbage collector variant differed across workload category levels for JRE JVM power use, but only weakly for JRE JVM energy consumption. Workload-dependent garbage collector effects were more consistently evident for power use, particularly under tiered execution, whereas energy consumption remained largely stable across collector variants when observations were pooled across execution modes.

In relation to **SRQ2.4**, the findings indicate that differences between interpretive and tiered execution substantially influenced the extent to which garbage collector-related effects emerged across workload category levels. For power use, tiered execution revealed statistically significant garbage collector effects across all workload categories, while interpretive execution largely suppressed these effects. For energy consumption, execution mode also conditioned the visibility of garbage collector effects, with statistically significant differences emerging primarily under tiered execution. Overall, execution mode represented a major determinant of

whether garbage collector-related workload effects were statistically observable.

Section Research Question Findings

SRQ2.3: The effects of garbage collector variant differed across workload category levels for JRE JVM power use, but only weakly for JRE JVM energy consumption. Workload-dependent garbage collector effects on power use were most evident within the `concurrency` and `database` workload categories under pooled execution and became statistically significant across all workload category levels under tiered execution. In contrast, energy consumption showed no statistically significant garbage collector effects under pooled execution.

SRQ2.4: Differences between interpretive and tiered execution substantially affected workload-dependent garbage collector effects on both JRE JVM power use and energy consumption. Garbage collector-related effects were consistently more visible under tiered execution, whereas interpretive execution largely suppressed collector-specific differences across workload category levels.

6.5 How this Chapter Answers RQ2

This chapter examined the extent to which the effects associated with OpenJDK JRE source version, GNU GCC build toolchain compiler source version, and garbage collector variant on JRE JVM power use and energy consumption were dependent upon workload category. In doing so, the chapter addressed **RQ2** by evaluating whether previously observed runtime configuration effects remained stable across workload categories or whether their magnitude, direction, and statistical significance varied according to the computational characteristics of the workload being executed. Because all analyses were further decomposed according

to interpretive and tiered execution, the chapter also evaluated the extent to which execution mode conditioned these workload-dependent relationships.

Across the analyses, the strongest and most consistent interaction effects were observed for OpenJDK JRE source version. For power use, OpenJDK JRE version 13 was consistently associated with elevated values across all workload category, although the magnitude of this increase differed substantially between workload categories. For energy consumption, OpenJDK JRE version effects were more selective, with statistically significant differences concentrated particularly within the `database` and `web` workload categories. These findings indicate that workload category moderated the relationship between OpenJDK JRE source version and both outcome variables, but that this moderation was broader and more systematic for power use than for energy consumption.

The GNU GCC build toolchain compiler source version interaction effects were more limited. For power use, statistically significant workload-dependent differences were concentrated primarily within the `concurrency` and `web` workload categories, with GNU GCC version 5 generally associated with lower power use relative to later compiler versions. For energy consumption, no substantive evidence of workload-dependent GNU GCC version effects was observed, with energy consumption remaining statistically stable across GNU GCC versions within workload categories. These findings indicate that GNU GCC build toolchain compiler source version exerted comparatively weak workload-dependent effects, particularly for energy consumption.

Garbage collector variant produced a different pattern. For power use, workload-dependent garbage collector effects were evident within the `concurrency` and `database` workload categories under pooled execution and became more widespread under tiered execution, where statistically significant differences were observed across all workload category levels. These effects were repeatedly associated with comparatively lower power use under `SGC`. For energy consumption, however, garbage collector effects were weaker. No statistically significant garbage collector

effects were observed under pooled execution, although effects became more visible under tiered execution within the `concurrency`, `database`, and `web` workload categories. These findings indicate that garbage collector variant exerted clearer workload-dependent effects on power use than on energy consumption.

Execution mode emerged as a major conditioning factor across the chapter. For OpenJDK JRE source version, execution mode influenced the magnitude and detectability of workload-dependent effects, particularly for energy consumption. For GNU GCC build toolchain compiler source version, execution mode strengthened workload-dependent effects for power use under interpretive execution, but did not meaningfully alter the GNU GCC version–energy consumption relationship. For garbage collector variant, execution mode was especially important: garbage collector-related effects on both power use and energy consumption were more visible under tiered execution, while interpretive execution largely suppressed collector-specific differences.

In relation to **RQ2**, the findings demonstrate that the effects associated with JRE JVM runtime configuration are not uniformly stable across workload category levels. Workload category moderated the relationship between runtime configuration and observed power use most clearly for OpenJDK JRE source version and garbage collector variant, while GNU GCC build toolchain compiler source version produced only limited workload-dependent effects. For energy consumption, workload-dependent effects were generally weaker and more strongly conditioned by execution mode, indicating that total energy consumption was less sensitive to some runtime configuration differences than instantaneous power use.

Overall, this chapter demonstrates that JRE JVM power use and energy consumption cannot be fully understood from runtime configuration factors in isolation. Instead, the effects of JRE source version, build toolchain compiler version, and garbage collector variant depend upon workload category and, in many cases, execution mode. These findings establish workload category and execution mode as important conditioning factors in the relationship between

JVM runtime configuration and observed power and energy outcomes.

Chapter Research Question Findings

RQ2: The effects associated with JRE JVM runtime configuration were dependent upon workload category, although the strength and consistency of these workload-dependent relationships varied substantially across runtime factors and outcome variables. OpenJDK JRE source version exhibited the strongest and most consistent workload-dependent effects, particularly for power use, with OpenJDK JRE version 13 repeatedly associated with elevated power use and, more selectively, elevated energy consumption. Garbage collector variant also demonstrated workload-dependent effects, most clearly for power use under tiered execution, whereas GNU GCC build toolchain compiler source version exhibited comparatively limited workload-dependent effects that were largely confined to power use. Across all analyses, energy consumption effects were generally weaker and more strongly conditioned by execution mode than corresponding power use effects. Collectively, these findings demonstrate that JRE JVM power use and energy consumption cannot be fully understood from runtime configuration factors in isolation, but must instead be understood as jointly conditioned by workload category and execution mode.

Chapter 7

Research Question 3: Model Development

This chapter addresses **Research Question 3 (RQ3)**, which examines the extent to which observed variation in JRE/JVM power use and energy consumption can be accounted for using regression models incorporating runtime configuration factors, workload category, and microarchitectural performance event measures. Whereas the preceding chapters examined the independent effects associated with individual runtime factors and the extent to which these effects were conditioned by workload category, the present chapter develops multivariate regression models that evaluate the collective contribution of these factors when considered simultaneously.

The objective of this chapter is to determine the extent to which JRE source version, GNU GCC build toolchain compiler version, JVM execution mode, garbage collector variant, workload category, and microarchitectural performance event measures can account for observed variation in JRE/JVM power use and energy consumption. In addition, these analyses seek to identify the relative contribution associated with each predictor class and to evaluate the stability and reliability of the resulting regression models.

To systematically evaluate these relationships, the chapter adopts a staged modelling framework. Regression models are first developed for pooled observations incorporating all experimental observations, followed by execution-mode specific models constructed separately for interpretive and tiered execution. Sepa-

rate models are developed for both power use and energy consumption, allowing direct comparison of the factors contributing most strongly to variation across each outcome variable.

Across all models, predictor specification follows the theoretically motivated empirical modelling framework introduced previously, while model refinement decisions are guided by consideration of predictor redundancy, coefficient stability, and overall model reliability. Model reliability is assessed through regression diagnostic procedures examining residual normality, linearity, heteroskedasticity, multicollinearity, and influential observations.

Collectively, the analyses presented in this chapter evaluate the extent to which the system-level factors, workload category, and lower-level microarchitectural performance event measures examined throughout this thesis can account for observed JRE/JVM power use and energy consumption outcomes, while identifying those predictors contributing most strongly to observed variation across both outcome measures.

7.1 How this Chapter Addresses RQ3

This chapter addresses Research Question 3 (RQ3), which asks:

> ***RQ3****: To what extent can variation in JRE/JVM power use and energy consumption be accounted for through regression modelling incorporating JRE source version, GNU GCC build toolchain compiler version, JVM execution mode, garbage collector variant, workload category, and microarchitectural performance event measures?*

RQ3 is addressed through a structured hierarchical multiple regression framework in which the combined effects associated with runtime configuration factors, workload category, and microarchitectural performance event measures are examined simultaneously to determine their collective contribution toward observed JRE/JVM power use and energy consumption outcomes.

Specifically, this chapter develops regression models for both power use and energy consumption, first using pooled observations and subsequently through execution-mode specific models constructed separately for interpretive and tiered execution. These analyses evaluate the extent to which the selected predictor variables collectively account for observed variation in the measured outcome variables while identifying the relative contribution associated with each predictor class.

Each section therefore contributes evidence toward answering RQ3 by determining the extent to which the proposed regression models account for observed variation, the relative contribution associated with each predictor group, and the extent to which model reliability is supported through regression diagnostic procedures and bootstrap resampling analyses.

Collectively, the analyses establish the extent to which regression modelling can successfully account for observed JRE/JVM power use and energy consumption outcomes, while identifying the runtime characteristics and microarchitectural performance measures contributing most strongly to variation across the observed outcome measures.

7.2 Power Use Model

This section principally addresses sub-research question **SRQ3.1**, which examines the extent to which linear statistical models incorporating runtime configuration factors, workload category, and microarchitectural performance event measures account for observed variation in JRE/JVM power use.

> **SRQ3.1:** ***To what extent can variation in JRE/JVM power use be accounted for through regression modelling incorporating JRE source version, GNU GCC build toolchain compiler version, JVM execution mode, garbage collector variant, workload category, and microarchitectural perfor-***

mance event measures?

To address this question, we develop a staged hierarchical multiple regression framework in which predictor variables are introduced incrementally to evaluate the collective contribution associated with successive predictor groups. Model reliability is subsequently assessed through regression diagnostic procedures and bootstrap resampling analyses.

7.2.1 Theory-Driven Model Specification

The regression models developed in this chapter were specified *a priori* based on the theoretical model (§4.2) and conceptual framework (§4.3) introduced in Chapter 4. Unlike exploratory modelling approaches in which predictor variables are selected through data-driven procedures, the models estimated in this study were constructed directly from the theoretically motivated relationships previously identified as potential determinants of JRE JVM power use and energy consumption.

Consistent with the theoretical framework, the models incorporate both experimentally manipulated runtime configuration variables and emergent execution characteristics observed during runtime. The runtime configuration variables represent directly controllable system-level factors and comprise OpenJDK JRE source version (JSV), GNU GCC build toolchain compiler version ($BTCV$), JVM execution mode (JEM), garbage collector variant (GCV), and workload category (WC). In addition, the models incorporate microarchitectural performance event measures representing emergent lower-level execution characteristics, specifically branch misprediction rate (BMR), L1 data cache load miss rate ($L1DLMR$), L2 data cache load miss rate ($L2DLMR$), and page fault rate ($PFMR$).

The objective of the regression analysis is therefore not to identify statistically convenient predictors, but to evaluate the extent to which the theoretically specified runtime configuration factors and microarchitectural performance event measures account for variation in observed JRE/JVM power use and energy consumption.

To evaluate variation in observed power use, we specify the following general linear model:

$$\begin{aligned} P_i = \beta_0 &+ \beta_1(JSV_i) + \beta_2(BTCV_i) \\ &+ \beta_3(JEM_i) + \beta_4(GCV_i) + \beta_5(WC_i) \\ &+ \beta_6(BMR_i) + \beta_7(L1DLMR_i) \\ &+ \beta_8(L2DLMR_i) + \beta_9(PFMR_i) + \varepsilon_i \end{aligned} \tag{7.1}$$

Having established the theoretical specification of the regression model, we next examine the hierarchical model development process, assessing the incremental explanatory contribution associated with progressively introducing the theoretically specified predictor groups.

7.2.2 Hierarchical Regression Results

Predictors were entered hierarchically according to a theoretically motivated layered systems architecture reflecting progressively deeper layers of system-level influence on observed JRE JVM power use.

The first predictor block represents foundational runtime implementation characteristics comprising OpenJDK JRE source version and GNU GCC build toolchain compiler version. These variables define the underlying executable runtime environment prior to program execution. The second block captures execution strategy characteristics, specifically JVM execution mode and garbage collector variant. These variables determine how program instructions are executed and how runtime memory management is performed during execution. The third block introduces workload category, representing the external computational demands imposed upon the configured runtime environment. Finally, the fourth block incorporates emergent microarchitectural performance event measures, specifically branch misprediction rate, L1 data cache load miss rate, L2 data cache load miss rate, and page fault rate. Unlike the preceding predictor

groups, these variables are not directly specified prior to execution but instead emerge dynamically through the interaction between workload category, runtime configuration, and the underlying hardware architecture.

This hierarchical structure therefore reflects progressively deeper layers of system behaviour, enabling assessment of the extent to which lower-level execution characteristics provide incremental explanatory power beyond the directly specified runtime configuration variables.

The explanatory contribution associated with each hierarchical predictor block is summarised in Table 7.1. Sequential introduction of predictor blocks permits assessment of the incremental explanatory contribution associated with progressively deeper layers of JRE JVM execution behaviour.

Table 7.1: Hierarchical regression model development for JRE JVM pooled power use. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block.

Model Block	R^2	**Adj.** R^2	ΔR^2	ΔF	p
Block 1: Runtime Implementation	0.113	0.111	0.113	–	–
Block 2: Execution Strategy	0.125	0.122	0.012	20.568	$< 0.001^{***}$
Block 3: Workload Category	0.448	0.446	0.324	579.883	$< 0.001^{***}$
Block 4: Microarchitecture	0.488	0.485	0.039	87.941	$< 0.001^{***}$
Final Model	0.488	0.485	–	–	$< 0.001^{***}$

Examination of hierarchical model progression reveals substantial differences in explanatory contribution across predictor blocks. The first predictor block, comprising foundational runtime implementation characteristics, explained approximately 11.3% of observed variation in JRE JVM power use. Introducing execution strategy variables in Block 2 produced only a modest increase in explained variance, increasing overall model explanatory power by approximately 1.2 percentage points.

The largest increase in explanatory power occurred following introduction of workload category in Block 3, which increased explained variance by approximately 32.4 percentage points. This indicates that workload category represents the strongest directly specified predictor group of observed JRE JVM power use

among the directly specified runtime configuration variables. The final block, incorporating emergent microarchitectural performance event measures, contributed an additional 3.9 percentage points of explained variance, increasing overall model explanatory power to approximately 48.8%.

Although hierarchical model progression quantifies the incremental explanatory contribution associated with each predictor block, examination of coefficient evolution across successive model stages additionally permits assessment of the extent to which predictor relationships remain stable as progressively deeper system characteristics are introduced. Table 7.2 therefore reports coefficient estimates associated with each hierarchical model stage.

Because preliminary diagnostic assessment indicated potential heterogeneity in residual variance across observations, coefficient estimates are reported using HC3 heteroskedasticity-consistent robust standard errors to ensure inferential robustness under conditions of possible heteroskedasticity.

Examination of coefficient evolution across hierarchical model stages reveals several important structural relationships. The elevated power use associated with OpenJDK JRE version 13 remained consistently positive and highly statistically significant across all four model stages, indicating a stable and robust relationship independent of subsequently introduced execution characteristics.

In contrast, several coefficients exhibited substantial instability as additional predictor blocks were introduced. Most notably, the coefficient associated with interpretive execution initially exhibited a positive and statistically significant relationship with power use when introduced in Block 2, but became statistically non-significant following incorporation of microarchitectural performance event measures in Block 4. Similarly, the functional workload coefficient reversed sign between Blocks 3 and 4, suggesting that part of the workload-specific variation initially associated with workload category may reflect differences in underlying microarchitectural performance event measures.

These coefficient shifts indicate that relationships observed among higher-level

Table 7.2: Hierarchical regression coefficients for pooled JRE JVM power use. Standard errors correspond to HC3 heteroskedasticity-consistent robust variance estimation.

	Block 1			Block 2			Block 3			Block 4		
Predictor	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**
Intercept	3.933	0.016	***	3.916	0.020	***	3.857	0.018	***	4.080	0.065	***
JRE 10	0.012	0.020		0.006	0.021		0.006	0.016		-0.002	0.016	
JRE 11	0.004	0.020		-0.001	0.021		-0.001	0.016		-0.001	0.016	
JRE 12	0.010	0.020		0.005	0.021		-0.003	0.017		-0.003	0.016	
JRE 13	0.389	0.017	***	0.383	0.018	***	0.376	0.015	***	0.348	0.017	***
JRE 14	0.030	0.021		0.025	0.022		0.017	0.017		0.038	0.018	*
GCC 6	0.054	0.016	***	0.054	0.016	***	0.054	0.013	***	0.057	0.012	***
GCC 7	0.053	0.016	***	0.053	0.016	***	0.053	0.013	***	0.056	0.012	***
GCC 8	0.048	0.016	**	0.048	0.016	**	0.048	0.013	***	0.042	0.013	**
INT	–	–		0.070	0.012	***	0.070	0.009	***	-0.028	0.036	
CPGC	–	–		-0.005	0.034		-0.005	0.026		0.087	0.032	**
G1GC	–	–		0.015	0.017		0.015	0.013		0.050	0.020	*
PGC	–	–		-0.001	0.017		-0.001	0.013		0.037	0.021	.
SGC	–	–		-0.062	0.016	***	-0.062	0.013	***	-0.033	0.017	.
Concurrency	–	–		–	–		0.513	0.016	***	0.506	0.017	***
Database	–	–		–	–		-0.120	0.016	***	-0.109	0.018	***
Functional	–	–		–	–		-0.081	0.013	***	0.029	0.048	
Scala	–	–		–	–		-0.103	0.014	***	-0.048	0.026	.
Web	–	–		–	–		0.512	0.014	***	0.570	0.018	***
BMR	–	–		–	–		–	–		-0.002	0.003	
L1DLMR	–	–		–	–		–	–		-0.019	0.010	.
L2DLMR	–	–		–	–		–	–		-0.008	0.002	***
PFMR	–	–		–	–		–	–		-1.194	0.497	*

Significance codes: *** $p < .001$; ** $p < .01$; * $p < .05$; . $p < .10$.

runtime configuration variables are partially conditioned by lower-level emergent execution characteristics arising during program execution.

Estimated Final Model

The final hierarchical regression model can be expressed as:

$$\begin{aligned}\widehat{PowerUse}_i = {} & 4.080 - 0.002\,JRE10_i - 0.001\,JRE11_i - 0.003\,JRE12_i \\ & + 0.348\,JRE13_i + 0.038\,JRE14_i + 0.057\,GCC6_i \\ & + 0.056\,GCC7_i + 0.042\,GCC8_i - 0.028\,INT_i \\ & + 0.087\,CPGC_i + 0.050\,G1GC_i + 0.037\,PGC_i - 0.033\,SGC_i \\ & + 0.506\,Concurrency_i - 0.109\,Database_i + 0.029\,Functional_i \\ & - 0.048\,Scala_i + 0.570\,Web_i - 0.002\,BMR_i \\ & - 0.019\,L1DLMR_i - 0.008\,L2DLMR_i - 1.194\,PFMR_i + \epsilon_i\end{aligned} \tag{7.2}$$

The estimated model coefficients are interpreted relative to the omitted reference categories corresponding to OpenJDK JRE 9, GCC 5, tiered execution, the default CGC garbage collector baseline, and the `apache` workload category.

7.2.3 Regression Diagnostics

To assess the statistical reliability of the pooled power use regression model, a series of standard diagnostic procedures were conducted examining model assumptions and overall specification behaviour. These diagnostics evaluate the extent to which the fitted regression model satisfies the principal assumptions underlying valid coefficient estimation and inferential interpretation.

Assessment of Linearity

Because the regression model incorporates both categorical runtime configuration factors and continuous microarchitectural performance event measures, assessment

of linearity focused principally on the continuous predictor variables. Linearity was assessed through visual inspection of residuals plotted against fitted values, Figure 7.1.



Figure 7.1: Linearity Diagnostics (Pooled Power Use Model): Residuals versus fitted values plot used to assess the adequacy of the linear regression specification. The majority of observations remained broadly centred around zero, although modest systematic residual structure was observed together with two observations exhibiting substantially elevated positive residuals. These observations indicate the presence of heterogeneous execution contexts not fully captured within the aggregated pooled model specification.

Visual examination of the residuals versus fitted values plot indicated that the majority of observations remained broadly distributed around zero across the central fitted value range, suggesting that the linear specification provided an adequate approximation of the dominant relationships represented within the pooled power use model.

However, modest systematic structure was observed within the residual pattern. In particular, two observations located at the lower fitted value range exhibited substantially elevated positive residuals relative to the remaining dataset. These observations correspond to specific execution contexts for which observed power use substantially exceeded model predictions. Because these observations were later identified through Cook's distance diagnostics as influential observations, their presence suggests that the departures from ideal linearity observed in the pooled model reflect structurally distinct execution contexts rather than widespread

misspecification of the underlying linear model form.

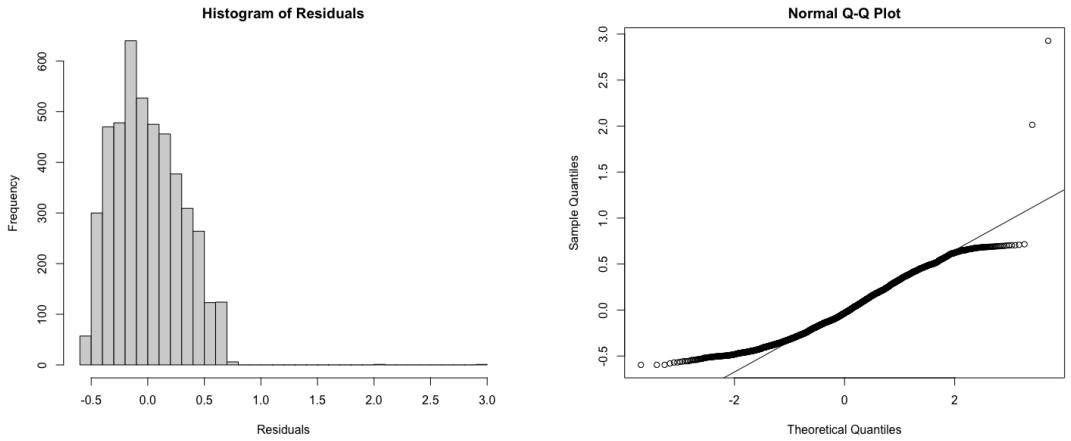
These observations suggest that although the overall linear specification provides a reasonable aggregate representation of the dominant predictor relationships, the pooled dataset contains heterogeneous execution contexts exhibiting partially distinct underlying relationships. This interpretation is consistent with the influential observation and sensitivity analyses reported subsequently, indicating that departures from ideal linearity likely reflect structural heterogeneity within the pooled dataset rather than simple misspecification of the underlying model form.

Assessment of Residual Distribution

Prior to interpreting model coefficients, the distributional characteristics of model residuals were examined to assess the extent to which the residual normality assumption underlying ordinary least squares regression was satisfied. Residual diagnostics were evaluated numerically through summary statistics and standardised residuals, and visually through examination of the residual frequency distribution and normal quantile–quantile (Q–Q) plot. Formal statistical assessment was additionally performed using the Shapiro–Wilk test. Because formal normality tests become increasingly sensitive to minor deviations from normality in large samples, primary interpretive emphasis was placed upon visual examination of the residual distribution.

Examination of raw residual summary statistics indicated that residuals were centred approximately around zero, with a mean residual of approximately zero and a median residual of -0.034. This suggests that the model exhibited no evidence of systematic overall prediction bias. However, asymmetry was observed in the residual range. The negative residual tail extended to approximately -0.60, whereas the positive residual tail extended substantially further to approximately 2.93, indicating that a relatively small number of observations exhibited substantially greater observed power use than predicted by the model.

Assessment of standardised residuals reinforced this pattern. The majority of



(a) Residual Histogram (b) Normal Q-Q Plot

Figure 7.2: Residual Distribution Diagnostics (Pooled Power Use Model): Histogram depicting the distribution of regression residuals associated with the pooled power use model (a), and normal Q–Q plot assessing conformity of residuals to the theoretical normal distribution (b).

observations remained concentrated within a relatively narrow range surrounding zero, suggesting that model fit was generally stable across most observations. However, the maximum standardised residual reached approximately 11.43, substantially exceeding conventional outlier thresholds, indicating the presence of a small number of observations for which observed power use substantially exceeded model predictions.

Visual examination of the residual histogram indicated that the majority of residuals followed an approximately bell-shaped central distribution, Figure 7.2 (a). However, a pronounced positive tail was clearly visible, indicating moderate positive skewness within the residual distribution. This pattern suggests that while the model provided reasonably accurate predictions across the majority of observations, a relatively small subset of observations exhibited systematically greater observed power use than predicted.

The normal Q–Q plot provided further evidence supporting this interpretation, Figure 7.2 (b). Residuals closely followed the theoretical normal reference line throughout the central quantiles, indicating approximate normality across the

majority of observations. However, substantial deviation was observed in the upper quantiles, where a small number of observations diverged sharply above the theoretical normal distribution. This indicates that departures from residual normality were primarily concentrated within the upper residual tail rather than reflecting general distributional misspecification across the full residual distribution.

Formal statistical assessment using the Shapiro–Wilk test indicated statistically significant deviation from normality ($W = 0.968$, $p < .001$). However, given the large sample size ($n = 4608$), this result was not considered substantively problematic, as the Shapiro–Wilk procedure is highly sensitive to even relatively minor distributional deviations under large-sample conditions. Consistent with the visual diagnostic procedures, the observed non-normality appears primarily attributable to a limited number of observations occupying the extreme upper residual distribution.

Taken collectively, residual diagnostic assessment indicates that the pooled power use regression model provided an adequate overall fit across the majority of observations. Observed departures from normality were not broadly distributed across the residual distribution, but instead appear attributable to a relatively small subset of observations exhibiting substantially larger positive residuals. This pattern suggests the presence of heterogeneous execution contexts within the pooled dataset, a possibility examined further through subsequent diagnostic assessment of residual variance and influential observations.

Assessment of Influential Observations

Cook's distance was used to identify observations exerting disproportionate influence over pooled model coefficient estimation. Using the conventional threshold of $4/n$, corresponding to approximately 0.00087, a total of 96 observations exceeded the influence threshold.

Inspection of influential observation distribution revealed clear non-random clustering across specific execution contexts. Of the 96 influential observations

identified, 83 observations (86.5%) originated from interpretive execution, while only 13 observations (13.5%) originated from tiered execution. Workload-level analysis similarly revealed substantial concentration within the `scala` workload category, which accounted for 57 observations (59.4%) exceeding the Cook's distance threshold, Table 7.3.

The observed concentration of influential observations within specific execution contexts suggests that model influence was not randomly distributed across the pooled dataset. Importantly, these influential observations do not represent anomalous or invalid measurements, but rather correspond to experimentally valid runtime configurations exhibiting disproportionately strong influence over pooled model parameter estimation.

Their systematic concentration within specific execution contexts indicates the presence of genuine structural variation in JRE JVM execution behaviour. This suggests that certain runtime configurations may exhibit partially distinct underlying execution relationships that are not fully captured within the aggregated pooled main-effects specification.

Table 7.3: Cross-classification of influential observations identified through Cook's distance analysis. Percentages are calculated relative to all influential observations ($n = 96$).

Workload Category	**HS**	**INT**	**Total**
Apache	1 (1.0%)	14 (14.6%)	15 (15.6%)
Concurrency	2 (2.1%)	6 (6.3%)	8 (8.3%)
Database	4 (4.2%)	0 (0.0%)	4 (4.2%)
Functional	0 (0.0%)	11 (11.5%)	11 (11.5%)
Scala	5 (5.2%)	52 (54.2%)	57 (59.4%)
Web	1 (1.0%)	0 (0.0%)	1 (1.0%)
Total	13 (13.5%)	83 (86.5%)	96 (100.0%)

Assessment of Homoskedasticity

Formal statistical assessment using the studentized Breusch–Pagan test indicated statistically significant evidence of non-constant residual variance within the pooled power use regression model ($BP = 1306.80$, $df = 22$, $p < .001$). This

indicates that residual variance was not constant across observations, suggesting the presence of heteroskedasticity within the fitted model.

Given the heterogeneous structure of the experimental design, incorporating multiple runtime implementations, execution strategies, garbage collector variants, workload category, and emergent microarchitectural performance event measures, this result was not unexpected. Rather than indicating random model instability, the observed heteroskedasticity likely reflects genuine structural differences in variance across distinct JRE JVM execution contexts operating under differing runtime conditions.

Because coefficient standard errors estimated under ordinary least squares assume constant residual variance, evidence of heteroskedasticity indicates that conventional model-based inferential estimates may be unreliable under the present data conditions. For this reason, the hierarchical regression coefficients reported previously in Table 7.2 were estimated and interpreted using HC3 heteroskedasticity-consistent robust standard errors, which provide more reliable coefficient inference under conditions of non-constant residual variance.

Assessment of Multicollinearity

Multicollinearity diagnostics were assessed in order to determine whether excessive linear dependency existed among the predictor variables included within the regression model. Because the predictor set incorporates both experimentally manipulated runtime configuration factors and emergent microarchitectural performance event measures, some degree of intercorrelation between predictors was anticipated.

Multicollinearity was assessed using adjusted Generalised Variance Inflation Factor (GVIF) statistics calculated for all predictors included within the fully specified pooled power use model. Following conventional practice, adjusted GVIF values substantially greater than 5 were treated as indicative of potentially problematic collinearity.

Table 7.4 summarises the adjusted GVIF statistics obtained for the pooled power use model.

Table 7.4: Adjusted Generalised Variance Inflation Factor statistics for predictors included within the pooled power use regression model.

Predictor	**Adjusted GVIF**
JRE Source Version	1.046
GNU GCC Version	1.003
Execution Mode	2.635
Garbage Collector Variant	1.037
Workload Category	1.123
Branch Misprediction Rate	2.951
L1 Data Cache Load Miss Rate	1.864
L2 Data Cache Load Miss Rate	1.815
Page Fault Rate	1.261

Inspection of the results indicated that no predictor exhibited evidence of problematic multicollinearity. The largest adjusted GVIF values were observed for branch misprediction rate (2.951) and execution mode (2.635), suggesting moderate association among predictors reflecting partially shared underlying execution processes. However, all values remained comfortably below commonly accepted thresholds associated with unstable coefficient estimation.

These findings indicate that multicollinearity is unlikely to substantially distort coefficient estimation or undermine interpretation of the subsequent regression modelling results. Consequently, all predictors were retained without modification for subsequent model estimation procedures.

Sensitivity Analysis

To further assess the robustness of the pooled regression specification, a sensitivity analysis was performed examining the effect of excluding observations identified as influential through Cook's distance diagnostics. A total of 96 observations exceeding the conventional influence threshold of $4/n$ were removed and the pooled regression model was subsequently re-estimated.

Comparison between the original pooled model and the sensitivity model

permits assessment of the extent to which model explanatory performance and coefficient estimation were affected by the relatively small subset of observations previously identified as exerting disproportionate influence over model estimation.

Table 7.5 summarises the comparison between the original pooled regression model and the sensitivity model estimated after removal of influential observations. Despite excluding only 96 observations (2.08% of the total dataset), substantial changes in overall model performance were observed. Explained variance increased from $R^2 = 0.488$ to $R^2 = 0.550$, while residual standard error decreased from 0.300 to 0.279.

These results indicate that a relatively small subset of observations contributed disproportionately to residual variance within the pooled regression specification. The improvement in explanatory performance suggests that the majority of observations conform more closely to the underlying structural relationships represented by the theoretical model.

Table 7.5: Final pooled power use regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. Coefficients are reported in watts. The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$.

	Full Model			Sensitivity Model			Dir.	Sig.
Predictor	β	**SE**	p	β	**SE**	p		
Intercept	4.080	0.065	$< .001$	4.214	0.026	$< .001$	Yes	Yes
JRE 10	-0.002	0.016	.897	0.001	0.015	.954	No	Yes
JRE 11	-0.001	0.016	.966	0.006	0.015	.705	No	Yes
JRE 12	-0.003	0.016	.859	-0.008	0.015	.583	Yes	Yes
JRE 13	0.348	0.017	$< .001$	0.355	0.016	$< .001$	Yes	Yes
JRE 14	0.038	0.018	.028	0.060	0.015	$< .001$	Yes	Yes
GCC 6	0.057	0.012	$< .001$	0.054	0.012	$< .001$	Yes	Yes
GCC 7	0.056	0.012	$< .001$	0.055	0.012	$< .001$	Yes	Yes
GCC 8	0.042	0.013	.001	0.035	0.012	.003	Yes	Yes
INT	-0.028	0.036	.435	-0.121	0.022	$< .001$	Yes	No
CPGC	0.087	0.032	.006	0.158	0.026	$< .001$	Yes	Yes
G1GC	0.050	0.020	.012	0.085	0.012	$< .001$	Yes	Yes
PGC	0.037	0.021	.077	0.074	0.012	$< .001$	Yes	No
SGC	-0.033	0.017	.051	-0.015	0.012	.210	Yes	No
Concurrency	0.506	0.017	$< .001$	0.504	0.014	$< .001$	Yes	Yes
Database	-0.109	0.018	$< .001$	-0.115	0.018	$< .001$	Yes	Yes
Functional	0.029	0.048	.540	0.149	0.015	$< .001$	Yes	No
Scala	-0.048	0.026	.071	-0.003	0.015	.862	Yes	No
Web	0.570	0.018	$< .001$	0.589	0.019	$< .001$	Yes	Yes
BMR	-0.002	0.003	.489	0.005	0.002	.035	No	No
L1DLMR	-0.019	0.010	.054	-0.053	0.007	$< .001$	Yes	No
L2DLMR	-0.008	0.002	$< .001$	-0.011	0.001	$< .001$	Yes	Yes
PFMR	-1.194	0.497	.016	-2.217	0.085	$< .001$	Yes	Yes

Dir. = coefficient direction remained unchanged.
Sig. = significance status retained (Y) or changed (C).

Several predictor coefficients also exhibited substantial changes following removal of influential observations. Most notably, interpretive execution changed from statistically non-significant in the full model ($\beta = -0.028$, $p = .435$) to strongly statistically significant in the sensitivity model ($\beta = -0.121$, $p < .001$). Similarly, the functional workload category level, branch misprediction rate, and L1 data cache load miss rate all became statistically significant following removal

of influential observations. The coefficient associated with page fault rate additionally increased substantially in magnitude, changing from $\beta = -1.194$ to $\beta = -2.217$.

These coefficient shifts indicate that the influential observations identified previously were not randomly distributed statistical anomalies, but instead represented structurally distinct execution contexts exerting disproportionate influence over pooled model estimation.

Taken collectively, the sensitivity analysis indicates that although the pooled regression model captures substantial variation in JRE JVM power use across a highly heterogeneous experimental dataset, a relatively small subset of execution contexts exhibits distinct underlying structural relationships that partially obscure broader patterns operating across the majority of observations.

Importantly, these influential observations correspond to experimentally valid runtime conditions rather than measurement error. Their disproportionate influence therefore suggests that JRE JVM power use is governed by partially heterogeneous execution relationships that vary across specific runtime configurations. More broadly, these findings reinforce the inherent complexity associated with modelling highly heterogeneous JRE JVM execution environments spanning multiple runtime implementations, workload category, and execution strategies.

Bootstrap Resampling Analysis

To further assess coefficient stability, a stratified non-parametric bootstrap procedure was performed using 1,000 resamples. Resampling was stratified by experimental design, defined by OpenJDK JRE source version, GNU GCC build toolchain compiler version, execution mode, garbage collector variant, and workload category. This approach preserves the structure of the experimental design while assessing the stability of coefficient estimates under repeated resampling.

Table 7.6 summarises the original coefficient estimates, bootstrap mean estimates, bootstrap standard errors, and percentile-based 95% confidence intervals

Table 7.6: Stratified bootstrap coefficient stability analysis for the pooled power use regression model.

Predictor	Orig. β	Boot. Mean	Boot. SE	CI_Lower	CI_Upper
(Intercept)	4.080	4.088	0.044	4.018	4.196
JRE 10	-0.002	-0.002	0.014	-0.029	0.025
JRE 11	-0.001	0.000	0.014	-0.026	0.026
JRE 12	-0.003	-0.003	0.014	-0.029	0.024
JRE 13	0.348	0.348	0.014	0.321	0.373
JRE 14	0.038	0.040	0.014	0.013	0.068
GCC6	0.057	0.057	0.010	0.037	0.077
GCC7	0.056	0.056	0.010	0.036	0.075
GCC8	0.042	0.042	0.011	0.022	0.064
INT	-0.028	-0.033	0.025	-0.087	0.011
CPGC	0.087	0.091	0.025	0.041	0.139
G1GC	0.050	0.052	0.014	0.027	0.082
PGC	0.037	0.040	0.015	0.012	0.070
SGC	-0.033	-0.031	0.013	-0.055	-0.005
Concurrency	0.506	0.505	0.015	0.474	0.535
Database	-0.109	-0.110	0.014	-0.135	-0.081
Functional	0.029	0.036	0.031	-0.013	0.115
Scala	-0.048	-0.043	0.019	-0.078	-0.001
Web	0.570	0.570	0.012	0.547	0.593
BMR	-0.002	-0.002	0.002	-0.006	0.003
L1DLMR	-0.019	-0.020	0.008	-0.036	-0.006
L2DLMR	-0.008	-0.008	0.001	-0.011	-0.007
PFMR	-1.194	-1.260	0.312	-2.094	-0.831

Note: Orig. β = Original Beta; Boot. Mean = Bootstrap Mean; Boot. SE = Bootstrap Standard Error;

for the pooled power use regression model.

The bootstrap results indicated a high degree of coefficient stability across most predictors. Original coefficient estimates were closely aligned with the corresponding bootstrap mean estimates, indicating that the fitted model coefficients were not strongly dependent upon a particular sample configuration. The elevated power use associated with OpenJDK JRE version 13 remained highly stable, with the bootstrap confidence interval excluding zero. Similarly, workload effects for `concurrency` and `web` remained large, positive, and stable across bootstrap resamples, while the `database` workload remained consistently negative.

The bootstrap analysis also reinforced the more cautious interpretation of several predictors. The confidence interval for interpretive execution included zero, consistent with its non-significant coefficient in the full model. Similarly, the confidence intervals for branch misprediction rate and the `functional` workload category level included zero, indicating weaker coefficient stability for these predictors within the pooled specification.

Overall, the bootstrap resampling analysis supports the reliability of the principal coefficient patterns observed in the pooled power use model. The strongest and most theoretically important relationships, particularly those associated with OpenJDK JRE version 13 and workload category, remained stable under stratified resampling. At the same time, predictors with confidence intervals crossing zero were interpreted cautiously, consistent with the broader diagnostic evidence that some relationships are partly conditioned by heterogeneous execution contexts.

7.2.4 Summary

This section addressed **SRQ3.1**, which examined the extent to which linear models incorporating selected runtime configuration factors, workload category, and microarchitectural performance event measures account for observed variation in JRE JVM power use.

The hierarchical regression analysis demonstrated that the theoretically specified predictor set accounted for a substantial proportion of observed variation in pooled JRE JVM power use. The final model explained approximately 48.8% of variance in power use, which is a notable level of explanatory performance given the substantial heterogeneity of the experimental dataset, comprising multiple JRE source versions, compiler build configurations, execution modes, garbage collector variants, workload category, and emergent microarchitectural performance event measures.

The largest incremental contribution was associated with workload category, which increased explained variance by approximately 32.4 percentage points. This

indicates that workload category represented the strongest directly specified predictor group within the pooled power use model. Runtime implementation variables accounted for a further 11.3% of observed variation, while execution strategy variables added a comparatively small additional contribution. Microarchitectural performance event measures contributed a further 3.9 percentage points, indicating that lower-level emergent execution characteristics provided additional explanatory value beyond the directly specified runtime configuration factors.

Coefficient analysis indicated that the elevated power use associated with OpenJDK JRE version 13 remained stable across all hierarchical model stages. In contrast, several coefficients changed following the introduction of microarchitectural performance event measures, indicating that relationships observed at higher levels of runtime configuration were partly conditioned by lower-level emergent execution characteristics.

Regression diagnostics indicated that the pooled model provided an adequate overall fit across the majority of observations, although departures from residual normality, heteroskedasticity, and influential observations were observed. These diagnostic findings were consistent with the heterogeneous structure of the experimental design rather than indicating simple model failure. HC3 robust standard errors were therefore used to support coefficient inference under non-constant residual variance.

Sensitivity analysis further demonstrated that a small subset of influential observations exerted disproportionate influence over pooled model estimation. Removal of 96 influential observations increased explained variance from $R^2 = 0.488$ to $R^2 = 0.550$, indicating that most observations conformed more closely to the underlying structural relationships represented by the theoretical model. However, the coefficient changes observed in the sensitivity model also indicated that these influential observations represented experimentally valid execution contexts with partially distinct relationships, rather than invalid measurements.

Bootstrap resampling analysis supported the stability of the principal coefficient

patterns. The strongest relationships, particularly those associated with OpenJDK JRE version 13 and workload category, remained stable under stratified resampling. Predictors whose bootstrap confidence intervals crossed zero, including interpretive execution, branch misprediction rate, and the `functional` workload category, were interpreted more cautiously.

Overall, the findings indicate that the theoretically specified pooled regression model accounted for a substantial proportion of observed variation in JRE JVM power use when runtime configuration factors, workload category, and emergent microarchitectural performance event measures were considered together. However, the diagnostic and sensitivity analyses also indicate that JRE JVM power use is characterised by partially heterogeneous execution relationships. The pooled model therefore provides a useful aggregate representation of power use across the full experimental dataset, while also demonstrating that some execution contexts exhibit relationships not fully captured by a single global specification.

Section Research Question Finding

SRQ3.1: The pooled regression model incorporating runtime configuration factors, workload category, and selected microarchitectural performance event measures accounted for a substantial proportion of observed variation in JRE JVM power use. The final model explained approximately 48.8% of observed variance, with workload category contributing the largest incremental increase in explained variance. Bootstrap analysis supported the stability of the principal coefficient patterns, particularly those associated with OpenJDK JRE version 13 and workload category. However, regression diagnostics and sensitivity analysis indicated that the pooled model contained structurally influential execution contexts, suggesting that JRE JVM power use is partly governed by heterogeneous execution relationships across runtime configurations and workload conditions.

7.3 Power Use Model: Interpretive Execution

This section principally addresses sub-research question **SRQ3.1a**, which examines the extent to which variation in JRE/JVM interpretive execution power use can be accounted for through regression modelling incorporating runtime configuration factors, workload category, and microarchitectural performance event measures:

> **SRQ3.1a:** ***To what extent can variation in JRE/JVM interpretive execution power use be accounted for through regression modelling incorporating JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, workload category, and microarchitectural performance event measures?***

To further examine whether relationships identified within the pooled power use model remained stable when restricting analysis to a single execution mode, the regression analysis was repeated separately for observations generated under interpretive execution only.

The interpretive execution model retains the same theoretically specified predictor set and hierarchical modelling structure introduced previously for the pooled power use analysis. However, because all observations correspond exclusively to interpretive execution, the execution mode predictor (JEM) is omitted from the model specification. This execution-mode-specific decomposition permits assessment of the extent to which predictor relationships observed within the pooled model remain stable under interpretive execution only.

7.3.1 Hierarchical Regression Results

Predictors were entered hierarchically according to the same theoretically motivated layered systems architecture introduced previously, reflecting progressively deeper layers of system-level influence on observed JRE JVM power use. Because

the present analysis is restricted exclusively to interpretive execution, execution mode is no longer included as an explanatory factor within the hierarchical modelling procedure. The block ordering follows that presented for the pooled analysis.

This hierarchical structure therefore permits assessment of the extent to which progressively deeper system characteristics account for variation in JRE JVM power use when analysis is restricted to a single execution mode.

Table 7.7: Hierarchical regression model development for JRE JVM interpretive execution power use. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block.

Model Block	R^2	**Adj.** R^2	ΔR^2	ΔF	p
Block 1: Runtime Implementation	0.078	0.075	0.078	–	–
Block 2: Execution Strategy	0.078	0.073	0.000	0.120	0.975
Block 3: Workload Category	0.519	0.516	0.441	518.833	$< 0.001^{***}$
Block 4: Microarchitecture	0.620	0.616	0.100	147.673	$< 0.001^{***}$
Final Model	0.620	0.616	–	–	$< 0.001^{***}$

The explanatory contribution associated with each hierarchical predictor block is summarised in Table 7.7. Examination of hierarchical model progression reveals a substantially different explanatory structure relative to the pooled regression model. The first predictor block, comprising foundational runtime implementation characteristics, explained approximately 7.8% of observed variation in JRE JVM power use, indicating a more limited contribution than observed within the pooled specification.

In contrast to the pooled model, introduction of garbage collector variant in Block 2 produced effectively no increase in explanatory power, with explained variance remaining unchanged following inclusion of garbage collector variant. This suggests that under interpretive execution, garbage collector variant contributes negligibly to explaining observed variation in power use.

The largest increase in explanatory power again occurred following introduction of workload category in Block 3, which increased explained variance

by approximately 44.1 percentage points. This increase exceeds that observed within the pooled specification, indicating that workload category accounts for a substantially larger proportion of observed variation in power use when execution mode heterogeneity is removed.

The final block, incorporating emergent microarchitectural performance event measures, contributed an additional 10.0% of explained variance, increasing overall model explanatory performance to approximately 62.0%. This represents a substantially larger incremental contribution than observed in the pooled specification and suggests that lower-level microarchitectural performance event measures play a more substantial explanatory role under interpretive execution.

Although hierarchical model progression quantifies the incremental explanatory contribution associated with each predictor block, examination of coefficient changes across successive model stages additionally permits assessment of the extent to which predictor relationships remain stable as progressively deeper execution characteristics are introduced. Table 7.8 therefore reports coefficient estimates associated with each hierarchical model stage.

As with the pooled specification, preliminary diagnostic assessment indicated evidence of heteroskedasticity. Coefficient estimates are therefore reported using HC3 heteroskedasticity-consistent robust standard errors to ensure inferential robustness under conditions of non-constant residual variance.

Examination of coefficient magnitude change across hierarchical model stages reveals several notable structural relationships. The elevated power use associated with OpenJDK JRE version 13 remained consistently positive and highly statistically significant across all four model stages, indicating a stable and robust relationship independent of subsequently introduced execution characteristics.

Several coefficients, however, exhibited substantial changes as progressively deeper system characteristics were introduced. Most notably, garbage collector effects remained uniformly non-significant throughout the early model stages but became statistically significant for several collector variants following introduction

Table 7.8: Hierarchical regression coefficients for interpretive execution JRE JVM power use. Standard errors correspond to HC3 heteroskedasticity-consistent robust variance estimation.

	Block 1			Block 2			Block 3			Block 4		
Predictor	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**
Intercept	3.944	0.027	***	3.947	0.034	***	4.050	0.028	***	4.490	0.033	***
JRE 10	0.016	0.035		0.016	0.038		0.016	0.027		0.005	0.024	
JRE 11	0.008	0.034		0.009	0.037		0.009	0.027		-0.008	0.025	
JRE 12	0.020	0.035		0.021	0.038		0.011	0.028		-0.004	0.025	
JRE 13	0.366	0.030	***	0.367	0.033	***	0.357	0.025	***	0.271	0.025	***
JRE 14	0.024	0.036		0.024	0.039		0.014	0.028		0.062	0.025	*
GCC 6	0.084	0.026	**	0.084	0.026	**	0.084	0.019	***	0.087	0.017	***
GCC 7	0.089	0.026	***	0.089	0.027	***	0.089	0.020	***	0.097	0.017	***
GCC 8	0.083	0.027	**	0.083	0.027	**	0.083	0.020	***	0.091	0.017	***
CPGC	–	–		0.002	0.058		0.002	0.042		-0.028	0.039	
G1GC	–	–		-0.001	0.030		-0.001	0.022		0.108	0.020	***
PGC	–	–		-0.001	0.028		-0.001	0.020		0.114	0.019	***
SGC	–	–		-0.011	0.028		-0.011	0.020		0.063	0.018	***
Concurrency	–	–		–	–		0.373	0.026	***	0.432	0.027	***
Database	–	–		–	–		-0.337	0.016	***	-0.191	0.020	***
Functional	–	–		–	–		-0.373	0.020	***	0.007	0.026	
Scala	–	–		–	–		-0.416	0.023	***	-0.212	0.032	***
Web	–	–		–	–		0.514	0.017	***	1.243	0.053	***
BMR	–	–		–	–		–	–		-0.018	0.003	***
L1DLMR	–	–		–	–		–	–		-0.006	0.010	
L2DLMR	–	–		–	–		–	–		-0.083	0.005	***
PFMR	–	–		–	–		–	–		-3.524	0.157	***

Significance codes: *** $p < .001$; ** $p < .01$; * $p < .05$; . $p < .10$.

of microarchitectural performance event measures in Block 4. This suggests that part of the relationship between garbage collector variant and observed power use may be conditioned indirectly through lower-level execution characteristics emerging during runtime.

workload category remained the dominant directly specified predictor group, although several workload coefficients exhibited substantial magnitude shifts following introduction of microarchitectural performance event measures. In particular, the coefficient associated with the `web` workload increased substantially following inclusion of the microarchitectural predictors, while the `functional` workload effect became statistically non-significant.

The microarchitectural predictors themselves exhibited comparatively strong explanatory influence. Branch misprediction rate, L2 data cache load miss rate, and page fault rate all emerged as highly statistically significant predictors within the fully specified model, while page fault rate exhibited the largest coefficient magnitude among all continuous predictors.

Taken collectively, these findings indicate that under interpretive execution, JRE JVM power use is more strongly conditioned by workload-specific computational demands and lower-level microarchitectural performance event measures than was observed within the pooled specification. More broadly, the substantial improvement in overall explanatory performance relative to the pooled model suggests that restricting analysis to a single execution mode reduces the structural heterogeneity previously observed within the aggregated model.

Estimated Final Model

The final hierarchical regression model can be expressed as:

$$\begin{aligned}\widehat{PowerUse}_i = {} & 4.490 + 0.005\,JRE10_i - 0.008\,JRE11_i - 0.004\,JRE12_i \\ & + 0.271\,JRE13_i + 0.062\,JRE14_i + 0.087\,GCC6_i \\ & + 0.097\,GCC7_i + 0.091\,GCC8_i - 0.028\,CPGC_i \\ & + 0.108\,G1GC_i + 0.114\,PGC_i + 0.063\,SGC_i \\ & + 0.432\,Concurrency_i - 0.191\,Database_i + 0.007\,Functional_i \\ & - 0.212\,Scala_i + 1.243\,Web_i - 0.018\,BMR_i \\ & - 0.006\,L1DLMR_i - 0.083\,L2DLMR_i - 3.524\,PFMR_i + \epsilon_i\end{aligned} \tag{7.3}$$

The estimated model coefficients are interpreted relative to the omitted reference categories corresponding to OpenJDK JRE 9, GCC 5, tiered execution, the default garbage collector baseline, and the `apache` workload category.

7.3.2 Regression Diagnostics

Residual diagnostic assessment indicated improved model behaviour relative to the pooled power use specification. Residuals remained closely centred around zero (mean residual approximately zero; median = -0.037), suggesting no evidence of systematic prediction bias. Examination of the residual frequency distribution indicated an approximately bell-shaped central distribution, with the majority of residuals concentrated close to zero.

A modest degree of positive skew remained visible within the residual distribution, with the positive residual tail extending slightly further than the corresponding negative tail, Figure 7.3. However, unlike the pooled regression model, no pronounced extreme tail behaviour or severe outlying residuals were observed. Standardised residuals ranged from approximately -2.10 to 2.90, indicating substantially reduced residual dispersion relative to the pooled specification.

Collectively, these results suggest that restricting analysis to interpretive execution reduced the structural heterogeneity previously observed within the aggregated pooled dataset.

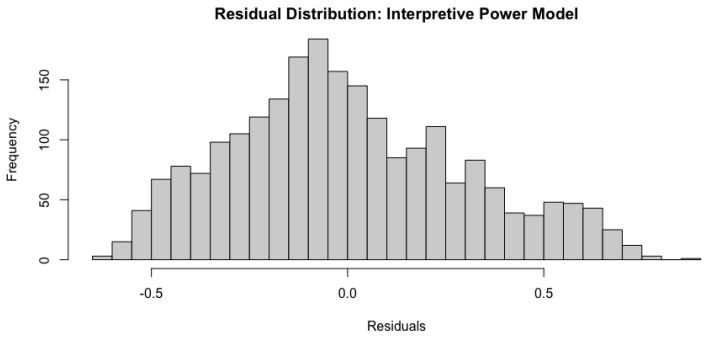


Figure 7.3: Residual Distribution Diagnostics (Interpretive Power Use Model): Histogram depicting the distribution of regression residuals associated with the interpretive power use model.

Formal assessment of residual normality using the Shapiro–Wilk test indicated statistically significant deviation from normality ($W = 0.979$, $p < .001$). However, given the large sample size ($n = 2256$) and the comparatively well-behaved residual distribution observed visually, this result was not considered substantively problematic.

Evidence of heteroskedasticity remained present within the interpretive execution model, with the Breusch–Pagan test indicating statistically significant non-constant residual variance ($BP = 530.17$, $p < .001$). Consequently, coefficient inference continued to rely upon HC3 heteroskedasticity-consistent robust standard errors.

Multicollinearity diagnostics indicated no evidence of problematic predictor dependency, with all adjusted GVIF values remaining below conventional thresholds. Although moderate association was observed for branch misprediction rate (2.14), L1 data cache load miss rate (2.00), and L2 data cache load miss rate (3.12), these values were not considered sufficiently large to threaten coefficient stability.

Sensitivity Analysis

The sensitivity analysis indicated that the interpretive execution model was substantially more stable than the pooled power use model. Following removal of influential observations, the majority of coefficient estimates remained directionally consistent and of broadly similar magnitude. The elevated power use associated with OpenJDK JRE version 13 remained positive and statistically significant, while the principal workload effects and microarchitectural performance event measures also remained stable.

The strongest workload-specific relationships were preserved following sensitivity analysis. The `web` workload remained strongly positively associated with power use, while the `database` and `scala` workload categories remained negatively associated. Similarly, branch misprediction rate, L2 data cache load miss rate, and page fault rate retained their negative and statistically significant relationships with observed power use.

The main change following removal of influential observations concerned L1 data cache load miss rate, which changed from a non-significant coefficient in the full interpretive model to a statistically significant negative coefficient in the sensitivity model. This suggests that the relationship between L1 data cache load miss activity and power use under interpretive execution may be somewhat sensitive to influential observations.

Table 7.9: Final interpretive power use regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$.

	Full Model			**Sensitivity Model**			**Dir.**	**Sig.**
Predictor	β	**SE**	p	β	**SE**	p		
Intercept	4.490	0.033	$< .001$	4.477	0.032	$< .001$	Y	Y
JRE 10	0.005	0.024	.849	0.027	0.023	.243	Y	Y
JRE 11	-0.008	0.025	.738	0.019	0.023	.406	N	Y
JRE 12	-0.004	0.025	.890	0.012	0.023	.601	N	Y
JRE 13	0.271	0.025	$< .001$	0.319	0.023	$< .001$	Y	Y
JRE 14	0.062	0.025	.015	0.082	0.024	$< .001$	Y	Y
GCC 6	0.087	0.017	$< .001$	0.089	0.016	$< .001$	Y	Y
GCC 7	0.097	0.017	$< .001$	0.095	0.016	$< .001$	Y	Y
GCC 8	0.091	0.017	$< .001$	0.087	0.016	$< .001$	Y	Y
CPGC	-0.028	0.039	.473	-0.025	0.033	.441	Y	Y
G1GC	0.108	0.020	$< .001$	0.121	0.019	$< .001$	Y	Y
PGC	0.114	0.019	$< .001$	0.129	0.018	$< .001$	Y	Y
SGC	0.063	0.018	$< .001$	0.079	0.017	$< .001$	Y	Y
Concurrency	0.432	0.027	$< .001$	0.411	0.026	$< .001$	Y	Y
Database	-0.191	0.020	$< .001$	-0.181	0.019	$< .001$	Y	Y
Functional	0.007	0.026	.773	-0.002	0.025	.946	N	Y
Scala	-0.212	0.032	$< .001$	-0.248	0.031	$< .001$	Y	Y
Web	1.243	0.053	$< .001$	1.263	0.049	$< .001$	Y	Y
BMR	-0.018	0.003	$< .001$	-0.014	0.003	$< .001$	Y	Y
L1DLMR	-0.006	0.010	.572	-0.028	0.009	.003	Y	C
L2DLMR	-0.083	0.005	$< .001$	-0.082	0.005	$< .001$	Y	Y
PFMR	-3.524	0.157	$< .001$	-3.377	0.153	$< .001$	Y	Y

Dir. = coefficient direction remained unchanged.
Sig. = significance status retained (Y) or changed (C).

Overall, the sensitivity analysis indicates that the interpretive execution model exhibits greater coefficient stability than the pooled power use model. This supports the interpretation that restricting analysis to a single execution mode reduces the structural heterogeneity observed within the pooled specification.

Bootstrap Validation

To further assess coefficient stability, a stratified non-parametric bootstrap procedure was performed using 1,000 resamples. Resampling was stratified according

Table 7.10: Stratified bootstrap coefficient stability analysis for the interpretive execution power use regression model.

Predictor	Orig. β	Boot. Mean	Boot. SE	CI_Lower	CI_Upper
(Intercept)	4.490	4.490	0.031	4.428	4.549
JRE 10	0.005	0.006	0.024	-0.039	0.052
JRE 11	-0.008	-0.007	0.023	-0.051	0.042
JRE 12	-0.004	-0.003	0.023	-0.048	0.041
JRE 13	0.271	0.273	0.024	0.225	0.318
JRE 14	0.062	0.063	0.024	0.015	0.108
GCC 6	0.087	0.087	0.015	0.058	0.118
GCC 7	0.097	0.097	0.016	0.066	0.128
GCC 8	0.091	0.091	0.016	0.059	0.122
CPGC	-0.028	-0.026	0.036	-0.092	0.046
G1GC	0.108	0.108	0.018	0.072	0.144
PGC	0.114	0.114	0.018	0.081	0.151
SGC	0.063	0.063	0.017	0.029	0.096
Concurrency	0.432	0.432	0.025	0.384	0.480
Database	-0.191	-0.191	0.018	-0.227	-0.154
Functional	0.007	0.008	0.024	-0.038	0.056
Scala	-0.212	-0.211	0.030	-0.268	-0.151
Web	1.243	1.244	0.044	1.163	1.331
BMR	-0.018	-0.018	0.003	-0.023	-0.012
L1DLMR	-0.006	-0.006	0.009	-0.023	0.011
L2DLMR	-0.083	-0.083	0.005	-0.092	-0.074
PRMR	-3.524	-3.524	0.134	-3.792	-3.273

Note: Orig. β = Original Beta; Boot. Mean = Bootstrap Mean; Boot. SE = Bootstrap Standard Error;

to the experimental design, defined by OpenJDK JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, and workload category. This procedure preserves the structure of the experimental design while assessing coefficient stability under repeated resampling.

Table 7.10 summarises the original coefficient estimates, bootstrap mean estimates, bootstrap standard errors, and percentile-based 95% confidence intervals for the interpretive execution power use regression model.

The bootstrap results indicated an exceptionally high degree of coefficient stability across the interpretive execution model. Original coefficient estimates remained almost perfectly aligned with the corresponding bootstrap mean es-

timates for all predictors, indicating that the fitted regression coefficients were not strongly dependent upon any particular sample configuration. Across nearly all predictors, repeated resampling produced remarkably consistent coefficient estimates, suggesting a highly stable underlying model structure.

The strongest relationships observed within the regression model remained highly stable under repeated resampling. The elevated power use associated with OpenJDK JRE version 13 remained consistently positive, with the bootstrap mean estimate ($\beta = 0.273$) remaining effectively identical to the original model coefficient ($\beta = 0.271$). Similarly, workload effects associated with the `concurrency`, `database`, `scala`, and particularly the strongly positive `web` workload category remained highly stable, with narrow confidence intervals consistently excluding zero. These findings provide further support for the earlier observation that workload category represented one of the strongest determinants of power use variation under interpretive execution.

The principal microarchitectural performance event measures also demonstrated substantial coefficient stability. Branch misprediction rate retained a stable negative relationship with power use, with confidence intervals remaining consistently below zero. L2 data cache load miss rate similarly retained a highly stable negative coefficient structure. Page fault rate exhibited particularly strong coefficient stability, with the bootstrap mean estimate remaining identical to the original coefficient estimate ($\beta = -3.524$), accompanied by a comparatively narrow bootstrap standard error ($SE = 0.134$) and confidence intervals consistently excluding zero. This provides strong evidence that page fault rate represented one of the most robust lower-level predictors within the interpretive execution model.

The bootstrap analysis also reinforced more cautious interpretation of weaker predictors. Confidence intervals associated with JRE versions 10, 11, and 12 all crossed zero, indicating comparatively weak coefficient stability for these runtime versions. Similarly, the CPGC garbage collector variant, the `functional` workload category, and L1 data cache load miss rate all exhibited confidence

intervals overlapping zero, suggesting comparatively limited evidence for stable coefficient structure for these predictors. In contrast, JRE version 14, all GNU GCC compiler variants, G1GC, PGC, and SGC all retained confidence intervals excluding zero, providing stronger evidence that these predictors contributed meaningfully to explaining observed power use variation.

Overall, the bootstrap resampling analysis provides strong evidence supporting the reliability of the interpretive execution power use regression model. Relative to the pooled model, coefficient estimates exhibited substantially greater stability under repeated resampling, further supporting the earlier interpretation that restricting analysis to a single execution mode reduces structural heterogeneity present within the aggregated pooled specification. Taken together, these findings provide additional support for the broader modelling observation developed throughout RQ3 that increasingly granular decomposition of execution context systematically improves coefficient stability and overall model robustness.

7.3.3 Summary

This section addressed **SRQ3.1a**, which examined the extent to which variation in JRE/JVM interpretive execution power use can be accounted for through regression modelling incorporating runtime configuration factors, workload category, and microarchitectural performance event measures.

The interpretive execution power use regression model accounted for a substantial proportion of observed variation in JRE/JVM power use, with the final model explaining approximately 62.0% of observed variance ($R^2 = .620$, adjusted $R^2 = .616$). This represented a marked improvement relative to the corresponding pooled specification, indicating that restricting analysis to a single execution mode reduced structural heterogeneity previously present within the aggregated model.

Hierarchical model development showed that workload category remained the dominant directly specified predictor group, accounting for the largest incremental increase in explained variance ($\Delta R^2 = .441$). In contrast, garbage collector variant

contributed negligibly when introduced independently, suggesting that under interpretive execution garbage collector effects were comparatively weak when considered before workload and microarchitectural performance event measures. The subsequent introduction of microarchitectural performance event measures produced a substantial additional increase in explained variance ($\Delta R^2 = .100$), indicating that lower-level execution characteristics exerted a stronger contribution under interpretive execution than was observed within the pooled specification.

Coefficient analysis demonstrated that the elevated power use associated with OpenJDK JRE version 13 remained consistently positive and statistically significant across all model stages, confirming a stable runtime-specific effect independent of subsequently introduced predictors. Workload effects also remained central to the model, particularly the strong positive association observed for the `web` workload category. Among the microarchitectural predictors, branch misprediction rate, L2 data cache load miss rate, and page fault rate emerged as statistically significant predictors, indicating that interpretive execution power use was strongly conditioned by lower-level execution characteristics.

Regression diagnostics and validation procedures supported the reliability of the interpretive execution model. Residuals remained closely centred around zero, residual dispersion was reduced relative to the pooled model, and no problematic multicollinearity was observed. Although residual non-normality and heteroskedasticity remained statistically significant, these issues were mitigated through visual diagnostic assessment, large-sample interpretation, and the use of HC3 heteroskedasticity-consistent robust standard errors. Sensitivity analysis and bootstrap validation further indicated substantial coefficient stability, with the main model relationships remaining stable under influential-observation removal and repeated stratified resampling.

Taken collectively, these findings indicate that variation in JRE/JVM interpretive execution power use can be accounted for to a substantial extent through the proposed regression modelling framework. More broadly, the results provide

further evidence that execution-mode-specific modelling improves model reliability and reveals structural relationships that are partially obscured within pooled specifications.

Section Research Question Finding

SRQ3.1a: Variation in JRE/JVM interpretive execution power use was substantially accounted for through regression modelling incorporating runtime configuration factors, workload category, and microarchitectural performance event measures. The final model explained approximately 62.0% of observed variance, with workload category and microarchitectural performance event measures providing the largest explanatory contributions. These findings indicate that interpretive execution power use is strongly conditioned by workload-specific computational demands and lower-level microarchitectural performance event measures, and that restricting analysis to a single execution mode improves model stability relative to the pooled specification.

7.4 Power Use Model: Tiered Execution

This section principally addresses sub-research question **SRQ3.1b**, which examines the extent to which variation in JRE/JVM tiered execution power use can be accounted for through regression modelling incorporating runtime configuration factors, workload category, and microarchitectural performance event measures:

> **SRQ3.1b:** ***To what extent can variation in JRE/JVM tiered execution power use be accounted for through regression modelling incorporating JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, workload category, and microarchitectural performance event measures?***

To further examine whether relationships identified within the pooled power use model remained stable when restricting analysis to a single execution mode, the regression analysis was repeated separately for observations generated under tiered execution only.

The tiered execution model retains the same theoretically specified predictor set and hierarchical modelling structure introduced previously for the pooled power use analysis. However, because all observations correspond exclusively to tiered execution, the execution mode predictor (JEM) is omitted from the model specification. This execution-mode-specific decomposition permits assessment of the extent to which predictor relationships observed within the pooled model remain stable under tiered execution only.

7.4.1 Hierarchical Regression Results

Predictors were entered hierarchically according to the same theoretically motivated layered systems architecture introduced previously, reflecting progressively lower-level sources of statistical association with observed JRE JVM power use. Because the present analysis is restricted exclusively to tiered execution, execution mode is no longer included as an explanatory factor within the hierarchical modelling procedure. The hierarchical block ordering therefore remains identical to that presented for the pooled and interpretive execution analyses.

This hierarchical structure permits assessment of the extent to which runtime implementation characteristics, garbage collector variant, workload category, and emergent microarchitectural performance event measures account for observed variation in JRE JVM power use when analysis is restricted to a runtime environment dominated by just-in-time compilation.

The explanatory contribution associated with each hierarchical predictor block is summarised in Table 7.11. In contrast to the interpretive execution model, the first predictor block, comprising foundational runtime implementation characteristics, accounted for a comparatively larger proportion of observed variation in

JRE JVM power use ($R^2 = .189$). This indicates that, under tiered execution, OpenJDK JRE source version and GNU GCC build toolchain compiler version were more strongly associated with observed power use than was the case under interpretive execution.

Introduction of garbage collector variant in Block 2 produced a modest but statistically meaningful increase in explanatory performance ($\Delta R^2 = .025$). This differs from the interpretive execution model, where garbage collector variant contributed negligibly when introduced independently. The result suggests that garbage collector variant retains a more visible explanatory role under tiered execution, although its contribution remains smaller than that associated with workload category.

The largest increase in explanatory performance occurred following introduction of workload category in Block 3, which increased explained variance by approximately 40.7 percentage points. This confirms that workload category remained the dominant directly specified predictor group within the tiered execution model. However, the incremental contribution of workload category was slightly smaller than that observed under interpretive execution, indicating that tiered execution power use was less exclusively dominated by workload category and more distributed across runtime implementation, garbage collector, and workload-level factors.

The final block, incorporating branch misprediction rate, L1 data cache load miss rate, L2 data cache load miss rate, and page fault rate, contributed an additional 3.0% of explained variance. This represents a much smaller incremental contribution than that observed within the interpretive execution model. Therefore, although microarchitectural performance event measures remained statistically relevant under tiered execution, they accounted for less additional variation once runtime implementation, garbage collector variant, and workload category had already been included.

Table 7.11: Hierarchical regression model development for JRE JVM tiered execution power use. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block.

Model Block	R^2	**Adj.** R^2	ΔR^2	ΔF	p
Block 1: Runtime Implementation	0.189	0.186	0.189	–	–
Block 2: Execution Strategy	0.213	0.209	0.025	40.836	$< 0.001^{***}$
Block 3: Workload Category	0.620	0.618	0.407	541.878	$< 0.001^{***}$
Block 4: Microarchitecture	0.650	0.647	0.030	49.378	$< 0.001^{***}$
Final Model	0.650	0.647	–	–	$< 0.001^{***}$

Taken together, the hierarchical model progression indicates that tiered execution power use exhibits a different explanatory structure from interpretive execution. Under tiered execution, runtime implementation characteristics and garbage collector variant accounted for a larger share of explained variation, while microarchitectural performance event measures contributed less incremental explanatory power. This suggests that the statistical relationships linking runtime configuration, workload category, and lower-level execution characteristics with observed power use differ materially across execution modes.

Although movement across the hierarchical model blocks quantifies the incremental explanatory contribution associated with each predictor block, examination of coefficient changes across successive model stages additionally permits assessment of the extent to which predictor relationships remain stable as progressively lower-level execution characteristics are introduced. Table 7.12 therefore reports coefficient estimates associated with each hierarchical model stage.

As with the previous regression models, preliminary diagnostic assessment indicated evidence of heteroskedasticity. Coefficient estimates are therefore reported using HC3 heteroskedasticity-consistent robust standard errors to ensure inferential robustness under conditions of non-constant residual variance.

Examination of coefficient magnitude change across hierarchical model stages reveals several notable structural relationships. Most notably, the elevated power use associated with OpenJDK JRE version 13 remained consistently positive

Table 7.12: Hierarchical regression coefficients for tiered execution JRE JVM power use. Standard errors correspond to HC3 heteroskedasticity-consistent robust variance estimation.

	Block 1			**Block 2**			**Block 3**			**Block 4**		
Predictor	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**
Intercept	3.928	0.018	***	3.947	0.022	***	3.734	0.017	***	3.675	0.071	***
JRE 10	0.001	0.022		0.004	0.023		0.004	0.016		-0.018	0.016	
JRE 11	-0.007	0.022		-0.004	0.023		-0.004	0.016		-0.018	0.015	
JRE 12	-0.008	0.021		-0.004	0.022		-0.010	0.015		-0.020	0.014	
JRE 13	0.404	0.018	***	0.407	0.019	***	0.401	0.014	***	0.401	0.018	***
JRE 14	0.030	0.022		0.033	0.023		0.028	0.016	.	0.024	0.017	
GCC 6	0.025	0.018		0.025	0.017		0.025	0.012	*	0.027	0.012	*
GCC 7	0.019	0.018		0.019	0.018		0.019	0.012		0.018	0.012	
GCC 8	0.015	0.018		0.015	0.018		0.015	0.012		-0.005	0.014	
CPGC	–	–		-0.007	0.038		-0.007	0.024		0.076	0.044	.
G1GC	–	–		0.025	0.018		0.025	0.013	*	0.058	0.015	***
PGC	–	–		-0.001	0.018		-0.001	0.013		0.025	0.016	
SGC	–	–		-0.114	0.017	***	-0.114	0.011	***	-0.099	0.014	***
Concurrency	–	–		–	–		0.648	0.016	***	0.625	0.019	***
Database	–	–		–	–		0.087	0.023	***	0.082	0.025	**
Functional	–	–		–	–		0.199	0.012	***	0.200	0.033	***
Scala	–	–		–	–		0.196	0.013	***	0.136	0.026	***
Web	–	–		–	–		0.510	0.011	***	0.412	0.029	***
BMR	–	–		–	–		–	–		-0.023	0.005	***
L1DLMR	–	–		–	–		–	–		0.084	0.019	***
L2DLMR	–	–		–	–		–	–		-0.004	0.001	**
PFMR	–	–		–	–		–	–		-0.443	0.355	

Significance codes: *** $p < .001$; ** $p < .01$; * $p < .05$; . $p < .10$.

and highly statistically significant across all four model stages, with coefficient magnitude remaining remarkably stable throughout the hierarchical modelling procedure. This indicates that the elevated power use associated with JRE version 13 represents a robust runtime-specific relationship that remains largely independent of subsequently introduced execution characteristics.

Several predictor relationships, however, changed substantially following introduction of progressively lower-level execution characteristics. In contrast to the interpretive execution model, garbage collector variant exhibited stronger and more stable relationships throughout the hierarchical modelling process. In particular, the Serial Garbage Collector (SGC) remained consistently negative and highly statistically significant from Block 2 onwards, while the G1GC collector variant became increasingly positive and statistically significant following introduction of workload category and microarchitectural performance event measures. This suggests that garbage collector variant retains a more substantial statistical association with observed power use under tiered execution than was observed under interpretive execution.

throughout the hierarchical modelling process, workload category remained the dominant directly specified predictor group, with all workload categories remaining statistically significant following introduction of microarchitectural performance event measures. However, several workload coefficients exhibited moderate magnitude reductions following introduction of Block 4 predictors. Most notably, the coefficients associated with the `scala` and `web` workload categories decreased following inclusion of the microarchitectural predictors, suggesting that part of the variation initially associated with workload category may be statistically associated with lower-level execution characteristics emerging during runtime. In addition, all workload categories exhibited higher power use relative to the `apache` workload reference category.

The microarchitectural performance event measures themselves exhibited a distinct coefficient structure relative to the interpretive execution model. Branch

misprediction rate, L1 data cache load miss rate, and L2 data cache load miss rate all emerged as statistically significant predictors within the fully specified model. In particular, L1 data cache load miss rate exhibited a positive and highly statistically significant coefficient, contrasting sharply with the interpretive execution model where this predictor remained statistically non-significant. In contrast, page fault rate, which exhibited a strong negative relationship under interpretive execution, became statistically non-significant under tiered execution once the full predictor structure was introduced.

Taken collectively, these findings indicate that JRE JVM power use under tiered execution is characterised by a distinct explanatory structure relative to interpretive execution. Under tiered execution, runtime implementation characteristics and garbage collector variant remained more strongly associated with observed power use, while microarchitectural performance event measures contributed comparatively weaker incremental explanatory performance. More broadly, the substantial differences observed across execution-specific coefficient structures provide further evidence that execution mode moderates the statistical relationships linking runtime configuration and lower-level execution characteristics with observed power use.

Estimated Final Model

The final hierarchical regression model can be expressed as:

$$\begin{aligned}\widehat{PowerUse}_i = 3.675 &- 0.018\,JRE10_i - 0.018\,JRE11_i - 0.020\,JRE12_i \\ &+ 0.401\,JRE13_i + 0.024\,JRE14_i + 0.027\,GCC6_i \\ &+ 0.018\,GCC7_i - 0.005\,GCC8_i + 0.076\,CPGC_i \\ &+ 0.058\,G1GC_i + 0.025\,PGC_i - 0.099\,SGC_i \\ &+ 0.625\,Concurrency_i + 0.082\,Database_i + 0.200\,Functional_i \\ &+ 0.136\,Scala_i + 0.412\,Web_i - 0.023\,BMR_i \\ &+ 0.084\,L1DLMR_i - 0.004\,L2DLMR_i - 0.443\,PFMR_i + \epsilon_i\end{aligned} \tag{7.4}$$

The estimated model coefficients are interpreted relative to the omitted reference categories corresponding to OpenJDK JRE 9, GCC 5, tiered execution, the default garbage collector baseline, and the `apache` workload category.

7.4.2 Regression Diagnostics

Residual diagnostic assessment indicated generally improved overall residual characteristics relative to the pooled power use specification. Residuals remained closely centred around zero (mean residual approximately zero; median = -0.025), suggesting no evidence of systematic prediction bias. Examination of the residual frequency distribution indicated a comparatively concentrated central distribution, with the majority of residuals clustered closely around zero.

Although the central residual distribution appeared comparatively well constrained, a noticeable degree of positive skew remained visible within the residual distribution, Figure 7.4. In contrast to the interpretive execution model, a more pronounced positive residual tail was observed. Standardised residuals ranged from approximately -2.49 to 7.01, indicating the presence of several comparatively large positive residual observations despite generally reduced dispersion across

the majority of observations. Collectively, these results suggest that restricting analysis to tiered execution reduced some of the structural heterogeneity observed within the pooled specification, although a small number of observations remained imperfectly captured by the fitted regression model.

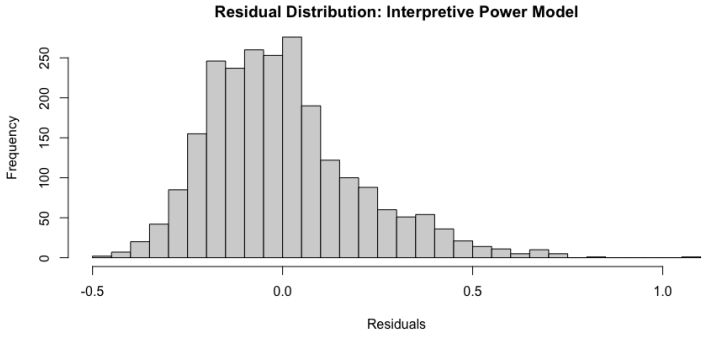


Figure 7.4: Residual Distribution Diagnostics (Tiered Power Use Model): Histogram depicting the distribution of regression residuals associated with the tiered execution power use model.

Formal assessment of residual normality using the Shapiro–Wilk test indicated statistically significant deviation from normality ($W = 0.960$, $p < .001$). This departure from normality was more pronounced than that observed within the interpretive execution model. However, given the large sample size ($n = 2352$) together with the relatively well-behaved central residual distribution observed visually, this departure from normality was not considered substantively problematic.

Evidence of heteroskedasticity remained present within the tiered execution model, with the Breusch–Pagan test indicating statistically significant non-constant residual variance ($BP = 482.46$, $p < .001$). Consequently, coefficient inference continued to rely upon HC3 heteroskedasticity-consistent robust standard errors. Comparison between ordinary least squares and HC3 robust coefficient estimation indicated that heteroskedasticity materially affected standard error estimation for several predictors, with coefficients associated with page fault rate and certain garbage collector variants becoming statistically non-significant following robust

variance adjustment.

Multicollinearity diagnostics indicated no evidence of problematic predictor dependency, with all adjusted GVIF values remaining comfortably below conventional diagnostic thresholds. The largest adjusted GVIF values were observed for branch misprediction rate (2.53) and L1 data cache load miss rate (2.51), suggesting modest shared variance between certain microarchitectural predictors. However, all values remained sufficiently low to indicate that multicollinearity was unlikely to compromise coefficient stability or distort parameter estimation within the fitted regression model.

Examination of influential observations indicated that 123 observations exceeded the conventional Cook's distance threshold ($4/n = 0.00170$), with a maximum Cook's distance of 1.589. This exceeds the maximum Cook's distance observed within both the pooled and interpretive execution models, indicating the presence of several comparatively influential observations within the tiered execution dataset.

Sensitivity Analysis

Sensitivity analysis was performed to assess the extent to which these influential observations affected coefficient stability within the fully specified regression model.

The sensitivity analysis indicated that the tiered execution power use model was more affected by influential observations than the corresponding interpretive execution model. Using the conventional Cook's distance threshold ($4/n = 0.00170$), 123 observations were excluded, reducing the analytical sample from $n = 2352$ to $n = 2229$. The model was then refitted using the same predictor specification and HC3 heteroskedasticity-consistent robust standard errors.

Removal of influential observations improved overall model fit, with R^2 increasing from .650 to .759 and adjusted R^2 increasing from .647 to .756. The residual standard error also decreased from 0.200 to 0.161, indicating that the sensitivity

model provided a tighter fit to the remaining observations.

Table 7.13: Final tiered execution power use regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$. Dependent variable is $\ln(energyWh)$.

	Full Model			**Sensitivity Model**			**Dir.**	**Sig.**
Predictor	β	**SE**	p	β	**SE**	p		
Intercept	3.675	0.071	$< .001$	3.611	0.027	$< .001$	Y	Y
JRE 10	-0.018	0.016	.247	-0.031	0.013	.018	Y	C
JRE 11	-0.018	0.015	.237	-0.037	0.012	.002	Y	C
JRE 12	-0.020	0.014	.154	-0.032	0.012	.008	Y	C
JRE 13	0.401	0.018	$< .001$	0.425	0.013	$< .001$	Y	Y
JRE 14	0.024	0.017	.147	0.005	0.013	.698	Y	Y
GCC 6	0.027	0.012	.024	0.030	0.010	.002	Y	Y
GCC 7	0.018	0.012	.120	0.019	0.010	.049	Y	C
GCC 8	-0.005	0.014	.691	-0.001	0.011	.900	Y	Y
CPGC	0.076	0.044	.086	0.093	0.022	$< .001$	Y	C
G1GC	0.058	0.015	$< .001$	0.071	0.010	$< .001$	Y	Y
PGC	0.025	0.016	.115	0.014	0.010	.161	Y	Y
SGC	-0.099	0.014	$< .001$	-0.068	0.010	$< .001$	Y	Y
Concurrency	0.625	0.019	$< .001$	0.640	0.017	$< .001$	Y	Y
Database	0.082	0.025	.001	0.081	0.019	$< .001$	Y	Y
Functional	0.200	0.033	$< .001$	0.260	0.010	$< .001$	Y	Y
Scala	0.136	0.026	$< .001$	0.162	0.012	$< .001$	Y	Y
Web	0.412	0.029	$< .001$	0.397	0.015	$< .001$	Y	Y
BMR	-0.023	0.005	$< .001$	-0.025	0.004	$< .001$	Y	Y
L1DLMR	0.084	0.019	$< .001$	0.111	0.008	$< .001$	Y	Y
L2DLMR	-0.004	0.001	.006	-0.005	0.001	$< .001$	Y	Y
PFMR	-0.443	0.355	.213	-0.806	0.076	$< .001$	Y	C

Dir. = coefficient direction remained unchanged.
Sig. = significance status retained (Y) or changed (C).

Several relationships remained stable following removal of influential observations. The power use associated with OpenJDK JRE version 13 remained positive and highly statistically significant, increasing slightly from $\beta = .401$ to $\beta = .425$. The principal workload effects also remained positive and statistically significant relative to the `apache` workload reference category, while branch misprediction rate, L1 data cache load miss rate, and L2 data cache load miss rate retained

their original directions and statistical significance.

However, the sensitivity analysis also revealed several notable changes. OpenJDK JRE versions 10, 11, and 12 became statistically significant negative predictors in the sensitivity model, whereas they were non-significant in the full model. Similarly, GCC 7 became statistically significant, while CPGC changed from marginal significance to a statistically significant positive coefficient. Page fault rate also changed substantially, moving from a non-significant coefficient in the full model to a statistically significant negative coefficient in the sensitivity model.

These changes indicate that a subset of influential observations affected the estimated strength and statistical significance of several predictors within the tiered execution model. Nevertheless, the most substantive relationships identified in the full model, particularly elevated power use associated with OpenJDK JRE version 13, the positive workload-category coefficients, and the microarchitectural associations involving branch misprediction, L1 data cache load misses, and L2 data cache load misses, remained directionally consistent.

Overall, the sensitivity analysis suggests that the tiered execution model contains a stronger influential-observation structure than the interpretive execution model. While the main explanatory pattern remained intact, the inferential significance of several secondary predictors changed following removal of influential cases. Consequently, coefficients associated with JRE versions 10–12, GCC 7, CPGC, and page fault rate should be interpreted cautiously within the full tiered execution model.

Bootstrap Validation

To further assess coefficient stability, a stratified non-parametric bootstrap procedure was performed using $1,000$ resamples. Resampling was stratified according to the experimental design, defined by OpenJDK JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, and workload category. This procedure preserves the structure of the experimental design while assessing

Table 7.14: Stratified bootstrap coefficient stability analysis for the tiered execution power use regression model.

Predictor	Orig. β	Boot. Mean	Boot. SE	CI_Lower	CI_Upper
(Intercept)	3.675	3.689	0.054	3.610	3.835
JRE 10	-0.018	-0.018	0.015	-0.046	0.012
JRE 11	-0.018	-0.017	0.015	-0.045	0.012
JRE 12	-0.020	-0.020	0.014	-0.046	0.007
JRE 13	0.401	0.399	0.015	0.369	0.429
JRE 14	0.024	0.026	0.016	-0.004	0.057
GCC 6	0.027	0.027	0.010	0.007	0.047
GCC 7	0.018	0.019	0.011	-0.002	0.040
GCC 8	-0.005	-0.006	0.013	-0.032	0.019
CPGC	0.076	0.085	0.034	0.028	0.169
G1GC	0.058	0.060	0.013	0.037	0.089
PGC	0.025	0.027	0.013	0.004	0.055
SGC	-0.099	-0.097	0.011	-0.117	-0.074
Concurrency	0.625	0.626	0.017	0.591	0.661
Database	0.082	0.081	0.020	0.041	0.118
Functional	0.200	0.207	0.024	0.174	0.276
Scala	0.136	0.142	0.020	0.110	0.193
Web	0.412	0.417	0.020	0.382	0.463
BMR	-0.023	-0.022	0.004	-0.030	-0.014
L1DLMR	0.084	0.080	0.014	0.043	0.099
L2DLMR	-0.004	-0.004	0.001	-0.007	-0.002
PFMR	-0.443	-0.521	0.255	-1.306	-0.254

Note: Orig. β = Original Beta; Boot. Mean = Bootstrap Mean; Boot. SE = Bootstrap Standard Error;

coefficient stability under repeated resampling.

Table 7.14 summarises the original coefficient estimates, bootstrap mean estimates, bootstrap standard errors, and percentile-based 95% confidence intervals for the tiered execution power use regression model.

The bootstrap results indicated a generally high degree of coefficient stability across the tiered execution model. Original coefficient estimates remained closely aligned with the corresponding bootstrap mean estimates for nearly all predictors, indicating that the fitted regression coefficients were not strongly dependent upon any particular sample configuration. This overall coefficient stability is notable given the influential observation sensitivity identified previously within the earlier sensitivity analysis.

Several of the strongest relationships observed within the full regression model remained highly stable under repeated resampling. The elevated power use associated with OpenJDK JRE version 13 remained consistently positive, with the bootstrap mean estimate ($\beta = 0.399$) remaining almost identical to the original model coefficient ($\beta = 0.401$). Similarly, workload coefficients associated with the `concurrency`, `database`, `functional`, `scala`, and `web` workload categories all remained highly stable, with narrow confidence intervals consistently excluding zero. These results reinforce the earlier finding that workload category represented one of the strongest determinants of observed power use variation under tiered execution.

The principal microarchitectural performance event measures also demonstrated substantial coefficient stability. Branch misprediction rate retained a stable negative relationship with power use, with confidence intervals remaining fully below zero. L1 data cache load miss rate remained positively associated with increased power use, while L2 data cache load miss rate retained a smaller but consistently negative coefficient structure. Together, these findings provide additional support for the robustness of the lower-level microarchitectural relationships identified within the full regression model.

The bootstrap analysis also provided additional clarification regarding several runtime configuration variables exhibiting comparatively weaker inferential stability. Confidence intervals associated with JRE versions 10, 11, 12, and 14 all crossed zero, indicating limited evidence of stable coefficient structure for these runtime versions. Similarly, GCC 7 and GCC 8 both exhibited confidence intervals overlapping zero, suggesting weaker coefficient stability relative to GCC 6, which retained a consistently positive confidence interval excluding zero. Among garbage collector variants, G1GC, PGC, and SGC all retained confidence intervals excluding zero, indicating comparatively stable coefficient effects. CPGC also retained a positive confidence interval excluding zero, although the comparatively larger bootstrap standard error ($SE = 0.034$) suggests somewhat greater coefficient

variability relative to the other garbage collector predictors.

Page fault rate produced the greatest evidence of coefficient instability among the microarchitectural predictors. Although the bootstrap confidence interval remained fully below zero, indicating a consistently negative overall relationship with power use, the comparatively large bootstrap standard error ($SE = 0.255$), together with the substantial difference between the original coefficient estimate ($\beta = -0.443$) and bootstrap mean estimate ($\beta = -0.521$), suggests that the estimated magnitude of this relationship remained comparatively sensitive to repeated resampling. This indicates that although the directional relationship appears robust, the precise magnitude of the page fault rate effect should be interpreted more cautiously.

Overall, the bootstrap resampling analysis provides strong evidence supporting the general reliability of the tiered execution power use regression model. Although the preceding sensitivity analysis indicated that several predictors were influenced by a subset of influential observations, repeated stratified resampling demonstrated that the broader coefficient structure remained highly stable. Taken together, these findings suggest that the tiered execution model exhibits a robust underlying explanatory structure, although several secondary runtime configuration predictors and page fault rate require more cautious interpretation than the dominant workload and microarchitectural predictors.

7.4.3 Summary

The tiered execution power use regression model demonstrated strong overall explanatory performance, accounting for approximately 65.0% of observed variation in JRE JVM power use. Although this explanatory performance exceeded that observed within the interpretive execution model, the subsequent diagnostic and validation procedures indicated a more complex coefficient structure, suggesting that several predictor relationships were more sensitive to influential observations despite the model's comparatively strong overall fit.

Hierarchical model development revealed a substantially different explanatory structure relative to the interpretive execution model. Runtime implementation characteristics accounted for a considerably larger proportion of explained variance ($\Delta R^2 = .189$), indicating that OpenJDK JRE source version and GNU GCC build toolchain compiler version were more strongly associated with power use under tiered execution. garbage collector variant also contributed additional explanatory power ($\Delta R^2 = .025$), contrasting with the negligible contribution observed under interpretive execution. Although workload category remained the dominant directly specified predictor group ($\Delta R^2 = .407$), its relative contribution was smaller than that observed within the interpretive model. In contrast, microarchitectural performance event measures contributed only a modest additional increase in explanatory performance ($\Delta R^2 = .030$), substantially smaller than the contribution observed under interpretive execution.

Examination of coefficient changes across hierarchical model stages demonstrated that the elevated power use associated with OpenJDK JRE version 13 remained consistently positive and highly statistically significant throughout all model stages, confirming a stable runtime-specific relationship independent of subsequently introduced execution characteristics. In contrast to the interpretive execution model, garbage collector variant retained a more substantial statistical association with observed power use, with the Serial Garbage Collector (SGC) remaining consistently negative and highly statistically significant, while G1GC became increasingly positive following introduction of progressively lower-level predictors. All workload categories remained positively associated with power use relative to the `apache` workload reference category, while branch misprediction rate, L1 data cache load miss rate, and L2 data cache load miss rate emerged as statistically significant microarchitectural predictors. Notably, L1 data cache load miss rate exhibited a strong positive relationship under tiered execution, contrasting with the comparatively weaker relationship observed under interpretive execution.

Regression diagnostics indicated generally acceptable model behaviour, with residuals remaining closely centred around zero and no evidence of problematic multicollinearity. However, residual normality departures were more pronounced than those observed within the interpretive execution model, and several comparatively large positive residuals remained visible. Formal testing also confirmed persistent heteroskedasticity, requiring continued reliance upon HC3 heteroskedasticity-consistent robust standard errors. Examination of influential observations revealed a comparatively stronger influential structure than was observed in either the pooled or interpretive execution models.

Model validation procedures revealed a more nuanced stability profile than that observed under interpretive execution. Sensitivity analysis demonstrated that removal of influential observations affected the statistical significance of several predictors, including OpenJDK JRE versions 10–12, GCC 7, CPGC, and page fault rate. Nevertheless, the structural relationships identified within the full model remained directionally stable, including the elevated power use associated with OpenJDK JRE version 13, the dominant workload-category effects, and the principal microarchitectural relationships involving branch misprediction rate, L1 data cache load miss rate, and L2 data cache load miss rate.

Bootstrap validation provided stronger evidence supporting overall coefficient reliability. The majority of original coefficient estimates remained closely aligned with the corresponding bootstrap mean estimates, and confidence intervals largely reinforced the principal inferential conclusions obtained from the full model. However, confidence intervals associated with several secondary runtime implementation and compiler variables crossed zero, indicating comparatively weaker coefficient stability for these predictors. Page fault rate produced a particularly complex result, becoming statistically non-significant under HC3-adjusted estimation while retaining a bootstrap confidence interval excluding zero, suggesting comparatively weaker inferential stability relative to the other microarchitectural predictors.

Taken collectively, these findings indicate that JRE JVM power use under tiered execution is characterised by a distinct explanatory structure relative to interpretive execution. Under tiered execution, runtime implementation characteristics and garbage collector variant accounted for a larger share of explained variation, while microarchitectural performance event measures contributed comparatively less additional explanatory power. More broadly, comparison across the pooled, interpretive, and tiered execution models provides further evidence that increasingly granular modelling of execution context systematically improves explanatory performance while simultaneously revealing execution-mode-specific structural relationships that remain partially obscured within aggregated pooled specifications.

Research Question Findings

SRQ3.1b: Hierarchical regression modelling demonstrated that JRE JVM power use under tiered execution could be accounted for with substantial explanatory performance ($R^2 = .650$). Compared with interpretive execution, runtime implementation characteristics and garbage collector variant contributed a larger proportion of explained variance, while workload category remained the dominant predictor group. Microarchitectural performance event measures provided a comparatively modest additional contribution following inclusion of the higher-level predictor blocks.

The tiered execution model exhibited a distinct explanatory structure relative to the corresponding interpretive execution model. Elevated power use associated with OpenJDK JRE version 13 remained consistently positive throughout all stages of hierarchical model development, while garbage collector variant retained stronger statistical associations than observed under interpretive execution. Although sensitivity analysis indicated that several secondary predictor coefficients were influenced by a subset of influential observations, bootstrap validation confirmed that the principal runtime, workload, and microarchitectural relationships remained broadly stable. Collectively, these findings provide further evidence that execution strategy moderates the statistical relationships governing observed JRE JVM power use.

7.5 Power Use Cross-Model Comparative Analysis

The preceding regression analyses examined the pooled, interpretive execution, and tiered execution power use models independently. While these individual analyses

establish the explanatory structure associated with each execution context, they do not directly quantify how these structures differ across model specifications. To facilitate this comparison, the following section examines the three final regression models from three complementary perspectives. First, comparative variance decomposition evaluates the relative explanatory contribution associated with each hierarchical predictor block across the three models. Second, observed-versus-fitted model correspondence provides a direct visual assessment of the extent to which the fitted values reproduce the structure of the observed power use measurements within each model specification. Third, comparative coefficient restructuring examines changes in individual regression coefficients to assess the extent to which execution mode alters the statistical relationships governing observed JRE/JVM power use. Together, these complementary analyses provide a more comprehensive assessment of differences in overall explanatory performance, model correspondence, and individual predictor relationships across the pooled, interpretive, and tiered execution specifications.

7.5.1 Comparative Variance Decomposition Analysis

To facilitate direct comparison across the execution-mode-specific regression models, Table 7.15 summarises the incremental variance contributions associated with each hierarchical predictor block across the three power use models.

Table 7.15: Comparative variance contributions across hierarchical power use regression models

Model	**Runtime**	**Exec./GC**	**Workload**	**Microarch.**	**Adj.** R^2
Pooled	0.113	0.012	0.324	0.039	0.485
Interpretive	0.078	0.000	0.441	0.100	0.616
Tiered	0.189	0.025	0.407	0.030	0.647

These comparative results reveal several important patterns regarding the relative contribution of hierarchical predictor blocks across the three power use models.

First, workload category consistently represented the dominant source of explained variance across all execution contexts. In each model, the introduction of workload category produced the largest incremental improvement in model fit, increasing explained variance by 32.4% within the pooled model, 44.1% within the interpretive execution model, and 40.7% within the tiered execution model. This finding indicates that workload category constituted the single strongest determinant of observed JRE JVM power use irrespective of execution regime.

Second, notable differences emerged in the contribution associated with runtime implementation characteristics. Within the tiered execution model, runtime implementation factors accounted for 18.9% of observed variance prior to inclusion of subsequent predictor blocks, substantially exceeding the corresponding contribution observed within the interpretive execution model (7.8%) and pooled model (11.3%). This pattern suggests that power use under tiered execution was more strongly conditioned by differences in underlying runtime implementation characteristics than under interpretive execution.

Third, the contribution associated with microarchitectural performance event measures differed substantially across execution modes. Within the interpretive execution model, inclusion of microarchitectural variables increased explained variance by an additional 10.0%, compared with only 3.9% within the pooled model and 3.0% within the tiered execution model. This finding indicates that power use observed under interpretive execution exhibited substantially greater sensitivity to emergent low-level system state characteristics captured through measured microarchitectural performance events.

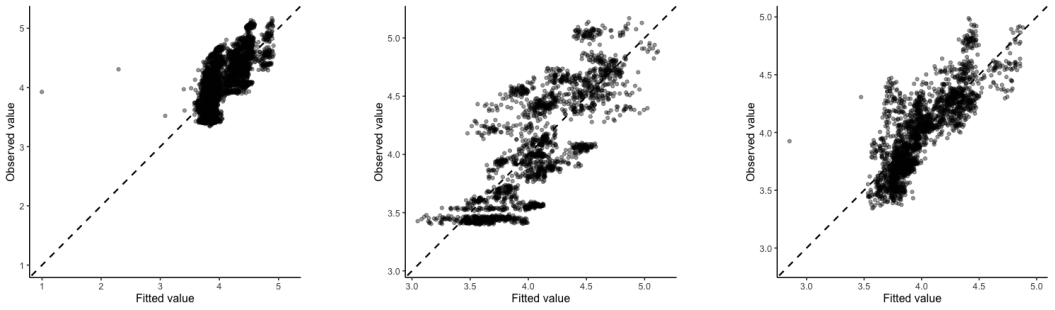
Finally, the comparatively weaker explanatory performance observed within the pooled execution model relative to both execution-mode-specific models suggests that aggregation across heterogeneous execution regimes obscured underlying execution-mode-dependent relationships. This pattern indicates that modelling JRE JVM power use separately according to execution mode permitted substantially improved statistical account of observed variation compared with the

corresponding pooled model specification.

Taken collectively, these comparative findings demonstrate that although workload category consistently represented the dominant determinant of JRE JVM power use, the relative importance of runtime implementation characteristics and emergent microarchitectural conditions differed substantially according to execution mode.

7.5.2 Observed-versus-Fitted Model Correspondence

The comparative variance decomposition presented above demonstrates that the execution-mode-specific power use models account for a greater proportion of observed variation than the corresponding pooled specification. To provide a direct visual assessment of this difference, Figure 7.5 compares the observed mean power use measurements with the values fitted by the final pooled, interpretive, and tiered execution regression models. For each observation, the horizontal axis represents the power use value fitted by the corresponding regression model, while the vertical axis represents the power use value measured experimentally. The dashed identity line, $y = x$, represents perfect correspondence between the observed and fitted values.



(a) Pooled **(b) Interpretive** **(c) Tiered**

Figure 7.5: Observed-versus-fitted correspondence for the final power use regression models: (a) pooled observations, (b) interpretive execution, and (c) tiered execution. The dashed identity line represents perfect correspondence between observed and fitted mean power use.

The pooled model, Figure 7.5(a), exhibits the greatest dispersion around the

identity line. Although the fitted values reproduce a substantial component of the overall power use structure, the correspondence is imperfect, with several observations exhibiting measured power use substantially greater than the corresponding fitted values. This visual pattern is consistent with the comparatively lower explanatory performance of the pooled specification ($R^2 = .488$, adjusted $R^2 = .485$) and with the residual and influential-observation diagnostics presented previously. Collectively, these results indicate that a single aggregated model does not represent all execution contexts equally well.

A clearer observed-versus-fitted relationship emerges when the dataset is decomposed according to JVM execution mode. For interpretive execution, Figure 7.5(b), the observations follow the identity line substantially more closely than within the pooled specification, consistent with the increase in explained variance to $R^2 = .620$ (adjusted $R^2 = .616$). Although structured clusters remain visible within the observed values, these are consistent with the heterogeneous combinations of runtime configuration and workload category represented within the analytical dataset. The overall diagonal correspondence nevertheless demonstrates that the interpretive execution model more closely represents the observed power use structure within this execution context.

The tiered execution model, Figure 7.5(c), similarly exhibits a pronounced diagonal relationship between fitted and observed power use, with the observations displaying comparatively close correspondence around the identity line. This is consistent with the tiered specification providing the strongest overall explanatory performance among the three power use models ($R^2 = .650$, adjusted $R^2 = .647$). The visual comparison therefore reinforces the numerical model-fit results: restricting the analysis to execution-mode-specific subsets produces substantially improved correspondence between fitted and observed power use relative to the aggregated pooled specification.

Importantly, these plots should be interpreted as visualisations of *in-sample model correspondence* rather than measures of out-of-sample predictive accuracy.

Table 7.16: Comparative coefficient restructuring across final power use regression models

Predictor	**Pooled**	**INT**	**Tiered**	**INT/Pooled**	**T./Pooled**	**INT/T.**
JRE 13	0.348	0.271	0.401	0.78	1.15	0.68
Concurrency	0.506	0.432	0.625	0.85	1.24	0.69
Web	0.570	1.243	0.412	2.18	0.72	3.02
BMR	-0.002	-0.018	-0.023	9.00	11.50	0.78
L1DLMR	-0.019	-0.006	0.084	0.32	4.42[a]	14.00[a]
L2DLMR	-0.008	-0.083	-0.004	10.38	0.50	20.75
PFMR	-1.194	-3.524	-0.443	2.95	0.37	7.95

[a] Coefficient sign reversal observed, absolute magnitudes used.
Note: INT/Pooled = Interpretive relative to Pooled; T./Pooled = Tiered relative to Pooled; INT/T.= Interpretive relative to Tiered;

The increasing alignment of observed and fitted values from the pooled to the execution-mode-specific models provides a visual complement to the variance decomposition and residual diagnostic analyses, supporting the conclusion that separating interpretive and tiered execution reduces the structural heterogeneity represented within the pooled power use model. Having established these differences in overall explanatory performance and observed-versus-fitted correspondence, the following analysis examines whether the underlying individual coefficient relationships are similarly restructured across the three model specifications.

7.5.3 Comparative Coefficient Restructuring Analysis

While the preceding comparative analysis examined differences in overall variance contribution across the pooled and execution-mode-specific power use regression models, coefficient-level comparisons provide additional insight into the extent to which execution strategy alters the underlying structural relationships governing observed JRE JVM power use. Table 7.16 summarises selected coefficients from the final pooled, interpretive execution, and tiered execution models, together with relative coefficient magnitude ratios quantifying the extent to which predictor relationships are amplified or attenuated across execution regimes.

Several notable structural differences emerge when comparing coefficient mag-

nitudes across the three final models. The OpenJDK JRE version 13 anomaly remained relatively stable across all three specifications, with coefficient magnitudes differing only modestly between execution contexts. Similarly, the `concurrency` workload coefficient exhibited comparatively limited variation, although tiered execution produced somewhat larger coefficient magnitudes relative to both pooled and interpretive execution. These findings suggest that certain high-level runtime and workload relationships remain comparatively stable regardless of execution strategy.

In contrast, several lower-level microarchitectural predictors exhibited substantial coefficient restructuring following execution-mode decomposition. Page fault rate demonstrated particularly pronounced variation across models. Relative to the pooled model, the magnitude of the page fault rate coefficient increased by approximately 2.95 times under interpretive execution, whereas under tiered execution the coefficient magnitude declined substantially, representing only approximately 37% of the pooled model magnitude. Direct comparison between execution modes reveals that the page fault rate coefficient under interpretive execution was approximately 7.95 times larger than under tiered execution.

A similarly pronounced restructuring effect was observed for L2 data cache load miss rate. Relative to the pooled model, interpretive execution increased the coefficient magnitude by approximately 10.38 times, whereas under tiered execution the coefficient magnitude declined to approximately half that observed within the pooled specification. Direct comparison between execution regimes indicates that the estimated L2 data cache load miss rate coefficient under interpretive execution was approximately 20.75 times larger than under tiered execution. This represents one of the largest coefficient magnitude differences observed across any predictor within the comparative analysis.

Workload-specific effects also exhibited substantial execution-dependent restructuring. The coefficient associated with the `web` workload category more than doubled under interpretive execution relative to the pooled model, while simulta-

neously declining under tiered execution. Direct comparison indicates that the `web` workload coefficient under interpretive execution was approximately 3.02 times larger than the corresponding coefficient observed under tiered execution. This suggests that execution strategy conditions the extent to which workload-specific computational demands contribute independently to observed power use variation.

An especially interesting result emerged for L1 data cache load miss rate. While both the pooled and interpretive execution models retained negative coefficient structure, the tiered execution model exhibited a coefficient sign reversal, producing a positive coefficient estimate. Although direct ratio comparisons must therefore be interpreted cautiously, this sign reversal provides additional evidence that execution strategy may fundamentally alter the observed statistical relationship between lower-level hardware execution characteristics and measured power use.

Overall, these findings provide strong evidence that execution strategy substantially conditions the structural relationships governing observed JRE JVM power use. Whereas several high-level runtime configuration effects remained comparatively stable across models, lower-level microarchitectural predictors exhibited substantial execution-dependent variation in coefficient magnitude, with interpretive execution consistently amplifying the observed influence of several hardware-level predictors relative to both pooled and tiered execution. More broadly, these findings suggest that execution strategy affects not only overall explanatory performance, but also the extent to which lower-level software–hardware interactions remain statistically observable within empirical power use models.

7.6 Energy Consumption Model

This section principally addresses sub-research question **SRQ3.2**, which examines the extent to which statistical models incorporating runtime configuration factors, workload category, and microarchitectural performance event measures account for observed variation in JRE/JVM energy consumption.

SRQ3.2: ***To what extent can variation in JRE/JVM energy consumption be accounted for through regression modelling incorporating JRE source version, GNU GCC build toolchain compiler version, JVM execution mode, garbage collector variant, workload category, and microarchitectural performance event measures?***

To address this question, we develop a staged hierarchical regression framework in which predictor variables are introduced incrementally to evaluate the collective contribution associated with successive predictor groups. Model reliability is subsequently assessed through regression diagnostic procedures and bootstrap resampling analyses.

Unlike the preceding power use analysis, preliminary modelling of raw energy consumption measurements indicated that the underlying data generating process associated with energy consumption differs fundamentally from that observed for instantaneous power use. Initial linear model estimation using untransformed energy consumption values ($energyWh$) produced substantially weaker explanatory performance and diagnostic evidence suggesting violation of standard linear modelling assumptions.

Further exploratory analysis demonstrated an extremely strong positive correlation between energy consumption and execution duration ($r \approx 0.99$), indicating that variation in observed energy consumption was overwhelmingly dominated by differences in execution time rather than instantaneous power demand. This result is theoretically consistent with the underlying physical relationship governing electrical energy consumption:

$$E = P \times t \tag{7.5}$$

where energy consumption is not itself an instantaneous system state, but rather a cumulative quantity generated multiplicatively through the interaction between power demand and execution duration.

This distinction has important implications for statistical modelling. Whereas power use represents an instantaneous additive system state well suited to conventional linear modelling approaches, energy consumption represents a cumulative duration-dependent process in which proportional rather than absolute variation becomes the dominant source of observed variance.

To better reflect this underlying multiplicative process, the dependent variable was transformed using the natural logarithm of observed energy consumption. The resulting log-linear specification substantially improved model explanatory performance while producing a representation more consistent with the physical process governing cumulative energy generation.

The analysis presented in this section therefore models *log-transformed energy consumption* rather than raw energy consumption values.

7.6.1 Theory-Driven Model Specification

The regression models developed in this chapter were specified *a priori* based on the theoretical model (§4.2) and conceptual framework (§4.3) introduced in Chapter 4. Unlike exploratory modelling approaches in which predictor variables are selected through data-driven procedures, the models estimated in this study were constructed directly from the theoretically motivated relationships previously identified as potential determinants of JRE JVM power use and energy consumption.

Consistent with the theoretical framework, the models incorporate both experimentally manipulated runtime configuration variables and emergent execution characteristics observed during runtime. The runtime configuration variables represent directly controllable system-level factors and comprise OpenJDK JRE source version (JSV), GNU GCC build toolchain compiler version ($BTCV$), JVM execution mode (JEM), garbage collector variant (GCV), and workload category (WC). In addition, the models incorporate microarchitectural performance event measures representing emergent lower-level execution characteristics, specifically

branch misprediction rate (*BMR*), L1 data cache load miss rate (*L1DLMR*), L2 data cache load miss rate (*L2DLMR*), and page fault rate (*PFMR*).

The objective of the regression analysis is therefore not to identify statistically convenient predictors, but rather to evaluate the extent to which the theoretically specified runtime configuration factors and emergent microarchitectural performance event measures account for variation in observed JRE JVM energy consumption.

To evaluate variation in observed energy consumption, we specify the following log-linear model:

$$\begin{aligned} \ln(E_i) = \beta_0 &+ \beta_1(JSV_i) + \beta_2(BTCV_i) \\ &+ \beta_3(JEM_i) + \beta_4(GCV_i) + \beta_5(WC_i) \\ &+ \beta_6(BMR_i) + \beta_7(L1DLMR_i) \\ &+ \beta_8(L2DLMR_i) + \beta_9(PFMR_i) + \varepsilon_i \end{aligned} \tag{7.6}$$

Having established the theoretical specification of the regression model, we next examine the hierarchical model development process, assessing the incremental explanatory contribution associated with progressively introducing the theoretically specified predictor groups.

7.6.2 Hierarchical Regression Results

Predictors were entered hierarchically according to the same theoretically motivated layered systems architecture introduced previously for the pooled power use analysis. This hierarchical structure reflects progressively deeper levels of system influence on observed JRE JVM energy consumption, beginning with foundational runtime implementation characteristics, followed by execution strategy variables, workload category, and finally emergent microarchitectural performance event measures observed during execution.

Table 7.17 summarises the incremental explanatory contribution associated

with each successive predictor block, while Table 7.18 presents the corresponding hierarchical coefficient estimates. Consistent with the preceding power use analysis, heteroskedasticity-consistent HC3 robust standard errors are reported for coefficient inference, while model fit statistics are derived from the original OLS estimation.

Table 7.17: Hierarchical regression model development for JRE JVM pooled energy consumption. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block.

Model Block	R^2	**Adj.** R^2	ΔR^2	ΔF	p
Block 1: Runtime Implementation	0.002	0.000	0.002	–	–
Block 2: Execution Strategy	0.624	0.623	0.622	4536.500	$< 0.001^{***}$
Block 3: Workload Category	0.861	0.861	0.238	1734.190	$< 0.001^{***}$
Block 4: Microarchitecture	0.874	0.874	0.013	116.850	$< 0.001^{***}$
Final Model	0.874	0.874	–	–	$< 0.001^{***}$

The first hierarchical model incorporating only OpenJDK JRE source version and GNU GCC build toolchain compiler version accounted for negligible variation in observed energy consumption ($R^2 = .002$), indicating that foundational runtime implementation characteristics alone provided effectively no explanatory power when modelling log-transformed energy consumption.

Table 7.18 reports the hierarchical coefficient development for the pooled execution energy consumption model using HC3 heteroskedasticity-consistent robust standard errors.

The Block 1 baseline model, incorporating only runtime implementation characteristics, revealed limited evidence of direct runtime implementation effects on observed energy consumption. OpenJDK JRE version 13 emerged as a statistically significant positive predictor relative to the JDK9 reference configuration ($\beta = 0.213$, $p < 0.01$), while JDK10 and JDK11 approached marginal statistical significance ($p < 0.10$). In contrast, GNU GCC build toolchain compiler source version exhibited no statistically significant effects, indicating that implementation-level runtime characteristics alone contributed relatively little explanatory power

Table 7.18: Hierarchical regression coefficients for pooled JRE JVM energy consumption (ln(*energyWh*)). Standard errors correspond to HC3 heteroskedasticity-consistent robust variance estimation.

	Block 1			**Block 2**			**Block 3**			**Block 4**		
Predictor	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**
Intercept	-1.834	0.069	***	-2.976	0.054	***	-2.368	0.035	***	-2.776	0.076	***
JRE 10	0.152	0.081	.	0.009	0.053		0.009	0.032		0.010	0.030	
JRE 11	0.141	0.081	.	-0.003	0.053		-0.003	0.032		-0.055	0.030	.
JRE 12	0.080	0.081		-0.064	0.053		0.005	0.032		-0.051	0.031	.
JRE 13	0.213	0.082	**	0.070	0.054		0.139	0.033	***	0.259	0.033	***
JRE 14	0.060	0.082		-0.083	0.054		-0.015	0.033		-0.061	0.032	.
GCC 6	-0.019	0.068		-0.019	0.042		-0.019	0.025		-0.018	0.024	
GCC 7	-0.022	0.068		-0.022	0.042		-0.022	0.025		-0.022	0.024	
GCC 8	-0.025	0.068		-0.025	0.042		-0.025	0.025		0.005	0.025	
INT	–	–		2.578	0.030	***	2.578	0.018	***	2.328	0.055	***
CPGC	–	–		0.002	0.086		0.002	0.051		0.118	0.052	*
G1GC	–	–		0.010	0.043		0.010	0.026		0.054	0.028	.
PGC	–	–		-0.014	0.043		-0.014	0.026		0.028	0.028	
SGC	–	–		-0.012	0.042		-0.012	0.026		0.005	0.026	
Concurrency	–	–		–	–		-0.044	0.026	.	-0.010	0.026	
Database	–	–		–	–		0.970	0.020	***	0.975	0.021	***
Functional	–	–		–	–		-1.582	0.028	***	-1.584	0.048	***
Scala	–	–		–	–		-1.352	0.028	***	-1.353	0.040	***
Web	–	–		–	–		-0.867	0.029	***	-1.176	0.036	***
BMR	–	–		–	–		–	–		0.038	0.005	***
L1DLMR	–	–		–	–		–	–		0.068	0.015	***
L2DLMR	–	–		–	–		–	–		0.008	0.002	***
PFMR	–	–		–	–		–	–		-1.043	0.440	*

Significance codes: *** $p < .001$; ** $p < .01$; * $p < .05$; . $p < .10$.

to observed variation in pooled energy consumption.

The introduction of execution strategy characteristics within Block 2, comprising execution mode and garbage collector variant, produced substantial coefficient restructuring. Most notably, previously observed JRE source version effects largely disappeared, with the JDK13 coefficient reducing substantially from $\beta = 0.213$ to $\beta = 0.070$ and losing statistical significance. By contrast, interpretive execution emerged as the dominant positive predictor within the model ($\beta = 2.578$, $p < 0.001$), indicating that execution mode represented the principal determinant of observed energy consumption variation at this stage of model development. Garbage collector coefficients remained statistically non-significant, suggesting negligible independent garbage collector effects prior to accounting for workload and lower-level execution-state characteristics.

Substantial coefficient restructuring emerged following introduction of workload category within Block 3. Several workload categories exhibited strong statistically significant effects relative to the `apache` reference category. In particular, the `database` workload was associated with substantially higher energy consumption ($\beta = 0.970$, $p < 0.001$), while `functional` ($\beta = -1.582$, $p < 0.001$), `scala` ($\beta = -1.352$, $p < 0.001$), and `web` ($\beta = -0.867$, $p < 0.001$) workloads were associated with substantially lower energy consumption. At this stage, OpenJDK JRE version 13 re-emerged as statistically significant ($\beta = 0.139$, $p < 0.001$), indicating that workload-specific execution characteristics conditioned the relationship between runtime implementation and observed energy consumption. Interpretive execution remained the dominant predictor, with its coefficient remaining unchanged in magnitude.

The introduction of microarchitectural performance event measures within Block 4 produced further notable coefficient restructuring. OpenJDK JRE version 13 increased substantially in magnitude from $\beta = 0.139$ to $\beta = 0.259$ ($p < 0.001$), while JDK11, JDK12, and JDK14 each approached marginal statistical significance ($p < 0.10$). By contrast, GNU GCC compiler source version remained consistently

non-significant across all model stages, indicating persistent absence of measurable compiler-level effects throughout hierarchical model development. The coefficient associated with interpretive execution also reduced from $\beta = 2.578$ to $\beta = 2.328$, suggesting that part of the substantial execution mode effect observed previously was shared with lower-level microarchitectural execution-state variation captured through the final model block.

Garbage collector coefficients also exhibited notable restructuring following introduction of microarchitectural performance event measures. Prior to Block 4, all garbage collector coefficients remained close to zero and statistically non-significant. Following introduction of microarchitectural predictors, the CPGC coefficient increased substantially and became statistically significant relative to the CMS reference configuration ($\beta = 0.118$, $p < 0.05$), while G1GC approached marginal statistical significance ($\beta = 0.054$, $p < 0.10$). This coefficient restructuring suggests that some garbage collector effects on observed energy consumption remained partially obscured prior to accounting for shared variance associated with lower-level execution-state characteristics emerging dynamically during program execution.

Workload category level coefficients remained among the strongest predictors within the final model, although several coefficients shifted following introduction of microarchitectural variables. The `database`, `functional`, and `scala` workload coefficients remained comparatively stable, while the coefficient associated with the `web` workload increased substantially in magnitude from $\beta = -0.867$ to $\beta = -1.176$ ($p < 0.001$). By contrast, the previously marginal `concurrency` workload effect observed in Block 3 disappeared following inclusion of microarchitectural predictors, suggesting that this variation was partially captured by lower-level execution-state characteristics.

Among microarchitectural performance event measures, branch misprediction rate emerged as a statistically significant positive predictor of energy consumption ($\beta = 0.038$, $p < 0.001$). Both L1 data cache load miss rate ($\beta = 0.068$, $p < 0.001$)

and L2 data cache load miss rate ($\beta = 0.008$, $p < 0.001$) were also positively associated with energy consumption. By contrast, page fault rate exhibited a statistically significant negative association with energy consumption ($\beta = -1.043$, $p < 0.05$), indicating that increasing page fault activity was associated with reduced observed energy consumption after controlling for all higher-level software configuration characteristics.

Overall, the hierarchical coefficient development indicates that pooled JRE JVM energy consumption was principally conditioned by execution mode and workload category, with interpretive execution representing the dominant system-level determinant of cumulative energy consumption. The repeated coefficient restructuring observed following introduction of microarchitectural performance event measures further suggests that the effects associated with higher-level software runtime configuration may remain partially obscured when lower-level emergent execution-state variation is not explicitly modelled.

Estimated Final Model

The final hierarchical regression model can be expressed as:

$$\begin{aligned}
\ln(\widehat{EnergyWh})_i = & -2.776 + 0.010\,JRE10_i - 0.055\,JRE11_i - 0.051\,JRE12_i \\
& + 0.259\,JRE13_i - 0.061\,JRE14_i - 0.018\,GCC6_i \\
& - 0.022\,GCC7_i + 0.005\,GCC8_i + 2.328\,INT_i \\
& + 0.118\,CPGC_i + 0.054\,G1GC_i + 0.028\,PGC_i \\
& + 0.005\,SGC_i - 0.010\,Concurrency_i + 0.975\,Database_i \\
& - 1.584\,Functional_i - 1.353\,Scala_i - 1.176\,Web_i \\
& + 0.038\,BMR_i + 0.068\,L1DLMR_i + 0.008\,L2DLMR_i \\
& - 1.043\,PFMR_i + \epsilon_i
\end{aligned} \tag{7.7}$$

The estimated model coefficients are interpreted relative to the omitted refer-

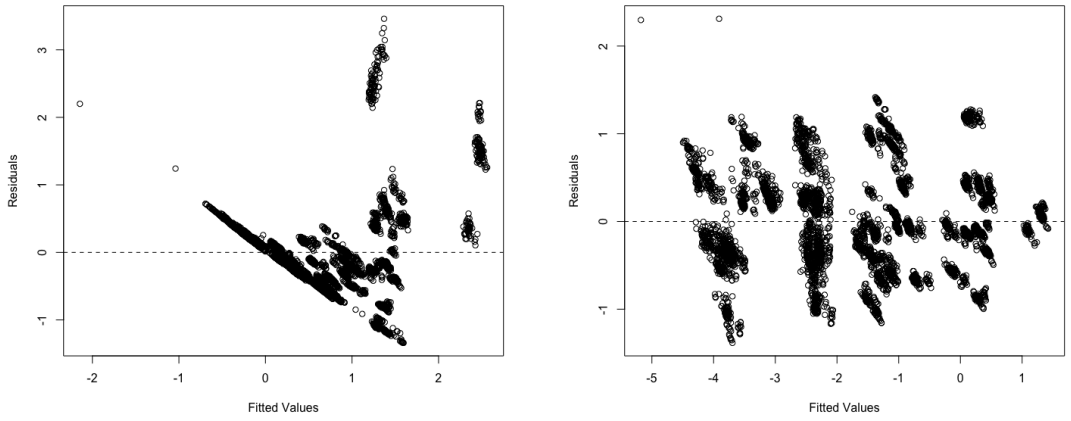
ence categories corresponding to OpenJDK JRE 9, GCC 5, tiered execution, the default CGC garbage collector baseline, and the `apache` workload category.

7.6.3 Regression Diagnostics

Assessment of Linearity

An important assumption underlying ordinary least squares regression is that the relationship between the dependent variable and predictor variables is appropriately represented through a linear functional form. Prior to estimation of the final pooled energy consumption model, preliminary regression analysis was initially performed using untransformed energy consumption ($energyWh$) as the dependent variable. However, diagnostic assessment indicated substantial deviation from linearity, suggesting that a conventional additive linear specification provided a poor representation of the underlying data generating process.

Figure 7.6 presents residual-versus-fitted plots comparing the preliminary untransformed energy consumption model against the final log-transformed model specification.



(a) Raw energy model (b) Log-transformed energy model

Figure 7.6: Residual-versus-fitted plots comparing linearity diagnostics for the preliminary untransformed energy consumption model and the final log-transformed energy consumption model.

Visual inspection of the untransformed energy consumption model revealed substantial violation of the linearity assumption. The residual distribution exhibited strong systematic structure, pronounced curvature, substantial clustering, and clear heterogeneity in residual spread across fitted values. Rather than displaying the approximately random residual distribution expected under ordinary least squares estimation, the residual structure indicates that the additive linear specification failed to adequately capture the underlying functional relationship governing observed variation in cumulative energy consumption.

To address this issue, the dependent variable was transformed using the natural logarithm of observed energy consumption, producing the final model specification based on $\ln(energyWh)$. Following transformation, the residual-versus-fitted distribution exhibited substantial improvement. Residuals were distributed more symmetrically around zero, systematic curvature was substantially reduced, and overall residual dispersion became considerably more homogeneous across the fitted value range.

The improvement associated with the transformed specification is further reflected in the substantial increase in explanatory performance. The preliminary model using untransformed energy consumption explained approximately 56.4% of observed variation ($R^2 = .564$), whereas the final log-transformed specification explained approximately 87.4% of observed variation ($R^2 = .874$), representing a substantial improvement in overall model fit.

To formally assess model specification, Ramsey's RESET test was applied to both models. The preliminary untransformed energy consumption model exhibited overwhelming evidence of functional form misspecification ($RESET = 1798$, $p < .001$), indicating that the additive linear specification failed to adequately represent the underlying statistical structure of the observed data. By contrast, the log-transformed model produced a non-significant RESET result ($RESET = 2.034$, $p = .131$), indicating no evidence of significant specification error following transformation.

The improved performance associated with logarithmic transformation is consistent with the underlying physical process governing cumulative energy consumption. Unlike instantaneous power use, which represents a momentary additive system state measured in watts, cumulative energy consumption is fundamentally governed by the multiplicative relationship between power demand and execution duration.

Because execution duration varied substantially across experimental conditions, particularly between interpretive and tiered execution modes, cumulative energy consumption does not naturally conform to an additive linear process. Rather, the underlying generative structure is inherently multiplicative, making logarithmic transformation theoretically and statistically appropriate.

Collectively, these results indicate that the final log-transformed specification provides a substantially better representation of the underlying statistical structure governing JRE JVM energy consumption and satisfies the linearity assumption considerably more appropriately than the original untransformed energy consumption model.

Assessment of Residual Distribution

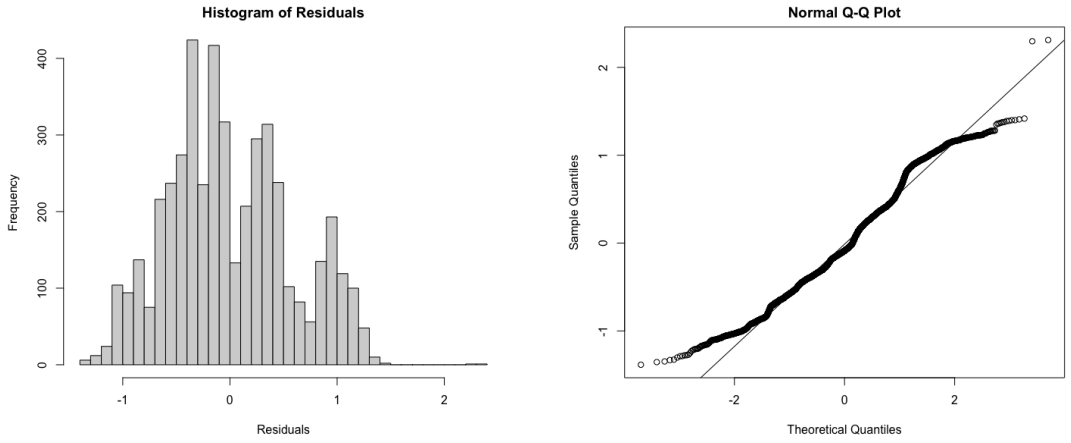
Ordinary least squares regression additionally assumes that model residuals are approximately normally distributed. Although coefficient estimation itself remains unbiased in the presence of moderate departures from normality, substantial deviation may affect statistical inference, particularly when conventional standard errors are used. Because coefficient inference throughout this study relies on HC3 heteroskedasticity-consistent robust standard errors, the residual distribution assessment is principally used to evaluate overall model adequacy rather than strict adherence to classical ordinary least squares assumptions.

Figure 7.7 presents the residual histogram and normal Q-Q plot for the final pooled energy consumption regression model based on the log-transformed dependent variable $\ln(energyWh)$.

Visual inspection of the residual histogram indicates an approximately symmetric central residual distribution, although moderate clustering and some departure from perfect normality remain evident. The presence of residual clustering is not unexpected given the heterogeneous experimental structure of the dataset, which incorporates multiple interacting runtime configurations, workload categories, execution modes, and microarchitectural performance event measures.

The normal Q-Q plot demonstrates close agreement with the theoretical normal distribution across the majority of central quantiles, indicating that the bulk of the residual distribution approximates normality reasonably well. Some deviation from the theoretical reference line is visible in the upper tail region, indicating the presence of a limited number of comparatively large positive residuals. However, these departures remain largely concentrated within the distribution tails rather than affecting the overall residual structure.

Statistical assessment using the Shapiro-Wilk normality test indicated statistically significant deviation from perfect normality ($W = 0.981$, $p < .001$). However, given the large sample size ($n = 4608$), the Shapiro-Wilk test is known to become highly sensitive to even minor departures from normality.



(a) Residual histogram (b) Normal Q-Q plot

Figure 7.7: Residual distribution diagnostics for the final pooled JRE JVM energy consumption regression model using the log-transformed dependent variable $\ln(energyWh)$.

Examination of standardised residuals further supports this interpretation. Standardised residual values ranged from approximately -2.39 to 4.67, indicating a relatively well-contained residual distribution with only a limited number of observations occupying the upper tail extremes.

Collectively, these diagnostic results suggest that although residuals do not conform perfectly to a theoretical normal distribution, the overall residual structure remains sufficiently well-behaved for reliable model estimation. The residual distribution demonstrates approximate central normality, with observed departures largely confined to the distribution tails and therefore unlikely to materially affect coefficient inference, particularly given the use of HC3 heteroskedasticity-consistent robust standard errors throughout the present analysis.

Assessment of Influential Observations

A further diagnostic consideration in regression modelling concerns the presence of influential observations capable of exerting disproportionate influence over estimated coefficient values. To assess the extent to which individual observations affected the stability of the final pooled energy consumption regression model, Cook's distance was examined for all observations included in the final model specification. Following conventional practice, potentially influential observations were identified using the threshold criterion:

$$D_i > \frac{4}{n} \tag{7.8}$$

where n represents the total number of observations included in the regression model.

For the present dataset ($n = 4608$), the corresponding threshold value was approximately 0.00087. Application of this criterion identified 130 observations exceeding the conventional influence threshold, representing approximately 2.82% of the total dataset.

Examination of the distribution of influential observations revealed substantial

variation across runtime conditions. Unlike the preceding pooled power use model, where influential observations were predominantly concentrated within interpretive execution, the present energy consumption model exhibited the opposite pattern. Of the 130 influential observations identified, 90 observations (69.2%) originated from tiered execution, while only 40 observations (30.8%) originated from interpretive execution.

The distribution of influential observations across OpenJDK JRE versions was similarly non-uniform. JRE versions 9 and 13 contributed the largest number of influential observations, accounting for 38 and 41 influential observations respectively, while comparatively fewer influential observations were associated with JRE versions 10, 11, and 12.

Workload category revealed particularly strong concentration effects. The `concurrency` workload category contributed the largest number of influential observations (52 observations), followed by `scala` (34 observations) and `functional` workloads (33 observations). By contrast, only a single influential observation was identified within both the `database` and `web` workload categories.

Table 7.19: Cross-classification of influential observations identified through Cook's distance analysis, detailing the number of influential observations belonging to each workload category. Percentages are calculated relative to all influential observations ($n = 130$).

Workload Category	**HS**	**INT**	**Total**
Apache	2 (1.5%)	7 (5.4%)	9 (6.9%)
Concurrency	52 (40.0%)	0 (0.0%)	52 (40.0%)
Database	1 (0.8%)	0 (0.0%)	1 (0.8%)
Functional	0 (0.0%)	33 (25.4%)	33 (25.4%)
Scala	34 (26.2%)	0 (0.0%)	34 (26.2%)
Web	1 (0.8%)	0 (0.0%)	1 (0.8%)
Total	90 (69.2%)	40 (30.8%)	130 (100.0%)

Table 7.19 further reveals a highly structured distribution of influential observations across workload category and execution mode. Unlike the preceding pooled power use model, influential observations within the energy consumption model exhibited strong execution mode partitioning. The majority of influential

observations associated with tiered execution were concentrated almost exclusively within the `concurrency` and `scala` workload categories, accounting collectively for 86 observations (66.2% of all influential observations). In contrast, influential observations under interpretive execution were overwhelmingly concentrated within the `functional` workload category, which contributed 33 observations (25.4% of all influential observations). Only the `apache` workload category exhibited influential observations under both execution modes. This highly structured distribution suggests that influential observations within the energy consumption model do not arise randomly across the experimental design, but instead reflect workload-specific execution characteristics operating differently under interpretive and tiered execution mode.

Collectively, these results indicate that although a moderate number of influential observations are present within the dataset, these observations do not materially destabilise the underlying regression structure. The final pooled energy consumption model therefore demonstrates substantial robustness to influential observation removal, providing further evidence that cumulative energy consumption exhibits a comparatively stable and predictable statistical structure under the present experimental conditions.

Assessment of Homoskedasticity

Ordinary least squares regression additionally assumes homoskedasticity, whereby residual variance remains approximately constant across the range of fitted values. Violation of this assumption may affect the reliability of conventional standard error estimation and associated significance testing. Because coefficient inference throughout the present analysis relies on HC3 heteroskedasticity-consistent robust standard errors, the assessment of homoskedasticity is principally used to evaluate overall model adequacy rather than strict adherence to classical ordinary least squares assumptions.

Figure 7.8 presents the residual-versus-fitted plot for the final pooled en-

ergy consumption regression model using the log-transformed dependent variable $\ln(energyWh)$.

Visual inspection indicates substantial improvement relative to the preliminary untransformed energy consumption model presented previously in Figure 7.6. Following logarithmic transformation, residual dispersion appears considerably more stable across the fitted value range, with no obvious funnel-shaped variance pattern or systematic variance expansion typically associated with severe heteroskedasticity.

Although some residual clustering remains visible, this structure is likely attributable to the heterogeneous experimental design underlying the dataset, which incorporates multiple interacting runtime configurations, workload categories, execution modes, and emergent microarchitectural performance event measures. Consequently, some localised variance heterogeneity across fitted value regions is not unexpected.

Formal statistical assessment using the studentised Breusch-Pagan test nevertheless indicated statistically significant heteroskedasticity ($BP = 815.79$, $df = 22$, $p < .001$), suggesting residual variance is not perfectly constant across all observations.

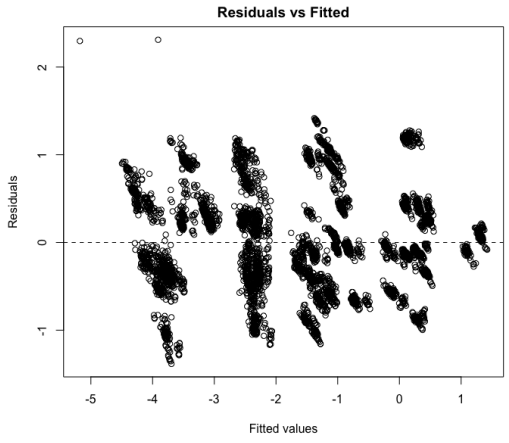


Figure 7.8: Residual-versus-fitted plot for the final pooled JRE JVM energy consumption regression model using the log-transformed dependent variable $\ln(energyWh)$.

Collectively, these results indicate that although formal homoskedasticity assumptions are not perfectly satisfied, the observed residual variance heterogeneity does not materially compromise model reliability. Furthermore, because all coefficient inference reported throughout this analysis is based on HC3 heteroskedasticity-consistent robust standard errors, the final pooled energy consumption model remains suitable for reliable coefficient estimation and inferential interpretation.

Assessment of Multicollinearity

Multicollinearity occurs when predictor variables exhibit substantial linear dependency, potentially inflating coefficient variance and reducing the interpretability of individual regression parameter estimates. To assess potential multicollinearity within the final pooled energy consumption regression model, generalised variance inflation factor diagnostics were examined for all predictors included in the final model specification.

Table 7.20 presents multicollinearity diagnostics for the final pooled model. Because several predictors comprise multi-level categorical variables represented through multiple dummy-coded regression terms, adjusted generalised variance inflation factor values ($GVIF^{1/(2\times Df)}$) are reported to permit direct comparison across predictors with differing degrees of freedom.

Inspection of the multicollinearity diagnostics indicates that all adjusted variance inflation factor values remain well below conventional threshold values typically associated with problematic multicollinearity. The largest adjusted values were observed for branch misprediction rate ($GVIF = 2.95$) and execution mode ($GVIF = 2.64$), followed by L1 data cache load miss rate ($GVIF = 1.86$) and L2 data cache load miss rate ($GVIF = 1.82$). However, even these values remain comfortably below commonly applied threshold values indicating substantive multicollinearity concerns.

The elevated values observed for execution mode and selected microarchitec-

tural performance event measures are theoretically expected, given that execution mode represents a major determinant of lower-level runtime execution behaviour and consequently influences emergent microarchitectural performance event measures observed during program execution.

Collectively, these results indicate that although moderate correlation exists among selected predictors, particularly between execution mode and lower-level microarchitectural performance measures, multicollinearity does not represent a substantive concern within the final pooled energy consumption regression model. The estimated regression coefficients may therefore be interpreted with confidence that predictor interdependence is unlikely to materially distort coefficient estimation or inferential interpretation.

Table 7.20: Multicollinearity diagnostics for the final pooled JRE JVM energy consumption regression model. Adjusted GVIF values are reported to permit comparison across predictors with differing degrees of freedom.

Predictor	**GVIF**	**Df**	**Adjusted GVIF**
JRE Source Version (JSV)	1.568	5	1.046
GCC Version (BTCV)	1.016	3	1.003
Execution Mode (JEM)	6.944	1	2.635
Garbage Collector (GCV)	1.333	4	1.037
Workload Category (WC)	3.195	5	1.123
Branch Misprediction Rate (BMR)	8.709	1	2.951
L1 Data Cache Load Miss Rate (L1DLMR)	3.475	1	1.864
L2 Data Cache Load Miss Rate (L2DLMR)	3.295	1	1.815
Page Fault Rate (PFMR)	1.591	1	1.261

Adjusted GVIF reported as $GVIF^{1/(2\times Df)}$.

Collectively, these results indicate that estimated regression coefficients may be interpreted with confidence that predictor interdependence is unlikely to materially distort coefficient estimation or inferential interpretation.

7.6.4 Sensitivity Analysis

To assess the robustness of the final hierarchical energy consumption model, a sensitivity analysis was performed by re-estimating the final regression model following removal of observations previously identified as potentially influential

through Cook's distance diagnostics. The results are presented in Table 7.21.

The objective of this procedure was to evaluate the extent to which the final coefficient estimates and overall model performance were disproportionately dependent upon a relatively small number of influential observations. Stability of the coefficient structure following observation removal provides evidence that the substantive relationships identified within the final model are not driven by a limited subset of extreme cases.

Re-estimation of the final model following removal of influential observations produced only limited changes in overall explanatory performance. The sensitivity model produced a marginal improvement in overall model fit ($R^2 = .886$) relative to the original full model ($R^2 = .874$), indicating that the explanatory structure identified within the original model remains stable following removal of influential cases.

Table 7.21: Final pooled energy consumption regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$. Dependent variable is $\ln(energyWh)$.

	Full Model			**Sensitivity Model**			**Dir.**	**Sig.**
Predictor	β	**SE**	p	β	**SE**	p		
Intercept	-2.776	0.076	$< .001$	-2.734	0.053	$< .001$	Y	Y
JRE 10	0.010	0.030	.732	0.011	0.029	.702	Y	Y
JRE 11	-0.055	0.030	.069	-0.047	0.030	.113	Y	C
JRE 12	-0.051	0.031	.098	-0.043	0.030	.153	Y	C
JRE 13	0.259	0.033	$< .001$	0.267	0.032	$< .001$	Y	Y
JRE 14	-0.061	0.032	.056	-0.027	0.030	.377	Y	C
GCC 6	-0.018	0.024	.450	-0.024	0.023	.299	Y	Y
GCC 7	-0.022	0.024	.362	-0.028	0.023	.233	Y	Y
GCC 8	0.005	0.025	.845	-0.013	0.023	.587	N	Y
INT execution	2.328	0.055	$< .001$	2.259	0.045	$< .001$	Y	Y
CPGC	0.118	0.052	.022	0.155	0.050	.002	Y	Y
G1GC	0.054	0.028	.058	0.090	0.024	$< .001$	Y	C
PGC	0.028	0.028	.327	0.078	0.024	.001	Y	C
SGC	0.005	0.026	.849	0.055	0.024	.021	Y	C
Concurrency	-0.010	0.026	.691	0.079	0.029	.006	N	C
Database	0.975	0.021	$< .001$	0.976	0.035	$< .001$	Y	Y
Functional	-1.584	0.048	$< .001$	-1.501	0.029	$< .001$	Y	Y
Scala	-1.353	0.040	$< .001$	-1.258	0.030	$< .001$	Y	Y
Web	-1.176	0.036	$< .001$	-1.214	0.038	$< .001$	Y	Y
BMR	0.038	0.005	$< .001$	0.050	0.005	$< .001$	Y	Y
L1DLMR	0.068	0.015	$< .001$	0.022	0.013	.103	Y	C
L2DLMR	0.008	0.002	$< .001$	0.013	0.002	$< .001$	Y	Y
PFMR	-1.043	0.440	.018	-1.954	0.164	$< .001$	Y	Y

Dir. = coefficient direction remained unchanged.
Sig. = significance status retained (Y) or changed (C).

The dominant positive association associated with interpretive execution remained largely unchanged ($\beta = 2.259$, $p < .001$) relative to the original model ($\beta = 2.328$, $p < .001$), indicating that execution mode continued to represent the principal determinant of cumulative energy consumption independent of influential observations.

Similarly, the positive coefficient associated with OpenJDK JRE version 13

remained highly stable ($\beta = 0.267$, $p < .001$), providing additional evidence that the elevated energy consumption behaviour previously observed for JRE version 13 does not arise from a small subset of anomalous observations.

At the workload level, the overall coefficient structure remained highly consistent with the original model, with the `database` workload category remaining strongly positively associated with energy consumption, while the `functional`, `scala`, and `web` workload categories remained strongly negatively associated relative to the reference category.

At the microarchitectural level, branch misprediction rate ($\beta = 0.050$, $p < .001$), L2 data cache load miss rate ($\beta = 0.013$, $p < .001$), and page fault rate ($\beta = -1.954$, $p < .001$) all retained statistically significant associations with energy consumption. Although L1 data cache load miss rate no longer remained statistically significant following removal of influential observations, the overall microarchitectural coefficient structure remained substantively stable.

Collectively, the sensitivity analysis indicates that the final tiered execution energy consumption model demonstrates strong robustness to influential observation removal. The principal substantive relationships identified within the original model remain largely unchanged, indicating that the overall explanatory structure observed within the final hierarchical regression model is not disproportionately driven by a limited number of influential cases.

Compared with the preceding pooled power use model, the pooled energy consumption model demonstrates substantially greater structural stability under sensitivity testing, providing further support for the earlier conclusion that cumulative energy consumption exhibits a more stable and predictable statistical structure than instantaneous power use.

7.6.5 Bootstrap Validation

To further assess the reliability of the final pooled energy consumption regression model, a stratified bootstrap resampling procedure was performed using repeated

Table 7.22: Stratified bootstrap coefficient stability analysis for the pooled energy consumption regression model.

Predictor	Orig. β	Boot. Mean	Boot. SE	CI_Lower	CI_Upper
(Intercept)	-2.776	-2.769	0.057	-2.872	-2.650
JRE 10	0.010	0.010	0.029	-0.048	0.066
JRE 11	-0.055	-0.055	0.030	-0.113	0.003
JRE 12	-0.051	-0.053	0.030	-0.109	0.006
JRE 13	0.259	0.257	0.030	0.197	0.310
JRE 14	-0.061	-0.062	0.030	-0.116	-0.005
GCC6	-0.018	-0.018	0.024	-0.065	0.031
GCC7	-0.022	-0.023	0.023	-0.067	0.021
GCC8	0.005	0.005	0.023	-0.043	0.050
INT	2.328	2.326	0.043	2.238	2.407
CPGC	0.118	0.119	0.048	0.024	0.210
G1GC	0.054	0.055	0.025	0.004	0.108
PGC	0.028	0.030	0.025	-0.017	0.077
SGC	0.005	0.005	0.024	-0.041	0.051
Concurrency	-0.010	-0.010	0.023	-0.057	0.034
Database	0.975	0.974	0.018	0.939	1.008
Functional	-1.584	-1.577	0.035	-1.639	-1.503
Scala	-1.353	-1.350	0.034	-1.417	-1.285
Web	-1.176	-1.176	0.026	-1.227	-1.125
BMR	0.038	0.038	0.005	0.029	0.047
L1DLMR	0.068	0.068	0.013	0.043	0.093
L2DLMR	0.008	0.008	0.002	0.004	0.011
PFMR	-1.043	-1.096	0.276	-1.840	-0.697

Note: Orig. β = Original Beta; Boot. Mean = Bootstrap Mean; Boot. SE = Bootstrap Standard Error;

random sampling with replacement while preserving the original experimental group structure. The objective of this procedure was to evaluate coefficient stability and assess the extent to which the final model estimates remained robust under repeated resampling of the observed dataset.

Table 7.22 presents the bootstrap coefficient estimates, including the original coefficient estimates obtained from the full model, mean bootstrap coefficient estimates, bootstrap standard errors, and percentile confidence intervals.

Overall, the bootstrap analysis demonstrates a high degree of coefficient stability across the final pooled energy consumption model. For nearly all predictors,

the bootstrap mean coefficient estimates remained extremely close to the original model estimates, indicating that the substantive relationships identified within the original regression model are highly stable under repeated resampling.

Among runtime configuration variables, the positive association previously observed for OpenJDK JRE version 13 remained highly stable, with the bootstrap mean estimate ($\beta = 0.257$) closely approximating the original model coefficient ($\beta = 0.259$). The associated confidence interval remained entirely positive, providing additional evidence that the elevated energy consumption associated with JRE version 13 represents a stable and reproducible effect rather than an artefact of sample-specific variation.

The strongest coefficient stability was observed for execution mode. Interpretive execution remained associated with an extremely stable positive coefficient, with the bootstrap mean estimate ($\beta = 2.326$) almost identical to the original model estimate ($\beta = 2.328$). The narrow bootstrap confidence interval further confirms that execution mode represents a highly reliable and dominant determinant of cumulative energy consumption.

Similarly, workload category variables demonstrated exceptionally strong coefficient stability. The `database` workload category retained a large positive coefficient ($\beta = 0.974$), while the `functional`, `scala`, and `web` workload categories all retained highly stable negative associations. In each case, bootstrap confidence intervals remained narrow and did not cross zero, indicating highly robust workload-dependent effects.

At the microarchitectural level, branch misprediction rate, L1 data cache load miss rate, and L2 data cache load miss rate all demonstrated strong coefficient stability, with bootstrap mean estimates closely matching the original model estimates and confidence intervals remaining consistently bounded away from zero. Although page fault rate exhibited comparatively greater variability, reflected in a wider bootstrap confidence interval, the interval remained entirely negative, indicating that the observed negative association remains statistically robust.

By contrast, several comparatively weaker predictors, including GNU GCC build toolchain compiler version, the parallel garbage collector, the serial garbage collector, and the `concurrency` workload category, exhibited bootstrap confidence intervals overlapping zero. This indicates comparatively lower coefficient stability for these predictors, although these variables do not substantially affect the overall explanatory structure of the final model.

Collectively, the bootstrap analysis provides strong evidence that the final pooled energy consumption model demonstrates a highly stable coefficient structure under repeated resampling. Compared with the preceding power use models, the present energy consumption model exhibits substantially greater parameter stability, providing further support for the earlier conclusion that cumulative energy consumption follows a more stable and predictable statistical structure than instantaneous power use.

7.6.6 Summary

This section addressed **SRQ3.2** by evaluating the extent to which hierarchical linear regression models incorporating runtime configuration factors, workload category, and microarchitectural performance event measures accounted for observed variation in JRE JVM energy consumption.

The analysis demonstrated that energy consumption required a different statistical treatment from power use. Preliminary modelling of untransformed energy consumption produced weaker explanatory performance and clear diagnostic evidence of functional form misspecification. By contrast, the log-transformed specification substantially improved model fit and diagnostic adequacy, supporting the theoretical argument that cumulative energy consumption behaves as a duration-dependent multiplicative process rather than an instantaneous additive system state.

The final hierarchical model accounted for a very large proportion of observed variation in log-transformed energy consumption ($R^2 = .874$, adjusted $R^2 = .874$).

Foundational runtime implementation variables alone accounted for negligible variation ($R^2 = .002$). The largest explanatory contribution emerged following inclusion of execution mode and garbage collector variant ($\Delta R^2 = .622$), indicating that execution strategy represented the dominant determinant of cumulative energy consumption. Workload category added further substantial explanatory power ($\Delta R^2 = .238$), while microarchitectural performance event measures provided a smaller but meaningful additional contribution ($\Delta R^2 = .013$).

Within the final model, interpretive execution was the dominant predictor of energy consumption ($\beta = 2.328$, $p < .001$), confirming that execution time duration effects strongly structure energy consumption. OpenJDK JRE version 13 also remained positively associated with elevated energy consumption ($\beta = 0.259$, $p < .001$), while the `database` workload category was associated with higher energy consumption and the `functional`, `scala`, and `web` workload categories were associated with lower energy consumption relative to `apache`. Branch misprediction rate, L1 data cache load miss rate, and L2 data cache load miss rate were positively associated with energy consumption, while page fault rate was negatively associated.

Regression diagnostics supported the adequacy of the log-transformed model specification. Linearity improved substantially relative to the untransformed model, and Ramsey's RESET test indicated no evidence of significant misspecification following transformation. Although residual normality and homoskedasticity assumptions were not perfectly satisfied, residual structure was sufficiently well behaved, and coefficient inference was protected through HC3 heteroskedasticity-consistent robust standard errors. Multicollinearity diagnostics indicated no substantive concern, with all adjusted GVIF values remaining below conventional thresholds.

Sensitivity analysis and stratified bootstrap validation further supported the robustness of the final model. Removal of influential observations did not materially alter the principal coefficient structure and modestly improved model fit ($R^2 =$

.886). Bootstrap estimates closely approximated the original coefficient estimates, with the strongest stability observed for execution mode, JRE version 13, major workload effects, and key microarchitectural predictors.

Collectively, the pooled energy consumption model provides strong evidence that the selected runtime configuration factors, workload category, and microarchitectural performance event measures account for a substantial proportion of observed variation in JRE JVM energy consumption. More importantly, the results demonstrate that energy consumption exhibits a more stable and predictable statistical structure than power use, supporting the broader methodological conclusion that power use and energy consumption should not be treated as statistically interchangeable metrics.

Research Question Findings

SRQ3.2: Hierarchical regression modelling demonstrated that runtime configuration factors, workload category, execution mode, and microarchitectural performance event measures collectively accounted for a substantial proportion of observed variation in JRE JVM energy consumption ($R^2 = .874$). Unlike power use, execution mode represented the dominant source of explained variance, reflecting the fundamental dependence of cumulative energy consumption upon execution duration. The results further demonstrated that cumulative energy consumption is more appropriately modelled using a log-linear specification than a conventional linear model. Logarithmic transformation substantially improved explanatory performance and model specification, supporting the theoretical interpretation of energy consumption as a cumulative multiplicative process rather than an instantaneous additive system state. Sensitivity analysis and bootstrap validation further demonstrated that the resulting regression model was highly stable and robust.

7.7 Energy Consumption Model: Interpretive Execution

To further examine whether relationships identified within the pooled energy consumption model differed across execution mode, the regression analysis was repeated separately for observations generated under interpretive execution only.

This section principally addresses sub-research question **SRQ3.2a**, which examines the extent to which variation in JRE/JVM interpretive execution energy consumption can be accounted for through regression modelling incorporating run-time configuration factors, workload category, and microarchitectural performance event measures:

> **SRQ3.2a:** ***To what extent can variation in JRE/JVM interpretive execution energy consumption be accounted for through regression modelling incorporating JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, workload category, and microarchitectural performance event measures?***

The interpretive execution model retains the same theoretically specified predictor set and hierarchical modelling structure introduced previously for the pooled energy consumption analysis. However, because all observations correspond exclusively to interpretive execution, the execution mode predictor (JEM) is omitted from the model specification. As with the preceding power use analysis, this execution-mode-specific decomposition permits assessment of the extent to which predictor relationships observed within the pooled model remain stable under interpretive execution only.

Because the energy consumption analysis employs the log-transformed energy consumption outcome variable introduced previously, this execution-mode-specific decomposition additionally permits assessment of whether the improved model specification observed for the pooled log-transformed energy consumption model

is maintained when analysis is restricted to the more homogeneous interpretive execution subset.

7.7.1 Hierarchical Regression Results

To further examine whether relationships identified within the pooled energy consumption model differed across execution mode, the regression analysis was repeated separately for observations generated under interpretive execution only.

The interpretive execution energy consumption model retains the same theoretically specified predictor set and hierarchical modelling structure introduced previously for the pooled energy consumption analysis. However, because all observations correspond exclusively to interpretive execution, the execution mode predictor (*JEM*) is omitted from the model specification. This execution-mode-specific decomposition permits assessment of the extent to which predictor relationships observed within the pooled analysis remain stable when restricting analysis to a single execution subset.

Predictors were entered hierarchically according to the same theoretically motivated ordering introduced previously, reflecting distinct categories of system-level influence on observed JRE JVM energy consumption. Predictor blocks progress from directly specified runtime implementation characteristics, to execution configuration factors, followed by workload-induced computational demand, and finally emergent microarchitectural performance event measures arising dynamically during program execution.

This hierarchical structure therefore permits assessment of the extent to which progressively deeper system characteristics account for variation in observed JRE JVM energy consumption under interpretive execution conditions.

Table 7.23 summarises the hierarchical regression model development for interpretive execution energy consumption. In contrast to the corresponding pooled energy consumption model, runtime implementation characteristics contributed negligibly to overall model performance. The baseline model incorporating only

JRE source version and GNU GCC build toolchain compiler version accounted for only 0.3% of observed variance, with adjusted explained variance slightly negative after model complexity adjustment.

Table 7.23: Hierarchical regression model development for JRE JVM interpretive execution energy consumption. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block.

Model Block	R^2	**Adj.** R^2	ΔR^2	$DeltaF$	p
Block 1: Runtime Implementation	0.003	-0.001	0.003	–	–
Block 2: Execution Strategy	0.003	-0.003	0.000	0.008	1.000
Block 3: Workload Category	0.608	0.605	0.605	926.225	$< 0.001^{***}$
Block 4: Microarchitecture	0.708	0.705	0.100	191.450	$< 0.001^{***}$
Final Model	0.708	0.705	–	–	$< 0.001^{***}$

The introduction of garbage collector variant within Block 2 produced no meaningful improvement in model performance, with explained variance remaining effectively unchanged ($\Delta R^2 \approx 0.000$), and the associated F-change statistic indicating no statistically significant improvement in model fit ($F = 0.008$, $p = 1.000$).

Substantial improvement emerged following introduction of workload category within Block 3. This predictor block increased explained variance by 60.5%, representing the single largest improvement observed across the hierarchical modelling procedure. The corresponding F-change statistic confirmed this contribution was highly statistically significant ($F = 926.225$, $p < 0.001$). This finding indicates that workload category represented the dominant determinant of energy consumption variation under interpretive execution conditions.

The introduction of microarchitectural performance event measures within Block 4 produced a further statistically significant improvement in model performance, increasing explained variance by an additional 10.0% ($F = 191.450$, $p < 0.001$). The final model accounted for 70.8% of total observed variance, with adjusted explained variance remaining high at 70.5%.

Taken collectively, these results indicate that unlike the corresponding pooled

model, variation in energy consumption under interpretive execution was overwhelmingly conditioned by workload category, with emergent microarchitectural performance characteristics providing substantial additional explanatory contribution, while implementation-level runtime configuration factors contributed negligibly.

Table 7.24 reports the transition of interpretive execution energy consumption model coefficients across each of the four hierarchical blocks using HC3 heteroskedasticity-consistent robust standard errors.

Among runtime implementation characteristics, only OpenJDK JRE source version 13 remained statistically significant within the final model. Relative to the JDK9 reference condition, JDK13 was associated with significantly higher energy consumption ($\beta = 0.461$, $p < 0.001$), while all remaining JRE source versions were statistically indistinguishable from the reference condition after controlling for all other predictors. GNU GCC build toolchain compiler source version exhibited no statistically significant effects, indicating that compiler-level implementation differences contributed negligibly to observed variation under interpretive execution.

Garbage collector variant exhibited several statistically significant effects. Relative to the CMS reference configuration, G1GC ($\beta = 0.142$, $p < 0.001$), PGC ($\beta = 0.138$, $p < 0.001$), and SGC ($\beta = 0.074$, $p = 0.028$) were associated with significantly higher energy consumption, while CPGC remained statistically non-significant.

Consistent with the hierarchical model development results, workload category exhibited the strongest systematic coefficient effects within the model. Relative to the `apache` workload reference category, the `database` workload was associated with substantially higher energy consumption ($\beta = 1.232$, $p < 0.001$), while `concurrency` workloads also exhibited significantly elevated energy consumption ($\beta = 0.413$, $p < 0.001$). In contrast, `functional` ($\beta = -1.242$, $p < 0.001$), `scala` ($\beta = -0.690$, $p < 0.001$), and `web` workloads ($\beta = -0.776$, $p < 0.001$) were

Table 7.24: Hierarchical regression coefficient development for interpretive execution JRE JVM energy consumption (HC3 robust standard errors)

	Block 1			**Block 2**			**Block 3**			**Block 4**		
Predictor	Coef	SE	Sig	Coef	SE	Sig	Coef	SE	Sig	Coef	SE	Sig
Intercept	-0.409	0.066	***	-0.406	0.081	***	0.153	0.055	**	-0.682	0.077	***
JRE 10	0.018	0.076	–	0.017	0.083	–	0.017	0.051	–	-0.011	0.042	–
JRE 11	0.008	0.076	–	0.007	0.083	–	0.007	0.052	–	-0.081	0.042	.
JRE 12	-0.062	0.076	–	-0.063	0.083	–	0.010	0.052	–	-0.083	0.043	.
JRE 13	0.078	0.076	–	0.078	0.083	–	0.150	0.053	**	0.461	0.045	***
JRE 14	-0.078	0.076	–	-0.079	0.083	–	-0.007	0.053	–	-0.048	0.045	–
GCC 6	-0.015	0.062	–	-0.015	0.063	–	-0.015	0.039	–	-0.016	0.034	–
GCC 7	-0.019	0.062	–	-0.019	0.063	–	-0.019	0.039	–	-0.018	0.034	–
GCC 8	-0.027	0.062	–	-0.027	0.063	–	-0.027	0.039	–	-0.024	0.034	–
CPGC	–	–	–	-0.002	0.129	–	-0.002	0.080	–	-0.035	0.065	–
G1GC	–	–	–	0.001	0.067	–	0.001	0.042	–	0.142	0.039	***
PGC	–	–	–	-0.004	0.062	–	-0.004	0.039	–	0.138	0.035	***
SGC	–	–	–	-0.005	0.062	–	-0.005	0.039	–	0.074	0.034	*
Concurrency	–	–	–	–	–	–	0.184	0.036	***	0.413	0.031	***
Database	–	–	–	–	–	–	1.106	0.030	***	1.232	0.031	***
Functional	–	–	–	–	–	–	-1.447	0.051	***	-1.242	0.052	***
Scala	–	–	–	–	–	–	-1.278	0.037	***	-0.690	0.053	***
Web	–	–	–	–	–	–	-1.279	0.031	***	-0.776	0.095	***
BMR	–	–	–	–	–	–	–	–	–	0.094	0.005	***
L1DLMR	–	–	–	–	–	–	–	–	–	-0.036	0.017	*
L2DLMR	–	–	–	–	–	–	–	–	–	-0.070	0.010	***
PFMR	–	–	–	–	–	–	–	–	–	-4.183	0.333	***

Significance codes: *** $p < .001$; ** $p < .01$; * $p < .05$; . $p < .10$.

associated with significantly lower energy consumption relative to the reference category.

Among microarchitectural performance event measures, branch misprediction rate exhibited a strong positive association with energy consumption ($\beta = 0.094$, $p < 0.001$). L1 data cache load miss rate exhibited a modest negative association ($\beta = -0.036$, $p = 0.037$), while L2 data cache load miss rate was associated with significantly lower energy consumption ($\beta = -0.070$, $p < 0.001$). Page fault rate exhibited the largest individual microarchitectural effect, with increasing page fault activity associated with substantially lower observed energy consumption ($\beta = -4.183$, $p < 0.001$).

Taken collectively, the final model indicates that energy consumption under interpretive execution was principally conditioned by workload category, with selective effects associated with JRE source version, garbage collector variant, and emergent microarchitectural performance event measures, while compiler-level implementation differences contributed negligibly to overall observed variation.

Estimated Final Model

The final hierarchical regression model can be expressed as:

$$\begin{aligned}
\ln(\widehat{EnergyWh})_i = \; & -0.682 - 0.011\,JRE10_i - 0.081\,JRE11_i - 0.083\,JRE12_i \\
& + 0.461\,JRE13_i - 0.048\,JRE14_i - 0.016\,GCC6_i \\
& - 0.018\,GCC7_i - 0.024\,GCC8_i - 0.035\,CPGC_i \\
& + 0.142\,G1GC_i + 0.138\,PGC_i + 0.074\,SGC_i \\
& + 0.413\,Concurrency_i + 1.232\,Database_i - 1.242\,Functional_i \\
& - 0.690\,Scala_i - 0.776\,Web_i + 0.094\,BMR_i \\
& - 0.036\,L1DLMR_i - 0.070\,L2DLMR_i - 4.183\,PFMR_i + \epsilon_i
\end{aligned} \tag{7.9}$$

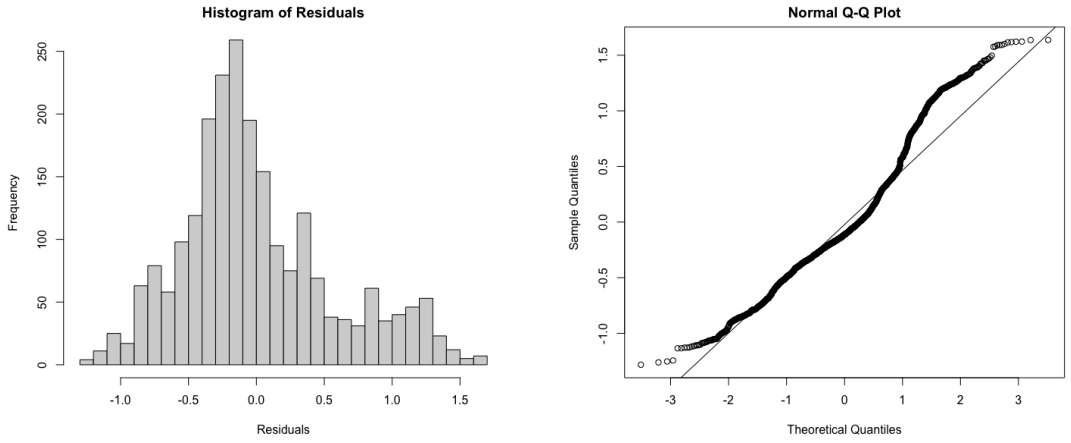
The estimated model coefficients are interpreted relative to the omitted reference categories corresponding to OpenJDK JRE 9, GCC 5, tiered execution, the

default garbage collector baseline, and the `apache` workload category.

7.7.2 Regression Diagnostics

Residual diagnostic assessment indicated generally acceptable model behaviour, although modest deviation from residual normality remained present. Residuals remained closely centred around zero (mean residual approximately zero; median $= -0.113$), suggesting no evidence of systematic prediction bias. Examination of the residual frequency distribution indicated a strong central concentration of residuals close to zero, with the majority of observations clustered between approximately -0.5 and 0.3.

A moderate degree of positive skew remained visible within the residual distribution, with the positive residual tail extending further than the corresponding negative tail, Figure 7.9 (a). Unlike the corresponding pooled energy consumption model, no pronounced extreme residual behaviour or severe outlying observations were evident. Standardised residuals ranged from approximately -1.28 to 1.64, indicating comparatively constrained residual dispersion.



(a) Residual Histogram (b) Normal Q-Q Plot

Figure 7.9: Residual Distribution Diagnostics (Interpretive Energy Consumption Model): Histogram of regression residuals (a) and corresponding normal Q-Q plot (b) for the interpretive execution energy consumption model.

Visual inspection of the corresponding normal Q-Q plot, Figure 7.9 (b), indi-

cated systematic deviation from normality, particularly within the upper residual quantiles, suggesting moderate departure from the normality assumption. Formal assessment using the Shapiro–Wilk test similarly indicated statistically significant deviation from normality ($W = 0.956$, $p < .001$). However, given the large sample size ($n = 2256$) and the comparatively moderate nature of the observed deviations, this result was not considered substantively problematic for model interpretation.

Evidence of heteroskedasticity remained present within the interpretive execution model, with the Breusch–Pagan test indicating statistically significant non-constant residual variance ($BP = 551.71$, $p < .001$). Consequently, coefficient inference relied upon HC3 heteroskedasticity-consistent robust standard errors throughout model estimation.

Sensitivity Analysis

To assess the extent to which the final interpretive execution energy consumption regression model was influenced by potentially influential observations, a sensitivity analysis was performed by re-estimating the full regression model after removing observations identified previously using Cook's distance, Table 7.25.

Table 7.25: Final interpretive execution energy consumption regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. Coefficients are reported on the log-transformed energy consumption scale ($\ln(energyWh)$). The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$.

	Full Model			**Sensitivity Model**			**Dir.**	**Sig.**
Predictor	β	**SE**	p	β	**SE**	p		
Intercept	-0.682	0.077	$< .001$	-0.686	0.073	$< .001$	Yes	Yes
JRE 10	-0.011	0.042	.793	-0.010	0.045	.828	Yes	Yes
JRE 11	-0.081	0.042	.055	-0.079	0.046	.085	Yes	Yes
JRE 12	-0.083	0.043	.053	-0.081	0.046	.080	Yes	Yes
JRE 13	0.461	0.045	$< .001$	0.464	0.051	$< .001$	Yes	Yes
JRE 14	-0.048	0.045	.285	-0.046	0.046	.323	Yes	Yes
GCC 6	-0.016	0.034	.638	-0.015	0.034	.655	Yes	Yes
GCC 7	-0.018	0.034	.596	-0.019	0.034	.568	Yes	Yes
GCC 8	-0.024	0.034	.475	-0.024	0.034	.481	Yes	Yes
CPGC	-0.035	0.065	.592	-0.034	0.070	.625	Yes	Yes
G1GC	0.142	0.039	$< .001$	0.142	0.037	$< .001$	Yes	Yes
PGC	0.138	0.035	$< .001$	0.139	0.035	$< .001$	Yes	Yes
SGC	0.074	0.034	.031	0.072	0.034	.035	Yes	Yes
Concurrency	0.413	0.031	$< .001$	0.415	0.049	$< .001$	Yes	Yes
Database	1.232	0.031	$< .001$	1.234	0.058	$< .001$	Yes	Yes
Functional	-1.242	0.052	$< .001$	-1.240	0.049	$< .001$	Yes	Yes
Scala	-0.690	0.053	$< .001$	-0.689	0.059	$< .001$	Yes	Yes
Web	-0.776	0.095	$< .001$	-0.772	0.129	$< .001$	Yes	Yes
BMR	0.094	0.005	$< .001$	0.094	0.006	$< .001$	Yes	Yes
L1DLMR	-0.036	0.017	.034	-0.036	0.020	.064	Yes	No
L2DLMR	-0.070	0.010	$< .001$	-0.070	0.012	$< .001$	Yes	Yes
PFMR	-4.183	0.333	$< .001$	-4.185	0.290	$< .001$	Yes	Yes

Dir. = coefficient direction remained unchanged.
Sig. = significance status retained (Yes) or changed (No).

The sensitivity analysis indicated a very high degree of model robustness following removal of influential observations. Overall explanatory performance remained essentially unchanged, with adjusted R^2 declining only marginally from 0.705 in the original full model to 0.704 following removal of influential observations. This negligible change suggests that the overall explanatory structure of the model was not materially dependent upon a small subset of influential observations.

Coefficient estimates also remained highly stable across the re-estimated model.

The elevated energy consumption associated with OpenJDK JRE version 13 remained virtually unchanged, increasing only slightly from the original coefficient estimate ($\beta = 0.461$) to the sensitivity model estimate ($\beta = 0.464$). This stability provides strong evidence that the previously identified JRE version 13 anomaly was not artificially driven by a small number of influential observations.

Similarly, workload category level coefficients remained highly stable following model re-estimation. The strong positive association observed for the `database` workload category and the strong negative coefficients associated with the `functional`, `scala`, and `web` workload categories all remained essentially unchanged. These results provide further evidence that workload category remained the dominant determinant of variation in observed energy consumption under interpretive execution.

The principal microarchitectural predictors also demonstrated substantial robustness under sensitivity testing. Branch misprediction rate, L2 data cache load miss rate, and page fault rate all retained coefficient estimates that were almost identical to those observed within the original full regression model. In particular, the page fault rate coefficient remained highly stable, changing only marginally from $\beta = -4.183$ within the original model to $\beta = -4.185$ following removal of influential observations. This finding provides strong evidence that the substantial lower-level hardware relationships identified previously were not dependent upon a small subset of extreme observations.

Only limited inferential variation was observed following model re-estimation. The coefficient associated with L1 data cache load miss rate remained similar in magnitude ($\beta = -0.036$) but exhibited a slight reduction in statistical significance, moving from conventional statistical significance within the original model to marginal significance ($p = .064$) within the sensitivity analysis. However, given the near-identical coefficient magnitude, this result suggests only modest inferential sensitivity rather than substantive structural instability.

Overall, the sensitivity analysis provides strong evidence supporting the reli-

ability of the final interpretive execution energy consumption regression model. Removal of influential observations produced only negligible changes in model fit, coefficient magnitude, and overall inferential structure. Taken together, these findings suggest that the interpretive execution energy consumption model exhibits a highly stable underlying explanatory structure and that the principal relationships identified within the full regression model are not materially dependent upon a small subset of influential observations.

Bootstrap Validation

To further assess coefficient stability, a stratified non-parametric bootstrap procedure was performed using 1,000 resamples. Resampling was stratified according to the experimental design, defined by OpenJDK JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, and workload category. This procedure preserves the structure of the experimental design while assessing coefficient stability under repeated resampling.

Table 7.26 summarises the original coefficient estimates, bootstrap mean estimates, bootstrap standard errors, and percentile-based 95% confidence intervals for the interpretive execution energy consumption regression model.

The bootstrap results indicated an exceptionally high degree of coefficient stability across the interpretive execution energy consumption model. Original coefficient estimates remained almost perfectly aligned with the corresponding bootstrap mean estimates for all predictors, indicating that the fitted regression coefficients were not strongly dependent upon any particular sample configuration. Across nearly all predictors, repeated resampling produced highly consistent coefficient estimates, suggesting a stable underlying model structure.

Several of the strongest relationships observed within the regression model remained highly stable under repeated resampling. The elevated energy consumption associated with OpenJDK JRE version 13 remained consistently positive, with the bootstrap mean estimate ($\beta = 0.460$) remaining effectively identical to

Table 7.26: Stratified bootstrap coefficient stability analysis for the interpretive execution energy consumption regression model.

Predictor	Orig. β	Boot. Mean	Boot. SE	CI_Lower	CI_Upper
(Intercept)	-0.682	-0.683	0.069	-0.817	-0.548
JRE 10	-0.011	-0.013	0.042	-0.097	0.065
JRE 11	-0.081	-0.083	0.042	-0.169	0.001
JRE 12	-0.083	-0.086	0.043	-0.172	-0.006
JRE 13	0.461	0.460	0.044	0.378	0.542
JRE 14	-0.048	-0.050	0.045	-0.139	0.036
GCC 6	-0.016	-0.017	0.033	-0.081	0.044
GCC 7	-0.018	-0.018	0.033	-0.085	0.046
GCC 8	-0.024	-0.025	0.034	-0.093	0.041
CPGC	-0.035	-0.036	0.062	-0.150	0.086
G1GC	0.142	0.141	0.038	0.066	0.220
PGC	0.138	0.138	0.034	0.069	0.206
SGC	0.074	0.074	0.032	0.011	0.142
Concurrency	0.413	0.413	0.028	0.358	0.466
Database	1.232	1.231	0.029	1.175	1.287
Functional	-1.242	-1.243	0.045	-1.332	-1.157
Scala	-0.690	-0.693	0.049	-0.791	-0.597
Web	-0.776	-0.783	0.082	-0.935	-0.626
BMR	0.094	0.094	0.005	0.085	0.103
L1DLMR	-0.036	-0.036	0.016	-0.065	-0.006
L2DLMR	-0.070	-0.069	0.008	-0.085	-0.053
PFMR	-4.183	-4.156	0.278	-4.675	-3.620

Note: Orig. β = Original Beta; Boot. Mean = Bootstrap Mean; Boot. SE = Bootstrap Standard Error;

the original model coefficient ($\beta = 0.461$). JRE version 12 also retained a stable negative coefficient structure, with confidence intervals excluding zero. In contrast, confidence intervals associated with JRE versions 10, 11, and 14 crossed zero, indicating comparatively weaker coefficient stability for these runtime versions.

Workload category effects demonstrated particularly strong coefficient stability. The `database` workload category retained a large positive coefficient with an exceptionally narrow confidence interval excluding zero, reinforcing the earlier finding that database-oriented workloads were strongly associated with elevated energy consumption under interpretive execution. Similarly, the `functional`, `scala`, and `web` workload categories all retained consistently negative coefficients

with comparatively narrow confidence intervals, while the `concurrency` workload category remained positively associated with increased energy consumption. These findings provide further evidence that workload category represented one of the dominant determinants of energy consumption variation within the interpretive execution model.

Garbage collector variant exhibited mixed coefficient stability. Confidence intervals associated with G1GC, PGC, and SGC all excluded zero, indicating stable positive coefficient structure for these runtime configurations. In contrast, the CPGC garbage collector variant exhibited confidence intervals crossing zero, suggesting comparatively weaker inferential stability. Similarly, all GNU GCC compiler version coefficients exhibited confidence intervals overlapping zero, indicating limited evidence that compiler source version contributed consistently to explaining energy consumption variation under interpretive execution.

The lower-level microarchitectural performance event measures demonstrated substantial coefficient stability. Branch misprediction rate retained a highly stable positive relationship with energy consumption, with narrow confidence intervals consistently excluding zero. L1 data cache load miss rate and L2 data cache load miss rate both retained stable negative coefficient structures under repeated resampling. Page fault rate exhibited particularly strong coefficient stability, with the bootstrap mean estimate ($\beta = -4.156$) remaining closely aligned with the original model coefficient ($\beta = -4.183$), accompanied by a comparatively narrow bootstrap standard error ($SE = 0.278$) and confidence intervals remaining consistently below zero. Taken together, these findings provide strong evidence that lower-level microarchitectural performance event measures contributed robustly to explaining observed variation in energy consumption.

Overall, the bootstrap resampling analysis provides strong evidence supporting the reliability of the interpretive execution energy consumption regression model. Relative to the corresponding pooled energy consumption model, coefficient estimates exhibited substantially greater stability under repeated resampling,

further supporting the broader modelling observation developed throughout RQ3 that restricting analysis to a single execution mode reduces structural heterogeneity and improves overall coefficient stability. More broadly, these findings reinforce the emerging pattern observed throughout the modelling analyses that workload category and lower-level microarchitectural predictors exhibit greater structural stability than higher-level runtime configuration variables when modelling JRE JVM energy consumption.

7.7.3 Summary

This section addressed **SRQ3.2a** by evaluating the extent to which a hierarchical regression model incorporating runtime configuration factors, workload category, and microarchitectural performance event measures accounted for observed variation in JRE JVM energy consumption under interpretive execution.

The final hierarchical model accounted for a substantial proportion of observed variation in log-transformed energy consumption ($R^2 = .708$, adjusted $R^2 = .705$). Runtime implementation characteristics alone explained negligible variation ($R^2 = .003$), while garbage collector variant contributed no meaningful improvement when introduced independently ($\Delta R^2 \approx 0.000$). The largest increase in explanatory performance occurred following introduction of workload category ($\Delta R^2 = .605$), confirming workload category as the dominant determinant of energy consumption under interpretive execution. Microarchitectural performance event measures provided a further substantial improvement in explanatory performance ($\Delta R^2 = .100$), indicating that lower-level execution characteristics contributed meaningfully beyond higher-level runtime configuration variables.

Within the final model, OpenJDK JRE version 13 remained positively associated with elevated energy consumption, while GNU GCC build toolchain compiler version exhibited no statistically significant effects. Relative to the CMS reference configuration, G1GC, PGC, and SGC were associated with significantly higher energy consumption. Workload category remained the strongest predictor

group, with the `database` and `concurrency` workload categories associated with higher energy consumption, while the `functional`, `scala`, and `web` workload categories were associated with lower energy consumption. Branch misprediction rate was positively associated with energy consumption, whereas L1 data cache load miss rate, L2 data cache load miss rate, and page fault rate exhibited negative associations.

Regression diagnostics indicated that the interpretive execution model provided an adequate statistical representation of the observed data. Although residual normality and homoskedasticity assumptions were not perfectly satisfied, residual behaviour remained well constrained and coefficient inference was protected through HC3 heteroskedasticity-consistent robust standard errors. Sensitivity analysis demonstrated that removal of influential observations produced negligible changes in model fit or coefficient structure, while stratified bootstrap resampling confirmed a high degree of coefficient stability across repeated samples.

Collectively, these findings indicate that restricting analysis to interpretive execution substantially reduced structural heterogeneity relative to the pooled energy consumption model. Under interpretive execution, energy consumption was principally conditioned by workload category and lower-level microarchitectural performance event measures, while runtime implementation variables contributed comparatively little explanatory power.

Research Question Findings

SRQ3.2a: Hierarchical regression modelling demonstrated that workload category and microarchitectural performance event measures accounted for the majority of observed variation in JRE JVM energy consumption under interpretive execution ($R^2 = .708$). Runtime implementation characteristics contributed negligibly, while microarchitectural performance event measures provided a substantial additional improvement in explanatory performance beyond workload category. The interpretive execution energy consumption model exhibited a highly stable explanatory structure. Sensitivity analysis and bootstrap validation demonstrated that the principal coefficient relationships remained robust following removal of influential observations and repeated resampling, indicating that the observed relationships are not artefacts of sample-specific variation.

7.8 Energy Consumption Model: Tiered Execution

This section principally addresses sub-research question **SRQ3.2b**, which examines the extent to which variation in JRE/JVM tiered execution energy consumption can be accounted for through regression modelling incorporating runtime configuration factors, workload category, and microarchitectural performance event measures:

> **SRQ3.2b:** ***To what extent can variation in JRE/JVM tiered execution energy consumption be accounted for through regression modelling incorporating JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, workload category, and microarchitectural performance event measures?***

To further examine whether relationships identified within the pooled energy consumption model remained stable when restricting analysis to a single execution mode, the regression analysis was repeated separately for observations generated under tiered execution only.

The tiered execution model retains the same theoretically specified predictor set and hierarchical modelling structure introduced previously for the pooled energy consumption analysis. However, because all observations correspond exclusively to tiered execution, the execution mode predictor (JEM) is omitted from the model specification. As with the preceding power use analysis, this execution-mode-specific decomposition permits assessment of the extent to which predictor relationships observed within the pooled model remain stable under tiered execution only.

Because the energy consumption analysis employs the log-transformed energy consumption outcome variable introduced previously, this execution-mode-specific decomposition additionally permits assessment of whether the improved model performance and diagnostic adequacy observed within the pooled log-transformed energy consumption model are maintained when analysis is restricted to the more homogeneous tiered execution subset.

7.8.1 Hierarchical Regression Results

Predictors were entered hierarchically according to the same theoretically motivated ordering introduced previously, reflecting distinct categories of system-level influence on observed JRE JVM energy consumption. Predictor blocks progress from directly specified runtime implementation characteristics, to execution configuration factors, followed by workload-induced computational demand, and finally emergent microarchitectural performance event measures arising dynamically during program execution.

This hierarchical structure therefore permits assessment of the extent to which successive layers of the system architecture account for variation in observed JRE

JVM energy consumption under tiered execution conditions.

Table 7.27 summarises the hierarchical regression model development for tiered execution energy consumption. As with the corresponding interpretive execution model, runtime implementation characteristics contributed negligibly to overall model performance. The baseline model incorporating only JRE source version and GNU GCC build toolchain compiler version accounted for only 0.3% of observed variance, with adjusted explained variance remaining slightly negative after model complexity adjustment.

Table 7.27: Hierarchical regression model development for JRE JVM tiered execution energy consumption. The table reports explained variance for each model block, adjusted explained variance, and change in explained variance relative to the preceding block.

Model Block	R^2	**Adj.** R^2	ΔR^2	ΔF	p
Block 1: Runtime Implementation	0.003	-0.001	0.003	–	–
Block 2: Execution Strategy	0.003	-0.002	0.000	0.633	0.639
Block 3: Workload Category	0.712	0.710	0.709	1261.514	$< 0.001^{***}$
Block 4: Microarchitecture	0.738	0.736	0.026	57.189	$< 0.001^{***}$
Final Model	0.738	0.736	–	–	$< 0.001^{***}$

The introduction of garbage collector variant within Block 2 produced no meaningful improvement in model performance, with explained variance remaining effectively unchanged ($\Delta R^2 \approx 0.000$), and the associated F-change statistic indicating no statistically significant improvement in model fit ($F = 0.633$, $p = 0.639$).

Substantial improvement emerged following introduction of workload category within Block 3. This predictor block increased explained variance by 70.9%, representing the single largest improvement observed across the hierarchical modelling procedure. The corresponding F-change statistic confirmed this contribution was highly statistically significant ($F = 1261.514$, $p < 0.001$). Compared with the corresponding interpretive execution model, workload category accounted for an even greater proportion of observed variation, indicating that workload category represented the dominant determinant of energy consumption under

tiered execution conditions.

The introduction of microarchitectural performance event measures within Block 4 produced a further statistically significant improvement in model performance, increasing explained variance by an additional 2.6% ($F = 57.189$, $p < 0.001$). Although statistically significant, this contribution was substantially smaller than that observed under interpretive execution, suggesting that lower-level hardware execution characteristics contributed comparatively little additional explanatory information once workload category had been accounted for. The final model accounted for 73.8% of total observed variance, with adjusted explained variance remaining high at 73.6%.

Taken collectively, these results indicate that variation in energy consumption under tiered execution was principally conditioned by workload category. In contrast to the corresponding interpretive execution model, microarchitectural performance event measures contributed comparatively little additional explanatory power following inclusion of workload category levels, suggesting that tiered execution substantially attenuated the influence of lower-level hardware execution variability on observed energy consumption.

Table 7.28 reports the transition of tiered execution energy consumption model coefficients across each of the four hierarchical blocks using HC3 heteroskedasticity-consistent robust standard errors.

Among runtime implementation characteristics, no OpenJDK JRE source version remained positively associated with energy consumption within the final model. Most notably, the substantial positive association previously observed for OpenJDK JRE version 13 under interpretive execution disappeared following restriction to tiered execution ($\beta = -0.023$, $p > .05$), suggesting that the earlier OpenJDK JRE version 13 anomaly was principally associated with interpretive execution rather than representing an intrinsic property of the runtime implementation itself. By contrast, OpenJDK JRE version 10 exhibited significantly lower energy consumption relative to the OpenJDK JRE version 9 reference con-

Table 7.28: Hierarchical regression coefficient development for tiered execution JRE JVM energy consumption ($\ln(energyWh)$). HC3 heteroskedasticity-consistent robust standard errors are reported for all coefficient estimates.

	Block 1			**Block 2**			**Block 3**			**Block 4**		
Predictor	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**	β	**SE**	**Sig.**
Intercept	-2.974	0.056	***	-2.970	0.070	***	-2.315	0.041	***	-2.273	0.100	***
JRE 10	0.002	0.067		0.005	0.070		0.005	0.037		-0.122	0.037	***
JRE 11	-0.011	0.066		-0.008	0.070		-0.008	0.037		-0.011	0.035	
JRE 12	-0.064	0.066		-0.061	0.070		0.004	0.038		0.012	0.035	
JRE 13	0.063	0.067		0.066	0.071		0.131	0.038	***	-0.023	0.041	
JRE 14	-0.087	0.067		-0.083	0.071		-0.018	0.038		0.013	0.037	
GCC 6	-0.023	0.056		-0.023	0.056		-0.023	0.030		-0.017	0.028	
GCC 7	-0.024	0.056		-0.024	0.056		-0.024	0.030		-0.022	0.028	
GCC 8	-0.024	0.056		-0.024	0.056		-0.024	0.030		-0.167	0.031	***
CPGC	–	–		0.010	0.116		0.010	0.061		0.070	0.072	
G1GC	–	–		0.016	0.057		0.016	0.030		0.082	0.032	*
PGC	–	–		-0.024	0.058		-0.024	0.031		0.038	0.032	
SGC	–	–		-0.020	0.057		-0.020	0.031		0.024	0.031	
Concurrency	–	–		–	–		-0.263	0.034	***	-0.482	0.043	***
Database	–	–		–	–		0.841	0.025	***	0.768	0.028	***
Functional	–	–		–	–		-1.712	0.024	***	-1.610	0.039	***
Scala	–	–		–	–		-1.423	0.041	***	-1.498	0.046	***
Web	–	–		–	–		-0.471	0.020	***	-0.300	0.060	***
BMR	–	–		–	–		–	–		-0.138	0.015	***
L1DLMR	–	–		–	–		–	–		0.146	0.038	***
L2DLMR	–	–		–	–		–	–		0.007	0.002	**
PFMR	–	–		–	–		–	–		-0.509	0.352	

Significance codes: *** $p < .001$; ** $p < .01$; * $p < .05$; . $p < .10$.

figuration ($\beta = -0.122$, $p < 0.001$). GNU GCC build toolchain compiler source version again exhibited limited overall influence, although GCC8 emerged as a statistically significant negative predictor ($\beta = -0.167$, $p < 0.001$), while GCC6 and GCC7 remained statistically non-significant.

Garbage collector variant exhibited comparatively weaker effects than those observed under interpretive execution. Relative to the CMS reference configuration, only G1GC remained positively associated with energy consumption ($\beta = 0.082$, $p < 0.05$), while CPGC, PGC, and SGC were all statistically non-significant. These findings suggest that garbage collector effects became substantially attenuated under tiered execution.

Consistent with the hierarchical model development results, workload category exhibited the strongest systematic coefficient effects within the model. Relative to the `apache` workload reference category, the `database` workload remained associated with substantially higher energy consumption ($\beta = 0.768$, $p < 0.001$), while the `functional` ($\beta = -1.610$, $p < 0.001$), `scala` ($\beta = -1.498$, $p < 0.001$), and `web` ($\beta = -0.300$, $p < 0.001$) workload categories remained associated with significantly lower energy consumption. In contrast to the corresponding interpretive execution model, however, the `concurrency` workload category exhibited a statistically significant negative association ($\beta = -0.482$, $p < 0.001$), representing a complete reversal of coefficient direction between the two execution modes.

Among microarchitectural performance event measures, branch misprediction rate exhibited a statistically significant negative association with energy consumption ($\beta = -0.138$, $p < 0.001$), contrasting with the corresponding positive relationship observed under interpretive execution. Conversely, both L1 data cache load miss rate ($\beta = 0.146$, $p < 0.001$) and L2 data cache load miss rate ($\beta = 0.007$, $p < 0.01$) exhibited positive associations with energy consumption, representing coefficient sign reversals relative to the interpretive execution model. Although page fault rate retained a negative coefficient, the relationship was no longer statistically significant ($\beta = -0.509$, $p > .05$), indicating substantial

attenuation of the strong page fault effect previously observed under interpretive execution.

Taken collectively, the final model indicates that energy consumption under tiered execution was principally conditioned by workload category, while runtime implementation, garbage collector variant, and microarchitectural performance event measures exhibited comparatively weaker and, in several cases, fundamentally different relationships from those observed under interpretive execution. The disappearance of the OpenJDK JRE version 13 anomaly together with multiple coefficient sign reversals across workload and microarchitectural predictors provides strong evidence that tiered execution substantially restructures the statistical relationships governing observed JRE JVM energy consumption.

Estimated Final Model

The final hierarchical regression model can be expressed as:

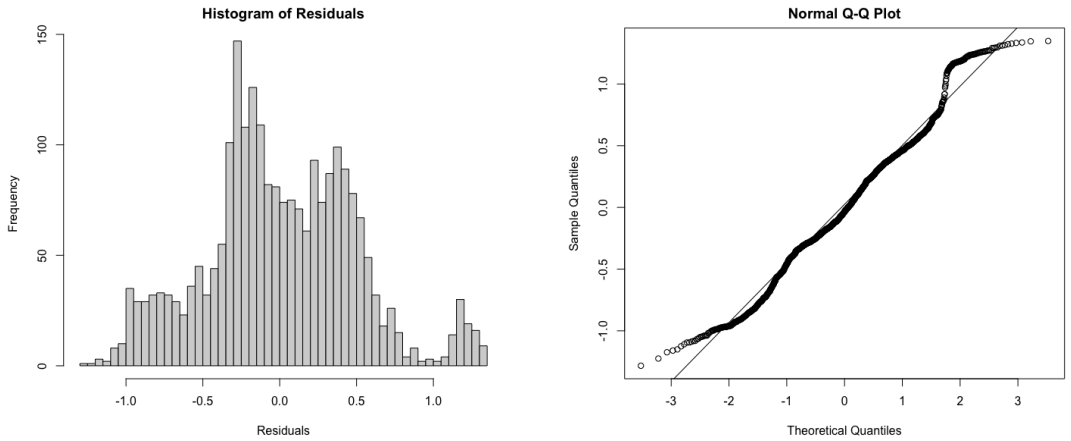
$$\begin{aligned}
\ln(\widehat{EnergyWh})_i = \; & -2.273 - 0.122\,JRE10_i - 0.011\,JRE11_i + 0.012\,JRE12_i \\
& - 0.023\,JRE13_i + 0.013\,JRE14_i - 0.017\,GCC6_i \\
& - 0.022\,GCC7_i - 0.167\,GCC8_i + 0.070\,CPGC_i \\
& + 0.082\,G1GC_i + 0.038\,PGC_i + 0.024\,SGC_i \\
& - 0.482\,Concurrency_i + 0.768\,Database_i - 1.610\,Functional_i \\
& - 1.498\,Scala_i - 0.300\,Web_i - 0.138\,BMR_i \\
& + 0.146\,L1DLMR_i + 0.007\,L2DLMR_i - 0.509\,PFMR_i + \epsilon_i
\end{aligned} \tag{7.10}$$

The estimated model coefficients are interpreted relative to the omitted reference categories corresponding to OpenJDK JRE 9, GCC 5, the CMS garbage collector, and the `apache` workload category.

7.8.2 Regression Diagnostics

Residual diagnostic assessment indicated generally well-behaved model structure, with evidence of modest improvement relative to the corresponding interpretive execution energy consumption model. Residuals remained closely centred around zero (mean residual approximately zero; median = −0.035), suggesting no evidence of systematic prediction bias. Examination of the residual frequency distribution indicated a strong central concentration of residuals close to zero, with the majority of observations clustered within a relatively narrow range around the central distribution.

A modest degree of positive skew remained visible within the residual distribution, although the positive residual tail was less pronounced than that observed previously within the interpretive execution model, Figure 7.10. No evidence of pronounced extreme residual behaviour or severe outlying observations was observed. Standardised residuals ranged from approximately −1.28 to 1.35, indicating comparatively constrained residual dispersion.



(a) Residual Histogram (b) Normal Q-Q Plot

Figure 7.10: Residual Distribution Diagnostics (Tiered Energy Consumption Model): Histogram of regression residuals (a) and corresponding normal Q-Q plot (b) for the tiered execution energy consumption model.

Visual inspection of the corresponding normal Q-Q plot indicated moderate

deviation from normality, primarily within the residual tails. Formal assessment using the Shapiro–Wilk test similarly indicated statistically significant deviation from normality ($W = 0.987$, $p < .001$). However, given the large sample size ($n = 2352$) and the comparatively well-behaved residual distribution observed visually, this result was not considered substantively problematic for model interpretation.

Sensitivity Analysis

To assess the extent to which the final tiered execution energy consumption regression model was influenced by potentially influential observations, a sensitivity analysis was performed by re-estimating the full regression model after removing observations identified previously using Cook's distance, Table 7.29.

Table 7.29: Final tiered execution energy consumption regression model with HC3 robust standard errors and sensitivity analysis excluding influential observations. Coefficients are reported on the log-transformed energy consumption scale ($\ln(energyWh)$). The full model reports HC3 heteroskedasticity-consistent robust standard errors. The sensitivity model excludes observations exceeding the Cook's distance threshold of $4/n$.

	Full Model			**Sensitivity Model**			**Dir.**	**Sig.**
Predictor	β	**SE**	p	β	**SE**	p		
Intercept	-2.273	0.100	$< .001$	-1.653	0.059	$< .001$	Yes	Yes
JRE 10	-0.122	0.037	$< .001$	-0.108	0.028	$< .001$	Yes	Yes
JRE 11	-0.011	0.035	.754	0.044	0.027	.111	No	Yes
JRE 12	0.012	0.035	.733	0.059	0.028	.033	Yes	No
JRE 13	-0.023	0.041	.575	-0.129	0.030	$< .001$	Yes	No
JRE 14	0.013	0.037	.719	0.097	0.028	$< .001$	Yes	No
GCC 6	-0.017	0.028	.548	-0.014	0.022	.522	Yes	Yes
GCC 7	-0.022	0.028	.428	-0.021	0.022	.338	Yes	Yes
GCC 8	-0.167	0.031	$< .001$	-0.164	0.024	$< .001$	Yes	Yes
CPGC	0.070	0.072	.330	0.085	0.051	.095	Yes	No
G1GC	0.082	0.032	.011	0.059	0.023	.012	Yes	Yes
PGC	0.038	0.032	.234	0.039	0.023	.094	Yes	No
SGC	0.024	0.031	.431	0.064	0.023	.005	Yes	No
Concurrency	-0.482	0.043	$< .001$	-0.290	0.030	$< .001$	Yes	Yes
Database	0.768	0.028	$< .001$	0.756	0.034	$< .001$	Yes	Yes
Functional	-1.610	0.039	$< .001$	-1.452	0.027	$< .001$	Yes	Yes
Scala	-1.498	0.046	$< .001$	-1.513	0.029	$< .001$	Yes	Yes
Web	-0.300	0.060	$< .001$	0.302	0.047	$< .001$	No	No
BMR	-0.138	0.015	$< .001$	-0.100	0.011	$< .001$	Yes	Yes
L1DLMR	0.146	0.038	$< .001$	-0.166	0.023	$< .001$	No	No
L2DLMR	0.007	0.002	.002	0.009	0.001	$< .001$	Yes	Yes
PFMR	-0.509	0.352	.149	-0.884	0.171	$< .001$	Yes	No

Dir. = coefficient direction remained unchanged.
Sig. = significance status retained (Yes) or changed (No).

The sensitivity analysis indicated that the overall explanatory structure of the tiered execution energy consumption model remained broadly stable following removal of influential observations. Overall explanatory performance changed only marginally, with adjusted R^2 increasing slightly from 0.736 in the original full model to 0.745 following removal of influential observations. This modest improvement suggests that the principal explanatory structure of the model was not materially dependent upon a relatively small subset of influential observations.

Several of the strongest relationships identified within the original regression model remained highly stable following model re-estimation. The significant negative association observed for GCC8 remained virtually unchanged ($\beta = -0.167$ to $\beta = -0.164$), while the principal workload category effects associated with the `database`, `functional`, and `scala` workload categories all retained both their direction and statistical significance. Similarly, branch misprediction rate and L2 data cache load miss rate exhibited only minor changes in coefficient magnitude, indicating that these relationships remained robust following removal of influential observations.

However, compared with the corresponding interpretive execution model, greater coefficient restructuring was evident following removal of influential observations. Several runtime implementation coefficients changed inferential status, with JRE 12, JRE 13, and JRE 14 becoming statistically significant within the sensitivity model despite remaining non-significant in the original model. In particular, the JRE 13 coefficient became significantly negative ($\beta = -0.129$, $p < 0.001$), suggesting that the absence of a JRE 13 effect within the original tiered model was influenced to some extent by the influential observations identified previously.

Greater sensitivity was also observed among selected garbage collector and workload coefficients. While G1GC remained statistically significant across both models, SGC became statistically significant following removal of influential observations, and both CPGC and PGC approached conventional significance thresholds. The `web` workload category exhibited the greatest structural change, reversing coefficient direction from a significant negative association within the original model ($\beta = -0.300$) to a significant positive association within the sensitivity model ($\beta = 0.302$). Similarly, L1 data cache load miss rate changed from a significant positive association ($\beta = 0.146$) to a significant negative association ($\beta = -0.166$), indicating substantial sensitivity of this relationship to influential observations. Page fault rate also became statistically significant following model re-estimation, although its coefficient direction remained unchanged.

Overall, the sensitivity analysis indicates that although the tiered execution energy consumption model retains a broadly stable explanatory structure, several individual coefficient estimates exhibit greater sensitivity to influential observations than observed within the corresponding interpretive execution model. The principal workload relationships and several key microarchitectural predictors remained robust, but the observed coefficient restructuring suggests that inference for a number of weaker runtime configuration effects should be interpreted with greater caution under tiered execution.

Additional examination of influential observation distribution revealed substantial clustering across specific workload categories rather than random dispersion throughout the dataset, Table 7.30.

Table 7.30: Distribution of influential observations identified within the tiered execution energy consumption sensitivity analysis. Influential observations were identified using Cook's distance threshold $4/n$. Percentages in the final row indicate the proportion of all influential observations attributable to each workload category.

JRE	**Apache**	**Conc.**	**Database**	**Func.**	**Scala**	**Web**	**Total**
JRE 9	2	9	1	0	23	1	36
JRE 10	0	1	0	0	20	6	27
JRE 11	0	8	0	0	16	0	24
JRE 12	0	9	0	0	16	0	25
JRE 13	0	0	0	0	16	4	20
JRE 14	1	9	1	0	22	0	33
Total	3	36	2	0	113	11	165
% Total	1.8%	21.8%	1.2%	0.0%	68.5%	6.7%	100.0%

Note: Conc. = Concurrency; Func. = Functional;

Of the observations identified as influential using Cook's distance, approximately 68% originated from the `scala` workload category, while a further substantial proportion were concentrated within the `concurrency` workload category. In contrast, no influential observations were identified within the `functional` workload category. This non-random distribution suggests that the greater coefficient sensitivity observed within the tiered execution energy consumption model may partially reflect heterogeneous execution behaviour associated with specific workload classes rather than general instability within the broader model structure.

Notably, influential observations were distributed relatively evenly across OpenJDK JRE versions, indicating that the previously observed coefficient instability associated with certain JRE predictors was unlikely to reflect disproportionate concentration within any individual runtime version.

Bootstrap Validation

To further assess coefficient stability, a stratified non-parametric bootstrap procedure was performed using 1,000 resamples. Resampling was stratified according to the experimental design, defined by OpenJDK JRE source version, GNU GCC build toolchain compiler version, garbage collector variant, and workload category. This procedure preserves the structure of the experimental design while assessing coefficient stability under repeated resampling.

Table 7.31 summarises the original coefficient estimates, bootstrap mean estimates, bootstrap standard errors, and percentile-based 95% confidence intervals for the tiered execution energy consumption regression model.

The bootstrap results indicated a generally high degree of coefficient stability across the tiered execution model. Original coefficient estimates remained closely aligned with the corresponding bootstrap mean estimates for the overwhelming majority of predictors, indicating that the fitted regression coefficients were not strongly dependent upon a particular sample configuration. The relatively small bootstrap standard errors observed across most predictors further suggest that the underlying coefficient structure remained highly stable under repeated resampling.

Several of the strongest relationships observed within the full regression model remained highly stable under repeated resampling. Workload coefficients associated with the `database`, `functional`, `scala`, and `web` workload categories all retained comparatively narrow confidence intervals that remained consistently separated from zero. In particular, the `database` and `functional` workload coefficients exhibited extremely high stability, with bootstrap mean estimates remaining almost identical to the original model coefficients. These findings provide further

evidence that workload category remained the principal determinant of observed energy consumption variation under tiered execution.

The principal microarchitectural performance event measures also demonstrated substantial inferential stability. Branch misprediction rate, L1 data cache load miss rate, and L2 data cache load miss rate all retained narrow confidence intervals excluding zero, indicating robust lower-level hardware relationships within the final regression model. This finding is particularly important given the stronger microarchitectural contribution previously identified within the hierarchical model development analysis.

Table 7.31: Stratified bootstrap coefficient stability analysis for the tiered execution energy consumption regression model.

Predictor	Orig. β	Boot. Mean	Boot. SE	CI_Lower	CI_Upper
(Intercept)	-2.273	-2.264	0.075	-2.398	-2.107
JRE 10	-0.122	-0.121	0.036	-0.193	-0.048
JRE 11	-0.011	-0.009	0.034	-0.078	0.052
JRE 12	0.012	0.012	0.035	-0.057	0.080
JRE 13	-0.023	-0.022	0.039	-0.102	0.053
JRE 14	0.013	0.016	0.035	-0.050	0.084
GCC 6	-0.017	-0.016	0.028	-0.068	0.039
GCC 7	-0.022	-0.021	0.029	-0.080	0.035
GCC 8	-0.167	-0.166	0.029	-0.220	-0.108
CPGC	0.070	0.079	0.065	-0.043	0.208
G1GC	0.082	0.085	0.030	0.024	0.143
PGC	0.038	0.041	0.031	-0.022	0.104
SGC	0.024	0.026	0.030	-0.032	0.086
Concurrency	-0.482	-0.478	0.038	-0.553	-0.396
Database	0.768	0.768	0.020	0.729	0.807
Functional	-1.610	-1.603	0.030	-1.653	-1.531
Scala	-1.498	-1.493	0.036	-1.558	-1.419
Web	-0.300	-0.297	0.041	-0.371	-0.212
BMR	-0.138	-0.136	0.011	-0.157	-0.113
L1DLMR	0.146	0.142	0.029	0.084	0.197
L2DLMR	0.007	0.007	0.001	0.003	0.009
PFMR	-0.509	-0.582	0.252	-1.361	-0.307

Note: Orig. β = Original Beta; Boot. Mean = Bootstrap Mean; Boot. SE = Bootstrap Standard Error;

The bootstrap analysis additionally clarified several weaker runtime config-

uration effects. Confidence intervals associated with JRE versions 11, 12, 13, and 14 all crossed zero, indicating comparatively weak coefficient stability for these runtime versions. Particularly notable is OpenJDK JRE version 13, which exhibited no stable coefficient structure under repeated resampling. This contrasts substantially with the corresponding interpretive execution model, where JRE version 13 remained strongly positive and highly stable. Similarly, confidence intervals associated with GCC 6, GCC 7, and several garbage collector variants, specifically CPGC, PGC, and SGC, also crossed zero, suggesting comparatively weaker inferential stability for these secondary runtime configuration variables.

Page fault rate again produced a particularly interesting result. Although the bootstrap confidence interval remained entirely separated from zero, indicating a stable negative relationship with energy consumption, page fault rate exhibited the largest bootstrap standard error within the model ($SE = 0.252$) together with the largest discrepancy between the original coefficient estimate ($\beta = -0.509$) and the corresponding bootstrap mean estimate ($\beta = -0.582$). This suggests comparatively greater coefficient instability for page fault rate relative to the other microarchitectural predictors, consistent with patterns observed previously across both the pooled and power use regression models.

Overall, the bootstrap resampling analysis provides strong evidence supporting the general reliability of the tiered execution energy consumption regression model. Although several secondary runtime configuration predictors exhibited comparatively weak coefficient stability, the broader coefficient structure remained highly robust under repeated stratified resampling. Taken together, these findings suggest that the tiered execution energy consumption model exhibits a generally stable underlying explanatory structure, while further reinforcing the emerging interpretation developed throughout this chapter that execution strategy conditions the extent to which lower-level system relationships remain statistically observable.

7.8.3 Summary

This section addressed **SRQ3.2b** by evaluating the extent to which a hierarchical regression model incorporating runtime configuration factors, workload category, and microarchitectural performance event measures accounted for observed variation in JRE JVM energy consumption under tiered execution.

The final hierarchical model accounted for a substantial proportion of observed variation in log-transformed energy consumption ($R^2 = .738$, adjusted $R^2 = .736$). As with the corresponding interpretive execution model, runtime implementation characteristics alone explained negligible variation ($R^2 = .003$), while garbage collector variant contributed no meaningful improvement when introduced independently ($\Delta R^2 \approx 0.000$). The largest increase in explanatory performance occurred following introduction of workload category ($\Delta R^2 = .709$), confirming workload category as the dominant determinant of energy consumption under tiered execution. Although microarchitectural performance event measures remained statistically significant, they contributed only a comparatively modest additional improvement in explanatory performance ($\Delta R^2 = .026$), indicating that lower-level hardware execution characteristics contributed substantially less additional explanatory information than observed under interpretive execution.

Within the final model, OpenJDK JRE version 13 no longer exhibited the elevated energy consumption observed previously under interpretive execution, while OpenJDK JRE version 10 and GCC8 were associated with significantly lower energy consumption relative to their respective reference categories. Garbage collector variant exhibited comparatively weak effects, with only G1GC remaining statistically significant. Workload category remained the dominant predictor group, with the `database` workload associated with higher energy consumption, while the `concurrency`, `functional`, `scala`, and `web` workload categories exhibited lower energy consumption relative to the `apache` reference category. Several microarchitectural predictors also exhibited fundamentally different relationships from those observed under interpretive execution, including complete coefficient

sign reversals for branch misprediction rate and both cache load miss rate measures.

Regression diagnostics indicated that the tiered execution model provided an adequate statistical representation of the observed data. Residual behaviour remained well constrained, with comparatively modest deviation from normality and no evidence of pronounced outlying residual behaviour. Although sensitivity analysis demonstrated greater coefficient restructuring than observed under the corresponding interpretive execution model, the dominant workload relationships and principal microarchitectural predictors remained broadly stable. Stratified bootstrap resampling similarly confirmed that the overall explanatory structure of the model remained reliable despite comparatively weaker stability among several secondary runtime configuration variables.

Collectively, these findings indicate that restricting analysis to tiered execution substantially altered the statistical relationships governing observed energy consumption. Workload category became the overwhelmingly dominant determinant of variation, while the additional explanatory contribution associated with lower-level microarchitectural performance event measures was markedly reduced. Together with the disappearance of the OpenJDK JRE version 13 anomaly and the observed coefficient sign reversals, these findings provide further evidence that tiered execution fundamentally restructures the statistical relationships governing JRE JVM energy consumption.

Research Question Findings

SRQ3.2b: Hierarchical regression modelling demonstrated that workload category accounted for the majority of observed variation in JRE JVM energy consumption under tiered execution ($R^2 = .738$). Runtime implementation characteristics contributed negligibly, while microarchitectural performance event measures provided only a comparatively modest additional contribution after workload category had been introduced. Compared with the corresponding interpretive execution model, tiered execution substantially altered the underlying coefficient structure. The previously observed OpenJDK JRE version 13 anomaly disappeared, several workload and microarchitectural predictors exhibited coefficient sign reversals, and lower-level hardware execution characteristics contributed markedly less additional explanatory power. Sensitivity analysis and bootstrap validation indicated that the overall explanatory structure remained reliable, although several secondary runtime configuration effects exhibited greater inferential sensitivity than under interpretive execution.

7.9 Energy Consumption Cross-Model Comparative Analysis

The preceding regression analyses examined the pooled, interpretive execution, and tiered execution energy consumption models independently. While these individual analyses establish the explanatory structure associated with each execution context, they do not directly quantify how these structures differ across model specifications. To facilitate this comparison, the following section examines the three final regression models from two complementary perspectives. First, comparative variance decomposition evaluates the relative explanatory contribution associated with each hierarchical predictor block across the three models. Second, comparative coefficient restructuring examines changes in individual regression coefficients to

assess the extent to which execution strategy alters the underlying statistical relationships governing observed JRE JVM energy consumption. Together, these complementary analyses provide a more comprehensive understanding of how execution mode moderates both overall explanatory performance and individual predictor relationships.

7.9.1 Comparative Variance Decomposition Analysis

To facilitate direct comparison across execution contexts, Table 7.32 summarises the relative variance contributions associated with each hierarchical predictor block across the pooled, interpretive execution, and tiered execution energy consumption regression models.

Table 7.32: Comparative variance contributions across hierarchical energy consumption regression models

Model	**Runtime**	**Exec./GC**	**Workload**	**Microarch.**	**Adj.** R^2
Pooled	0.002	0.622	0.238	0.013	0.874
Interpretive	0.003	0.000	0.605	0.100	0.705
Tiered	0.003	0.000	0.709	0.026	0.736

A clear pattern emerges when comparing the variance structures across the three models. In contrast to the earlier power use models, runtime implementation characteristics contributed negligibly to explaining observed variation in energy consumption across all three specifications. The runtime implementation block, comprising OpenJDK JRE source version and GNU GCC build toolchain compiler version, accounted for less than 0.3% of variance in each model. This suggests that, unlike instantaneous power demand, variation associated with underlying runtime implementation characteristics contributes comparatively little to explaining cumulative energy consumption once execution duration effects are considered through the log-transformed energy consumption specification.

The pooled energy consumption model exhibited a markedly different variance structure relative to both execution-mode-specific models. Within the pooled

model, the second hierarchical block accounted for the largest proportion of explained variance ($\Delta R^2 = 0.622$). Because this block contains both execution mode and garbage collector variant, and earlier analyses demonstrated comparatively weak garbage collector effects, this substantial variance contribution strongly suggests that differences between interpretive and tiered execution represented the dominant determinant of overall energy consumption variation within the aggregated pooled specification. This finding is consistent with earlier results demonstrating substantial reductions in energy consumption associated with tiered execution relative to interpretive execution.

Once execution mode heterogeneity is removed through execution-mode-specific decomposition, workload category emerges as the dominant explanatory factor. Within the interpretive execution model, workload category accounted for approximately 60.5% of explained variance, while within the tiered execution model this contribution increased further to approximately 70.9%. These findings indicate that once execution strategy is held constant, the computational demands imposed by workload category become the principal determinant governing observed variation in cumulative energy consumption.

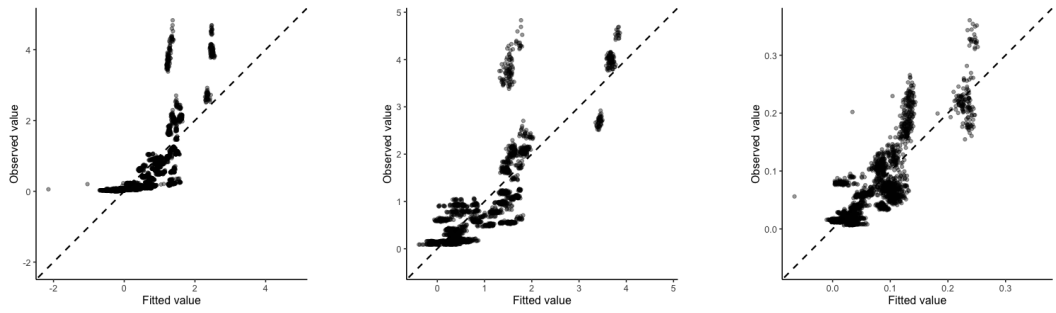
Microarchitectural performance event measures contributed comparatively differently across the three model specifications. Within the interpretive execution model, the microarchitectural predictor block accounted for an additional 10.0% of explained variance, representing a substantial secondary contribution beyond workload category. However, under tiered execution this contribution declined sharply to only 2.6%. This suggests that lower-level microarchitectural variation exerts substantially greater explanatory influence under interpretive execution, whereas tiered execution substantially reduces the extent to which lower-level hardware execution characteristics contribute independently to explaining energy consumption variation.

Taken together, these comparative results reveal a fundamentally different explanatory structure for energy consumption relative to the earlier power use

models. Whereas power use exhibited a more distributed multi-layer dependency involving runtime configuration, workload category, and lower-level microarchitectural behaviour, cumulative energy consumption appears primarily governed by execution strategy and workload category. More broadly, these findings provide further evidence supporting the emerging interpretation developed throughout this chapter that increasingly granular decomposition of execution context systematically reduces structural heterogeneity while revealing distinct underlying explanatory mechanisms governing JRE JVM energy consumption.

7.9.2 Observed-versus-Fitted Model Correspondence

The comparative variance decomposition presented above demonstrates that the explanatory structure governing energy consumption differs substantially between the pooled and execution-mode-specific model specifications. To provide a direct visual assessment of overall model correspondence, Figure 7.11 compares the observed log-transformed energy consumption values with those fitted by the final pooled, interpretive, and tiered execution regression models. For each observation, the horizontal axis represents the value fitted by the corresponding regression model, while the vertical axis represents the experimentally observed value on the same log-transformed energy consumption scale. The dashed identity line, $y = x$, represents perfect correspondence between observed and fitted values.



(a) Pooled **(b) Interpretive** **(c) Tiered**

Figure 7.11: Observed-versus-fitted correspondence for the final log-transformed energy consumption regression models: (a) pooled observations, (b) interpretive execution, and (c) tiered execution. The dashed identity line represents perfect correspondence between observed and fitted ln(energyWh) values.

The pooled energy consumption model, Figure 7.11(a), exhibits strong overall correspondence between observed and fitted values, consistent with the comparatively high explanatory performance of the final pooled log-transformed model ($R^2 = .874$, adjusted $R^2 = .874$). Although structured clustering remains visible within the observed values, the observations exhibit a clear alignment with the identity line across the fitted range. Unlike the corresponding pooled power use model, the strong explanatory performance of the pooled energy model reflects, in substantial part, the systematic distinction between interpretive and tiered execution, with execution mode forming part of the pooled model specification and accounting for a large proportion of observed energy consumption variance.

When the observations are decomposed according to execution mode, the interpretive execution model, Figure 7.11(b), continues to exhibit a clear relationship between observed and fitted log-transformed energy consumption, although greater dispersion around the identity line is evident than within the pooled specification. This is consistent with the final interpretive model accounting for approximately 70.8% of observed variance ($R^2 = .708$, adjusted $R^2 = .705$). The reduction in overall explained variance relative to the pooled model should not be interpreted as evidence that the interpretive specification is less appropriate. Rather, execution-mode decomposition removes the substantial between-mode variation that accounts for much of the explanatory performance of the pooled energy model, thereby allowing the statistical structure operating specifically within interpretive execution to be examined.

The tiered execution model, Figure 7.11(c), similarly exhibits clear correspondence between fitted and observed log-transformed energy consumption. Its explanatory performance ($R^2 = .738$, adjusted $R^2 = .736$) is moderately greater than that observed for the interpretive execution model. Consistent with the preceding variance decomposition, this model is dominated principally by workload category, which accounted for approximately 70.9% of observed variance when introduced into the hierarchical specification, compared with approximately 60.5%

under interpretive execution.

The visual comparison therefore reveals a different cross-model pattern from that observed for power use. For power use, execution-mode decomposition increased overall explained variance relative to the pooled specification. For energy consumption, the pooled model instead retains the strongest overall explanatory performance because differences between interpretive and tiered execution themselves account for a substantial proportion of cumulative energy consumption variation. Once this between-mode variation is removed, the execution-mode-specific models expose the explanatory structure operating within each execution context, with workload category becoming the dominant source of explained variance.

Importantly, these plots represent *in-sample model correspondence* rather than out-of-sample predictive accuracy. Their purpose is to provide a visual complement to the reported R^2 statistics and residual diagnostic analyses by illustrating the extent to which each fitted model reproduces the observed structure of log-transformed energy consumption. Having established these differences in overall explanatory structure and model correspondence, the following analysis examines whether the individual coefficient relationships are similarly restructured across the pooled, interpretive, and tiered model specifications.

7.9.3 Comparative Coefficient Restructuring Analysis

While the preceding comparative variance decomposition analysis examined differences in overall explanatory structure across the pooled and execution-mode-specific energy consumption regression models, coefficient-level comparisons provide additional insight into the extent to which execution strategy alters the underlying structural relationships governing observed JRE JVM energy consumption. Table 7.33 summarises selected coefficients from the final pooled, interpretive execution, and tiered execution models, together with relative coefficient magnitude ratios quantifying the extent to which predictor relationships are amplified,

attenuated, or structurally reconfigured across the model specifications.

Table 7.33: Comparative coefficient restructuring across final energy consumption regression models

Predictor	**Pooled**	**INT**	**Tiered**	**INT/Pooled**	**T./Pooled**	**INT/T.**
JRE 13	0.259	0.461	-0.023	1.78	0.09[a]	20.04[a]
Concurrency	-0.010	0.413	-0.482	41.30[a]	48.20[a]	0.86[a]
Database	0.975	1.232	0.768	1.26	0.79	1.60
Functional	-1.584	-1.242	-1.610	0.78	1.02	0.77
Web	-1.176	-0.776	-0.300	0.66	0.26	2.59
BMR	0.038	0.094	-0.138	2.47[a]	3.63[a]	0.68[a]
L1DLMR	0.068	-0.036	0.146	0.53[a]	2.15	0.25[a]
PFMR	-1.043	-4.183	-0.509	4.01	0.49	8.22

[a] Coefficient sign reversal observed, absolute magnitudes used.
Note: INT/Pooled = Interpretive relative to Pooled; T./Pooled = Tiered relative to Pooled; INT/T.= Interpretive relative to Tiered;

Several substantial structural differences emerge when comparing coefficient magnitudes across the three final energy consumption models. Unlike the earlier power use models, several predictors exhibited not only large coefficient magnitude differences but, in several cases, complete coefficient sign reversals across the different model specifications. This suggests that execution strategy may alter not merely the strength of predictor relationships, but also the underlying direction of the observed statistical relationships governing cumulative energy consumption.

One particularly notable result concerns OpenJDK JRE version 13. Within the pooled model, JRE version 13 remained positively associated with energy consumption ($\beta = 0.259$), while under interpretive execution this relationship strengthened substantially ($\beta = 0.461$), representing an approximate 1.78-fold increase in coefficient magnitude. However, under tiered execution the coefficient collapsed almost entirely ($\beta = -0.023$), producing an effective sign reversal and suggesting that the elevated energy consumption behaviour previously associated with JRE version 13 becomes largely absent once execution occurs under tiered compilation conditions.

Workload-specific relationships also exhibited substantial execution-dependent restructuring. The `database` workload coefficient remained comparatively stable across all three models, with only modest variation in coefficient magnitude,

suggesting relatively stable workload effects regardless of execution strategy. In contrast, the `concurrency` workload category exhibited one of the strongest structural changes observed within the comparative analysis. While the pooled model produced a near-zero negative coefficient ($\beta = -0.010$), interpretive execution produced a substantial positive coefficient ($\beta = 0.413$), whereas tiered execution produced a comparatively large negative coefficient ($\beta = -0.482$). This sign reversal pattern suggests that execution strategy fundamentally alters the relationship between workload-specific computational behaviour and observed cumulative energy consumption.

Several lower-level microarchitectural predictors exhibited similarly pronounced restructuring effects. Page fault rate again demonstrated particularly strong execution-dependent variation. Relative to the pooled model, interpretive execution increased the magnitude of the page fault rate coefficient by approximately 4.01 times, while under tiered execution the coefficient magnitude declined substantially, representing approximately half the magnitude observed within the pooled specification. Direct comparison between the three model specifications indicates that the page fault rate coefficient under interpretive execution was approximately 8.22 times larger than under tiered execution. This closely mirrors the pattern observed previously within the power use models, providing further evidence that interpretive execution exposes substantially stronger lower-level hardware relationships.

Branch misprediction rate and L1 data cache load miss rate produced particularly interesting results. Branch misprediction rate remained positively associated with energy consumption within both the pooled and interpretive execution models, but exhibited a coefficient sign reversal under tiered execution, becoming negatively associated with observed energy consumption. L1 data cache load miss rate similarly exhibited coefficient sign reversal, transitioning from a positive coefficient within the pooled model to a negative coefficient under interpretive execution before returning to a substantially larger positive coefficient under

tiered execution. These repeated sign reversals suggest that execution strategy may substantially alter the extent to which specific lower-level hardware execution characteristics contribute independently to cumulative energy consumption behaviour.

Taken together, these comparative results provide strong evidence that execution strategy substantially conditions the structural relationships governing observed JRE JVM energy consumption. Relative to the earlier power use models, the energy consumption models exhibited even greater coefficient restructuring across the three model specifications, with several predictors demonstrating substantial magnitude changes together with outright reversals in coefficient direction. More broadly, these findings reinforce the emerging interpretation developed throughout this chapter that execution strategy influences not only overall explanatory performance, but also fundamentally alters the statistical structure through which software configuration, workload category, and lower-level microarchitectural behaviour combine to govern cumulative energy consumption.

7.10 How this Chapter Answers RQ3

This chapter addressed Research Question **RQ3**, which asked:

> **RQ3:** ***To what extent can variation in JRE/JVM power use and energy consumption be accounted for through regression modelling incorporating JRE source version, GNU GCC build toolchain compiler version, JVM execution mode, garbage collector variant, workload category, and microarchitectural performance event measures?***

To address this question, hierarchical multiple regression models were developed separately for power use and energy consumption using pooled observations and execution-mode-specific datasets. Model reliability was subsequently evaluated through regression diagnostics, sensitivity analyses, and stratified bootstrap

validation.

Collectively, the results demonstrate that the proposed regression framework successfully accounted for substantial variation in both JRE JVM power use and energy consumption. However, the explanatory structures governing the two outcome variables differed markedly.

For power use, the pooled regression model explained approximately 48.8% of observed variation, increasing to 62.0% under interpretive execution and 65.0% under tiered execution following decomposition by execution mode. Across all three models, workload category consistently represented the dominant directly specified predictor group, while microarchitectural performance event measures contributed additional explanatory power beyond the higher-level runtime configuration variables. Restricting analysis to individual execution modes substantially improved explanatory performance, indicating that execution-mode-specific modelling reduced structural heterogeneity present within the aggregated pooled dataset.

For energy consumption, substantially stronger overall explanatory performance was achieved. The pooled log-transformed energy consumption model explained approximately 87.4% of observed variation, while the execution-mode-specific models explained 70.8% and 73.8% of variation under interpretive and tiered execution respectively. Within the pooled specification, execution mode represented the dominant source of explained variance, whereas workload category became the principal explanatory factor once execution mode was held constant. Compared with the power use models, energy consumption exhibited greater statistical regularity, improved model fit, and more stable residual behaviour, supporting the suitability of the log-transformed specification for modelling cumulative energy consumption.

Cross-model comparative analysis demonstrated that execution strategy fundamentally altered both the explanatory structure and coefficient relationships governing observed JRE JVM power use and energy consumption. Variance

decomposition showed that interpretive execution consistently exhibited greater sensitivity to lower-level microarchitectural performance event measures, whereas tiered execution placed greater explanatory emphasis on higher-level runtime and workload category.

Coefficient restructuring analysis further demonstrated that several predictor relationships changed substantially across the pooled and execution-mode-specific contexts, including complete coefficient sign reversals for selected workload and microarchitectural predictors within the energy consumption models. These findings indicate that execution strategy influences not only overall explanatory performance but also the statistical relationships through which runtime configuration, workload category, and lower-level emergent hardware execution behaviour combine to determine observed outcomes.

Regression diagnostics, sensitivity analyses, and bootstrap validation collectively supported the reliability of the fitted regression models. Although evidence of heteroskedasticity and influential observations was identified within several pooled specifications, HC3 robust standard errors, sensitivity analyses, and stratified bootstrap resampling demonstrated that the principal coefficient structures remained broadly stable.

Together, these findings indicate that the observed relationships represent robust characteristics of the experimental data rather than artefacts of individual observations or particular sample configurations.

Chapter Research Question Findings

RQ3: Hierarchical regression modelling incorporating runtime configuration factors, workload category, and microarchitectural performance event measures successfully accounted for substantial variation in both JRE JVM power use and energy consumption. Workload category consistently represented the dominant directly specified predictor group, while execution mode emerged as the principal determinant of pooled energy consumption. Decomposition by execution mode improved model performance and revealed distinct explanatory structures for interpretive and tiered execution.

Cross-model comparative analysis demonstrated that execution strategy fundamentally moderates the statistical relationships governing both outcome variables. Interpretive execution exhibited substantially greater sensitivity to lower-level microarchitectural performance event measures, whereas tiered execution attenuated these relationships and altered the coefficient structure governing observed power use and energy consumption. Regression diagnostics, sensitivity analyses, and bootstrap validation further demonstrated that the principal explanatory relationships remained robust despite the heterogeneous experimental design.

Chapter 8

Discussion

This chapter synthesises and interprets the empirical findings presented in Chapters 5, 6, and 7. Rather than reiterating individual statistical results, the discussion integrates evidence across the three research questions to examine the broader software–hardware relationships emerging from the study and to establish how these findings support the contributions to knowledge identified in Chapter 1.

The discussion is organised thematically rather than according to the sequential structure of the research questions. This approach enables findings from the independent analyses, workload-conditioned interaction analyses, and hierarchical regression models to be considered collectively where they address common explanatory themes. Particular attention is given to the uneven influence of runtime implementation characteristics, the conditioning role of workload, the restructuring associated with JVM execution mode, and the relationship between configured software characteristics, emergent microarchitectural behaviour, and observed JRE/JVM power use and energy consumption.

Throughout the thematic discussion, the principal empirical findings are also compared directly with related published results identified in Chapter 3, enabling areas of convergence, divergence, and extension to be made explicit and providing an external context within which the findings of the present study can be evaluated.

The chapter begins by revisiting the research questions and establishing the relationship between the empirical analyses, the thematic structure of the discussion, and the contributions arising from the research. It then considers the

extent to which runtime configuration effects are consistently expressed before examining workload category as a conditioning context and JVM execution mode as an important context within which software–hardware relationships become differently observable. The discussion subsequently considers the analytical role of microarchitectural performance events and uses the distinctive OpenJDK 13 observations as a multi-level example of how electrical, workload-conditioned, microarchitectural, and multivariable evidence can be synthesised without assuming an underlying causal mechanism.

The later thematic sections draw these empirical findings together through the hierarchical modelling framework, examining how the introduction of progressively deeper analytical layers changes both explained variance and the relationships attributed to previously represented factors. Power use and energy consumption are then considered comparatively to establish the distinct explanatory and statistical structures associated with the two electrical outcomes. Throughout these sections, evidence from the three research questions is synthesised at the point at which each corresponding contribution to knowledge becomes established, providing explicit traceability between the empirical findings, their interpretation within the discussion, and the contribution claims presented earlier in the thesis.

The chapter concludes by considering the practical implications arising from these findings, before critically examining the limitations of the study and their implications for the interpretation and generalisability of the results. Finally, directions for future research are identified, including opportunities to extend the experimental and analytical framework developed in this thesis and to investigate the software–hardware relationships identified through the empirical analyses in greater depth.

8.1 Revisiting the Research Questions

The overarching objective of this thesis was to investigate how software implementation decisions, runtime execution characteristics, workload composition,

and emergent microarchitectural performance measures collectively influence JRE/JVM power use and energy consumption on contemporary ARM-based embedded computing platforms. Three research questions guided this investigation.

Research Question 1 examined the independent effects associated with JRE source version, GNU GCC build toolchain compiler version, JVM execution mode, garbage collector variant, workload category, and selected microarchitectural performance event measures. **Research Question 2** extended this analysis by examining whether these software configuration effects remained consistent across different workload classes or were conditioned by application characteristics. Finally, **Research Question 3** investigated the extent to which these factors could be integrated within hierarchical regression models capable of explaining variation in both power use and energy consumption.

Collectively, the empirical findings demonstrate that software configuration exerts a measurable influence on both power use and energy consumption. However, these influences are neither uniform nor independent. Instead, the magnitude and direction of observed effects depend upon interactions between runtime implementation, workload category, execution mode, and the underlying hardware architecture. Furthermore, the hierarchical modelling analyses demonstrate that the relative explanatory importance of these factors differs substantially between average instantaneous power use and cumulative energy consumption, indicating that the two electrical outcomes exhibit distinct explanatory and statistical structures despite their underlying physical relationship.

The following sections discuss these findings thematically, synthesising evidence across all three research questions to develop a broader interpretation of the software–hardware relationships identified throughout this study. Rather than revisiting the research questions sequentially, the discussion progresses across the principal analytical themes emerging from their combined findings: the uneven influence of runtime configuration, the conditioning role of workload category, the restructuring associated with JVM execution mode, the analytical role of

emergent microarchitectural behaviour, the distinctive OpenJDK 13 characteristic, the context-dependent relationships revealed through hierarchical modelling, and the differing explanatory structures governing power use and energy consumption.

This thematic progression also provides the basis for synthesising the contributions to knowledge identified in Chapter 1, §1.5. Evidence relevant to individual contributions frequently extends across more than one research question and analytical level; consequently, the contribution claims are developed through the convergence of evidence from the independent, interaction, and multivariable analyses rather than being associated exclusively with individual statistical tests. Where each contribution reaches its principal point of synthesis within the discussion, its relationship to the preceding empirical evidence is explicitly identified. In this way, the chapter progresses from interpretation of the individual findings towards the broader empirical, analytical, and methodological contributions arising from the research.

8.2 Runtime Configuration Effects Are Real but Highly Uneven

The principal conclusion arising from the investigation of the main effects (Research Question 1) is that software runtime configuration exerts a genuine and measurable influence on both JRE/JVM power use and energy consumption. Across the empirical analyses presented in Chapters 5, 6, and 7, statistically significant differences were repeatedly observed between alternative runtime configurations, confirming that implementation decisions made during runtime construction and execution influence the power use and energy consumption characteristics of software executing on an embedded ARM platform. This finding is broadly consistent with previous research demonstrating that JVM implementation decisions influence power use and energy consumption through factors including interpretive execution, just-in-time compilation, garbage collection strategy, JVM version, and

runtime configuration. However, as discussed in Chapter 3, much of the existing literature has considered these factors in isolation, with the majority of studies concentrating on execution time performance or cumulative energy consumption rather than simultaneously considering instantaneous power use, cumulative energy consumption, workload category, execution mode, and microarchitectural performance measures [99, 105, 112, 113].

The principal insight emerging from these analyses, however, is that the influence of runtime configuration is far from uniform. Rather than contributing equally to observed variation, the individual runtime configuration dimensions examined throughout this study exhibited markedly different levels of influence. OpenJDK source version produced substantial differences in power use, driven primarily by the distinctive characteristics associated with OpenJDK version 13 (Chapter 5, §5.2), whereas GNU GCC build toolchain compiler version generally exhibited comparatively modest effects that were often restricted to particular execution contexts (Chapter 5, §5.3). Similarly, execution mode produced some of the largest differences observed throughout the study, particularly in relation to energy consumption, while garbage collector variant exerted a comparatively smaller influence (Chapter 5, §5.4). Consequently, the empirical evidence does not support the view that all software configuration choices contribute equally to power use and energy consumption. Instead, their relative importance depends upon both the runtime context in which they are evaluated and the outcome measure under consideration.

Among the runtime implementation variables investigated, OpenJDK source version produced the most distinctive version-specific effect, driven overwhelmingly by the characteristics associated with OpenJDK 13. These characteristics were observed across multiple stages of the investigation, including the non-parametric analyses, workload interaction analyses, hierarchical regression models, and microarchitectural performance event analyses. Previous studies examining JVM version effects have similarly reported measurable differences between successive JVM

releases, although their conclusions have not been entirely consistent. For example, Ournani et al. [112] reported that newer JVM distributions generally exhibited improved energy characteristics across successive releases, whereas Shimchenko et al. [113] identified contrasting results between different OpenJDK versions and attributed those differences to implementation changes affecting compiler optimisation strategies. These inconsistencies suggest that JVM evolution does not necessarily produce monotonic improvements in power use or energy consumption. The present findings reinforce this interpretation. Rather than observing gradual changes across successive OpenJDK releases, the empirical analysis identified a highly localised implementation-specific effect centred on OpenJDK 13, while the remaining versions exhibited comparatively stable characteristics. Furthermore, the recurrence of this effect across descriptive analyses, workload interaction analyses, hierarchical regression models, and microarchitectural performance measures strengthens the conclusion that the observed differences reflect a genuine implementation-specific characteristic rather than random experimental variation. The broader implications of this finding, together with its potential relationship to changes introduced within the OpenJDK implementation, are considered later in Section 8.6.

By comparison, the influence associated with GNU GCC build toolchain compiler version was considerably weaker than might be anticipated from the broader compiler optimisation literature. Previous research has demonstrated that compiler selection and optimisation strategy can influence execution characteristics across large software systems. Studies examining Geant4, Linux kernel builds, and embedded applications have shown measurable performance improvements arising from newer GCC releases and alternative optimisation strategies [118, 30, 122, 123, 124]. However, these studies have predominantly focused on execution time differences rather than the power use or energy consumption of JRE and JVM binaries themselves. Although isolated statistically significant differences were identified under particular execution conditions within the present study

(Chapter 5, §§§5.3.1), especially during interpretive execution, compiler version rarely emerged as a dominant explanatory factor. This suggests that, once a functioning JVM has been constructed, differences attributable solely to the compiler toolchain contribute comparatively little to observed power use and energy consumption relative to subsequent runtime implementation and execution decisions. Consequently, the present findings extend previous performance-oriented compiler research by demonstrating that improvements in compiler generation do not necessarily translate into proportionally large differences in runtime power use or energy consumption.

Similarly, garbage collector variant demonstrated measurable but generally modest effects. Significant differences in power use were identified for particular collector implementations, whereas energy consumption proved substantially less sensitive to garbage collector variant. This observation broadly aligns with the evolution of the garbage collection literature. Historically, garbage collector research has concentrated primarily on throughput, latency, and memory footprint, with comparatively limited attention directed towards power use and energy consumption [144, 31, 50]. More recent investigations have demonstrated that garbage collector selection can influence energy consumption, although those effects have generally been shown to depend strongly upon workload category, heap configuration, and JVM implementation [112, 113]. One plausible interpretation of the present findings concerns differences in collector execution strategy, particularly the extent to which collection work is performed serially, in parallel, or concurrently. Collectors employing greater levels of parallelism or concurrent execution may increase instantaneous processor activity during garbage collection, potentially contributing to the elevated power use observed for configurations such as CPGC and G1GC relative to the baseline CGC collector. Conversely, the lower power use associated with SGC is consistent with its simpler serial collection strategy. However, because the present study did not directly measure garbage collection activity or thread utilisation, this interpretation should be regarded as

a plausible architectural explanation rather than a mechanism directly established by the empirical measurements. More importantly, the findings reinforce previous observations that garbage collector variants cannot be interpreted independently of the workload and runtime environment within which they execute.

The comparatively modest garbage collector effects observed here should also be interpreted against earlier studies that attributed a considerably larger proportion of energy consumption to garbage collection activity itself. Contreras et al. [146], for example, estimated that garbage collection accounted for approximately 37% of total energy consumption for SPECjvm98 workloads and approximately 32% for DaCapo workloads. These estimates are not directly comparable with the present analysis, since Contreras et al. instrumented individual JVM services whereas the present study measures system-level electrical outcomes associated with alternative collector configurations. Nevertheless, both studies establish that memory-management strategy can contribute meaningfully to JVM energy behaviour.

More directly comparable are the findings of Ournani et al. [112], who demonstrated that the electrical consequences of garbage collector selection were workload dependent and that the default G1 collector was not uniformly associated with the lowest energy consumption, with Serial and Parallel collectors producing more favourable outcomes for several workloads. The present findings provide partial convergence with this result: SGC was repeatedly associated with comparatively lower power use, while the magnitude and statistical detectability of garbage collector relationships varied substantially according to workload category and execution mode. However, the absence of a strong corresponding pooled energy-consumption effect in the present study further demonstrates that collector-related differences in instantaneous power demand do not necessarily translate directly into equivalent differences in cumulative energy consumption.

Execution mode emerged as one of the most consequential dimensions investigated, although its influence differed substantially between power use and

energy consumption. The distinction between interpretive and tiered execution consistently produced substantial differences across the descriptive analyses and hierarchical regression models, with particularly pronounced differences observed in relation to cumulative energy consumption. This finding is broadly consistent with more than three decades of JVM research demonstrating the fundamental differences between interpretive execution and just-in-time compilation [13, 143, 59]. However, previous work has overwhelmingly concentrated on execution time as a measure of performance, compiler optimisation, and interpreter implementation, with comparatively little attention directed towards power use and energy consumption under modern JVM implementations [105, 29]. The present findings extend this literature by demonstrating that execution mode influences considerably more than execution speed. As shown throughout the subsequent analyses, decomposing observations according to execution mode substantially alters the magnitude and explanatory importance of several microarchitectural performance measures, indicating that interpretive and tiered execution establish markedly different contexts for the expression of software–hardware relationships. Consequently, execution mode represents considerably more than a simple runtime option; it defines an important operational context within which software implementation is translated into observable hardware activity.

Collectively, these findings provide a direct answer to Research Question 1.

Research Question 1: Discussion Synthesis

Software runtime configuration exerts a measurable influence on both power use and energy consumption, but the individual configuration dimensions contribute at markedly different levels of explanatory importance. OpenJDK implementation and execution mode generally exhibited more substantial relationships than GNU GCC build toolchain compiler version or garbage collector variant, while workload category emerged as an additional and particularly influential determinant of both outcomes. Runtime configuration should therefore not be regarded as a collection of equally influential implementation choices, nor should the apparent importance of individual configuration dimensions be assumed to remain constant across differing execution conditions.

More fundamentally, when the main effects, workload-conditioned analyses, and multivariate models are considered collectively, the apparent influence of individual runtime configuration dimensions cannot be regarded as a fixed property of a particular software configuration. Rather, the magnitude and, in some cases, direction of these relationships depend upon the broader software–hardware context within which they are observed. This interpretation becomes increasingly important throughout the remainder of this chapter, where workload category, execution mode, and microarchitectural performance measures are shown to progressively reshape the apparent influence of software implementation decisions. Ultimately, the combined evidence indicates that JRE/JVM power use and energy consumption emerge through relationships spanning multiple layers of the software and hardware stack rather than through isolated runtime components considered independently.

8.3 Workload Category Condition Runtime Relationships

Building upon the main effects identified in Chapter 5, the interaction analyses undertaken to address Research Question 2 demonstrate that the influence of runtime configuration cannot be understood independently of the workload being executed. Across the main effects analyses (Chapter 5), interaction analyses (Chapter 6), and hierarchical regression models (Chapter 7), workload category repeatedly emerged as one of the principal sources of variation in both power use and energy consumption. More importantly, workload did not simply contribute additional explanatory power alongside the runtime configuration variables. Instead, it conditioned the extent to which differences associated with alternative JRE implementations, garbage collector variants, and, to a lesser extent, GNU GCC build toolchain compiler versions were expressed. These findings suggest that workload should not be regarded merely as another explanatory variable within the experimental design. Rather, it defines the computational context within which software implementation decisions influence power use and energy consumption.

This interpretation is broadly consistent with an increasing body of literature recognising workload as an important determinant of JVM power use and energy consumption. Previous studies have demonstrated workload-dependent differences across Java I/O libraries, concurrent collections, garbage collector variants, JVM distributions, and design patterns, collectively indicating that implementation characteristics can vary substantially across application domains [109, 111, 112, 113, 114]. However, much of this literature has considered workload primarily as a moderating factor when evaluating individual runtime components. Consequently, relatively little attention has been directed towards understanding workload as a systems-level influence capable of conditioning multiple layers of runtime implementation simultaneously.

Several of these published findings correspond particularly closely with the workload-conditioning effects observed here. Ournani et al. [112] found that the relative energy characteristics associated with garbage collector selection changed according to the executing workload, while Shimchenko et al. [113] similarly identified workload as a moderating factor in the relationship between execution performance and energy consumption. Lambert et al. [29] further demonstrated, using the Renaissance Benchmark Suite, that workload type represented the principal differentiator of the execution-time gap between interpretive and tiered execution. The present findings therefore reinforce an observation appearing across several independent studies: relationships attributed to JVM configuration or execution strategy cannot necessarily be separated from the computational characteristics of the workload through which those runtime mechanisms are exercised. The contribution of the present study is to demonstrate that this conditioning extends simultaneously across multiple runtime configuration factors and differs between power use and energy consumption.

The present findings both reinforce and extend these observations. Workload category accounted for a substantial source of variation throughout the empirical analyses and contributed considerable explanatory power within the regression models developed for both power use and energy consumption. However, the importance of workload extended beyond differences in average power use or average energy consumption between application classes. Equivalent runtime configuration choices produced markedly different patterns of power use and energy consumption across workload category levels, with the interaction analyses demonstrating that the magnitude and statistical detectability of runtime configuration effects varied according to workload category (Chapter 6), §6.5.

The interaction analyses undertaken to determine whether workload conditions runtime configuration effects provide particularly compelling evidence for this interpretation. The strongest and most systematic workload-dependent relationships were associated with OpenJDK JRE source version. For power use, OpenJDK 13

consistently exhibited elevated values across all workload category levels, but the magnitude of this increase varied substantially between workload categories. The effect was more pronounced within the apache, database, functional, and scala categories and comparatively smaller within concurrency and web workloads. For energy consumption, the relationship was more selective, with statistically significant JRE version effects concentrated within particular workload categories, especially database and web. Workload therefore conditioned not only the magnitude of the OpenJDK 13 effect but also, particularly for energy consumption, whether JRE version differences were statistically detectable.

The remaining runtime configuration factors exhibited more selective workload-dependent relationships. GNU GCC build toolchain compiler version demonstrated comparatively limited moderation, with workload-dependent power use differences concentrated primarily within the concurrency and web categories and little substantive evidence of corresponding effects for energy consumption. garbage collector variant exhibited a different pattern. Workload-dependent garbage collector effects were more clearly evident for power use and became particularly pronounced under tiered execution, whereas interpretive execution largely suppressed collector-specific differences. Energy consumption remained less sensitive to garbage collector variant, although some workload-dependent differences became visible following decomposition by execution mode. Consequently, workload conditioning was neither equally strong across runtime configuration dimensions nor equivalent between power use and energy consumption.

This distinction is important because it demonstrates that workload dependency should itself not be treated as a uniform phenomenon. The interaction analyses do not indicate simply that "workload matters". Rather, they show that the extent to which workload conditions a runtime relationship depends upon the implementation factor being examined, the outcome measure under consideration, and the execution mode within which that relationship is observed. OpenJDK JRE version exhibited the broadest workload-dependent relationships, particularly

for power use; GNU GCC build toolchain compiler version exhibited comparatively limited workload dependence; and garbage collector effects were strongly influenced by the distinction between interpretive and tiered execution. Workload therefore forms one component of a broader conditional structure through which runtime implementation characteristics become observable.

From a systems perspective, these findings are consistent with the fundamentally different computational demands imposed by different application classes. Compute-intensive workloads, memory-intensive workloads, concurrent applications, and web-oriented applications exercise different combinations of instruction execution, memory hierarchy utilisation, branch prediction, object allocation, and garbage collection. Consequently, equivalent runtime implementations operate under substantially different computational conditions depending upon the workload presented to the virtual machine. The present findings therefore suggest that differences in workload alter not only overall power use and energy consumption but also the extent to which these outcomes are sensitive to changes in runtime configuration.

This interpretation also provides a useful perspective from which to reconsider previous studies examining individual runtime components. Much of the existing literature has necessarily focused on evaluating isolated factors, including JVM version, garbage collector selection, compiler configuration, execution mode, or application-level implementation decisions. While these studies provide valuable evidence concerning the effects associated with specific runtime choices, the present findings suggest that the magnitude of those reported effects should be interpreted within the context of the workloads on which they were evaluated. It is therefore plausible that some relationships reported within the existing literature reflect, at least in part, characteristics of the workloads included within those experimental designs rather than universally applicable properties of the runtime component itself. Establishing the generality of previously reported effects therefore requires consideration of a sufficiently diverse range of application domains.

Collectively, these findings provide a direct answer to Research Question 2.

Research Question 2: Discussion Synthesis

The effects associated with runtime configuration are not uniformly stable across workload categories. Workload category conditioning was strongest and most systematic for OpenJDK JRE source version, particularly in relation to power use, while GNU GCC build toolchain compiler version exhibited comparatively limited workload-dependent effects and garbage collector relationships were additionally conditioned by execution mode. For energy consumption, workload-dependent runtime effects were generally more selective and more strongly influenced by execution mode than those observed for power use. Workload should therefore be understood as the computational context within which runtime configuration effects are expressed rather than simply as an additional independent determinant of power use and energy consumption.

More broadly, the findings provide little basis for assuming that a single runtime configuration will exhibit equivalent power use or energy consumption characteristics across all Java workload classes. The consequences associated with implementation choices, including JRE version, garbage collector variant, execution mode, and, to a lesser extent, build toolchain compiler version, depend upon the computational characteristics of the workload under consideration. This observation has important implications for both runtime optimisation and experimental methodology, since conclusions derived from individual benchmarks or narrowly defined application domains may not necessarily generalise to broader deployment scenarios.

While workload defines the computational demands placed upon the runtime environment, the manner in which those demands are translated into lower-level execution characteristics depends fundamentally upon the execution mode adopted by the JVM. The interaction analyses already indicate that execution mode can alter the magnitude and statistical detectability of workload-dependent runtime effects. The following section therefore examines this relationship directly, considering how interpretive and tiered execution restructure the relationships between software implementation and the underlying hardware.

Contribution to Knowledge: C2

The preceding synthesis supports **C2** by demonstrating that workload category represents more than an independent source of variation in JRE/JVM power use and energy consumption. The magnitude and statistical detectability of relationships associated with runtime configuration vary across workload category levels and, in several cases, across execution modes and electrical outcomes. Workload is therefore identified as a conditioning computational context within which runtime configuration relationships are expressed, limiting the extent to which findings obtained from individual benchmarks or narrowly defined workload domains can be generalised.

8.4 Execution Mode Restructures Software–Hardware Relationships

Synthesising the findings across the main effects analyses (Chapter 5), workload interaction analyses (Chapter 6), and hierarchical regression models (Chapter 7), one of the clearest conclusions emerging from this research is that execution mode represents considerably more than a runtime configuration option controlling how Java bytecode is executed. Rather, the collective evidence demonstrates that interpretive and tiered execution establish substantially different contexts within which relationships between software implementation, microarchitectural performance characteristics, power use, and energy consumption are expressed. Execution-mode decomposition repeatedly altered the magnitude, explanatory contribution, and, in some cases, statistical detectability of relationships observed within the pooled analyses. Consequently, execution mode should be understood as a defining characteristic of the software–hardware execution context rather than simply another implementation choice within the JVM.

This interpretation both aligns with and extends the existing literature. Previous research has extensively documented the execution time advantages associated

with modern just-in-time compilation, improvements in interpreter dispatch mechanisms, and the evolution of tiered compilation strategies [13, 54, 143, 59]. More recent work has similarly demonstrated substantial execution time differences between interpretive and tiered execution for contemporary JVM workloads [29]. However, the overwhelming emphasis of this literature has been on execution performance rather than understanding how execution strategy influences the relationships between software implementation, processor activity, power use, and energy consumption. Consequently, execution mode has generally been considered primarily as a determinant of execution performance rather than as a factor capable of restructuring the broader statistical relationships observed across the software–hardware system.

There is also earlier energy-specific evidence against which the present execution-mode findings can be compared. Vijaykrishnan et al. [103] reported substantially lower total energy consumption under JIT execution than under interpretive execution, although their analysis did not normalise the observed energy difference for the corresponding difference in execution duration. Cao et al. [105] similarly demonstrated that JVM execution services, including interpretation and just-in-time compilation, can represent substantial components of overall execution time and energy consumption. The present findings are directionally consistent with this earlier evidence in demonstrating a pronounced energy distinction between interpretive and dynamically compiled execution. However, they extend it by separating instantaneous power use from cumulative energy consumption and by showing that execution mode also restructures the relationships between runtime configuration, workload category, and lower-level microarchitectural performance event measures. In particular, the dominant contribution of execution mode to pooled energy consumption should therefore be interpreted partly in the context of the substantially different execution durations associated with the two execution strategies, rather than as evidence of an equivalent difference in instantaneous power demand.

The present findings suggest that this perspective is incomplete. Decomposing the observations according to interpretive and tiered execution revealed markedly different explanatory structures despite retaining the same underlying experimental factors. This restructuring was particularly evident within the power use regression models (Chapter 7), §7.3 and §7.4. The comparative variance decomposition demonstrated that the microarchitectural predictor block accounted for an additional 10.0% of explained variance under interpretive execution, compared with only 3.0% under tiered execution (Chapter 7), §7.5). Conversely, the initial runtime implementation block accounted for 18.9% of observed power use variance under tiered execution compared with 7.8% under interpretive execution. These results indicate that the relative explanatory importance of higher-level runtime implementation characteristics and lower-level microarchitectural performance measures changed substantially according to execution mode.

Coefficient-level comparisons provide further evidence of this restructuring (Chapter 7, §§7.5.3). The relationship associated with page fault rate (PFMR), for example, increased in magnitude from approximately -1.19 within the pooled power use model to -3.52 under interpretive execution, while declining to approximately -0.44 under tiered execution. Consequently, the interpretive PFMR coefficient was approximately eight times larger in magnitude than its tiered counterpart. A similarly pronounced restructuring was observed for L2 data cache load miss rate, whose coefficient increased from approximately -0.008 within the pooled model to -0.083 under interpretive execution, but declined to approximately -0.004 under tiered execution. These differences were therefore not confined to changes in overall model fit; execution-mode decomposition revealed substantial changes in the estimated relationships between individual microarchitectural performance measures and observed power use (Chapter 7, Section 7.5).

Importantly, these execution-mode-dependent relationships were not simply artefacts of unstable coefficient estimation. Bootstrap validation of the inter-

pretive power use model demonstrated substantial stability among the principal microarchitectural coefficients, with the PFMR estimate remaining effectively unchanged under repeated resampling and its confidence interval remaining consistently separated from zero. Branch misprediction rate and L2 data cache load miss rate similarly retained stable negative coefficient structures. This provides additional evidence that the stronger microarchitectural relationships observed under interpretive execution represent persistent characteristics of the fitted model specification rather than relationships attributable to a particular sample configuration (Chapter 7, Section 7.3).

The interaction analyses provide complementary evidence that this restructuring extends beyond the regression coefficients themselves. Execution-mode decomposition altered the magnitude and statistical detectability of workload-dependent runtime effects. OpenJDK JRE version effects varied between interpretive and tiered execution, while garbage collector-related workload effects were substantially more visible under tiered execution and were largely suppressed under interpretation. GNU GCC build toolchain compiler effects exhibited a different pattern, becoming more pronounced for power use under interpretive execution while remaining comparatively weak overall. Execution mode therefore did not affect every runtime relationship in the same direction. Rather, it changed the conditions under which particular software implementation characteristics became statistically observable.

Collectively, these findings demonstrate that execution mode changes more than the magnitude of observed power use. It changes the statistical structure through which software implementation and lower-level processor characteristics are associated with the observed outcome. Under interpretive execution, the substantially greater incremental contribution of the microarchitectural predictor block, together with the increased magnitude and stability of several lower-level coefficients, indicates greater statistical observability of microarchitectural relationships. Under tiered execution, these relationships generally contributed

less additional explanatory information, while higher-level runtime implementation characteristics accounted for a greater proportion of power use variation. Execution mode therefore appears to determine the level within the software–hardware stack at which variation becomes most statistically visible.

From a systems perspective, this pattern is consistent with the fundamental differences between interpretation and tiered compilation. During interpretive execution, application bytecode continues to be processed through the JVM interpreter, whereas tiered execution progressively compiles frequently executed code into optimised native machine code. It is therefore plausible that dynamic compilation and optimisation mediate some of the lower-level variability that remains more directly observable during interpretive execution. Under this interpretation, tiered execution does not eliminate microarchitectural activity or its relationship with power use; rather, the optimisation processes associated with dynamic compilation may alter the extent to which variation in individual microarchitectural performance measures remains independently associated with the observed outcome. The present measurements do not directly identify the optimisation mechanisms responsible for this pattern, and this interpretation should therefore be regarded as a systems-level explanation consistent with, rather than directly demonstrated by, the empirical evidence.

Perhaps the most significant implication arising from these findings concerns the statistical observability of software–hardware relationships. The substantially larger contribution of microarchitectural predictors under interpretive execution indicates that lower-level processor characteristics were more strongly expressed within the corresponding statistical models. Tiered execution, by comparison, attenuated several of these relationships. This distinction is important because an architectural characteristic may remain present within the underlying system while becoming less statistically distinguishable within an observed outcome when execution occurs through a dynamically optimising runtime. Consequently, the absence of a large statistical relationship under one execution strategy should not

necessarily be interpreted as evidence that the corresponding lower-level system characteristic is unimportant.

This observation also has methodological implications. Where the objective of an experiment is to investigate relationships between software execution and lower-level architectural characteristics, execution mode may materially influence the extent to which those relationships are statistically observable. The present findings indicate that interpretive execution provided a context in which several microarchitectural relationships with power use were more strongly expressed than under tiered execution. This does not establish that interpretive execution will necessarily provide greater sensitivity to architectural modifications, since greater variation in observed microarchitectural measures does not itself guarantee greater experimental sensitivity. However, it suggests that execution-mode decomposition may provide a more informative experimental framework than exclusive reliance upon pooled or tiered observations when investigating software–hardware relationships. Establishing whether this pattern generalises to alternative processor architectures, JVM implementations, or hardware modifications represents an important direction for future research.

Collectively, these findings provide a direct answer to the execution-mode component of Research Question 1 while also extending the workload-conditioned relationships identified through Research Question 2 and the explanatory structures established through Research Question 3.

Execution Mode: Discussion Synthesis

Execution mode is not merely another runtime configuration variable influencing power use and energy consumption. Rather, it defines an execution context within which the statistical relationships between software implementation and lower-level processor characteristics are expressed. For power use in particular, interpretive execution exposed substantially stronger microarchitectural relationships, whereas tiered execution shifted explanatory importance towards higher-level runtime implementation characteristics. The corresponding implications for energy consumption, and the extent to which its explanatory structure differs from that governing power use, are considered separately in Section 8.8.

Contribution to Knowledge: C3

The preceding synthesis supports **C3** by demonstrating that JVM execution mode alters the statistical observability of relationships across the software–hardware stack. Decomposition into interpretive and tiered execution changes variance contributions, coefficient magnitudes, statistical detectability, and, for some relationships, coefficient direction. The contribution lies in showing that execution strategy defines an important analytical context within which software–hardware relationships become differently observable, such that relationships estimated from pooled observations cannot automatically be assumed to characterise those present within the constituent execution contexts.

8.5 Microarchitectural Behaviour Reveals Hidden Runtime Characteristics

The analyses of microarchitectural performance event measures provide a deeper perspective on the runtime relationships identified throughout this study. Whereas power use and energy consumption describe the electrical consequences of execution at the system level, measures such as branch misprediction rate, L1 and L2 data

cache load miss rates, and page fault rate provide insight into the lower-level execution characteristics through which those outcomes emerge. The results demonstrate that these measures do not simply associate with power use and energy consumption. Their distributions also differed systematically according to runtime implementation and, particularly, execution mode, revealing execution characteristics that were not apparent from the electrical outcome measures alone (Chapter 5, §5.6).

This distinction is important because runtime configurations that appear relatively similar when assessed only through aggregate power use or energy consumption can nevertheless exhibit substantially different microarchitectural performance event measure magnitudes. Across the main effects analyses (Chapter 5), interpretive execution consistently produced substantially greater branch misprediction rates and moderately greater L1 data cache load miss rates, whereas tiered execution produced substantially greater L2 data cache load miss rates. The latter difference was especially pronounced, with average L2 data cache load miss rates under tiered execution approximately five times those observed under interpretive execution. These patterns demonstrate that interpretive and tiered execution do not simply differ in execution duration or electrical demand; they generate materially different processor-level execution profiles.

These execution-mode differences can be compared directly with earlier microarchitectural studies of JVM interpretation and compiled execution. Radhakrishnan et al. [162] reported consistently greater branch misprediction under interpretive execution than under JIT execution across their assessed workloads and branch-prediction mechanisms, while Li et al. [163] similarly identified substantially greater branch-misprediction behaviour for JVM interpretation. The present finding that interpretive execution exhibits markedly greater branch misprediction therefore provides contemporary ARM-based evidence consistent with this established distinction between the two execution strategies.

The corresponding cache results, however, are not uniformly consistent with

the earlier literature. Radhakrishnan et al. [162] reported higher L1 data-cache miss rates under JIT execution than under interpretation, an observation also discussed in relation to the changing data-cache composition produced by dynamic compilation. Earlier work by Hsieh et al. [18] and Hu et al. [167] similarly demonstrated that cache behaviour differs substantially between interpretive and compiled JVM execution and is influenced by the redistribution of bytecode, application data, and generated native code across the cache hierarchy. In the present study, interpretive execution instead exhibited moderately greater L1 data-cache load miss rates, while tiered execution produced substantially greater L2 data-cache load miss rates. This divergence is important rather than contradictory: the earlier studies employed substantially different JVM implementations, benchmark suites, and processor architectures, whereas the present analysis evaluates contemporary OpenJDK releases using Renaissance workloads on an ARM Cortex-A72 platform. The contrast therefore reinforces the importance of evaluating microarchitectural relationships within the specific runtime, workload, and hardware context in which they are observed.

The significance of these observations becomes clearer when the microarchitectural measures are considered in relation to power use and energy consumption. The relationships were highly heterogeneous. Branch misprediction rate and L1 data cache load miss rate generally exhibited comparatively weak or inconsistent associations with the power use and energy consumption outcomes, whereas L2 data cache load miss rate exhibited stronger execution-mode-dependent relationships and page fault rate produced some of the strongest and most consistent associations observed among the microarchitectural measures (Chapter 7, §7.2 and §7.6). Consequently, the findings do not support a simple interpretation in which greater microarchitectural event activity necessarily corresponds to greater power use or energy consumption. Instead, the relevance of an individual event depends upon both the type of event being measured and the execution context within which it occurs.

This is particularly evident when pooled relationships are compared with execution-mode-specific analyses. Several relationships that appeared substantial when interpretive and tiered observations were combined changed markedly once the execution modes were examined separately. For example, the pooled relationship between L2 data cache load miss rate and energy consumption initially suggested that greater miss activity was associated with lower energy consumption. However, decomposition by execution mode revealed that this negative pooled relationship was largely attributable to the structural separation between the two execution strategies: tiered execution simultaneously exhibited substantially greater L2 data cache load miss rates and substantially lower cumulative energy consumption. Within the tiered subset itself, the relationship between L2 data cache load miss rate and energy consumption was instead positive. This demonstrates that an apparently meaningful pooled microarchitectural relationship may represent differences between execution contexts rather than a stable within-context relationship (Chapter 5, § 7.9).

This observation has broader implications for the interpretation of hardware performance counters within managed runtime systems. Microarchitectural event measures should not necessarily be treated as independent causal determinants of power use or energy consumption. Rather, within the present study they are more appropriately understood as emergent characteristics of execution generated through the interaction of workload, runtime implementation, execution mode, and processor architecture. Their value therefore lies partly in revealing how higher-level software choices are manifested at lower levels of the system. A branch misprediction, cache miss, or page fault does not by itself identify the software mechanism responsible for the event, but systematic changes in these measures can provide evidence that apparently higher-level runtime differences are accompanied by materially different processor-level execution characteristics.

The hierarchical regression models reinforce this interpretation. Once incorporated alongside runtime configuration and workload category, microarchitectural

performance event measures contributed additional explanatory information, although the magnitude of that contribution depended strongly upon execution mode. For power use, the microarchitectural block accounted for an additional 10.0% of explained variance under interpretive execution compared with approximately 3.0% under tiered execution. Several individual coefficients also changed substantially across model specifications, particularly those associated with L2 data cache load miss rate and page fault rate. The microarchitectural measures therefore provided more than an additional set of predictors; they exposed execution-mode-dependent statistical structure that remained partially obscured within the higher-level runtime variables alone (Chapter 7, § 7.5).

A particularly important example concerns page fault rate. Although page fault activity exhibited comparatively little variation across OpenJDK JRE versions, its relationship with the electrical outcome measures was comparatively strong and remained especially pronounced under interpretive execution. Within the interpretive power use model, the page fault coefficient was substantially larger in magnitude than within either the pooled or tiered model and remained highly stable under bootstrap resampling. This contrast demonstrates an important distinction between variability in the event itself and its statistical association with the observed outcome. A microarchitectural performance event measure does not need to vary dramatically between runtime versions to contribute meaningfully to explaining variation once workload and execution context are taken into account.

The OpenJDK 13 observations provide a complementary example of the diagnostic value of microarchitectural analysis. The elevated power use associated with this JRE version was accompanied by distinctive differences in several lower-level execution measures, most clearly branch misprediction rate and L1 data cache load miss rate. OpenJDK 13 exhibited the lowest average branch misprediction rate among the assessed JRE versions and also exhibited a distinctly reduced L1 data cache load miss rate relative to the remaining implementations. These findings are particularly important because they demonstrate that the

OpenJDK 13 power use difference was accompanied by systematic changes in processor-level execution characteristics rather than appearing solely within the electrical measurements. The microarchitectural evidence therefore provides an additional basis for interpreting OpenJDK 13 as exhibiting a distinctive implementation-specific execution characteristic. The potential explanation for this pattern is considered separately in Section 8.6.

Care is nevertheless required when interpreting these relationships mechanistically. Hardware performance event measures describe characteristics of processor and memory-system activity, but they do not directly identify the software mechanisms responsible for those events. The present study did not, for example, instrument individual compiler transformations, interpreter dispatch paths, allocation events, or operating-system memory-management decisions sufficiently to establish a direct causal pathway from a specific runtime implementation change to a particular microarchitectural event and subsequently to power use or energy consumption. The observed relationships should therefore be interpreted as evidence of systematic association and execution structure rather than proof of a specific causal mechanism.

Taken collectively, the microarchitectural analyses provide an important additional layer of evidence within the broader software–hardware framework developed in this thesis. Runtime configuration and workload category determine the higher-level conditions under which execution takes place, while microarchitectural performance event measures provide observable evidence of how those conditions are expressed within the processor and memory hierarchy. Their relationships with power use and energy consumption are neither uniform nor independent of execution context, but their inclusion reveals structural differences that would remain hidden if analysis were restricted to runtime configuration and electrical outcomes alone. In this sense, microarchitectural performance measures function as an important analytical bridge between software implementation and measured system-level electrical characteristics.

Contribution to Knowledge: C4

The preceding synthesis supports **C4** by demonstrating that microarchitectural performance events provide an analytical connection between configured runtime characteristics and the processor-level behaviour emerging during execution. Differences in branch misprediction, cache miss, and page fault behaviour reveal execution characteristics that are not apparent from system-level electrical measurements alone, while their relationships vary according to workload and execution context. Their value therefore lies not simply in their use as additional statistical predictors, but in providing observable indicators through which higher-level software characteristics can be related to lower-level execution behaviour and associated electrical outcomes.

Contribution to Knowledge: C1

The preceding synthesis, across §8.2 through §8.5, supports contribution **C1** by demonstrating that JRE/JVM power use and energy consumption cannot be characterised through any single runtime or execution factor in isolation. Across runtime implementation, workload category, execution mode, and emergent microarchitectural behaviour, the findings reveal a multi-layer explanatory structure in which the relative importance and statistical expression of individual factors vary according to analytical context and electrical outcome. The contribution therefore lies in providing an integrated empirical characterisation of JRE/JVM electrical behaviour across multiple interacting levels of the managed runtime system.

8.6 Interpreting the OpenJDK 13 Behaviour

One of the most distinctive findings emerging from this research concerns the behaviour associated with OpenJDK JRE version 13. Across the main effects, workload-conditioned analyses, and hierarchical regression models, this runtime version repeatedly separated from the remaining OpenJDK releases, particularly

in relation to power use. Importantly, the consistency of this observation across several independent analytical approaches indicates that the OpenJDK 13 effect should not be regarded simply as an artefact of one statistical comparison or a small number of atypical benchmark executions. Rather, the collective evidence suggests that OpenJDK 13 exhibited a distinctive implementation-specific execution characteristic within the ARM-based experimental environment examined in this study.

The clearest manifestation of this behaviour was observed for power use. Across the pooled main-effects analysis (Chapter 5), OpenJDK 13 exhibited significantly greater median power use than every other assessed JRE version, while no statistically significant pairwise differences remained among the other JRE releases. The magnitude of the difference between OpenJDK 13 and the remaining versions was approximately 0.44 W. Moreover, the elevated power use was not accompanied by unusually high dispersion. OpenJDK 13 simultaneously exhibited the highest power use and the lowest variability across the pooled and execution-mode-specific distributions. Consequently, the observed separation cannot readily be explained by a small number of extreme measurements or by unusually high variability within the OpenJDK 13 observations. (Chapter 5, §5.2).

The OpenJDK 13 observation can also be considered in relation to previously reported JVM-version effects. Ournani et al. [112] and Shimchenko et al. [113] both identified measurable differences in electrical characteristics between JVM releases, although their findings concerning the relative behaviour of particular versions were not consistent. Ournani et al. reported a general tendency towards improved energy characteristics across successive JVM releases, while also identifying local regressions between particular versions. Shimchenko et al., by contrast, reported that OpenJDK 13 exhibited more favourable energy consumption than version 16, whereas the corresponding results reported by Ournani et al. indicated the opposite ordering for these releases. The present findings therefore emerge within an existing body of evidence showing that JVM-version effects are not necessarily

monotonic or stable across experimental contexts. Rather than identifying a general progression towards higher or lower electrical demand across OpenJDK versions, the present analysis similarly identifies a strongly localised version-specific characteristic. The particularly distinctive contribution of the present findings is that the OpenJDK 13 relationship can subsequently be traced across power use, workload category, execution mode, microarchitectural performance event measures, and multivariable model specifications.

This conclusion is reinforced by the regression modelling. OpenJDK 13 remained positively associated with power use after runtime configuration, workload category, execution mode, and selected microarchitectural performance event measures were incorporated simultaneously within the hierarchical models. The estimated coefficient remained comparatively stable across the pooled, interpretive, and tiered power use specifications, indicating that the elevated power use associated with OpenJDK 13 was not eliminated after accounting statistically for these additional explanatory factors (Chapter 7, §7.5). Under interpretive execution, bootstrap resampling produced an OpenJDK 13 coefficient almost identical to the original estimate, while the corresponding confidence interval remained clearly separated from zero. The power use effect should therefore be regarded as a particularly robust feature of the observed dataset rather than one attributable to model instability.

The microarchitectural analyses, (Chapter 5, §5.6), provide an additional and particularly informative perspective. OpenJDK 13 did not simply exhibit elevated electrical demand; it was also associated with a distinctive lower-level execution profile. Most notably, it exhibited the lowest branch misprediction rate of all assessed JRE versions and substantially reduced L1 data cache load miss activity relative to the remaining implementations. These characteristics are important because they demonstrate that the power use anomaly coincided with systematic changes in processor-level execution characteristics. However, they also caution against simplistic interpretations of hardware performance events. Lower

branch misprediction and L1 cache miss rates might conventionally be regarded as favourable execution characteristics, yet in this instance they coincided with the highest observed power use. The findings therefore demonstrate that individual microarchitectural event rates cannot be interpreted as direct proxies for reduced electrical demand.

The same evidence also places important limits on the interpretation of the OpenJDK 13 effect. L2 data cache load miss behaviour did not uniquely distinguish OpenJDK 13 from the remaining JRE releases. Although OpenJDK 13 exhibited the lowest observed central tendency, differences in L2 cache miss activity between JRE versions were comparatively small and were not statistically significant. Similarly, page fault activity varied only minimally between JRE versions. The OpenJDK 13 behaviour therefore cannot be described as a generalised reduction or increase across all measured microarchitectural activity. Rather, its distinctive lower-level characteristics were concentrated particularly in branch prediction and L1 data-cache behaviour (Chapter 5, §5.6).

The workload interaction analyses further demonstrate that the OpenJDK 13 effect was not expressed uniformly across application contexts. For power use, OpenJDK 13 remained elevated across all workload category levels, but the magnitude of the difference varied substantially between them. The effect was particularly pronounced within some workload categories and considerably smaller within others. Energy consumption exhibited an even more conditional pattern, with statistically significant JRE version differences concentrated within particular workload classes and predominantly involving OpenJDK 13. Workload composition therefore influenced the extent to which the implementation-specific characteristics of OpenJDK 13 became visible within the electrical outcomes (Chapter 6, §6).

Execution-mode decomposition provides perhaps the most important evidence for interpreting the energy consumption behaviour. Within the pooled energy consumption model, OpenJDK 13 retained a positive coefficient of approximately 0.259 (Chapter 7, §7.6). When analysis was restricted to interpretive execution,

this relationship strengthened substantially to approximately 0.461 (Chapter 7, §7.7). Under tiered execution, however, the coefficient effectively disappeared, falling to approximately −0.023 (Chapter 7, §7.8). The corresponding bootstrap analysis similarly demonstrated that the OpenJDK 13 energy coefficient was strongly positive and stable under interpretive execution but did not retain a stable coefficient structure under tiered execution (Chapter 7, §§§7.7.2 and §§§7.8.2). This represents a substantially different pattern from that observed for power use, where the OpenJDK 13 coefficient remained comparatively stable across the three model specifications.

This distinction is central to interpreting the anomaly. The evidence does not indicate that OpenJDK 13 simply consumed more energy irrespective of execution context. Instead, its elevated energy consumption became especially pronounced when execution remained within the interpreter and was largely removed once execution occurred under tiered compilation. The interpretive energy relationship also remained effectively unchanged following removal of influential observations, providing strong evidence that the observed OpenJDK 13 effect was not artificially generated by a small number of influential cases. The findings therefore point towards an execution-strategy-dependent characteristic whose consequences are amplified during interpretive execution but substantially attenuated when dynamic compilation is permitted.

From a systems perspective, this pattern raises the possibility that the OpenJDK 13 observations originate from implementation changes affecting execution paths exercised differently under interpretation and tiered compilation. Such an explanation would be consistent with the simultaneous presence of elevated power use, distinctive branch prediction and L1 cache characteristics, and the substantially stronger energy coefficient observed under interpretive execution. However, the present experiment was not designed to isolate individual HotSpot implementation changes, interpreter routines, generated machine-code sequences, or compiler transformations. The empirical evidence can therefore establish that

OpenJDK 13 exhibited a distinctive and execution-context-dependent profile, but it cannot identify the specific source-code change responsible for that profile.

This distinction is particularly important when considering individual OpenJDK development issues as possible explanations. A number of AArch64-specific HotSpot changes occurred around the OpenJDK 13 development period [211], demonstrating that architecture-specific execution paths were actively evolving. Such changes provide useful context for the plausibility of an implementation-level explanation, but temporal correspondence alone is insufficient to establish causality. Identifying a particular OpenJDK change as responsible for the observed behaviour would require targeted source-level comparison or controlled reconstruction of runtime versions with and without the relevant modification. Consequently, individual HotSpot changes should be presented as potential directions for subsequent investigation rather than as explanations established by the present results.

The OpenJDK 13 findings are nevertheless important beyond the behaviour of one runtime release. They demonstrate how a comparatively subtle implementation difference can remain strongly observable in system-level power measurements while also producing a distinctive microarchitectural signature. At the same time, the contrast between the persistent power use effect and the energy effect illustrates why power use and energy consumption should not be treated as interchangeable indicators of runtime behaviour. The former remained elevated across execution contexts, whereas the latter depended substantially upon how the runtime translated bytecode into executable machine instructions.

More broadly, the OpenJDK 13 case illustrates the value of analysing managed runtime systems across multiple explanatory layers. Had the investigation been restricted to power use, OpenJDK 13 would simply have appeared as an anomalously high-power runtime release. Incorporating workload category demonstrated that the magnitude of this effect depended upon application context; microarchitectural analysis revealed accompanying differences in lower-level execution characteristics;

and execution-mode-specific regression modelling demonstrated that its energy implications were substantially amplified under interpretation but largely absent under tiered execution. The apparent anomaly therefore becomes more informative when interpreted not as an isolated numerical outlier, but as evidence of a runtime implementation whose characteristics were expressed differently across the software–hardware execution stack.

OpenJDK JRE Source Version: Research Question Synthesis

Collectively, the evidence supports a cautious but substantive interpretation of the OpenJDK 13 behaviour. Its elevated power use represents a robust implementation-specific effect within the experimental environment, accompanied by distinctive branch prediction and L1 data-cache characteristics and conditioned in magnitude by workload. Its energy consumption consequences were considerably more dependent upon execution mode, becoming strongly expressed under interpretive execution but largely disappearing under tiered execution. While the precise HotSpot mechanism responsible cannot be established from the present data, the convergence of electrical, microarchitectural, workload-conditioned, and regression evidence indicates that the observation reflects a genuine execution characteristic rather than statistical noise or isolated measurement artefact.

More broadly, the OpenJDK 13 case contributes directly to answering the research questions by demonstrating that JRE source version can produce measurable differences in JRE/JVM electrical characteristics, but that the manifestation of those differences depends upon the level and context of analysis. In relation to RQ1, the findings establish a distinctive JRE-version effect in power use accompanied by systematic differences in selected microarchitectural performance event measures. In relation to RQ2, they demonstrate that the magnitude and statistical visibility of this implementation-specific effect are conditioned by workload category and execution mode. Finally, in relation to RQ3, the persistence of the OpenJDK 13 power use coefficient across model specifications, contrasted with the substantial restructuring of its energy consumption coefficient between interpretive and tiered execution, demonstrates that runtime implementation effects cannot be interpreted independently of either execution context or the electrical outcome being modelled.

Contribution to Knowledge: C7

The preceding synthesis supports **C7** through convergent evidence that OpenJDK version 13 exhibits a distinctive execution characteristic within the experimental environment. Its elevated power use persists across independent comparisons, workload-conditioned analyses, hierarchical modelling, execution-mode decomposition, sensitivity assessment, and bootstrap validation, while accompanying differences in selected microarchitectural measures provide evidence of a distinctive execution profile. The substantially greater dependence of its energy consumption relationship upon execution mode further demonstrates the value of evaluating runtime-version characteristics across multiple analytical levels and separately for power use and energy consumption. The precise implementation mechanism remains unresolved and represents a target for subsequent source-level investigation.

8.7 Hierarchical Modelling Reveals Context-Dependent Relationships

Perhaps one of the most important findings emerging from the empirical modelling presented in this dissertation is that the statistical relationships governing JRE JVM power use and energy consumption are not fixed properties of individual runtime characteristics. Instead, their apparent explanatory importance depends upon the wider system context in which they are evaluated, the execution strategy under which observations are generated, and the response variable being modelled. The findings therefore indicate that runtime implementation, execution mode, workload category, and microarchitectural behaviour should not be interpreted as independent determinants whose effects remain stable across analytical settings. Rather, their observed relationships are progressively restructured as additional software and hardware layers are incorporated into the analysis.

Previous investigations of JRE JVM behaviour have commonly examined

specific runtime mechanisms or system characteristics in relative isolation. Studies of managed runtime systems have investigated garbage collection strategy [146, 112, 113], compiler optimisation and execution strategy [13, 59], workload characteristics [39, 43], and microarchitectural behaviour [134, 161]. Similarly, research examining Raspberry Pi power use has demonstrated that processor utilisation and hardware performance event measures can provide useful predictors of system-level power demand [182, 183, 188]. Collectively, this literature establishes that individual software and hardware characteristics can influence execution behaviour and associated power use or energy consumption. However, the narrower analytical scope necessarily adopted within many earlier studies provides comparatively limited evidence regarding whether the apparent importance of these characteristics remains stable once multiple layers of the managed runtime system are considered simultaneously.

The hierarchical modelling framework developed in this dissertation (Chapter 7) addressed this issue by introducing predictors according to a theoretically motivated layered systems architecture. Foundational runtime implementation characteristics were considered first, followed by execution strategy, workload category, and emergent microarchitectural performance event measures. This ordering reflected the conceptual progression from software characteristics specified before execution towards lower-level computational behaviour emerging dynamically from the interaction between the workload, the configured runtime environment, and the underlying processor architecture.

The resulting analyses demonstrate that the successive predictor blocks did not merely account for additional proportions of outcome variance. Their introduction also altered the statistical relationships associated with predictor groups entered at earlier stages of the modelling process. Coefficients changed in magnitude as additional blocks were incorporated, some predictors gained or lost statistical significance, and several relationships differed in direction across the final model specifications. The hierarchical analyses therefore indicate that lower system layers

do not simply contribute additional explanatory power; they can also restructure the statistical relationships observed at higher layers of the managed runtime system.

This restructuring is especially apparent when workload category is incorporated into the models. Workload category accounted for substantial additional variation in both power use and energy consumption, but their importance was not confined to this direct contribution. Introducing workload category also changed the apparent relationships associated with runtime source version, build toolchain compiler version, execution mode, and garbage collector variant. This indicates that the effects attributed to runtime configuration cannot be interpreted independently of the computational demands imposed by the executing workload. A runtime characteristic that appears influential within a model excluding workload may exhibit a reduced, strengthened, or otherwise altered relationship once systematic differences in workload behaviour are represented.

The subsequent incorporation of microarchitectural performance event measures produced a similar effect at a deeper analytical layer. Branch misprediction rate, data-cache load miss rates, and page fault rate represented emergent rather than directly configured characteristics of execution. Although their incremental contributions to explained variance varied across the models, their inclusion frequently changed the magnitude or statistical interpretation of higher-level runtime coefficients. These results suggest that part of the variation initially associated with runtime configuration or workload category is statistically related to the lower-level processor behaviour through which those higher-level characteristics are expressed during execution.

The findings therefore provide empirical support for the layered conceptual framework underlying this dissertation. The software, runtime, workload, and hardware layers represented within that framework do not act as isolated sources of variation. Instead, characteristics at higher levels are expressed through, and statistically conditioned by, behaviour emerging at lower levels. From this

perspective, hierarchical model development serves not only as a method for calculating incremental variance contributions but also as a means of examining how explanatory relationships are reorganised as progressively deeper layers of the managed runtime system become observable.

The comparison between pooled and execution-mode-specific models provides particularly strong evidence of this context dependence. Relationships identified within the pooled analyses were not consistently reproduced when observations were separated according to interpretive and tiered execution (Chapter 7, §7.5, §7.9). Several predictors exhibited substantial differences in coefficient magnitude across the pooled, interpretive, and tiered model specifications. Other predictors changed statistical significance and, in some cases, coefficient direction. These differences demonstrate that relationships estimated from aggregated observations can conceal important structural variation associated with the execution strategy under which those observations were generated.

Under interpretive execution, workload category and microarchitectural performance event measures generally exhibited stronger relationships with the response variables. In the power use analysis, the microarchitectural block accounted for approximately 10% of additional variance under interpretive execution, compared with approximately 3% under tiered execution. Several associated coefficients, including those for L2 data-cache load miss rate and page fault rate, were also substantially larger within the interpretive specification than within either the pooled or tiered models (Table 7.16). A similar pattern was observed for interpretive energy consumption, where the microarchitectural block provided a considerably larger incremental contribution than it did within the pooled model (Table 7.32).

Tiered execution exhibited a different explanatory structure. Runtime implementation characteristics accounted for a comparatively larger proportion of variation within the tiered power model, while the additional contribution of the microarchitectural predictor block was more limited. These findings should not be interpreted as demonstrating that microarchitectural behaviour is absent or

unimportant under tiered execution. Rather, they indicate that its statistical relationship with the measured outcomes is less pronounced within the present dataset and model specification. The tiered and interpretive analyses therefore reveal distinct configurations of explanatory relationships rather than simply differing levels of the same effects.

Execution strategy consequently functions as more than an additional categorical predictor within the pooled models. The separate analyses demonstrate that it defines materially different analytical contexts within which the relationships between runtime configuration, workload category, microarchitectural behaviour, power use, and energy consumption are estimated. The effect of execution strategy is therefore visible not only through its pooled regression coefficient but through the broader restructuring of coefficients and variance contributions across the execution-mode-specific model specifications.

These findings complement the multidimensional perspective advanced by Blackburn et al. [31], who argued that a multifaceted methodology incorporating the underlying architecture, the compilers involved in code generation, virtual machine characteristics and implementations, memory management strategies, and the nature of the executed workloads is necessary to fully understand the complex interactions governing managed runtime execution. More recently, Papadakis et al. [161] similarly argued that contemporary JVM workloads should be understood through both high-level workload characteristics and lower-level microarchitectural behaviour. The present research extends these perspectives by demonstrating how the explanatory importance of these factors changes when they are analysed simultaneously within a hierarchical systems framework, and by showing that their explanatory importance varies according to execution strategy and model specification. The results also extend earlier Raspberry Pi power modelling research by Paniego et al. [182, 183, 188]. Whereas that work demonstrated the predictive value of utilisation measures and hardware performance events for predominantly native system activity, the present findings show that managed

runtime environments introduce additional layers of runtime implementation and execution strategy that alter the statistical observability of software–hardware relationships.

Contribution to Knowledge: C5

The preceding synthesis provides the empirical support for **C5**, completing the layered hierarchical framework developed within the Methodology. By progressively representing runtime implementation, execution configuration, workload category, and emergent microarchitectural behaviour, the framework demonstrates that successive analytical layers do not merely contribute additional explained variance but can restructure relationships attributed to previously represented factors. Combined with pooled and execution-mode-specific modelling and subsequent robustness assessment, this provides a systematic analytical framework for identifying explanatory relationships that remain comparatively stable across contexts and those whose apparent importance depends upon the level of the managed runtime system represented within the model.

8.8 Distinct Explanatory Structures Governing Power Use and Energy Consumption

One of the most important consequences of the context-dependent relationships revealed through the hierarchical modelling analysis is that power use and energy consumption exhibit substantially different explanatory structures. Although the two measures are physically related through the relationship

$$E = P \times t, \tag{8.1}$$

the factors that account for variation in instantaneous power use are not identical to those accounting for variation in cumulative energy consumption.

For power use, workload category consistently represented the dominant source of additional explained variance. Within the pooled model, workload category accounted for approximately one third of the observed variation beyond that explained by runtime implementation and execution configuration. Workload remained the dominant hierarchical block within both execution-mode-specific power models, accounting for an especially large increase in explained variance under interpretive execution. These findings indicate that instantaneous power demand is strongly differentiated by the type of computational activity being performed.

Execution mode, by contrast, contributed comparatively little additional explanatory power to the pooled power model. This does not imply that interpretive and tiered execution generate identical power relationships. As demonstrated by the separate models, execution strategy substantially altered the coefficients and variance contributions associated with the remaining predictors. Its principal importance for power use was therefore expressed through the restructuring of other relationships rather than through a large isolated contribution within the pooled variance decomposition.

The explanatory hierarchy for energy consumption was markedly different. Within the pooled energy model, execution mode accounted for the largest increase in explained variance, substantially exceeding the contribution of runtime implementation characteristics, workload category, or microarchitectural performance event measures. This dominant contribution is consistent with the substantial duration differences between interpretive and tiered execution. Because energy accumulates throughout the complete execution interval, differences in execution time are incorporated directly into the observed energy outcome.

Workload category also made a substantial contribution to explaining energy consumption, particularly within the interpretive model. However, its position within the overall explanatory hierarchy differed from that observed for power use. For power, workload was the dominant system-level predictor across the pooled

and execution-specific analyses. For pooled energy consumption, execution mode was dominant, with workload providing the next major explanatory contribution. The variance decomposition therefore demonstrates that the factors most strongly differentiating instantaneous demand are not necessarily those most strongly differentiating cumulative consumption.

This distinction is reinforced by the different functional forms required to model the two response variables. Power use was modelled satisfactorily using linear regression on the original measurement scale. Energy consumption, however, exhibited substantial improvement after logarithmic transformation. The raw linear energy model produced weaker explanatory performance and evidence of functional misspecification, whereas the log-linear model produced substantially greater explained variance and no corresponding evidence of misspecification under the Ramsey RESET assessment (Chapter 7, §7.6.3).

The requirement for different functional forms is consistent with the physical and statistical properties of the two outcomes. Power represents the rate at which energy is transferred and can therefore be interpreted as an instantaneous system state measured in watts. Within the present experimental setting, variation in power use was reasonably represented as an additive combination of runtime configuration, workload, and microarchitectural performance event measures. Energy consumption, in contrast, accumulates over the duration of execution and is jointly determined by both power demand and elapsed time. Its generative structure is therefore inherently multiplicative rather than purely additive.

The near-perfect empirical relationship observed between execution duration and energy consumption ($r \approx 0.99$), established during the energy consumption model development in Chapter 7, § 7.6, further supports this interpretation. Because the Raspberry Pi platform operated within a comparatively bounded power range, differences in execution duration produced substantial differences in cumulative energy even when differences in instantaneous power were relatively modest. Consequently, a runtime configuration or workload can exhibit a limited

effect upon average power use while producing a much larger effect upon total energy consumption through prolonged execution time.

This distinction is consistent with previous research demonstrating that changes in execution performance, instantaneous power demand, and cumulative energy consumption need not occur proportionally. Pinto et al. [108], for example, observed that increasing the number of application threads could increase power demand while simultaneously reducing total energy consumption where the corresponding reduction in execution time was proportionally greater than the increase in power use. Pallister et al. [30] similarly demonstrated on embedded ARM hardware that improvements in execution time resulting from compiler optimisation did not necessarily produce proportional reductions in energy consumption. More recently, Shimchenko et al. [113] identified workload category as a moderating factor in the relationship between execution-time performance and JVM energy consumption.

The present findings provide complementary evidence from a managed-runtime configuration perspective. They demonstrate not only that power use, execution duration, and energy consumption can respond differently to individual implementation choices, but that the factors accounting statistically for variation in instantaneous power use differ systematically from those accounting for cumulative energy consumption. The distinct hierarchical variance structures and model specifications identified in the present study therefore extend the earlier empirical observation that lower power use or shorter execution time cannot, in isolation, be assumed to imply proportionately lower cumulative energy consumption.

These findings demonstrate why power use and energy consumption should not be treated as statistically interchangeable response variables. A factor associated with increased power use does not necessarily produce a proportionately similar increase in energy consumption, because the resulting cumulative outcome also depends upon execution duration. Conversely, a configuration exhibiting slightly lower instantaneous power may still consume substantially more energy

when associated with a sufficiently longer execution interval. The direction and magnitude of relationships observed for power therefore cannot be assumed to transfer directly to energy.

The differing explanatory hierarchies also clarify why optimisation claims based exclusively on power measurements require careful interpretation. Lower instantaneous power demand cannot by itself establish lower cumulative energy consumption, just as shorter execution time does not necessarily imply proportionately lower instantaneous power. Evaluation of managed runtime systems should therefore distinguish explicitly between changes affecting the rate of energy use and those affecting the duration over which that demand persists.

Collectively, these findings provide empirical evidence that power use and energy consumption possess distinct explanatory and statistical structures. Power use was principally differentiated by workload and was satisfactorily represented through an additive linear specification. Energy consumption was strongly differentiated by execution mode and required a log-linear specification consistent with its cumulative and duration-dependent character. Future investigations of managed runtime systems should therefore develop outcome-specific models and should avoid assuming that analytical approaches or substantive interpretations derived from instantaneous power use will transfer directly to cumulative energy consumption.

Contribution to Knowledge: C6

The preceding synthesis supports **C6** by demonstrating that power use and energy consumption should not be treated as statistically interchangeable representations of JRE/JVM electrical behaviour. Their explanatory structures differ in the relative importance of runtime, execution, workload, and microarchitectural factors, while their statistical properties require different model specifications. The contribution therefore lies in establishing that conclusions concerning predictor importance and explanatory relationships for one electrical outcome cannot automatically be transferred to the other, despite the underlying physical relationship between power use and cumulative energy consumption.

8.9 Implications for the Analysis of Managed Runtime Systems

Beyond the substantive findings concerning JRE JVM power use and energy consumption, this research makes a methodological contribution through the analytical framework used to investigate managed runtime execution. Rather than examining individual runtime configuration factors, workload category, or microarchitectural performance measures in isolation, the study integrates these characteristics within a common experimental and statistical framework. This provides a means of progressing from the identification of individual effects, through the examination of workload-dependent relationships, towards multivariable models representing multiple layers of the managed runtime system simultaneously.

An important feature of this approach is the combination of pooled and execution-mode-specific analysis. Interpretive and tiered execution were initially examined within the complete experimental dataset, but were subsequently separated where required to determine whether relationships observed within pooled data remained evident within the underlying execution contexts. This distinction

proved methodologically important. Across the analyses, decomposition according to execution mode altered the magnitude, statistical significance, and, in some instances, direction of observed relationships. Consequently, relationships identified within aggregated managed-runtime measurements cannot necessarily be assumed to characterise the constituent execution environments. The use of execution-mode-specific analysis therefore provides a means of identifying structural heterogeneity that may otherwise remain concealed within pooled observations.

The hierarchical regression framework represents a further methodological contribution. Predictors were introduced according to a theoretically motivated ordering extending from runtime implementation characteristics, through execution configuration and workload category, to emergent microarchitectural performance event measures. This structure enabled the contribution of each analytical layer to be evaluated through changes in explained variance. More importantly, however, it permitted examination of how the relationships associated with previously entered predictors changed as progressively deeper layers of the system were represented. Hierarchical modelling was therefore used not solely as a procedure for partitioning explained variance, but as a means of examining the stability and restructuring of explanatory relationships across different levels of the managed runtime system.

This distinction is methodologically significant because a coefficient estimated within a restricted model may partly represent variation subsequently associated with workload category or lower-level processor behaviour. The progressive modelling strategy provides a means of observing this restructuring rather than interpreting coefficients from a single final model as isolated properties of individual runtime factors. Comparison of the pooled, interpretive, and tiered specifications further extends this approach by demonstrating whether such relationships remain stable when the analytical context is changed. The resulting framework therefore combines variance decomposition, coefficient comparison, and execution-context decomposition as complementary means of examining managed-runtime behaviour.

A further contribution concerns the treatment of power use and energy con-

sumption as distinct analytical outcomes. Although physically related, the results demonstrated that the two measures exhibited different statistical structures and different explanatory hierarchies. Power use could be represented satisfactorily on its original measurement scale, whereas energy consumption required logarithmic transformation to obtain an appropriately specified regression model. This distinction reflects the cumulative and duration-dependent nature of energy consumption and demonstrates that a common model form should not be imposed upon both electrical outcomes simply because they are physically related. The methodological implication is that model specification should be guided by the properties of the outcome being analysed rather than by an assumption that power use and energy consumption represent interchangeable measures of runtime electrical behaviour.

The study also adopts multiple complementary forms of statistical evidence rather than relying upon a single analytical procedure. Independent and workload-conditioned relationships were first established through the analyses addressing RQ1 and RQ2, before the simultaneous relationships between runtime, workload, and microarchitectural performance event measures were examined through the hierarchical models developed for RQ3. Heteroscedasticity-consistent inference was employed where model diagnostics indicated departures from constant residual variance, while sensitivity analyses were used to determine whether substantive conclusions depended upon influential observations. Stratified bootstrap resampling provided an additional assessment of coefficient stability while preserving the experimental configuration structure represented within the dataset. The convergence or divergence of findings across these analytical stages provides a more informative basis for interpretation than reliance upon statistical significance within any single model.

This multi-stage approach is particularly illustrated by the OpenJDK 13 findings. Its elevated power use was first identifiable through independent comparison, subsequently shown to vary in magnitude according to workload, and finally examined after simultaneous adjustment for runtime configuration, workload cate-

gory, and microarchitectural performance measures. The persistence of the power relationship across these different analytical stages provided substantially stronger evidence of a systematic runtime-specific effect than would have been available from a single omnibus comparison. Conversely, the changing OpenJDK 13 energy relationship across pooled, interpretive, and tiered model specifications demonstrated how decomposition can identify conditionality that would be obscured by interpretation of the aggregate result alone. The methodological value therefore lies not in any individual statistical technique, but in the progressive triangulation of evidence across analytical levels.

The microarchitectural analyses provide a related methodological insight. Hardware performance event measures should not necessarily be interpreted as independent causal predictors of electrical outcomes. Within a managed runtime environment, these measures are themselves emergent characteristics of interactions between workload, runtime implementation, execution strategy, and processor architecture. Their analytical value therefore extends beyond their direct association with power use or energy consumption. Changes in their distributions and in their relationships across execution contexts can provide evidence of underlying execution characteristics that are not visible from system-level electrical measurements alone. Incorporating these measures within a layered analytical framework consequently provides a bridge between configured software characteristics and the processor-level behaviour through which execution is realised.

Collectively, these methodological elements provide a framework for investigating managed-runtime power use and energy consumption as system-level outcomes emerging from multiple interacting analytical layers. The principal methodological contribution is therefore not the application of any individual statistical method, but their integration within a structured progression from independent effects, through conditional relationships, to hierarchical and execution-mode-specific modelling. This approach enables researchers to distinguish effects that remain

comparatively stable across analytical contexts from those whose apparent importance depends upon workload, execution strategy, lower-level processor behaviour, or the electrical outcome being modelled. In doing so, it provides a methodological basis for moving beyond isolated factor comparisons towards a more integrated analysis of software–hardware relationships within managed runtime systems.

8.10 Practical Implications

The findings of this research have practical implications for the evaluation, configuration, and analysis of managed runtime systems operating on resource-constrained and energy-conscious computing platforms. Although the study was designed primarily to develop empirical understanding rather than to prescribe universally optimal JRE/JVM configurations, the results demonstrate that runtime implementation decisions can influence both power use and energy consumption and that these influences are strongly dependent upon execution and workload context. Consequently, evaluation of managed runtime systems should extend beyond execution performance alone and consider the electrical consequences associated with alternative runtime configurations. The principal practical implications arising from the research are summarised below.

PI1: Runtime Configuration Factors Should Not Be Treated as Equally Influential.

The empirical findings demonstrate that the assessed runtime configuration factors do not contribute equally to variation in electrical outcomes. OpenJDK JRE source version produced a pronounced power use difference associated particularly with OpenJDK 13, whereas GNU GCC build toolchain compiler version generally exhibited considerably smaller and more context-dependent relationships. Garbage collector variant similarly produced comparatively modest effects that varied according to execution and workload context. JVM execution mode, by

contrast, represented a particularly important determinant of energy consumption. Evaluation or optimisation effort should therefore not assume equivalent electrical consequences across all runtime configuration parameters. Instead, attention should be directed towards those configuration dimensions demonstrated to exert substantive influence for the electrical outcome, workload, and execution context under consideration.

PI2: Runtime Evaluation Should Reflect the Intended Workload Context.

The strong role of workload category demonstrates that the electrical consequences associated with runtime configuration cannot be evaluated independently of the computational context in which those configurations operate. The magnitude and statistical detectability of several runtime relationships changed across the assessed workload category levels, meaning that evaluation using a single benchmark or narrowly defined workload may provide an incomplete representation of the electrical characteristics likely to emerge across a broader application environment. Where runtime configurations are being compared for deployment, benchmarking should therefore incorporate workloads representative of the intended application domain and, where possible, multiple workloads exhibiting different computational characteristics. This becomes particularly important when relatively small configuration effects are being assessed, since workload-dependent variation may exceed or alter the apparent influence associated with the runtime factor itself.

PI3: JVM Execution Mode Can Serve as Both an Operational and Analytical Dimension.

JVM execution mode has practical implications extending beyond the direct energy consumption difference observed between interpretive and tiered execution. The execution-mode-specific analyses demonstrated substantial restructuring of the relationships between runtime characteristics, workload, microarchitectural performance events, and electrical outcomes. For power use in particular, several lower-level microarchitectural relationships became substantially more statistically observable under interpretive execution, while runtime implementation characteristics accounted for a greater proportion of variation under tiered execution. Consequently, experimental evaluations that aggregate interpretive and tiered observations may obscure relationships present within either execution context, and execution-mode-specific analysis should be considered where the objective is to investigate relationships between runtime implementation and lower-level processor behaviour.

Interpretive execution should not, however, be assumed to provide universally greater experimental sensitivity. Greater dispersion within interpretive observations may itself introduce additional variability, and the relationships identified in this study remain specific to the assessed JRE/JVM, workloads, and ARM-based platform. Rather, the findings indicate that execution mode can be treated as an experimental design dimension. Tiered execution provides an appropriate context for examining behaviour under dynamically optimised runtime operation, whereas interpretive execution may provide a complementary analytical context in which some lower-level software–hardware relationships become more statistically observable when dynamic compilation is removed from the execution pathway. Evaluating both contexts may therefore reveal whether relationships remain stable or become attenuated, amplified, or otherwise restructured following the

introduction of runtime optimisation.

PI4: Power Use and Energy Consumption Should Be Evaluated as Distinct Electrical Outcomes.

Although power use and energy consumption are physically related, the empirical analyses demonstrate that they exhibit substantially different explanatory and statistical structures. Workload category accounted for the largest additional contribution to explained variation within the power use models, whereas JVM execution mode dominated the pooled energy consumption model. Energy consumption was also strongly associated with execution duration and required a different statistical specification from power use. Consequently, a runtime configuration associated with comparatively favourable power use should not automatically be assumed to produce correspondingly favourable energy consumption. Selection of an electrical measure should instead reflect the practical objective of the evaluation: power use where instantaneous electrical demand or power constraints are of primary concern, and energy consumption where cumulative electrical demand over the duration of execution represents the principal consideration. Where both objectives are relevant, the two outcomes should be evaluated separately rather than treated as interchangeable measures of runtime electrical behaviour.

PI5: Microarchitectural Performance Events Can Provide a Diagnostic Layer for Runtime Evaluation.

Microarchitectural performance event measures provide practical analytical information beyond their inclusion as predictors within statistical models. Branch misprediction, cache miss, and page fault measures exposed differences between runtime configurations and execution

contexts that were not apparent from electrical measurements alone. Where unexpected power use or energy consumption characteristics are identified, these measures can therefore provide an additional analytical layer through which differences in processor-level execution behaviour can be investigated. The OpenJDK 13 findings illustrate this diagnostic value: its elevated power use was accompanied by distinctive branch prediction and L1 data-cache characteristics, allowing the electrical observation to be situated within a broader execution profile even though the measurements cannot establish the source-level mechanism responsible. Microarchitectural performance events should therefore be regarded as complementary diagnostic indicators rather than direct causal explanations of electrical behaviour.

PI6: Managed Runtime Evaluation Should Adopt a Layered and Context-Sensitive Approach.

Taken together, the findings support a layered approach to the practical evaluation of managed runtime systems. Runtime configurations should be assessed using workloads representative of their intended deployment context, power use and energy consumption should be considered as distinct electrical outcomes, and execution strategy should be explicitly represented where it materially changes the relationships under investigation. Where deeper explanation is required, microarchitectural performance events can provide complementary evidence connecting higher-level runtime configuration with lower-level processor execution. The practical implication is therefore not the identification of a single universally preferable JRE/JVM configuration, but the provision of an evidence-based analytical approach through which runtime configuration choices can be evaluated according to the workload, execution context, hardware platform, and electrical

objective relevant to a particular deployment.

Collectively, Practical Implications **PI1**–**PI6** translate the empirical and methodological contributions of this thesis into a set of principles for the evaluation of managed runtime systems. They indicate that meaningful assessment of JRE/JVM electrical behaviour requires consideration not only of which runtime configuration is employed, but also of the workload under which it is evaluated, the execution strategy through which it operates, the electrical outcome being optimised or constrained, and the lower-level processor characteristics accompanying execution. These implications should nevertheless be interpreted within the experimental boundaries of the present investigation. The following section therefore considers the principal limitations that constrain the generalisability and interpretation of the findings.

8.11 Limitations

The findings and contributions arising from this research should be interpreted within the boundaries of the experimental and analytical design. Chapter 4, Section 4.12, considers threats to internal, external, construct, and conclusion validity and describes the methodological controls employed to mitigate their influence. The purpose of the present section is therefore not to repeat those methodological threats, but to consider the principal limitations that remain following completion of the empirical analysis and the extent to which they constrain the interpretation, generalisability, and practical application of the findings.

L1: Hardware Platform and Architectural Scope.

All experiments were conducted using Raspberry Pi 4 Model B devices incorporating the ARM Cortex-A72 microarchitecture. The controlled use of a common hardware platform was necessary to isolate relation-

ships associated with runtime configuration, workload, execution mode, and emergent microarchitectural behaviour; however, it also establishes an important boundary around the resulting conclusions. The coefficient relationships, relative variance contributions, and execution-mode-specific differences identified in this research should not be assumed to transfer unchanged to alternative ARM implementations, x86-64 processors, server-class systems, or heterogeneous and accelerated computing platforms. The contributions of the thesis are therefore most appropriately interpreted as demonstrating the existence and analytical importance of these relationships within the assessed ARM-based environment, rather than establishing architecture-independent effect magnitudes.

L2: Runtime and Compiler Version Scope.

The experimental design incorporated OpenJDK source versions 9 through 14 and GNU GCC build toolchain compiler versions 5 through 8, representing a deliberately controlled range of runtime and compiler implementations. These versions enabled systematic comparison of implementation differences and exposed the distinctive OpenJDK 13 characteristic identified throughout the thesis. However, the findings cannot establish whether the same relationships persist within more recent OpenJDK releases, alternative JVM implementations, or JRE/JVM instances constructed using other compiler families and optimisation strategies. This limitation is particularly relevant to implementation-specific findings: the OpenJDK 13 observation represents a robust empirical characteristic within the assessed environment, but should not be interpreted as evidence of a general property extending to other OpenJDK releases or JVM implementations.

L3: Workload Representativeness.

The Renaissance Benchmark Suite provides a diverse collection of contemporary JVM workloads and enabled workload category to be incorporated explicitly within the experimental design. Nevertheless, benchmark applications remain abstractions of real deployment environments, and the six workload categories employed in this research cannot represent the full diversity of enterprise, scientific, embedded, mobile, cloud-native, and interactive JVM applications. This limitation is particularly important given the findings associated with Research Question 2: because runtime configuration relationships were themselves shown to vary according to workload category level, conclusions derived from the assessed benchmark set should not be assumed to generalise uniformly to workload domains that were not represented. The demonstrated importance of workload conditioning therefore simultaneously constitutes an empirical finding and places a boundary around the generalisability of the other runtime configuration effects identified in the study.

L4: Cold-Start and Execution-State Scope.

Benchmark workloads were evaluated under cold-start conditions, with a single benchmark repetition performed within each JRE/JVM invocation. This design was necessary to permit controlled comparison between interpretive and tiered execution and to capture the electrical consequences associated with the complete execution interval. It does, however, mean that the tiered observations incorporate runtime initialisation, profiling, compilation, and progressive optimisation rather than representing exclusively steady-state JIT-compiled execution. Consequently, the substantial energy consumption differences observed between interpretive and tiered execution should be interpreted as

characteristics of the assessed cold-start execution process and should not automatically be extrapolated to long-running applications in which JIT compilation has reached a comparatively stable operating condition. Similarly, the execution-mode-dependent statistical observability identified in this thesis requires replication under warm and steady-state execution before its persistence across different stages of JVM execution can be established.

L5: Within-Condition Electrical Replication.

The main experimental design prioritised systematic coverage of a large runtime configuration and workload space, with each fully specified experimental condition represented by a single power use and energy consumption execution. Separate reliability assessments involving repeated executions were undertaken to establish the stability and reproducibility of the experimental platform; however, these assessments do not provide configuration-specific estimates of within-condition electrical variability across the complete experimental design. Consequently, the principal analyses characterise variation across runtime configurations and workloads rather than decomposing measurement variability within every individual experimental condition. Replicated electrical measurements across the full design would permit more direct estimation of within-condition uncertainty, but would also substantially increase the experimental measurement burden associated with the factorial configuration space.

L6: Measurement and Explanatory Resolution.

Power use and energy consumption were measured directly at the Raspberry Pi power boundary, providing system-level electrical observations but excluding supporting infrastructure such as networking

equipment, measurement hosts, and external services. Similarly, the selected microarchitectural performance event measures provide observable indicators of processor execution behaviour rather than direct observations of the software or hardware mechanisms producing that behaviour. This distinction is particularly important when interpreting relationships such as those associated with OpenJDK 13. The convergence of electrical, workload-conditioned, microarchitectural, and multivariable evidence supports the conclusion that a distinctive execution characteristic is present, but the experimental measurements cannot identify the source-level HotSpot implementation mechanism responsible. The microarchitectural measures should therefore be interpreted as analytical indicators through which execution characteristics can be investigated rather than as direct causal explanations of the observed electrical outcomes.

L7: Statistical and Causal Interpretation.

The statistical models developed for Research Question 3 are subject to limitations inherent in explanatory modelling of observational relationships. Although the hierarchical framework accounted for substantial proportions of variation in both power use and energy consumption, the models necessarily represent simplified approximations of considerably more complex runtime and processor behaviour. Predictor relationships may reflect both direct and indirect associations arising from relationships between runtime configuration, workload, execution strategy, and emergent processor behaviour, and the regression coefficients should therefore not be interpreted as estimates of isolated causal effects. The use of execution-mode-specific models, heteroscedasticity-consistent inference, sensitivity analysis, and stratified bootstrap validation strengthens confidence in the stability

of many of the identified relationships, but these procedures do not transform statistical associations into causal mechanisms. Residual unexplained variation further indicates that additional software, hardware, temporal, and environmental factors remain outside the scope of the present modelling framework.

L8: Residual Environmental and Experimental Variation.

Some residual experimental variability remains unavoidable despite the controlled design. In particular, processor temperature was not explicitly controlled or incorporated as an experimental factor, leaving the possibility that thermal variation contributed to differences observed between individual executions. Similarly, processor clock frequency was not fixed or continuously recorded during the retained experimental analyses. The Raspberry Pi platforms were permitted to employ their normal processor-frequency management behaviour, meaning that dynamic frequency scaling and hardware-level throttling could occur during benchmark execution. Consequently, transient differences in processor frequency cannot be excluded as a potential source of uncontrolled variation between individual executions.

Minor board-level manufacturing variation and background operating-system activity similarly cannot be eliminated completely. Reliability assessments, standardised hardware and software configurations, common operating-system images, controlled benchmark procedures, and separate collection of electrical and microarchitectural measurements were employed to reduce these influences, and the resulting stability assessments provide evidence that they did not dominate the experimental observations. Nevertheless, these controls reduce rather than eliminate potential uncontrolled variation.

Collectively, Limitations **L1**–**L8** define the scope within which the contributions of this thesis should be interpreted rather than undermining the empirical relationships from which those contributions arise. The study establishes that runtime implementation, workload, execution strategy, and emergent microarchitectural behaviour are associated with JRE/JVM electrical outcomes in systematic but context-dependent ways within the assessed experimental environment. What remains unresolved is the extent to which the magnitude, direction, and statistical observability of these relationships persist across alternative processor architectures, contemporary runtime implementations, broader workload domains, replicated experimental conditions, and warm or steady-state execution. These boundaries provide a direct basis for the future research directions considered in the following section.

8.12 Future Research

The findings, contributions, practical implications, and limitations identified in this thesis provide several directions through which the present research can be extended. These directions are not limited to addressing the methodological boundaries represented by Limitations **L1**–**L8**; they also provide opportunities to test the generalisability of the contributions established in this research, investigate the mechanisms underlying the observed relationships, and extend the experimental and analytical framework to new runtime and computing environments. The principal directions for future research are outlined below.

FR1: Cross-Architecture Replication of Runtime Electrical Relationships.

A direct extension of this research would be to replicate the experimental framework across alternative processor architectures and hardware platforms. The present findings were established using the ARM Cortex-A72 microarchitecture and therefore provide a controlled char-

acterisation of JRE/JVM electrical behaviour within that environment. Replication using newer ARM implementations, x86-64 processors, and server-class systems would establish whether the relative importance of runtime implementation, workload category, execution mode, and microarchitectural performance events persists across different processor architectures. Of particular interest would be whether the execution-mode-dependent statistical observability identified in this thesis remains apparent when processor microarchitecture, cache organisation, branch prediction structures, and memory systems differ. Such research would directly address Limitation **L1** while testing the architectural generalisability of Contributions **C1**, **C3**, and **C4**.

FR2: Extension to Contemporary and Alternative Managed Runtime Implementations.

Future research should extend the runtime implementation space beyond the OpenJDK and GNU GCC versions examined in the present study. Replication using contemporary OpenJDK releases would establish whether the implementation-dependent relationships identified across OpenJDK versions 9 through 14 remain observable within later generations of HotSpot. The framework could also be extended to alternative JVM implementations, JRE/JVM instances constructed using different compiler toolchains, and alternative runtime optimisation configurations. Such comparisons would help distinguish relationships that are specific to individual runtime implementations from those that represent more persistent characteristics of managed runtime execution. This direction would directly address Limitation **L2** and provide an important test of the broader applicability of Contributions **C1**, **C2**, and **C5**.

FR3: Broader Workload and Deployment-Context Validation.

The strong workload-conditioned relationships identified throughout this thesis make broader workload validation a particularly important direction for future research. The experimental framework could be extended beyond the Renaissance Benchmark Suite to incorporate additional benchmark suites and representative enterprise, scientific, embedded, server, and cloud-oriented applications. Evaluation within less controlled deployment environments could additionally examine whether the relationships observed experimentally remain detectable when applications operate alongside operating-system activity, networking, containerisation, virtualisation, and competing workloads. Rather than treating workload merely as a source of experimental variation, future research should continue to represent workload category explicitly within the analytical framework. This would address Limitation **L3** while providing a direct test of Contribution **C2**, particularly the proposition that workload represents an important conditioning context through which runtime configuration relationships are expressed.

FR4: Longitudinal Analysis of JVM Warm-Up and Steady-State Execution.

A particularly important extension concerns the temporal development of JRE/JVM electrical behaviour during execution. The present research evaluated workloads under cold-start conditions and therefore captured runtime initialisation, profiling, dynamic compilation, and progressive optimisation within the measured execution interval. Future experiments could instead collect repeated or continuous measurements across successive benchmark iterations, allowing power use, energy consumption, execution time, and microarchitectural perfor-

mance events to be examined as the JVM transitions from initial execution towards increasingly optimised and comparatively stable operation. Such an approach would establish whether the execution-mode-specific relationships identified in this thesis persist, diminish, or change as JIT compilation progresses. It would also allow the substantial energy consumption difference between interpretive and tiered execution observed under the present experimental conditions to be reconsidered across applications of differing execution duration. This direction directly addresses Limitation **L4** and would extend Contributions **C3** and **C6** by introducing execution state and time as additional analytical dimensions.

FR5: Replicated Electrical Measurement and Within-Configuration Variability.

Future experimental designs could incorporate repeated electrical measurements for each fully specified runtime and workload configuration. Whereas the present study prioritised broad systematic coverage of the factorial configuration space, repeated observations within individual conditions would permit direct estimation of within-configuration electrical variability in addition to the between-configuration relationships examined in this thesis. This would enable the use of variance-component or multilevel modelling approaches capable of distinguishing variation associated with repeated executions, hardware instances, workloads, and runtime configurations. Such an extension would be particularly valuable when investigating relatively small configuration effects whose magnitude may approach the natural variability of repeated execution. This direction would directly address Limitation **L5** and strengthen the inferential foundation of the layered analytical framework represented by Contribution **C5**.

FR6: Deeper Microarchitectural and Runtime-Level Investigation of Observed Electrical Characteristics.

The microarchitectural performance event measures employed in this thesis provided an analytical bridge between higher-level runtime configuration and lower-level processor execution, but they do not identify the software mechanisms responsible for the observed relationships. Future research could extend this analytical layer through a broader range of hardware performance counters, runtime instrumentation, JIT compilation logs, generated-code analysis, and source-level examination of HotSpot implementation differences. The distinctive OpenJDK 13 characteristic provides a particularly suitable case for such investigation. Its elevated power use, distinctive branch prediction and L1 data-cache characteristics, workload-conditioned expression, and execution-mode-dependent energy relationship provide convergent evidence of a distinctive execution characteristic, but the present study cannot establish the mechanism responsible. Combining electrical measurement with progressively deeper runtime and processor instrumentation could therefore investigate how implementation-level differences propagate through runtime execution to become observable as microarchitectural and electrical differences. This direction addresses the explanatory boundary represented by Limitation **L6** while extending Contributions **C4** and **C7**.

FR7: Execution Mode as an Experimental Dimension for Software–Hardware Analysis.

One of the broader research directions arising from this thesis concerns the potential use of execution strategy as an experimental dimension rather than solely as a runtime configuration choice. The present findings demonstrate that several software–hardware relationships changed

substantially when analysis was separated between interpretive and tiered execution. Future research could explicitly investigate whether controlled comparison between less dynamically optimised and dynamically optimised execution contexts can improve understanding of relationships that become attenuated, amplified, or otherwise restructured following runtime optimisation. Within the JVM, this could involve comparison of interpretive execution with different JIT compilation tiers and optimisation states. More broadly, the principle could be investigated in other managed and dynamically executed environments that provide distinguishable interpretation, compilation, or runtime optimisation pathways. Such work would test whether the execution-mode-dependent statistical observability identified in this thesis represents a characteristic specific to the assessed HotSpot environment or a more general analytical phenomenon associated with dynamically optimised software systems. This direction therefore extends Contribution **C3** beyond its present empirical setting while remaining explicitly subject to cross-platform and cross-runtime validation.

FR8: Development of Context-Dependent Power and Energy Decision Indicators.

A further practical research direction concerns the development of power and energy-aware indicators capable of supporting runtime configuration and deployment decisions. Earlier stages of this research considered whether JRE/JVM configurations could be represented using comparatively simple power or energy classifications. However, the findings of the revised thesis demonstrate that such classifications would require substantial caution. The electrical relationships associated with individual runtime configurations were shown to vary

according to workload category, JVM execution mode, and the electrical outcome under consideration, while the limitations of the present experimental design additionally constrain generalisation beyond the assessed ARM-based platform and OpenJDK environment.

Future research could therefore investigate whether larger and more heterogeneous datasets, incorporating repeated measurements, additional processor architectures, more recent and alternative runtime implementations, broader workload domains, and different execution states, reveal sufficiently stable relationships to support empirically validated runtime-selection indicators. Rather than assigning a fixed power or energy label to a runtime configuration, such mechanisms may need to adopt probabilistic or context-dependent decision rules in which classifications or recommendations are conditional upon the workload, execution strategy, hardware platform, and electrical objective under consideration.

Such work would provide an opportunity to revisit the original power- and energy-flag concept within a substantially stronger empirical framework. It would also permit investigation of whether runtime-selection indicators are best represented as discrete classifications, confidence-based recommendations, or predictive decision-support mechanisms derived from multivariable models. This direction therefore extends the Practical Implications arising from the present thesis while directly addressing the current limitation that the available dataset is not sufficiently broad to support general-purpose power- or energy-based configuration decisions.

FR9: Phase-Resolved Electrical Profiling of JVM Execution.

A valuable direction for future research would be to extend the present experimental framework through phase-resolved electrical profiling of

JVM execution. Such an investigation could synchronise power use and energy consumption measurements with explicitly instrumented JVM execution events, including application start-up, interpretive execution, JIT compilation, transitions between compilation tiers, deoptimisation, and subsequent compiled execution.

This could be achieved through JVM instrumentation interfaces, such as the Java Virtual Machine Tool Interface (JVMTI), together with complementary JVM logging or tracing facilities capable of exposing timestamped runtime events. These events could then be aligned with the high-resolution electrical measurements collected at the platform power boundary, permitting changes in power use and cumulative energy consumption to be associated more directly with specific phases of JVM execution. Such an approach would provide a substantially finer-grained view of the temporal mechanisms through which interpretation, dynamic compilation, and compiled execution contribute to overall JRE/JVM power use and energy consumption.

Collectively, Future Research Directions **FR1**–**FR9** provide a progression from replication and validation towards deeper explanatory and conceptual investigation. Cross-architecture, runtime, and workload replication would establish the extent to which the relationships identified in this thesis generalise beyond the present experimental environment, while longitudinal and replicated measurement designs would strengthen understanding of their temporal stability and within-condition variability. Deeper integration of electrical measurement, microarchitectural observation, and runtime-level instrumentation could subsequently investigate the mechanisms through which implementation differences become expressed at the processor and system electrical levels. Finally, treating execution mode as an experimental dimension provides an opportunity to test whether the differences in statistical observability identified within this research extend beyond the specific JRE/JVM environment examined here. In this way, future research can progress

from validating the empirical contributions established in this thesis towards developing a more general understanding of how managed runtime implementation, dynamic optimisation, workload, and processor behaviour collectively shape software electrical characteristics.

Chapter 9

Conclusion

This thesis investigated the extent to which software-controlled characteristics of the Java Runtime Environment and its associated Java Virtual Machine are associated with variation in electrical power use and energy consumption during the execution of representative JVM workloads on an ARM-based embedded computing platform. In doing so, the research examined OpenJDK source version, GNU GCC build toolchain compiler version, JVM execution mode, garbage collector variant, workload category, and selected microarchitectural performance event measures within a common experimental and analytical framework. Rather than treating the JRE/JVM as a single software layer, the research considered how variation across these different dimensions is reflected in the power use and energy consumption characteristics observed during workload execution.

The research was structured around three complementary research questions. RQ1 established the independent relationships associated with the principal runtime, workload, and microarchitectural dimensions examined within the study. RQ2 extended this analysis by determining the extent to which relationships associated with runtime configuration were conditioned by workload category. RQ3 subsequently integrated these dimensions within a theory-driven hierarchical modelling framework, allowing their relative explanatory contributions to be examined across pooled observations and separately under interpretive and tiered execution. Collectively, this progression provided a structured means of moving from the identification of individual relationships, through their workload

dependence, to an integrated assessment of their explanatory importance.

9.1 Conclusions in Relation to the Research Questions

The findings associated with RQ1 demonstrate that the software-controlled dimensions examined within this research are not equally associated with variation in JRE/JVM power use and energy consumption. Workload category produced substantial differences in both electrical outcomes, while JVM execution mode emerged as a particularly important distinction for energy consumption. OpenJDK source version was generally associated with comparatively limited variation, with the notable exception of OpenJDK 13, which exhibited a distinctive and consistently elevated power-use characteristic within the experimental environment. GNU GCC build toolchain compiler version and garbage collector variant generally exhibited smaller or more context-dependent relationships. The analysis of microarchitectural performance event measures further demonstrated that electrical variation can be examined alongside lower-level processor activity, although the strength and direction of these relationships differed according to the measure, electrical outcome, and execution mode considered.

RQ2 demonstrated that aggregated effects alone do not provide a complete characterisation of runtime-related electrical variation. Relationships associated with JRE/JVM configuration were found in several cases to depend upon the workload category under which execution occurred. This was particularly evident for OpenJDK source version and garbage collector variant in relation to power use, while energy-consumption relationships were generally more selective. The OpenJDK 13 power-use characteristic remained visible across workload category levels, although its magnitude varied between workload contexts. These findings establish workload not simply as another source of variation, but as an important analytical context within which runtime configuration relationships should be

interpreted.

RQ3 demonstrated that the relative explanatory importance of the dimensions examined within this research changes substantially according to both the electrical outcome and the execution mode represented within the model specification. For power use, workload category accounted for a substantial proportion of the explained variation across the pooled, interpretive, and tiered specifications. Microarchitectural performance event measures contributed considerably more explanatory information under interpretive execution than under tiered execution, while runtime configuration factors accounted for a greater proportion of variation within the tiered power-use specification. These differences demonstrate that relationships between software configuration, workload, processor activity, and power use are not statistically equally observable across JVM execution modes.

The energy-consumption models revealed a different explanatory structure. Within the pooled model, JVM execution mode accounted for a substantial proportion of the explained variation, reflecting the cumulative and duration-dependent nature of energy consumption and the considerable execution differences between interpretive and tiered execution. Once the observations were decomposed by execution mode, workload category became the dominant explanatory dimension, while the contribution of the microarchitectural measures again differed substantially between interpretive and tiered execution. The improved suitability of the logarithmic energy-consumption specification further demonstrated that power use and energy consumption, despite their underlying physical relationship, should not be assumed to exhibit equivalent statistical structures.

9.2 Overall Contribution of the Research

Taken together, the findings establish a multi-layer empirical characterisation of JRE/JVM power use and energy consumption, demonstrating that the relationships associated with runtime configuration, workload context, execution mode, and microarchitectural activity differ in their magnitude and explanatory

importance. The principal contribution of the research therefore extends beyond identifying whether individual runtime factors are associated with electrical variation. It demonstrates that these relationships are layered, context dependent, and differ according to whether power use or energy consumption is being examined.

A particularly important finding concerns JVM execution mode. Interpretive and tiered execution differ not only in their direct relationship with energy consumption, but also in the extent to which other software and microarchitectural relationships remain statistically observable once analysis is restricted to each execution mode. Several lower-level microarchitectural relationships became substantially more pronounced under interpretive execution, whereas corresponding relationships were attenuated under tiered execution. This provides empirical support for treating execution mode as an important analytical dimension when examining software–hardware relationships within managed runtime environments, while recognising that greater statistical observability does not itself establish greater underlying hardware sensitivity or causality.

The research also demonstrates the analytical value of incorporating selected microarchitectural performance event measures alongside higher-level runtime and workload factors. These measures provide an intermediate analytical layer between software configuration and observed electrical outcomes, enabling differences in power use and energy consumption to be considered alongside changes in processor-level activity. Their explanatory contribution was nevertheless strongly dependent upon execution mode and outcome, reinforcing the importance of interpreting such measures within the wider runtime and workload context rather than as independent proxies for electrical behaviour.

The OpenJDK 13 findings provide a further example of the value of this layered approach. Its elevated power use was observed across independent analyses, workload-conditioned analyses, and the hierarchical regression models, while accompanying differences were also identified within selected microarchitectural measures. In contrast, its energy-consumption relationship was substantially more

dependent upon execution mode, becoming strongly expressed under interpretive execution but largely disappearing within the tiered model specification. Although the experimental evidence does not establish the underlying HotSpot mechanism responsible for these observations, their recurrence across multiple levels of analysis supports their interpretation as a distinctive implementation-specific execution characteristic within the experimental environment rather than an isolated statistical or measurement artefact.

Methodologically, the thesis contributes a structured approach through which these relationships can be examined. The progression from independent effects, through workload-conditioned analysis, to theory-driven hierarchical modelling provides a framework for separating different levels of empirical evidence while retaining their relationship to a common conceptual structure. The distinction between the theoretical specification of the regression models and their subsequently estimated empirical forms, together with execution-mode-specific decomposition, sensitivity analysis, heteroskedasticity-consistent inference, and stratified bootstrap validation, provides a transparent basis from which the stability and explanatory contribution of the observed relationships can be assessed.

9.3 Concluding Perspective

The findings of this thesis demonstrate that the electrical characteristics of managed runtime execution cannot be adequately represented by considering individual software configuration choices in isolation. Power use and energy consumption emerge from the combined context within which a workload is executed, encompassing runtime implementation, execution strategy, memory-management configuration, workload category, and the processor activity accompanying execution. Importantly, the relative importance and statistical observability of these dimensions change according to whether instantaneous power use or cumulative energy consumption is examined and according to whether execution occurs interpretively or through tiered execution.

Within the boundaries of the experimental environment, workload category emerged as a particularly important source of variation, execution mode fundamentally distinguished the energy-consumption characteristics of JVM execution, and selected microarchitectural measures provided additional explanatory information whose contribution was especially pronounced under interpretive execution. At the same time, the distinctive OpenJDK 13 findings demonstrate that implementation-specific runtime characteristics can remain observable even when broader version effects are comparatively limited. These results caution against treating JRE/JVM version, compiler toolchain, garbage collection, execution mode, workload, or processor activity as interchangeable or uniformly influential dimensions of electrical behaviour.

More broadly, this research demonstrates the value of examining managed runtime systems through multiple analytical layers rather than reducing their electrical characteristics to a single configuration factor or outcome measure. By connecting runtime configuration and workload context with microarchitectural observation and direct electrical measurement, the thesis provides an empirical and methodological basis for more context-sensitive investigation of software power use and energy consumption. The resulting evidence reinforces the need to treat power use and energy consumption as related but analytically distinct outcomes and to recognise execution mode and workload context as central dimensions when evaluating electrical behaviour within managed runtime environments.

The thesis therefore concludes that understanding JRE/JVM power use and energy consumption requires consideration not only of *which* runtime configuration is executed, but also of *what* workload is executed, *how* that workload is executed by the JVM, and *which* processor-level characteristics remain observable under those execution conditions. It is through the combined consideration of these dimensions that the electrical characteristics of managed runtime execution can be more comprehensively understood.

Appendix A

Statistical Analysis Code

This appendix documents the availability and organisation of the R-language statistical analysis code used throughout this thesis. The complete collection of scripts supporting the descriptive, inferential, interaction, and regression analyses is publicly available through the accompanying GitHub repository:

`https://github.com/jonathan-lambert/PhD-RScripts`

The repository contains the executable R-language implementations used to support the analyses reported in Chapters 5, 6, and 7. These scripts provide the computational implementation of the statistical procedures described within Chapter 4 and allow the analyses reported throughout the thesis to be inspected and reproduced independently of the dissertation text.

The codebase supports the three principal stages of the empirical analysis. First, scripts associated with Research Question 1 implement the descriptive and inferential procedures used to examine differences in JRE/JVM power use, energy consumption, and microarchitectural performance event measures across the experimental factors. These analyses include distributional assessment, non-parametric comparison procedures, post-hoc comparisons, and the calculation of effect magnitudes.

Second, scripts associated with Research Question 2 implement the workload-conditioned analyses used to determine whether the effects associated with JRE source version, GNU GCC build toolchain compiler version, and garbage collector

variant remain consistent across workload category levels. These scripts apply the corresponding descriptive and inferential procedures within workload-conditioned subsets of the experimental dataset.

Third, scripts associated with Research Question 3 implement the hierarchical multiple linear regression modelling procedures used to account for variation in power use and energy consumption. These include model construction, dummy coding of categorical predictors, hierarchical block development, regression diagnostics, heteroscedasticity-consistent inference, sensitivity analysis, and stratified bootstrap resampling used to assess coefficient stability and associated uncertainty.

The statistical scripts were developed in conjunction with the methodological procedures described in Sections 4.10.3, 4.10.4, and 4.11.7. Representative code fragments are retained within Chapter 4 where they assist explanation of particular analytical procedures; however, the repository provides the complete executable codebase rather than reproducing extensive source listings within the thesis itself.

The repository should therefore be considered a supporting research artefact associated with this dissertation. Its purpose is to provide transparency regarding the computational implementation of the statistical analyses and to support reproducibility of the results presented throughout the empirical chapters. The statistical interpretation of the resulting outputs remains documented within Chapters 5, 6, and 7, while the repository provides the corresponding computational procedures used to generate those analyses.

Appendix A

Raspberry Pi 4 Model B Configuration File

To ensure configuration consistency across all experimental nodes, the same Raspberry Pi boot configuration file (`config.txt`) was deployed on every Raspberry Pi 4 Model B within the experimental cluster. The configuration file controlled low-level hardware and firmware settings, including device interfaces and graphics subsystem configuration. Listing A.1 presents the complete configuration file used throughout this study.

Listing A.1: Raspberry Pi boot configuration file (`config.txt`) deployed across all experimental nodes

```
# For more options and information see
# http://rpf.io/configtxt
# Some settings may impact device functionality. See link above for d

# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1

# uncomment this if your display has a black border of unused pixels
# and your display can output without overscan
#disable_overscan=1
```

```
# uncomment the following to adjust overscan. Use positive numbers if
# goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16

# uncomment to force a console size. By default it will be display's
# overscan.
#framebuffer_width=1280
#framebuffer_height=720

# uncomment if hdmi display is not detected and composite is being ou
#hdmi_force_hotplug=1

# uncomment to force a specific HDMI mode (this will force VGA)
#hdmi_group=1
#hdmi_mode=1

# uncomment to force a HDMI mode rather than DVI. This can make audio
# DMT (computer monitor) modes
#hdmi_drive=2

# uncomment to increase signal to HDMI, if you have interference, bla
# no display
#config_hdmi_boost=4

# uncomment for composite PAL
```

```
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800

dtparam=i2c_arm=on

dtparam=audio=on

[pi4]
dtoverlay=vc4-fkms-v3d
max_framebuffers=2

[all]
#dtoverlay=vc4-fkms-v3d
```

REFERENCES

[1] Raspberry Pi Ltd, "Raspberry Pi 4 Model B Specifications," [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, [Accessed: Sep. 15, 2026].

[2] Jetperch LLC, "Joulescope DC Energy Analyser," [Online]. Available: https://www.joulescope.com, [Accessed: Sep. 28, 2022].

[3] TP-Link Technologies Co., Ltd., *TL-SG116 16-Port Gigabit Desktop Switch Datasheet*, TP-Link Technologies Co., Ltd., 2026, [Accessed: Jun. 4, 2026]. [Online]. Available: https://www.tp-link.com/us/support/download/tl-sg116/

[4] J. Lambert, "PhD Data Joulescope Acquisition Script," [Online]. Available: https://github.com/jonathan-lambert/, 2026, gitHub repository. [Accessed: Jun. 7, 2026].

[5] A. Prokopec, A. Rosà, D. Leopoldseder, G. Duboscq, P. Tůma, M. Studener, L. Bulej, Y. Zheng, A. Villazón, D. Simon, T. Würthinger, and W. Binder, "Renaissance: Benchmarking suite for parallel applications on the JVM," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 31–47. [Online]. Available: https://doi.org/10.1145/3314221.3314637

[6] B. Venners, *Inside the Java Virtual Machine*. McGraw-Hill, 1998.

[7] K. J. Gough and D. Corney, "Evaluating the Java Virtual Machine as a Target for Languages Other Than Java," in *Modular Programming Languages*, W. Weck and J. Gutknecht, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 278–290.

[8] Cloud Foundry Foundation, "Top languages for enterprise application development," Cloud Foundry Research Report. [Online]. Available: https://www.cloudfoundry.org/wp-content/uploads/Developer-Language-Report_FINAL.pdf, 2018, [Accessed: Apr. 24, 2021].

[9] Azul Systems, Inc., "Java at speed: The Azul Zing JVM optimized for modern hardware," [Online]. Available: https://www.azul.com/resources-hub/whitepapers, [Accessed: Apr. 27, 2021].

[10] S. Cass, "Top programming languages 2020," IEEE Spectrum. [Online]. Available: https://spectrum.ieee.org/top-programming-language-2020, 2020, [Accessed: Apr. 29, 2021].

[11] TIOBE Software BV, "TIOBE Index for April 2021," [Online]. Available: https://www.tiobe.com/tiobe-index/, 2021, [Accessed: Apr. 23, 2021].

[12] W. Gu, N. A. Burns, M. T. Collins, and W. Y. P. Wong, "The evolution of a high-performing Java Virtual Machine," *IBM Systems Journal*, vol. 39, no. 1, pp. 135–150, 2000.

[13] J. Aycock, "A brief history of just-in-time," *ACM Comput. Surv.*, vol. 35, no. 2, p. 97–113, 06 2003. [Online]. Available: https://doi.org/10.1145/857076.857077

[14] D. Gregg, J. F. Power, and J. Waldron, "Platform independent dynamic Java Virtual Machine analysis: the Java Grande Forum benchmark suite," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 3-5, pp. 459–484, 2003. [Online]. Available: http://mural.maynoothuniversity.ie/10224/

[15] S. L. P. Jones, N. Ramsey, and F. Reig, "C–: A portable assembly language that supports garbage collection," in *Principles and Practice of Declarative Programming, International Conference PPDP'99, Paris, France, September 29 - October 1, 1999, Proceedings*, ser. Lecture Notes in Computer Science, G. Nadathur, Ed., vol. 1702. Berlin, Heidelberg: Springer, 1999, pp. 1–28. [Online]. Available: https://doi.org/10.1007/10704567_1

[16] C. Daly, J. Horgan, J. Power, and J. Waldron, "Platform independent dynamic Java Virtual Machine analysis: the Java grande forum benchmark suite," in *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, 2001, pp. 106–115.

[17] T. Cramer, R. Friedman, T. Miller, D. Seberger, R. Wilson, and M. Wolczko, "Compiling Java Just in Time," *IEEE Micro*, vol. 17, no. 3, p. 36–43, 05 1997. [Online]. Available: https://doi.org/10.1109/40.591653

[18] C. A. Hsieh, M. T. Conte, T. L. Johnson, J. C. Gyllenhaal, and W. mei W. Hwu, "A study of the cache and branch performance issues with running

Java on current hardware platforms," in *In Proceedings of COMPCON*. Piscataway, NJ, USA: IEEE, 1997, pp. 211–216.

[19] D. Gollapudi, "Java Platform, Standard Edition JRockit to HotSpot Migration Guide," Oracle. [Online]. Available: https://docs.oracle.com/javacomponents/jrockit-hotspot/migration-guide/index.html, Jun. 2015, release 8, E63235-01. [Accessed: Apr. 20, 2022].

[20] P. Costanza, C. Herzeel, and W. Verachtert, "A comparison of three programming languages for a full-fledged next-generation sequencing tool," *BMC Bioinformatics*, vol. 20, no. 1, p. 301, 2019. [Online]. Available: https://doi.org/10.1186/s12859-019-2903-5

[21] B. Oancea, I. G. Rosca, T. Andrei, and A. I. Iacob, "Evaluating Java performance for linear algebra numerical computations," *Procedia Computer Science*, vol. 3, pp. 474–478, 2011, world Conference on Information Technology. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050910004552

[22] L. Lin, Z. Xu, H. Huan, Z. Jian, and L. Li-Xin, "An empirical comparison of implementation efficiency of iterative and recursive algorithms of fast fourier transform," in *Machine Learning and Intelligent Communications*, M. Guan and Z. Na, Eds. Cham: Springer International Publishing, 2021, pp. 73–81.

[23] M. Källén and T. Wrigstad, "Performance of an OO compute kernel on the JVM: Revisiting Java as a language for scientific computing applications," in *Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*, ser. MPLR 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 144–156. [Online]. Available: https://doi.org/10.1145/3357390.3361026

[24] J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, "Chapter 2 - high-performance embedded computing," in *Embedded Computing for High Performance*, J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, Eds. Boston: Morgan Kaufmann, 2017, pp. 17 – 56. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128041895000028

[25] M. Wolf, "Chapter 2 - CPUs," in *High-Performance Embedded Computing (Second Edition)*, second edition ed., M. Wolf, Ed. Boston: Morgan Kaufmann, 2014, pp. 59 – 138. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124105119000022

[26] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[27] G. Polito, N. Palumbo, S. Labsari, S. Ducasse, and P. Tesone, "Interpreter register autolocalisation: Improving the performance of efficient interpreters," in *Companion Proceedings of the 6th International Conference on the Art, Science, and Engineering of Programming*, ser. Programming '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–5. [Online]. Available: https://doi.org/10.1145/3532512.3532518

[28] K. R. Bowers and D. R. Kaeli, "Characterizing the SPEC JVM98 benchmarks on the Java Virtual Machine," Northeastern University, Department of Electrical and Computer Engineering, Boston, MA, USA, Tech. Rep. ECE-CEG-98-026, Sep. 1998, [Online]. Available: https://ece.northeastern.edu/groups/nucar/publications/javafin.pdf.

[29] J. Lambert, R. Monahan, and K. Casey, "Repositioning tiered HotSpot execution performance relative to the interpreter," [Online]. Available: https://arxiv.org/abs/2303.04157, 2023.

[30] J. Pallister, S. J. Hollis, and J. Bennett, "Identifying Compiler Options to Minimize Energy Consumption for Embedded Platforms," *The Computer Journal*, vol. 58, no. 1, pp. 95–109, 11 2013. [Online]. Available: https://doi.org/10.1093/comjnl/bxt129

[31] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer *et al.*, "The DaCapo benchmarks: Java benchmarking development and analysis," in *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 2006, pp. 169–190.

[32] T. L. Floyd, *Principles of Electric Circuits: Conventional Current Version*, 9th ed. Upper Saddle River, NJ: Pearson, 2010.

[33] M. A. Hardy, *Regression with Dummy Variables*, ser. Quantitative Applications in the Social Sciences. Newbury Park, CA, USA: SAGE Publications, 1993, no. 93.

[34] R. F. Stärk, J. Schmid, and E. Börger, *Java and the Java Virtual Machine: definition, verification, validation*. Springer Science & Business Media, 2012.

[35] F. Yellin and T. Lindholm, *The Java Virtual Machine Specification*. Addison Wesley, 1996.

[36] C. Herder and J. Dujmovic, "Workload characterization using metrics based on instruction grouping," *International Journal of Computer and Information Sciences*, vol. 5, no. 1, 2004.

[37] H. McGhan and M. O'Connor, "PicoJava: A direct execution engine for Java bytecode," *Computer*, vol. 31, no. 10, pp. 22–30, 1998.

[38] J. Waldron and O. Harrison, "Analysis of virtual machine stack frame usage by Java methods." in *IMSA*. Citeseer, 1999, pp. 271–274.

[39] J. Waldron, "Dynamic bytecode usage by object oriented Java programs," in *Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 29 (Cat. No.PR00275)*, 1999, pp. 384–393.

[40] J. Waldron, C. Daly, D. Gray, and J. Horgan, "Comparison of factors influencing bytecode usage in the Java Virtual Machine," in *Second International Conference and Exhibition on the Practical Application of Java, Manchester, UK*, 2000.

[41] S. Byrne, C. Daly, D. Gregg, and J. Waldron, "Dynamic analysis of the Java Virtual Machine method invocation architecture," *Database*, vol. 96, no. 8.3, pp. 91–7, 2002.

[42] T. Dowling, J. F. Power, and J. Waldron, "Relating static and dynamic measurements for the Java Virtual Machine instruction set," *Recent Advances in Simulation, Computational Methods and Soft Computing*, 2002.

[43] D. Gregg, J. Power, and J. Waldron, "Platform independent dynamic Java Virtual Machine analysis: the Java Grande Forum benchmark suite," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 3-5, pp. 459–484, 2003.

[44] D. O'Donoghue, A. Leddy, J. F. Power, and J. Waldron, "Bi-gram analysis of Java bytecode sequences," in *Proceedings Second Workshop on Intermediate Representation Engineering for the Java Virtual Machine*. National University of Ireland Maynooth, 2002, pp. 187–192.

[45] D. O'Donoghue and J. F. Power, "Identifying and evaluating a generic set of superinstructions for embedded Java programs," in *International Conference on Embedded Systems and Applications (ESA '04)*, June 2004, pp. 192–198.

[46] D. Gregg, J. Power, and J. Waldron, "A method-level comparison of the Java Grande and SPEC JVM98 benchmark suites," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 7-8, pp. 757–773, 2005.

[47] J. Lambert and J. F. Power, "An analysis of basic blocks within SPECjvm98 applications," Department of Computer Science, National University of Ireland Maynooth, Maynooth, Co. Kildare, Ireland, Technical Report NUIM-CS-TR-2005-15, Nov. 2005. [Online]. Available: https://norma.ncirl.ie/1395/1/NUIM-CS-TR-2005-15.pdf

[48] J. Park, J. Park, W. Song, S. Yoon, B. Burgstaller, and B. Scholz, "Treegraph-based instruction scheduling for stack-based virtual machines," *Electronic Notes in Theoretical Computer Science*, vol. 279, no. 1, pp. 33–45, 2011.

[49] J. Cohen, *Statistical power analysis for the behavioral sciences*. Academic Press, 2013.

[50] P. Lengauer, V. Bitto, H. Mössenböck, and M. Weninger, "A comprehensive Java benchmark study on memory and garbage collection behavior of DaCapo, DaCapo Scala, and SPECjvm2008," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ser. ICPE '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 3–14. [Online]. Available: https://doi.org/10.1145/3030207.3030211

[51] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, and G. Bierman, *The Java Language Specification Java SE 20 Edition*. Oracle, 2023.

[52] T. Lindholm, F. Yellin, G. Bracha, A. Buckley, and D. Smith, *The Java® Virtual Machine Specification Java SE 20 Edition*. Oracle, 2023.

[53] Oracle, "Java Platform, Standard Edition & Java Development Kit Version 14 API Specification," [Online]. Available: https://docs.oracle.com/en/java/javase/14/docs/api/index.html, 2020, [Accessed: Sep. 20, 2023].

[54] K. Casey, A. Ertl, and D. Gregg, "Optimizing indirect branch prediction accuracy in virtual machine interpreters," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 29, no. 6:37, 2007. [Online]. Available: http://mural.maynoothuniversity.ie/10188/

[55] E. Gagnon and L. Hendren, "Effective inline-threaded interpretation of Java bytecode using preparation sequences," in *International Conference on Compiler Construction*. Springer, 2003, pp. 170–184.

[56] B. Davis and J. Waldron, "A survey of optimisations for the Java Virtual Machine," in *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java*, ser. PPPJ '03. USA: Computer Science Press, Inc., 2003, p. 181–183.

[57] T. Kotzmann, C. Wimmer, H. Mössenböck, T. Rodriguez, K. Russell, and D. Cox, "Design of the Java HotSpot™ client compiler for Java 6," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 5, no. 1, pp. 1–32, 2008.

[58] S. Oaks, *Java Performance: The Definitive Guide: Getting the Most Out of Your Code.* " O'Reilly Media, Inc.", 2014.

[59] V. Ivanov, "JVM JIT-compiler Overview," [Online]. Available: https://cr.openjdk.org/~vlivanov/talks/2015_JIT_Overview.pdf, Oracle Corp., [Accessed: Mar. 5, 2023].

[60] J. Rose, "OpenJDK Wiki: Performance Tactic Index," [Online]. Available: https://wiki.openjdk.org/display/HotSpot/PerformanceTacticIndex, 2009, [Accessed: Sep. 4, 2023].

[61] Oracle, "Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide," [Online]. Available: https://docs.oracle.com/javase/9/gctuning/toc.htm, 2017, [Accessed: Sep. 4, 2023].

[62] B. J. Evans, *Java: The Legend*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015.

[63] M. Reinhold, "JDK enhancement proposal index," [Online]. Available: https://openjdk.java.net/jeps/0, [Accessed: May 26, 2022].

[64] S. Johansson, "JDK enhancement proposal 344: Parallel full GC for G1," [Online]. Available: https://openjdk.java.net/jeps/344, [Accessed: May 26, 2022].

[65] E. Helin, "JDK enhancement proposal 344: Abortable mixed collections for G1," [Online]. Available: https://openjdk.java.net/jeps/344, [Accessed: May 26, 2022].

[66] R. S. Rodrigo Bruno, Thomas Schatzl, "JDK enhancement proposal 346: Promptly return unused committed memory from G1," [Online]. Available: https://openjdk.java.net/jeps/346, [Accessed: May 26, 2022].

[67] S. Kim, "JDK enhancement proposal 345: NUMA-aware memory allocation for G1," [Online]. Available: https://openjdk.java.net/jeps/345, [Accessed: May 26, 2022].

[68] I. Lam, "JDK enhancement proposal 310: Application class-data sharing," [Online]. Available: https://openjdk.java.net/jeps/310, [Accessed: May 26, 2022].

[69] I. L. Jiangli Zhou, Calvin Cheung, "JDK enhancement proposal 341: Default CDS archives," [Online]. Available: https://openjdk.java.net/jeps/341, [Accessed: May 26, 2022].

[70] C. C. Jiangli Zhou, Ioi Lam, "JDK enhancement proposal 346: Dynamic CDS archives," [Online]. Available: https://openjdk.java.net/jeps/350, [Accessed: May 26, 2022].

[71] M. C. Carlisle and A. Rogers, "Software caching and computation migration in Olden," *ACM SIGPLAN Notices*, vol. 30, no. 8, pp. 29–38, 1995.

[72] B. Cahoon and K. S. McKinley, "Data flow analysis for software prefetching linked data structures in Java," in *Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques*. IEEE, 2001, pp. 280–291.

[73] reprogrammer, "The JOlden Benchmark Suite," GitHub repository. [Online]. Available: https://github.com/reprogrammer/jolden, [Accessed: Sep. 15, 2026].

[74] J. M. Bull, L. A. Smith, M. D. Westhead, D. S. Henty, and R. A. Davey, "A benchmark suite for high performance Java," *Concurrency: Practice and Experience*, vol. 12, no. 6, pp. 375–388, 2000.

[75] Roldan Pozo and Bruce R. Miller, "SciMark 2.0: A Java benchmark for scientific and numerical computing," National Institute of Standards and Technology (NIST), 2000, [Online]. Available: https://math.nist.gov/scimark2/.

[76] Standard Performance Evaluation Corporation, "SPECjbb2015," [Online]. Available: https://www.spec.org/jbb2015/index.html, 2015, [Accessed: Sep. 7, 2023].

[77] Standard Performance Evaluation Corporation, "SPECjEnterprise 2018 Web Profile," [Online]. Available: https://www.spec.org/jenterprise2018web/, 2018, [Accessed: Sep. 15, 2026].

[78] Standard Performance Evaluation Corporation, "SPECjvm2008," [Online]. Available: https://www.spec.org/jvm2008/index.html, 2008, [Accessed: Sep. 7, 2023].

[79] I. Gouy, "The Computer Language Benchmarks Game," [Online]. Available: https://sschakraborty.github.io/benchmark/index.html, [Accessed: Sep. 4, 2023].

[80] B. Goel, S. A. McKee, and M. Själander, "Chapter two - techniques to measure, model, and manage power," in *Advances in Computers*, A. Hurson and A. Memon, Eds. Elsevier, 2012, vol. 87, pp. 7 – 54. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012396528800002X

[81] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati, "Portable, scalable, per-core power estimation for intelligent resource management," in *International Conference on Green Computing*, 2010, pp. 135–146.

[82] J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, "Chapter 4 - source code analysis and instrumentation," in *Embedded Computing for High Performance*, J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, Eds. Boston: Morgan Kaufmann, 2017, pp. 99 – 135. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128041895000041

[83] J. Nunez-Yanez and G. Lore, "Enabling accurate modeling of power and energy consumption in an ARM-based system-on-chip," *Microprocessors and Microsystems*, vol. 37, no. 3, pp. 319 – 332, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933113000021

[84] Standard Performance Evaluation Corporation (SPEC), "Power and Performance Benchmark Methodology V2.2," [Online]. Available: https://www.spec.org/power_ssj2008/docs/SPECpower-Methodology.pdf, 7001 Heritage Village Plaza, Suite 225 Gainesville, VA 20155, USA, 2014, [Accessed: Jun. 3, 2020].

[85] Standard Performance Evaluation Corporation, "SPECpower_ssj2008 Design Overview," [Online]. Available: https://www.spec.org/power/docs/SPECpower_ssj2008-Design_Overview.pdf, May 2012, [Accessed: Sep. 15, 2026].

[86] Y. Mao, "Detailed power measurement with ARM embedded boards," Master's thesis, University of Maine, Orono, ME, USA, 2018, [Accessed:

Sep. 15, 2026]. [Online]. Available: https://digitalcommons.library.umaine.edu/etd/2878

[87] M. Tanner, R. Eckel, and I. Senevirathne, "Enhanced low current, voltage, and power dissipation measurements via Arduino Uno microcontroller with modified commercially available sensors," in *APS March Meeting Abstracts*, 2016.

[88] B. Dezfouli, I. Amirtharaj, and C.-C. C. Li, "EMPIOT: An energy measurement platform for wireless IoT devices," *Journal of Network and Computer Applications*, vol. 121, pp. 135 – 148, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804518302479

[89] P. S. Munoz, N. Tran, B. Craig, B. Dezfouli, and Y. Liu, "Analyzing the resource utilization of AES encryption on IoT devices," in *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2018, pp. 1200–1207.

[90] S. Abdulsalam, D. Lakomski, Q. Gu, T. Jin, and Z. Zong, "Program energy efficiency: The impact of language, compiler and implementation choices," in *International Green Computing Conference*. Piscataway, NJ, USA: IEEE, Nov 2014, pp. 1–6.

[91] J. J. Merelo-Guervós, I. Blancas-Alvarez, P. A. Castillo, G. Romero, P. García-Sánchez, V. M. Rivas, M. García-Valdez, A. Hernández-Águila, and M. Román, "Ranking the performance of compiled and interpreted languages in genetic algorithms," in *International Conference on Evolutionary Computation Theory and Applications*, vol. 2. SCITEPRESS, 2016, pp. 164–170.

[92] G. G. Magalhaes, A. L. Sartor, A. F. Lorenzon, P. O. A. Navaux, and A. C. Schneider Beck, "How programming languages and paradigms affect performance and energy in multithreaded applications," in *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2016, pp. 71–78.

[93] H. Eichhorn, J. L. Cano, F. McLean, and R. Anderl, "A comparative study of programming languages for next-generation astrodynamics systems," *CEAS Space Journal*, vol. 10, no. 1, pp. 115–123, 2018.

[94] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. a. P. Fernandes, and J. a. Saraiva, "Energy efficiency across programming languages: How do energy, time, and memory relate?" in *Proceedings of the 10th ACM SIGPLAN International Conference on Software*

Language Engineering, ser. SLE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 256–267. [Online]. Available: https://doi.org/10.1145/3136014.3136031

[95] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, "Ranking programming languages by energy efficiency," *Science of Computer Programming*, vol. 205, p. 102609, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642321000022

[96] H. Rich, "What you need to know about the JITServer from IBM Semeru Runtimes – The basics," [Online]. Available: https://developer.ibm.com/blogs/all-you-need-to-know-about-the-openj9-jitserver/, Jun 2022, [Accessed: Feb. 28, 2023].

[97] A. G. Graham, J.-P. Legault, H. Soontiens, J. Brown, S. A. MacKay, G. W. Dueck, K. B. Kent, K. Konno, and D. Maier, "Evaluating the performance of the Eclipse OpenJ9 JVM JIT compiler on AArch64," in *2022 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2022, pp. 411–418.

[98] M. Arnold, S. Fink, D. Grove, M. Hind, and P. F. Sweeney, "Adaptive optimization in the jalapeño JVM," *SIGPLAN Not.*, vol. 35, no. 10, p. 47–65, oct 2000. [Online]. Available: https://doi.org/10.1145/354222.353175

[99] G. Chen, R. Shetty, M. Kandemir, N. Vijaykrishnan, M. Irwin, and M. Wolczko, "Tuning garbage collection in an embedded Java environment," in *Proceedings Eighth International Symposium on High Performance Computer Architecture*. Piscataway, NJ, USA: IEEE, 2002, pp. 92–103.

[100] C. S. Ballapuram, *Semantics-oriented low power architecture*. Georgia Institute of Technology, 2008.

[101] S. Georgiou, M. Kechagia, P. Louridas, and D. Spinellis, "What are your programming language's energy-delay implications?" in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 303–313. [Online]. Available: https://doi.org/10.1145/3196398.3196414

[102] D. Mossé, N. Aboughazaleh, B. Childers, and R. Melhem, "Energy conservation in memory hierarchies using power-aware cached-DRAM," IDepartment of Computer Science, University of Pittsburgh, Tech. Rep. TR-05-123, 2005.

[103] N. Vijaykrishnan, M. Kandemir, S. Kim, S. Tomar, A. Sivasubramaniam, and M. J. Irwin, "Energy behavior of Java applications from the memory perspective," in *Java (TM) Virtual Machine Research and Technology Symposium (JVM 01)*, 2001.

[104] S. Lafond and J. Lilius, "An energy consumption model for an embedded Java Virtual Machine," in *Architecture of Computing Systems - ARCS 2006*, W. Grass, B. Sick, and K. Waldschmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 311–325.

[105] T. Cao, S. M. Blackburn, and K. S. McKinley, "Virtual machine services: An opportunity for hardware customization," *Technology*, vol. 32, no. 32nm, p. 45nm, 2011.

[106] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "Runtime monitoring of software energy hotspots," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, pp. 160–169.

[107] G. Pinto, F. Soares-Neto, and F. Castor, "Refactoring for energy efficiency: A reflection on the state of the art," in *2015 IEEE/ACM 4th International Workshop on Green and Sustainable Software*. IEEE, 2015, pp. 29–35.

[108] G. Pinto, F. Castor, and Y. D. Liu, "Understanding energy behaviors of thread management constructs," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, 2014, pp. 345–360.

[109] G. Rocha, F. Castor, and G. Pinto, "Comprehending energy behaviors of Java I/O APIs," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, pp. 1–12.

[110] M. Longo, A. V. Rodriguez, C. M. Mateos Diaz, and A. O. Zunino Suarez, "Reducing energy usage in resource-intensive Java-based scientific applications via micro-benchmark based code refactorings," *Computer Science And Information Systems*, vol. 16, no. 2, pp. 541–564, 2023.

[111] W. Oliveira, R. Oliveira, F. Castor, G. Pinto, and J. P. Fernandes, "Improving energy-efficiency by recommending Java collections," *Empirical Software Engineering*, vol. 26, pp. 1–45, 2021.

[112] Z. Ournani, M. C. Belgaid, R. Rouvoy, P. Rust, and J. Penhoat, "Evaluating the impact of Java Virtual Machines on energy consumption," in *Proceedings*

of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2021, pp. 1–11.

[113] M. Shimchenko, M. Popov, and T. Wrigstad, "Analysing and predicting energy consumption of garbage collectors in OpenJDK," in *Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes*, 2022, pp. 3–15.

[114] D. C. Bree and M. Ó. Cinnéide, "Removing decorator to improve energy efficiency," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 902–912.

[115] G. Chen, M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and M. Wolczko, "Adaptive garbage collection for battery-operated environments." in *Java Virtual Machine Research and Technology Symposium*, 2002, pp. 1–12.

[116] T. Cao, S. M. Blackburn, T. Gao, and K. S. McKinley, "The yin and yang of power and performance for asymmetric hardware and managed software," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 225–236, 2012.

[117] R. Kumar and P. K. Singh, "An approach for compiler optimization to exploit instruction level parallelism," in *Advanced Computing, Networking and Informatics- Volume 2*, M. Kumar Kundu, D. P. Mohapatra, A. Konar, and A. Chakraborty, Eds. Cham: Springer International Publishing, 2014, pp. 509–516.

[118] Marcon, Caterina, Smirnova, Oxana, and Muralidharan, Servesh, "Impact of different compilers and build types on Geant4 simulation execution time," *EPJ Web Conf.*, vol. 245, p. 05037, 2020. [Online]. Available: https://doi.org/10.1051/epjconf/202024505037

[119] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, F. Behner, L. Bellagamba, J. Boudreau, L. Broglia, A. Brunengo, H. Burkhardt, S. Chauvie, J. Chuma, R. Chytracek, G. Cooperman, G. Cosmo, P. Degtyarenko, A. Dell'Acqua, G. Depaola, D. Dietrich, R. Enami, A. Feliciello, C. Ferguson, H. Fesefeldt, G. Folger, F. Foppiano, A. Forti, S. Garelli, S. Giani, R. Giannitrapani, D. Gibin, J. Gómez Cadenas, I. González, G. Gracia Abril, G. Greeniaus, W. Greiner, V. Grichine, A. Grossheim, S. Guatelli, P. Gumplinger, R. Hamatsu, K. Hashimoto, H. Hasui, A. Heikkinen, A. Howard, V. Ivanchenko, A. Johnson, F. Jones, J. Kallenbach, N. Kanaya,

M. Kawabata, Y. Kawabata, M. Kawaguti, S. Kelner, P. Kent, A. Kimura, T. Kodama, R. Kokoulin, M. Kossov, H. Kurashige, E. Lamanna, T. Lampén, V. Lara, V. Lefebure, F. Lei, M. Liendl, W. Lockman, F. Longo, S. Magni, M. Maire, E. Medernach, K. Minamimoto, P. Mora de Freitas, Y. Morita, K. Murakami, M. Nagamatu, R. Nartallo, P. Nieminen, T. Nishimura, K. Ohtsubo, M. Okamura, S. O'Neale, Y. Oohata, K. Paech, J. Perl, A. Pfeiffer, M. Pia, F. Ranjard, A. Rybin, S. Sadilov, E. Di Salvo, G. Santin, T. Sasaki, N. Savvas, Y. Sawada, S. Scherer, S. Sei, V. Sirotenko, D. Smith, N. Starkov, H. Stoecker, J. Sulkimo, M. Takahata, S. Tanaka, E. Tcherniaev, E. Safai Tehrani, M. Tropeano, P. Truscott, H. Uno, L. Urban, P. Urban, M. Verderi, A. Walkden, W. Wander, H. Weber, J. Wellisch, T. Wenaus, D. Williams, D. Wright, T. Yamada, H. Yoshida, and D. Zschiesche, "Geant4—a simulation toolkit," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, no. 3, pp. 250–303, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900203013688

[120] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: an overview," in *Proceedings of 1994 IEEE Symposium on Low Power Electronics*, 1994, pp. 38–39.

[121] D. Branco and P. R. Henriques, "Impact of GCC optimization levels in energy consumption during program execution," *Acta Electrotechnica et Informatica*, vol. 16, no. 1, pp. 20–26, 2016.

[122] P. Yuan, Y. Guo, and X. Chen, "Rethinking compiler optimizations for the Linux kernel," in *Proceedings of the 2015 Workshop on Programming Languages and Operating Systems (PLOS '15)*. Association for Computing Machinery, 2015.

[123] M. Ugur, C. Jiang, A. Erf, T. Ahmed Khan, and B. Kasikci, "One profile fits all: Profile-guided Linux kernel optimizations for data center applications," *SIGOPS Oper. Syst. Rev.*, vol. 56, no. 1, p. 26–33, Jun. 2022. [Online]. Available: https://doi.org/10.1145/3544497.3544502

[124] M. Palkowski and M. Gruzewski, "Time and energy benefits of using automatic optimization compilers for NPDP tasks," *Electronics*, vol. 12, no. 17, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/17/3579

[125] K. Venugopal, G. Manjunath, and V. Krishnan, "{sEc}: A portable interpreter optimizing technique for embedded Java Virtual Machine," in *2nd*

Java Virtual Machine Research and Technology Symposium (Java VM 02), 2002.

[126] C. Wimmer, C. Stancu, P. Hofer, V. Jovanovic, P. Wögerer, P. B. Kessler, O. Pliss, and T. Würthinger, "Initialize once, start fast: application initialization at build time," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–29, 2019.

[127] J. Lambert, R. Monahan, and K. Casey, "Accidental choices—How JVM choice and associated build tools affect interpreter performance," *Computers*, vol. 11, no. 6, p. 96, 2022.

[128] R.-G. Urma, "Alternative languages for the JVM. a look at eight features from eight JVM languages," [Online]. Available: https://www.oracle.com/technical-resources/articles/{J}ava/architect-languages.html, Jul 2014, [Accessed: Apr. 23, 2021].

[129] Oracle, "Oracle GraalVM Enterprise Edition 22: Truffle Language Implementation Framework," [Online]. Available: https://docs.oracle.com/en/graalvm/enterprise/22/docs/graalvm-as-a-platform/language-implementation-framework/, 2022, [Accessed: Apr. 20, 2022].

[130] A. Prokopec, A. Rosa, D. Leopoldseder, G. Duboscq, P. Tuma, M. Studener, L. Bulej, Y. Zheng, A. Villazon, D. Simon *et al.*, "On evaluating the Renaissance benchmarking suite: Variety, performance, and complexity," *arXiv preprint arXiv:1903.10267*, 2019.

[131] A. Prokopec, A. Rosà, D. Leopoldseder, G. Duboscq, P. Tůma, M. Studener, L. Bulej, Y. Zheng, A. Villazón, D. Simon *et al.*, "Renaissance: A modern benchmark suite for parallel applications on the JVM," in *Proceedings Companion of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, 2019, pp. 11–12.

[132] T. A. Proebsting, "Optimizing an ANSI C interpreter with superoperators," in *22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. New York, NY, USA: ACM Press, 1995, pp. 322–332.

[133] M. Maierhofer and M. A. Ertl, "Local Stack Allocation," in *CC '98: Proceedings of the 7th International Conference on Compiler Construction*. London, UK: Springer-Verlag, 1998, pp. 189–203.

[134] M. A. Ertl and D. Gregg, "The behavior of efficient virtual machine interpreters on modern architectures," in *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, ser. Euro-Par '01. Berlin, Heidelberg: Springer-Verlag, 2001, p. 403–412.

[135] M. A. Ertl and D. Gregg, "The structure and performance of efficient interpreters," *Journal of Instruction-Level Parallelism*, vol. 5, pp. 1–25, 2003.

[136] M. Berndl, B. Vitale, M. Zaleski, and A. Brown, "Context threading: a flexible and efficient dispatch technique for virtual machine interpreters," in *International Symposium on Code Generation and Optimization.* Piscataway, NJ, USA: IEEE, 2005, pp. 15–26.

[137] G. Savrun-Yeniçeri, W. Zhang, H. Zhang, E. Seckler, C. Li, S. Brunthaler, P. Larsen, and M. Franz, "Efficient hosted interpreters on the JVM," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 1, feb 2014. [Online]. Available: https://doi.org/10.1145/2532642

[138] S. Ritter, "A Matter of Interpretation: From Bytecodes to Machine Code in the JVM," [Online]. Available: https://www.{A}zul.com/blog/a-matter-of-interpretation-from-bytecodes-to-machine-code-in-the-jvm/, Azul, 2022, [Accessed: Mar. 7, 2023].

[139] V. Simonis, "Template- vs. C++-Interpreter shootout," [Online]. Available: https://web.archive.org/web/20090214031632/https://weblogs.{J}ava.net/blog/simonis/archive/2007/11/template_vs_cin.html, 2007, [Accessed: Mar. 7, 2023].

[140] D. Gregg, M. A. Ertl, and A. Krall, "Implementing an efficient Java interpreter," in *High-Performance Computing and Networking*, B. Hertzberger, A. Hoekstra, and R. Williams, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 613–620.

[141] Y. Shi, D. Gregg, A. Beatty, and M. A. Ertl, "Virtual machine showdown: stack versus registers," in *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, 2005, pp. 153–163.

[142] Y. Shi, K. Casey, M. A. Ertl, and D. Gregg, "Virtual machine showdown: Stack versus registers," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 4, no. 4, pp. 1–36, 2008.

[143] J. Rose, "Performance tactic index," [Online]. Available: https://wiki.openjdk.org/display/HotSpot/PerformanceTacticIndex, 2009, [Accessed: Apr. 26, 2023].

[144] L. Dykstra, W. Srisa-an, and J. M. Chang, "An analysis of the garbage collection performance in sun's HotSpot/sup TM/Java Virtual Machine," in *Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference (Cat. No. 02CH37326)*. IEEE, 2002, pp. 335–339.

[145] S. Tavakolisomeh, R. Bruno, and P. Ferreira, "BestGC: An automatic GC selector," *IEEE Access*, 2023.

[146] G. Contreras and M. Martonosi, "Techniques for real-system characterization of Java Virtual Machine energy and power behavior," in *2006 IEEE International Symposium on Workload Characterization*. IEEE, 2006, pp. 29–38.

[147] S. Bhattacharya, K. Rajamani, and M. Gupta, "Power-performance implications of software runtime bloat: A case study with the SPECpower_ssj2008 benchmark," IBM Research, Tech. Rep., 2010.

[148] P. Pufek, H. Grgić, and B. Mihaljević, "Analysis of garbage collection algorithms and memory management in Java," in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2019, pp. 1677–1682.

[149] Z. Cai, S. M. Blackburn, M. D. Bond, and M. Maas, "Distilling the real cost of production garbage collectors," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2022, pp. 46–57.

[150] J. W. Davidson, J. R. Rabung, and D. B. Whalley, "Relating static and dynamic machine code measurements," *IEEE Transactions on Computers*, vol. 41, no. 4, pp. 444–454, 1992.

[151] D. Gregg, J. Power, and J. Waldron, "Benchmarking the Java virtual architecture: The SPEC JVM98 benchmark suite," *Java Microarchitectures*, pp. 1–18, 2002.

[152] J. Horgan, J. Power, and J. Waldron, "Measurement and analysis of runtime profiling data for Java programs," in *Proceedings First IEEE International Workshop on Source Code Analysis and Manipulation*. IEEE, 2001, pp. 122–130.

[153] B. Davis, A. Beatty, K. Casey, D. Gregg, and J. Waldron, "The case for virtual register machines," in *Proceedings of the 2003 workshop on Interpreters, virtual machines and emulators*, 2003, pp. 41–49.

[154] L. Ardito and M. Torchiano, "Creating and evaluating a software power model for Linux single board computers," in *2018 IEEE/ACM 6th International Workshop on Green And Sustainable Software (GREENS)*, 2018, pp. 1–8.

[155] J. J. Dujmović, "Evaluation and design of benchmark suites," *State-of-the-Art in Performance Modeling and Simulation: Theory, Techniques, and Tutorials*, pp. 287–323, 1998.

[156] J. J. Dujmovic and I. Dujmovic, "Evolution and evaluation of SPEC benchmarks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 2–9, 1998.

[157] J. J. Dujmovic, "Universal benchmark suites," in *MASCOTS'99. Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems.* IEEE, 1999, pp. 197–205.

[158] J. J. Dujmovic and C. Herder, "Java workload characterization using bytecodes," in *Int. CMG Conference*, 2001, pp. 205–216.

[159] T. H. Romer, D. Lee, G. M. Voelker, A. Wolman, W. A. Wong, J.-L. Baer, B. N. Bershad, and H. M. Levy, "The structure and performance of interpreters," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS VII. New York, NY, USA: Association for Computing Machinery, 1996, p. 150–159. [Online]. Available: https://doi.org/10.1145/237090.237175

[160] D. Gregg, M. A. Ertl, and A. Krall, "A fast Java interpreter," in *Proceedings of the Workshop on Java optimisaton strategies for embedded systems (JOSES), Genoa.* Citeseer, 2001.

[161] O. Papadakis, A. Andronikakis, N. Foutris, M. Papadimitriou, A. Stratikopoulos, F. Zakkak, P. Xekalakis, and C.-E. Kotselidis, "A multi-faceted memory analysis of Java benchmarks," in *20th International Conference on Managed Programming Languages & Runtimes*, 2023.

[162] R. Radhakrishnan, N. Vijaykrishnan, L. K. John, A. Sivasubramaniam, J. Rubio, and J. Sabarinathan, "Java runtime systems: Characterization and architectural implications," *IEEE Transactions on computers*, vol. 50, no. 2, pp. 131–146, 2001.

[163] T. Li, R. Bhargava, and L. K. John, "Adapting branch-target buffer to improve the target predictability of Java code," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 2, no. 2, pp. 109–130, 2005.

[164] E. Rohou, B. N. Swamy, and A. Seznec, "Branch prediction and the performance of interpreters—don't trust folklore," in *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2015, pp. 103–114.

[165] C. Kim, S. Kim, H. G. Cho, D. Kim, J. Kim, Y. H. Oh, H. Jang, and J. W. Lee, "Short-circuit dispatch: Accelerating virtual machine interpreters on embedded processors," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 291–303, 2016.

[166] M. A. Ertl, "Stack caching for interpreters," in *Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation*, 1995, pp. 315–327.

[167] S. Hu *et al.*, "Cache performance in Java Virtual Machines: A study of constituent phases," in *2002 IEEE International Workshop on Workload Characterization.* IEEE, 2002, pp. 81–90.

[168] C. Wimmer, M. Haupt, M. L. Van De Vanter, M. Jordan, L. Daynès, and D. Simon, "Maxine: An approachable virtual machine for, and in, Java," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, jan 2013. [Online]. Available: https://doi.org/10.1145/2400682.2400689

[169] S. Crooijmans, M. Lubbers, and P. Koopman, "Reducing the power consumption of IoT with task-oriented programming," in *Trends in Functional Programming*, W. Swierstra and N. Wu, Eds. Cham: Springer International Publishing, 2022, pp. 80–99.

[170] I. Ashraf, M. Umer, R. Majeed, A. Mehmood, W. Aslam, M. N. Yasir, and G. S. Choi, "Home automation using general purpose household electric appliances with Raspberry Pi and commercial smartphone," *PloS one*, vol. 15, no. 9, pp. e0 238 480–e0 238 480, 2020.

[171] G. Coleman, W. Salter, and M. Walsh, "OpenWeedLocator (OWL): an open-source, low-cost device for fallow weed detection," *Scientific Reports*, vol. 12, no. 1, 2022, cited by: 4; All Open Access, Gold Open Access, Green Open Access. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85122560652&doi=10.1038%2fs41598-021-03858-9&partnerID=40&md5=e1960f2fae049106c8ecb0feabc5c1ed

[172] L. Chechlinski, B. Siemiatkowska, and M. Majewski, "A system for weeds and crops identification-reaching over 10 FPS on Raspberry Pi with the usage of MobileNets, DenseNet and custom modifications," *Sensors (Basel, Switzerland)*, vol. 19, no. 17, 2019.

[173] K. Joshitha, S. Nishanthini, B. Harshini, and T. Sadhana, "Raspberry Pi based crop and weed classification using machine learning algorithm," in *IOP Conference Series: Earth and Environmental Science*, vol. 980, 2022. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85125841008&doi=10.1088%2f1755-1315%2f980%2f1%2f012029&partnerID=40&md5=2906d2fc9da4e754b064c165f83dd2ad

[174] J. Y. Kim, H. Abdel-Haleem, Z. Luo, and A. Szczepanek, "Open-source electronics for plant phenotyping and irrigation in controlled environment," *Smart Agricultural Technology*, vol. 3, p. 100093, 2023.

[175] M. F. Kabir and S. Roy, "Real-time vehicular accident prevention system using deep learning architecture," *Expert systems with applications*, vol. 206, p. 117837, 2022.

[176] L. Skolrood, Y. Wang, S. Zhang, and Q. Wei, "Single-molecule and particle detection on true portable microscopy platforms," *Sensors and actuators reports*, vol. 4, p. 100063, 2022.

[177] Z. Wu, Y. Chen, B. Zhao, X. Kang, and Y. Ding, "Review of weed detection methods based on computer vision," *Sensors*, vol. 21, no. 11, 2021. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85106240294&doi=10.3390%2fs21113647&partnerID=40&md5=8dbf08a24aec888f938ced63282733d9

[178] M. Lupión, A. Polo-Rodríguez, J. Medina-Quero, J. F. Sanjuan, and P. M. Ortigosa, "On the limits of conditional generative adversarial neural networks to reconstruct the identification of inhabitants from IoT low-resolution thermal sensors," *Expert Systems with Applications*, vol. 203, p. 117356, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422007084

[179] S. D. Icagic and G. S. Kvascev, "A smart alcoholmeter sensor based on deep learning visual perception," *Sensors*, vol. 22, no. 19, 2022.

[180] J. C. Tovar, J. S. Hoyer, A. Lin, A. Tielking, S. T. Callen, S. Elizabeth Castillo, M. Miller, M. Tessman, N. Fahlgren, J. C. Carrington, D. A. Nusinow, and M. A. Gehan, "Raspberry Pi–powered imaging for plant phenotyping," *Applications in plant sciences*, vol. 6, no. 3, pp. e1031–n/a, 2018.

[181] F. Astudillo-Salinas, D. Barrera-Salamea, A. Vázquez-Rodas, and L. Solano-Quinde, "Minimizing the power consumption in Raspberry Pi to use as a remote WSN gateway," in *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, 2016, pp. 1–5.

[182] J. M. Paniego, L. Libutti, M. Pi Puig, F. Chichizola, L. C. De Giusti, M. Naiouf, and A. E. De Giusti, "Modelado estadístico de potencia usando contadores de rendimiento sobre Raspberry Pi," in *XXIV Congreso Argentino de Ciencias de la Computación (La Plata, 2018).*, 2018.

[183] J. M. Paniego, L. Libutti, M. Pi Puig, F. Chichizola, L. C. De Giusti, M. Naiouf, and A. E. De Giusti, "Modelado de potencia en placas SBC: integración de diferentes generaciones Raspberry Pi," in *XXV Congreso Argentino de Ciencias de la Computación (CACIC 2019, Universidad Nacional de Río Cuarto)*, 2019.

[184] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The nas parallel benchmarks," *Int. J. High Perform. Comput. Appl.*, vol. 5, no. 3, p. 63–73, Sep 1991. [Online]. Available: https://doi.org/10.1177/109434209100500306

[185] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '09. USA: IEEE Computer Society, 2009, p. 44–54. [Online]. Available: https://doi.org/10.1109/IISWC.2009.5306797

[186] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.728

[187] J. Somarribas, E. Meneses, and K. Olivas, "Predictive power consumption model for compute intensive applications in clustered ARM A53 embedded systems," in *2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)*, 2020, pp. 1–4.

[188] J. M. Paniego, L. Libutti, M. P. Puig, F. Chichizola, L. De Giusti, M. Naiouf, and A. De Giusti, "Unified power modeling design for various Raspberry Pi generations analyzing different statistical methods," in *Computer Science – CACIC 2019*, P. Pesado and M. Arroyo, Eds. Cham: Springer International Publishing, 2020, pp. 53–65.

[189] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XII. New York, NY, USA: Association for Computing Machinery, 2006, p. 185–194. [Online]. Available: https://doi.org/10.1145/1168857.1168881

[190] F. Kaup, P. Gottschling, and D. Hausheer, "PowerPi: Measuring and modeling the power consumption of the Raspberry Pi," in *39th Annual IEEE Conference on Local Computer Networks*, 2014, pp. 236–243.

[191] G. Bekaroo and A. Santokhee, "Power consumption of the Raspberry Pi: A comparative analysis," in *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, 2016, pp. 361–366.

[192] M. Cloutier, C. Paradis, and V. Weaver, "A Raspberry Pi cluster instrumented for fine-grained power measurement," *Electronics*, vol. 5, no. 4, p. 61, Sep 2016. [Online]. Available: http://dx.doi.org/10.3390/electronics5040061

[193] W. Hajji and F. P. Tso, "Understanding the performance of low power Raspberry Pi cloud for big data," *Electronics*, vol. 5, no. 2, p. 29, 2016.

[194] S. E. Alves Filho, A. M. F. Burlamaqui, R. V. Aroca, and L. M. G. Gonçalves, "NPi-cluster: A low power energy-proportional computing cluster architecture," *IEEE Access*, vol. 5, pp. 16 297–16 313, 2017.

[195] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, "HPL: A portable implementation of the high-performance Linpack benchmark for distributed-memory computers," [Online]. Available: https://www.netlib.org/benchmark/hpl/, 2008, [Accessed: Sep. 15, 2026].

[196] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[197] S. Plimpton, P. Crozier, and C. Trott, "miniMD: Molecular dynamics mini-app," GitHub repository. [Online]. Available: https://github.com/Mantevo/miniMD, [Accessed: Sep. 15, 2026].

[198] T. Shizukuishi and K. Matsubara, "An efficient tinification of the Linux kernel for minimizing resource consumption," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1228–1237. [Online]. Available: https://doi.org/10.1145/3341105.3373913

[199] I. Amirtharaj, T. Groot, and B. Dezfouli, "Profiling and improving the duty-cycling performance of Linux-based IoT devices," *CoRR*, vol. abs/1808.10097, 2018. [Online]. Available: http://arxiv.org/abs/1808.10097

[200] Raspberry Pi Ltd, "BCM2711 ARM Peripherals," [Online]. Available: https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf, 2022, [Accessed: Sep. 15, 2026].

[201] SanDisk, "SanDisk Ultra microSDHC UHS-I Card, 32 GB, A1, Class 10 (SDSQUA4-032G-GN6MA)," [Online]. Available: https://www.sandisk.com/en-ie/products/memory-cards/microsd-cards/sandisk-ultra-uhs-i-microsd?sku=SDSQUA4-032G-GN6MA, [Accessed: Jun. 6, 2026].

[202] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, 1st ed. John Wiley & Sons, 1991.

[203] OpenJDK, "Archived releases," [Online]. Available: https://wiki.openjdk.org/spaces/JDKUpdates/pages/42598676/Archived+Releases, 2026, [Accessed: Jun. 9, 2026].

[204] H. Oi, "A comparative study of JVM implementations with SPECjvm2008," in *2010 Second International Conference on Computer Engineering and Applications*, vol. 1, 2010, pp. 351–357.

[205] M. Hollander, D. A. Wolfe, and E. Chicken, *Nonparametric statistical methods*, 3rd ed. John Wiley & Sons, 2014.

[206] G. W. Corder and D. I. Foreman, *Nonparametric Statistics: A Step-by-Step Approach*, 1st ed. Hoboken, NJ: Wiley, 2009.

[207] W. D. Berry, *Understanding regression assumptions*. Sage, 1993, vol. 92.

[208] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. Chapman and Hall/CRC, 1994.

[209] J. Fox and S. Weisberg, "Bootstrapping regression models in R," Appendix to *An R Companion to Applied Regression*, 3rd ed., 2018. [Online]. Available: https://www.john-fox.ca/Companion/appendices/Appendix-Bootstrapping.pdf

[210] A. Canty and B. Ripley, *boot: Bootstrap Functions*, 2025, r package version 1.3-32. [Online]. Available: https://CRAN.R-project.org/package=boot

[211] OpenJDK, "JDK-8233466: aarch64: Remove unnecessary load of mdo when profiling return and parameter types," OpenJDK JDK Bug System. [Online]. Available: https://bugs.openjdk.org/browse/JDK-8233466, 2019, [Accessed: Aug. 29, 2026].