Roskilde University

IMT

Energy consumption of .Net Object Relational Mappers

Patrik Kolesár

supervised by
Maja Hanne Kirkeby and João Paulo Fernandes

June 01, 2023

Abstract

This thesis critically investigates the energy efficiency implications of Object Relational Mapping (ORM) frameworks within the .NET landscape, particularly focusing on Dapper, Entity Framework, and nHibernate. The primary objectives involve evaluating their energy consumption and execution time, exploring the impact of various query types and table sizes. Data was gathered through controlled experimental setups, running numerous database queries across different frameworks, query types, and table sizes. The data were subjected to descriptive statistics, non-parametric tests, and visual aids for comprehensive analysis.

Findings within this research reveal that Dapper outperforms in terms of energy efficiency compared to nHibernate and Entity Framework, suggesting its potential role in energy-conscious software development. Additionally, the type of query plays a significant role in energy consumption and execution time, with "Get" operations showing higher energy usage and increased execution time. Larger tables were also found to consume more energy during data retrieval processes.

The insights gained from this study provide a valuable contribution to understanding the energy efficiency nuances among different ORM frameworks, informing software engineers in making strategic decisions related to ORM framework selection, query type usage, and data management. These findings bear significant implications for both the academic field and the industry, enabling the advancement towards sustainable software development practices and contributing to the broader dialogue on energy efficiency.

Contents

1 Introduction

The energy consumption of data centers remains a significant concern in the digital sector, especially in the context of energy consumption, which is one the main focus of this master's thesis. Projections indicated a potential increase in power usage from approximately 33 GW to around 89 GW in the last decade [9]. Other studies suggest that data centers globally consumed around 400 TWh of electricity in 2018 [10]. Looking ahead, projections estimate a substantial growth in data center electricity consumption, with a potential usage exceeding 2000 TWh by 2030 [9] [10]. Additionally, it is acknowledged that the digital sector, including computing infrastructure, plays a significant role in new electricity use and CO2 emissions [8]. The energy efficiency of Object relational mapping (ORM) frameworks needs to address the challenges posed

by data centers, such as their substantial power requirements and the hypothesis that servers deliver only about 30% of their nominal electrical efficiency improvements, leading to reduced electricity costs and carbon emissions [23]. By examining these factors and aligning with the growing concern for sustainable energy practices, this research aims to contribute to the development of energy-efficient ORM frameworks that can help mitigate the environmental impact of data centers and other factors in the digital sector.

The concept of Object Relational Mapper (ORM) refers to a design pattern utilized in object-oriented programming languages to facilitate seamless access to relational databases. Implementations of ORM exist for various programming languages, providing developers with a range of options. The fundamental functionality of an ORM includes support for a specific persistence engine and essential CRUD operations. Additionally, advanced ORM features may encompass custom-SQL extensions to enhance query building capabilities. In essence, an ORM library leverages the object-oriented paradigm within a specific programming language, enabling developers to construct queries that retrieve data and map it into corresponding object types [27].

In recent years, there has been a growing recognition of the importance of incorporating environment-friendly practices, such as reducing power consumption, paper usage, and carbon emissions, in the software development process [28]. However, there is a noticeable gap in the existing literature and research when it comes to studying energy efficiency in Object-Relational Mapping (ORM) frameworks, particularly in the context of Dapper, Entity Framework, and nHibernate. The lack of empirical studies and comparative evaluations specifically focused on these frameworks' energy consumption leaves a significant knowledge gap. Therefore, this master thesis aims to bridge this gap by conducting an in-depth investigation into the energy efficiency of these ORM frameworks. By providing valuable insights and comparative analysis, this research aims to contribute to the development of sustainable software practices and inform decision-making processes in the selection and optimization of ORM frameworks.

The research conducted in this master thesis was carefully planned using the Goal-Question-Metrics (GQM) approach [12], a recognized methodology for designing and conducting empirical studies. The research question (RQ) that drove the experimentation can be succinctly stated as follows: "What is the impact of ORM frameworks, specifically Dapper, Entity Framework, and nHibernate, on energy consumption and performance?" This research question serves as the guiding principle for the investigation into the energy efficiency implications of these ORM frameworks within the .Net framework, with a specific focus on the essential CRUD (Create, Read, Update, Delete) operations. The aim of the research is to evaluate and compare potential differences among the chosen frameworks looking at their energy consumption and performance considering different metrics like table sizes or chosen CRUD operations.

2 Related work

In the domain of performance evaluation for .NET Object Relational Mappers (ORMs), previous studies [11] [17] [34] have primarily focused on comparing the time performance and memory consumption [30] of different ORM tools, highlighting their features and capabilities within a .NET environment. These studies have contributed valuable insights into query execution time, transaction management, and data processing. However, to the best of our knowledge, none of these studies specifically considered the energy efficiency implications of the ORM frameworks. In contrast, our current research extends this knowledge gap by investigating both time performance and energy consumption, offering a comprehensive evaluation to guide developers in selecting ORM frameworks that optimize performance and energy efficiency in their applications. Additionally, a notable contribution to the field comes from [20], where the study not only explores performance analysis but also examines multiple databases, providing a broader perspective on the capabilities and limitations of .NET-based Object-Relational Mapping frameworks across different database systems.

Other researches address practical performance issues [13] and offer benefits compared to plain SQL approaches [15]. However, there are also known disadvantages, and the choice between simpler ORMs like Dapper and more complex ones like EF Core or NHibernate depends on the desired trade-off between performance and functionality. None of the studies provide a comprehensive comparison among all three ORMs and analyze multiple CRUD calls with varying complexity [13] [15] [30].

In the field of energy efficiency evaluation for Object Relational Mappers (ORMs), it is worth noting two relevant research studies. First, [25] focuses on the energy efficiency of different ORM approaches within PHP applications, providing an empirical evaluation of their energy consumption patterns. Although this study diverges from our focus on .NET-based ORMs, it shares a common interest in energy efficiency and highlights the importance of considering energy optimization in ORM usage across various programming languages. Second, [24] explores the impact of database-related practices on the energy efficiency of software applications, providing insights into how certain practices can contribute to improved energy efficiency. While their study does not directly investigate ORMs or the .NET environment, it complements our research focus on energy optimization in the context of ORM frameworks in the .NET ecosystem. Together, these studies emphasize the significance of considering energy efficiency in ORM usage and software development practices, providing valuable insights for energy-efficient software engineering.

When it comes to energy efficiency in database systems, several studies have investigated power-performance tradeoffs and energy estimation of relational operations [32] [33]. Lang and Patel explored the concept of eco-friendly database management systems [22], while Chen

et al. focused on integrating renewable energy to achieve green databases [14]. Xu further contributed to the field by proposing a power-aware database management system [31]. Additionally, Song et al. compared and analyzed the energy efficiency of cloud databases and parallel databases [29]. These studies provide valuable insights into the energy efficiency considerations and strategies in the broader context of database systems.

3 Experiment Context

To shed light on the energy efficiency implications of these widely embraced ORM frameworks within the .NET landscape, this study meticulously examines the distinctive attributes of Dapper [3], Entity Framework [5], and nHibernate [7]. These frameworks have gained significant popularity among developers in the .NET community, as evidenced by their impressive download statistics from the NuGet package repository. For instance, Entity Framework has been downloaded over 212 million times, showcasing its widespread adoption and recognition [4]. Dapper, with over 228 million downloads, is highly favored for its lightweight and high-performance nature [2]. Similarly, nHibernate, with over 30 million downloads, is renowned for its advanced features and flexibility [6].

All of the frameworks share common functionality such as automatic mapping of database entities to object models, support for CRUD operations (Create, Read, Update, Delete), and the ability to perform database queries using object-oriented syntax. They also provide features for managing database connections, transaction handling, and caching mechanisms to optimize performance.

While frameworks like Dapper, Entity Framework, or nHibernate offer numerous advantages in data management within relational database management systems (RDBMS), they also exhibit certain drawbacks. In their pursuit of simplifying usage and abstracting the complexities of SQL, these frameworks occasionally obscure advanced functionalities and features inherent in SQL, thereby limiting developers' control and flexibility [16].

Despite their shared functionality, Dapper, Entity Framework, and nHibernate exhibit distinct characteristics that set them apart. Dapper prides itself on its lightweight and high-performance nature, relying on raw SQL queries and leveraging ADO.NET for efficient data retrieval. On the other hand, Entity Framework offers a rich set of features, including a powerful object-relational mapping designer, support for advanced querying capabilities through LINQ (Language Integrated Query), and automatic database schema generation. Meanwhile, nHibernate is known for its extensive support for complex mapping scenarios, including inheritance, associations, and advanced caching mechanisms providing a flexible and customizable

ORM solution, often preferred in enterprise-level applications.

Moreover, the significance of energy efficiency in the digital sector cannot be understated. Energy consumption has a direct impact on cost savings, environmental sustainability, and overall system performance. Hence, it is crucial to evaluate the energy efficiency implications of these ORM frameworks within the context of the .NET ecosystem.

In this research, the focus specifically lies on the energy performance trade-offs associated with the essential CRUD (Create, Read, Update, Delete) operations, which form the foundation of data manipulation in software applications. By rigorously evaluating the energy consumption profiles and conducting comparative analyses, valuable insights will be provided into the energy efficiency implications of these ORM frameworks, specifically within the .NET domain. The goal is to identify the framework that best balances performance and energy efficiency, enabling developers to make informed decisions in their ORM framework selection and optimization processes.

4 Experiment design

4.1 Experiment definition

According to the established Goal-Question-Metrics (GQM) approach [9], and building upon the design proposed by Procaccianti [25], the main objective of this investigation is to analyze the impact of three ORM frameworks, namely Dapper, Entity Framework, and nHibernate, on energy consumption and performance in software development. The study aims to provide insights into the energy efficiency implications of these frameworks from a software engineer's perspective, facilitating the identification of best practices and recommendations for optimizing data management processes. The research question guiding this study is: "What is the impact of Dapper, Entity Framework, and nHibernate on performance and energy consumption?".

To achieve this goal, the study utilizes specific metrics to assess their influence. These metrics include Energy Consumption, measuring the energy consumed by the software system in joules (J), and Execution Time, capturing the duration of an experimental unit in milliseconds (ms). By analyzing these metrics, the study aims to gain a comprehensive understanding of the performance and energy consumption characteristics of the examined ORM frameworks. Additionally, the study calculates Power Consumption (P) as the product of average energy consumption (E) and execution time (t), providing insights into the overall energy efficiency performance.

4.2 Variable selection

In the "Variable selection" section, the study carefully considers the selection of independent variables that can be manipulated and controlled. The primary variable of interest is the **ORM framework**, which includes three different frameworks: **Dapper, Entity Framework, and nHibernate**. These frameworks are chosen based on their popularity and significance in the .NET development community.

To address potential confounding factors and ensure a comprehensive analysis, two additional variables are identified. The first is the **Query type** issued to the database, encompassing the fundamental CRUD operations:**Create, Read (Get and GetById), Update, and Delete**. By considering different query types, the impact of each ORM framework on various data manipulation scenarios can be assessed.

The second independent variable is the **Table size** involved in the queries. Three distinct table sizes are considered:**Small, Medium, and Large**. The categorization is based on the table size in kilobytes (KB), with small tables being less than 100 KB, medium tables ranging from 100 KB to less than 1 MB, and large tables exceeding 1 MB. By examining different table sizes, the potential influence of data volume on the performance and energy consumption of the ORM frameworks can be explored.

These carefully selected independent variables and their corresponding treatments enable a comprehensive investigation into the energy efficiency implications of the Dapper, Entity Framework, and nHibernate ORM frameworks. By systematically examining the effects of different query types and table sizes, this study aims to provide valuable insights into the performance and energy consumption characteristics of these frameworks. Furthermore, by considering potential confounding variables and controlling the selected factors, a rigorous analysis of the variables' impacts on energy efficiency can be ensured.

By incorporating these variables into the study design, a deeper understanding of how different factors affect the performance and energy consumption of ORM frameworks in various data management scenarios can be gained.

4.3 Sample selection

The "Sample selection" section outlines the process of selecting samples from the AdventureWorks2019 database [1] for the experiment. AdventureWorks2019 is a widely used Microsoft database that showcases the capabilities of Microsoft's SQL Server. It contains multiple tables with millions of rows, offering a comprehensive dataset for evaluation.

Convenience sampling was employed to select the samples from AdventureWorks2019, allowing for the choice of relevant tables that align with the research objectives. This approach enables the evaluation of the energy efficiency implications of the ORM frameworks under investigation.

The selected tables from AdventureWorks2019 are as follows:

Table: Locations:

- Characteristics: A small table with 16 rows and 6 columns
- Size: 32.0 KB
- Data: 8.0 KB

Table: Purchases:

- Characteristics: A medium-sized table with 4,012 rows and 13 columns
- Size: 496.0 KB
- Data: 336.0 KB

Table: Addresses:

- Characteristics: A large table with 19,614 rows and 9 columns
- Size: 5812.5 KB
- Data: 2784.0 KB

It is important to note that the provided row numbers and sizes represent the initial state of the tables before the start of the experiment. It is also acknowledged that some test cases involve the creation and subsequent deletion of rows, which may impact the final measurements.

By selecting these specific tables from AdventureWorks2019, the study ensures a diverse representation of table sizes and characteristics, enabling a comprehensive evaluation of the energy efficiency implications of the ORM frameworks in different data management scenarios.

4.4 Instrumentation

The experiment for this research was conducted using a Lenovo ThinkPad X260 laptop with the model identifier 20F6007WMD. The laptop was equipped with an Intel Core i5-6200U CPU running at a base frequency of 2.30GHz. The system had 16 GB of DDR4 RAM for efficient memory operations. The primary storage device used was a SanDisk SD8TB8U2 SSD with a capacity of 256GB, providing fast and reliable storage. The research was performed on this laptop running Ubuntu 22.04 as the operating system. Power consumption measurements were captured using the RAPL (Running Average Power Limit) interface, a powerful tool for measuring and monitoring power consumption in cloud computing servers [21]. Additionally, for the execution time measurements, code instrumentation was employed, with timestamps logged at the end of each experimental run to capture the duration of the execution.

4.5 Experiment execution

To ensure accurate data interpretation, the execution time was measured in milliseconds using RAPL's high-resolution timer, as well as energy consumption that was recorded in joules using the RAPL interface. These measurements provided precise and granular data for analysis.

In order to capture a sufficient number of query executions, batches of 1000 consecutive queries were performed for each experimental condition. This approach allowed for a robust as-

sessment of the average energy consumption per query, enabling meaningful comparisons between the ORM frameworks.

To maintain consistency and eliminate any potential interference, each test script initiated a fresh database connection and generated new objects, if applicable to the ORM frameworks being evaluated. Caching of objects and queries was disabled to ensure accurate measurements and prevent any performance optimizations that could skew the results.

Furthermore, careful consideration was given to the selection and design of the specific queries used in the experiments. The queries were chosen to be representative of real-world scenarios, encompassing various levels of complexity, diversity, and relevance to software applications. This approach aimed to provide a comprehensive evaluation of the ORM frameworks' performance and energy consumption under different query conditions.

Additionally, efforts were made to control external influences during the data collection process, such as minimizing background processes and ensuring a stable and controlled testing environment.

Due to time limitations and resource constraints, two trials per experimental unit was conducted. While multiple trials would have provided more statistical power, the careful design and execution of the experiments aimed to maximize the reliability and validity of the results within the available resources.

By adhering to these rigorous experimental procedures, this study aimed to generate meaningful and reliable data for the subsequent analysis and interpretation of the results, ensuring that the conclusions drawn are based on sound experimental evidence.

Software versions:

- .NET Framework version 7.0.203
- Microsoft SQL Server version 16.0.1050.5
- Dapper version 2.0.123
- Entity Framework version 7.0.5
- nHibernate version 5.4.2

These specific versions were chosen based on their stability, compatibility, and relevance to the research objectives. It is important to note that the choice of software versions may impact the performance and energy efficiency results, and using different versions in future experiments may yield different outcomes.

4.6 Hypothesis formulation

The hypotheses are formulated in a two-tailed fashion. The following hypotheses examine the energy consumption, execution time, and the effect of different frameworks, query types, and table sizes. To test whether there are statistically significant differences between chosen entities, ANOVA (Analysis of Variance) [18] is conducted. In the case where the data is not normally distributed, a (non-parametric) permutation test [19] is conducted.

4.6.1 Hypothesis 1: Framework Energy Consumption

Null hypothesis H10: The energy consumption of the frameworks does not differ significantly from each other.

$$\mu_{\text{E,dapper}} = \mu_{\text{E,entity framework}} = \mu_{\text{E,nHibernate}}$$

Alternative hypothesis H1a: The energy consumption of at least one of the three frameworks significantly differs from the other two.

$$\exists \text{ i, j with i} \neq j \land i, j \in \{\text{dapper, entity framework, nHibernate}\} \mid \mu_{E,i} \neq \mu_{E,j} \quad (1)$$

4.6.2 Hypothesis 2: Framework Execution Time

Null hypothesis H20: The execution time of the frameworks does not differ significantly from each other.

$$\mu_{\text{t,dapper}} = \mu_{\text{t,entity framework}} = \mu_{\text{t,nHibernate}}$$

Alternative hypothesis H2a: The execution time of at least one of the three frameworks significantly differs from the other two.

$$\exists \text{ i, j with i} \neq j \land i, j \in \{\text{dapper, entity framework, nHibernate}\} \mid \mu_{t,i} \neq \mu_{t,j} \quad (2)$$

4.6.3 Hypothesis 3: Query Types Energy Consumption

Null hypothesis H30: There is no significant difference in energy consumption between the different query types: Create, Read (Get, GetById), Update, and Delete.

$$\mu_{\text{E,create}} = \mu_{\text{E,read}} = \mu_{\text{E,update}} = \mu_{\text{E,delete}}$$

Alternative hypothesis H3a: There is a significant difference in energy consumption between at least two of the different query types.

$$\exists \text{ i, j with i} \neq j \land i, j \in \{\text{create, read, update, delete}\} \mid \mu_{E,i} \neq \mu_{E,j} \quad (3)$$

4.6.4 Hypothesis 4: Query Types Execution Time

Null hypothesis H40: There is no significant difference in time between the different query types: Create, Read, Update, and Delete.

$$\mu_{\text{t,create}} = \mu_{\text{t,read}} = \mu_{\text{t,update}} = \mu_{\text{t,delete}}$$

Alternative hypothesis H4a: There is a significant difference in time between at least two of the different query types.

$$\exists \text{ i, j with i} \neq j \land i, j \in \{\text{create, read, update, delete}\} \mid \mu_{t,i} \neq \mu_{t,j} \quad (4)$$

4.6.5 Hypothesis 5: Table Size Energy Consumption

Null hypothesis H50: The size of the queried table does not have a significant effect on energy consumption.

$$\mu_{\text{E,small}} = \mu_{\text{E,medium}} = \mu_{\text{E,large}}$$

Alternative hypothesis H5a: The size of the queried tables has a significant effect on energy consumption.

$$\exists \text{ i, j with i} \neq j \land i, j \in \{\text{small}, \text{medium}, \text{large}\} \mid \mu_{E,i} \neq \mu_{E,j} \quad (5)$$

4.6.6 Hypothesis 6: Table Size Execution Time

Null hypothesis H60: The execution time of the different table sizes does not differ significantly from each other.

$$\mu_{\text{t,small}} = \mu_{\text{t,medium}} = \mu_{\text{t,large}}$$

Alternative hypothesis H6a: The execution time of at least one of the table sizes significantly differs from the other two.

$$\exists \text{ i, j with i} \neq j \land i, j \in \{\text{small}, \text{medium}, \text{large}\} \mid \mu_{t,i} \neq \mu_{t,j} \quad (6)$$

Note: The decision to conduct ANOVA or an alternative non-parametric test will depend on the distribution of the data. If the data deviates from normality, we will employ a permutation test.

5 Results

In this section, the study's findings are presented, summarizing the dataset's characteristics using descriptive statistics. The distribution of response variables is explored, and hypothesis testing is conducted to assess the validity of the formulated hypotheses. Visual aids, such as boxplots, are used to enhance data visualization. These results contribute valuable insights to the field and have practical implications for future research and applications.

5.1 Statistics

The section presents the descriptive statistics for energy consumption, execution time, and energy consumption, which were derived from the collected data. The statistics include the mean, median, standard deviation (σ), and coefficient of variation (CV) for each variable. It is worth noting that energy consumption exhibited relatively minor variations, indicating a consistent energy usage throughout the experiment. On the other hand, execution time showed more sig-

nificant fluctuations, suggesting varying performance levels across different treatments. These findings highlight the substantial impact of treatment variations on system performance, which consequently influenced the overall energy consumption. The descriptive statistics provide a comprehensive overview of the characteristics of these variables and shed light on the relationship between energy consumption, execution time, and power consumption in this study.

Total CPU Energy Consumption:

- Mean: 78.92J
- Median: 39.53J
- Standard Deviation (σ): 126.54J
- Coefficient of Variation (CV): 160.34J

Total Execution Time:

- Mean: 29553.12ms
- Median: 8522.54ms
- Standard Deviation (σ): 58160.40ms
- Coefficient of Variation (CV): 196.80ms

Total Power Consumption:

- Mean: 4.06W
- Median: 4.61W
- Standard Deviation (σ): 1.42W
- Coefficient of Variation (CV): 34.90W

Upon conducting the Shapiro-Wilk normality test [26] on our dataset, we made some interesting observations. For the variables "Get", "Create", "Dapper", "EntityFramework", "nHibernate", "Purchases", and "Addresses," the test indicated that the data does not follow a normal distribution (p ¡ 0.05), as evidenced by the low p-values and W-statistics less than 0.5. However, for the variables "GetById", "Update", "Delete", and "Locations," the test did not provide sufficient evidence to reject the null hypothesis, suggesting that the data may adhere to a normal distribution. These findings highlight the importance of considering the distributional characteristics of the data when selecting appropriate statistical methods for analysis. Based on these facts it was decided to perform permutation tests as we could not find enough evidence for normal distribution of data.

Please note, the box plots presented in this section have been adjusted to enhance their readability by extending the whisk values. The standard box plots including outliers can be consulted in Appendix A, under the 'Additional Statistics' section (see Figures 11 through 16). For a more comprehensive visualization of the differences among frameworks, operations, and tables, refer to the histograms of mean differences provided in the same section (see Figures 17 through 26).

Hypothesis 1 results: Framework Energy Consumption:
In Hypothesis 1, the energy consumption of different frameworks was examined without considering the query types or table sizes. The overall means of the energy consumption for each framework were compared. The permutation test revealed no significant difference among the frameworks. Therefore, the null hypothesis failed

to be rejected. Figure 1a displays a box plot depicting the distribution of energy consumption samples, grouped by framework. The box plot provides a visual representation of the range, median, quartiles offering insights into the distribution of energy consumption across different frameworks.

Empirical probabilities between different frameworks where A = Dapper, B = nHibernate, and C = Entity Framework:

Empirical Probability (AB): 0.4153
Empirical Probability (AC): 0.367
Empirical Probability (BC): 0.4385

The group means for each treatment are as follows:

Framework A: 72.872798J
Framework B: 79.479555J
Framework C: 84.401406J

Hypothesis 2 results: Framework Execution Time:
Hypothesis 2 aimed to assess the difference in execution time among the selected frameworks. The permutation test does not demonstrate a significant difference among the frameworks. Thus, the null hypothesis cannot be rejected. Figure 1b presents a box plot illustrating the distribution of execution time samples, grouped by framework. The box plot provides a visual representation of the range, median, quartiles allowing for insights into the distribution of execution times across different frameworks.

Empirical probabilities between different frame-

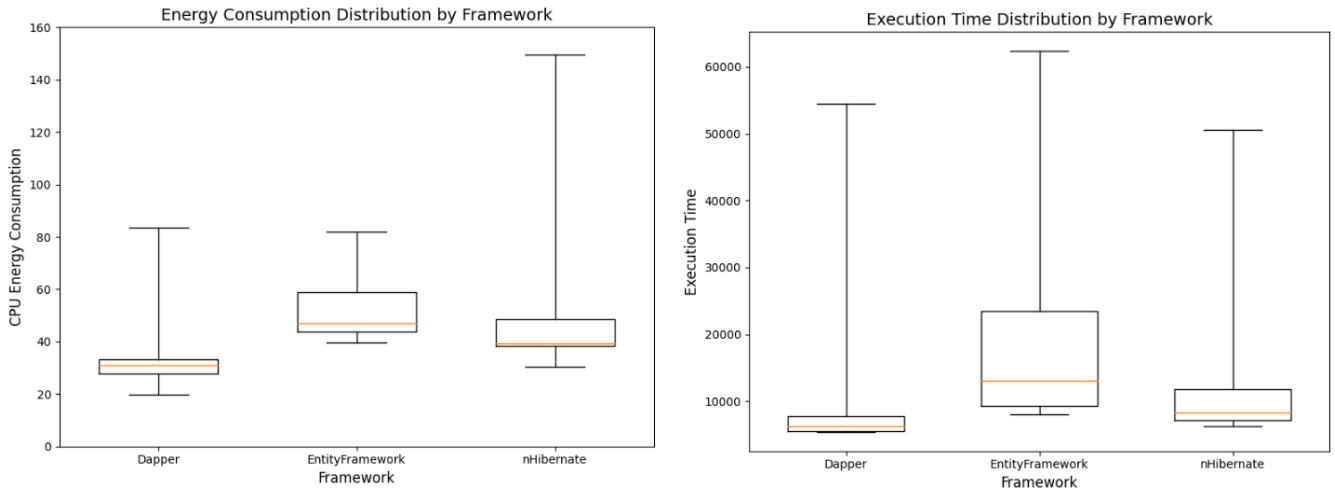

Figure 1: (a) Energy Consumption in J per framework (b) Execution time in ms per framework

works where A = Dapper, B = nHibernate, and C = Entity Framework:

Empirical Probability (AB): 0.5035
Empirical Probability (AC): 0.4682
Empirical Probability (BC): 0.4593

The group means for each treatment are as follows:

Framework A: 29067.558889ms
Framework B: 29045.221852ms
Framework C: 30546.564815ms

Hypothesis 3 results: Query Types Energy Consumption:
For Hypothesis 3, the energy consumption difference among the query types was investigated without considering frameworks or table sizes. The permutation test identified a noticeable difference among the query types (especially for Get operation). Therefore, the null hypothesis can be rejected. Figure 2a illustrates a box plot representing the distribution of energy consumption samples, grouped by query type. The box plot helps understand the range and variation in energy consumption for each query type, highlighting the differences in energy usage across different types of operations.

Empirical probabilities between different frameworks where A = Get, B = GetById, C = Create, D = Update, and E = Delete :

Empirical Probability (AB): 1.0
Empirical Probability (AC): 1.0
Empirical Probability (AD): 1.0
Empirical Probability (AE): 1.0
Empirical Probability (BC): 0.5107
Empirical Probability (BD): 0.5014
Empirical Probability (BE): 0.4569
Empirical Probability (CD): 0.5003
Empirical Probability (CE): 0.4478
Empirical Probability (DE): 0.4531

The group means for each treatment are as follows:

Query Type A: 219.422058J
Query Type B: 39.054369J
Query Type C: 38.306769J
Query Type D: 38.660007J
Query Type E: 39.374871J

Hypothesis 4 results: Query Types Execution Time:
Hypothesis 4 aimed to assess the effect of query types on the execution time without considering frameworks or table sizes. The permutation test identified a noticeable difference among the query types (especially with Get and Delete operations). Therefore, the null hypothesis can be rejected. Figure 2b showcases a box plot displaying the distribution of execution time samples, grouped by query type. The box plot helps in understanding the range and variation in execution times for each query type, highlighting the differences in execution duration across different types of operations.

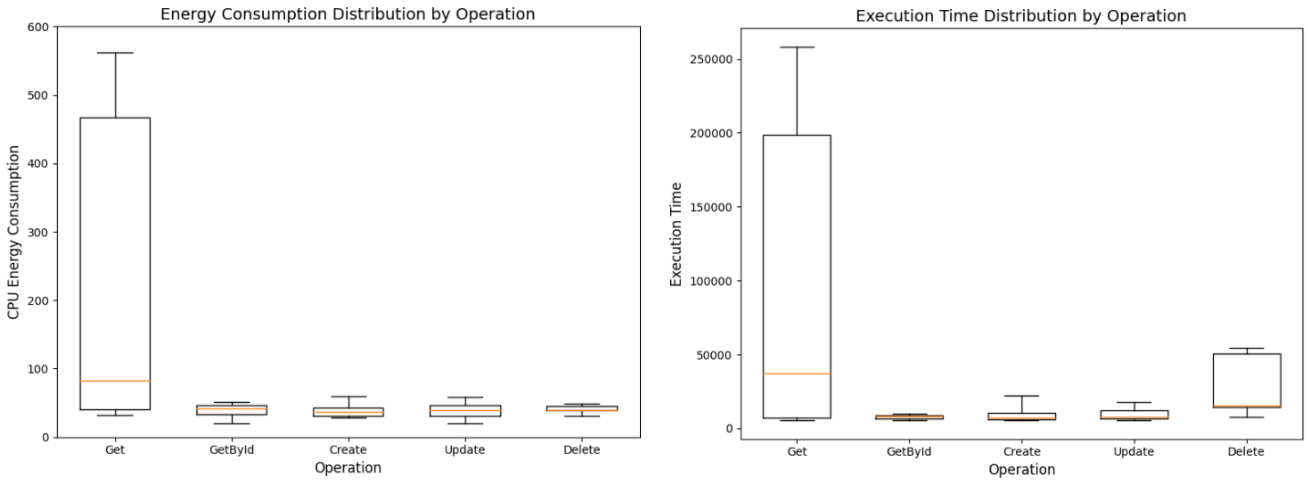

Figure 2: (a) Energy Consumption in J per operation (b) Execution time in ms per operation

Empirical probabilities between different frameworks where A = Get, B = GetById, C = Create, D = Update, and E = Delete:

Empirical Probability (AB): 1.0
Empirical Probability (AC): 1.0
Empirical Probability (AD): 1.0
Empirical Probability (AE): 0.9996
Empirical Probability (BC): 0.4693
Empirical Probability (BD): 0.4598
Empirical Probability (BE): 0.2086
Empirical Probability (CD): 0.4893
Empirical Probability (CE): 0.2195
Empirical Probability (DE): 0.2325

The group means for each treatment are as follows:

Query Type A: 91512.231111ms
Query Type B: 8048.633333ms
Query Type C: 9610.167778ms
Query Type D: 10067.179444ms
Query Type E: 27501.613333ms

Hypothesis 5 results: Table Size Energy Consumption:
In Hypothesis 5, the effect of table size (Small, Medium, Large) on energy consumption was investigated. The permutation test identified a noticeable difference among the table sizes. Thus, the null hypothesis can be rejected. Figure 3a displays a box plot representing the distribution of energy consumption samples, grouped by table size. The box plot provides a visual representation of the energy consumption distribution for each table size, allowing observation of the vari-

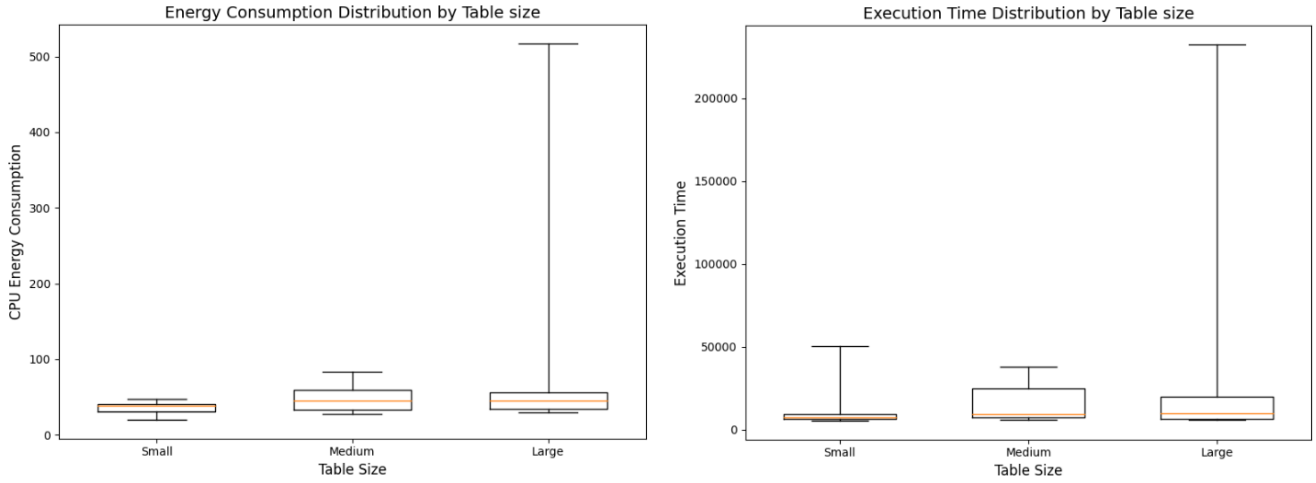

Figure 3: (a) Energy Consumption in J per table (b) Execution time in ms per table

ations in energy usage across different table sizes.

Empirical probabilities between different frameworks where A = Small Table(Locations), B = Medium Table(Purchases), and C = Large Table(Addresses):

Empirical Probability (AB): 0.3121
Empirical Probability (AC): 0.0003
Empirical Probability (BC): 0.0025

The group means for each treatment are as follows:

Table A: 36.221592J
Table B: 54.441248J
Table C: 146.090920J

Hypothesis 6 results: Table Size Execution Time:
Hypothesis 6 aimed to assess the effect of table size (Small, Medium, Large) on the execution time for all frameworks. The permutation test identified a noticeable difference among the tables. Therefore, the null hypothesis can be rejected. Figure 3b presents a box plot illustrating the distribution of execution time samples, grouped by table size. The box plot provides a visual representation of the execution time distribution for each table size, allowing observation of the variations in execution duration across different table sizes.

Empirical probabilities between different frameworks where A = Small Table(Locations), B = Medium Table(Purchases), and C = Large Table(Addresses):

Empirical Probability (AB): 0.4112
Empirical Probability (AC): 0.0014
Empirical Probability (BC): 0.0021

Table A: 13208.922593ms
Table B: 16864.110370ms
Table C: 58586.312593ms

The group means for each treatment are as follows:

6 Discussion

In this study, the focus was on analyzing the energy efficiency implications of three widely-used ORM frameworks in the .NET landscape: Dapper, Entity Framework, and nHibernate. The goal was to understand how these frameworks impact power consumption and execution time. The metrics used for evaluation included energy consumption, execution time, operation type, and table size. By conducting this research, valuable insights were gained to assist software engineers in making informed decisions when selecting an ORM framework and optimizing energy efficiency in their applications.

The descriptive statistics reveal the characteristics of the dataset, with energy consumption exhibiting minor variations and execution time showing slightly more significant fluctuations. The Shapiro-Wilk normality test indicates that the data distribution varies across different variables. Hypothesis testing results indicate that there is no significant difference in energy consumption among the frameworks, while there is a mild difference in execution time. The analysis of query types reveals noticeable differences in energy consumption and execution time, with Get operations having the highest impact. Additionally, table size has a significant effect on energy consumption and execution time. These results contribute to a better understanding of the performance and energy efficiency implications of these ORM frameworks, enabling readers to better understand how to optimize energy consumption in their applications.

An interesting finding from the analysis is the notable variation in energy consumption and execution time levels among the frameworks (Dapper, Entity Framework, and nHibernate) compared to the overall mean level. Figure 4a presents a plot which reveals that Dapper demonstrates better energy consumption for this research compared to both nHibernate and Entity Framework, indicating its potential for energy-efficient software development. Moreover, Fihure 4b shows that Dapper and nHibernate exhibit similar execution time levels, suggesting comparable performance in terms of time efficiency. In contrast, Entity Framework stands out with significantly higher execution time and energy consumption, indicating potential performance and energy drawbacks. However, it is worth mention-

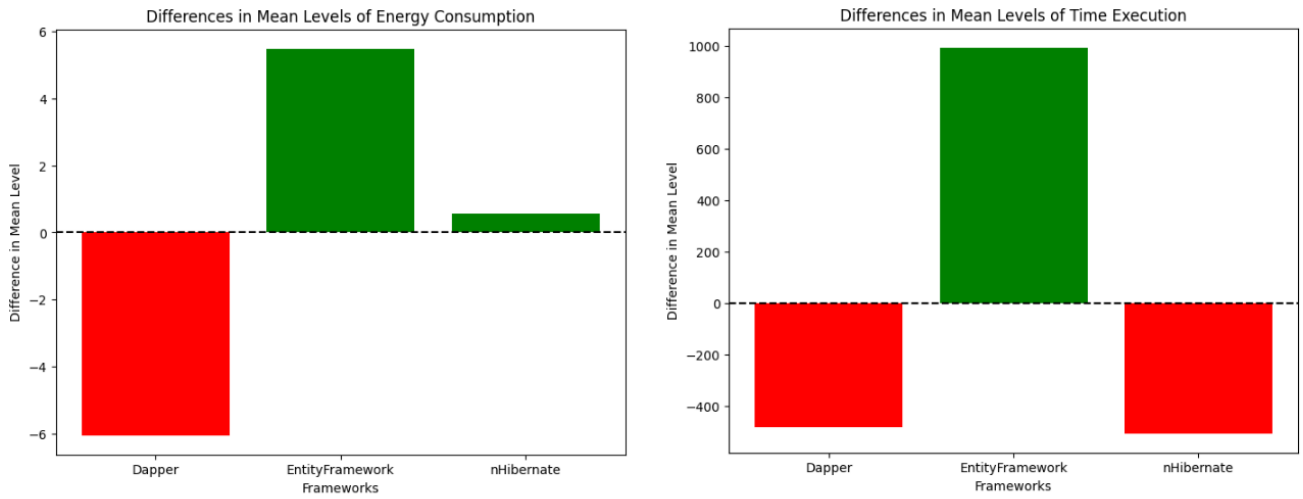

Figure 4: (a) Differences in mean levels of Energy Consumption per framework. (b) Differences in mean levels of Execution time per framework.

ing that Entity Framework enjoys widespread popularity in the development community and offers a rich set of features and functionalities. Its user-friendly interface and extensive tooling support make it still an attractive choice for developers seeking convenience and productivity in their projects. Therefore, when choosing a framework, developers and engineers should consider the trade-off between performance, energy consumption, popularity, and functionality. By selecting frameworks with better energy consumption and comparable execution time, they can strike a balance between efficient performance, energy efficiency, functionality, and the practical considerations of popularity and ease of use.

The analysis of the two plots in Figure 5 sheds light on the performance variations among different operations within the studied ORM frameworks (Dapper, Entity Framework, nHibernate). The plot Figure 5a highlights a significant disparity in energy consumption between the "Get" operation and the other operations (GetById, Create, Update, and Delete) among all the frameworks. The "Get" operation exhibits noticeably higher energy consumption, which can be attributed to the retrieval of data from larger tables. In contrast, the energy consumption levels for the remaining operations remain relatively consistent. Similarly, the plot Figure 5b unveils that the "Get" operation has the longest execution time, followed by the "Delete" operation, while the execution times for the other operations are comparable. These findings underscore the importance of optimizing the energy consump-

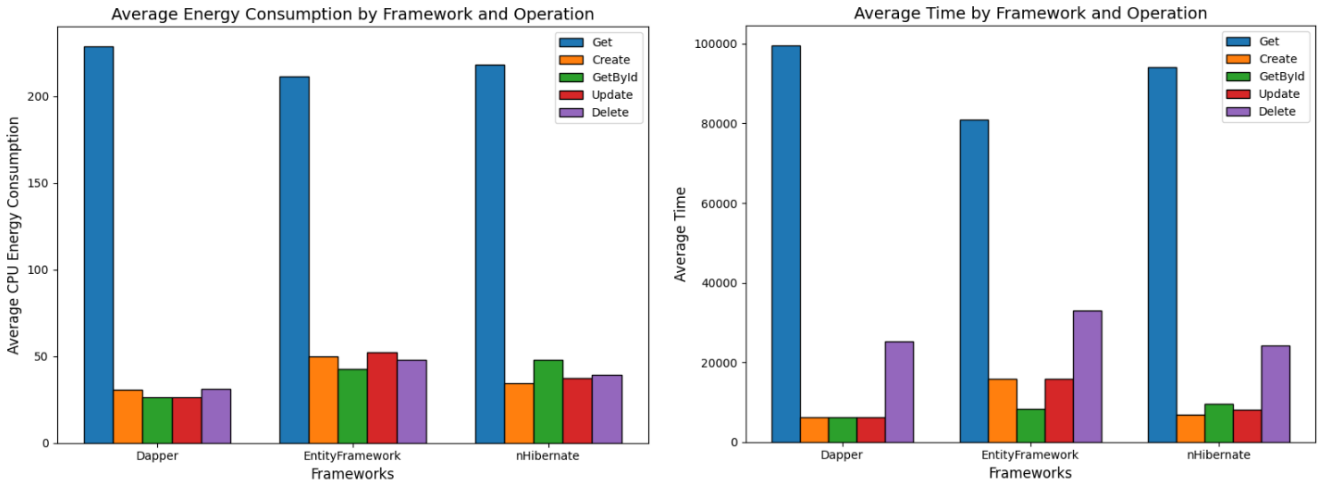

Figure 5: (a) Average Energy Consumption in J per framework and operation (b) Average Execution time in ms per framework and operation

tion and execution time of the "Get" operation.

The analysis of the two additional plots in Figure 6 provides valuable insights into the energy consumption and execution time variations among different tables within the studied ORM frameworks. In the plot Figure 6a, the energy consumption levels across the tables vary significantly. Notably, the "Locations" table exhibits lower energy consumption across all three frameworks compared to the "Purchases" and "Addresses" tables. This discrepancy suggests that the "Locations" table demonstrates superior energy efficiency in terms of resource utilization, possibly due to its smaller data volume. Similarly, in the plot Figure 6b, the "Locations" table demonstrates faster execution time across all frameworks, while the "Purchases" and "Addresses" tables exhibit slightly longer execution times. These findings imply that the "Locations" table offers better performance in terms of execution time, which can be attributed to the reduced data volume and complexity within the table.

The analysis of the additional two plots in Figure 7 provides valuable insights into the energy consumption and execution time variations across different operations and tables. The first plot Figure 7a reveals intriguing trends in average energy consumption by table and operation. In the "Locations" table, the energy consumption remains relatively consistent across all operations, suggesting that table size has minimal impact on energy consumption. However, for the "Purchases" and "Addresses" tables, energy consumption for "Get" operation increases significantly as the table size grows larger, while the energy consumption for the other operations

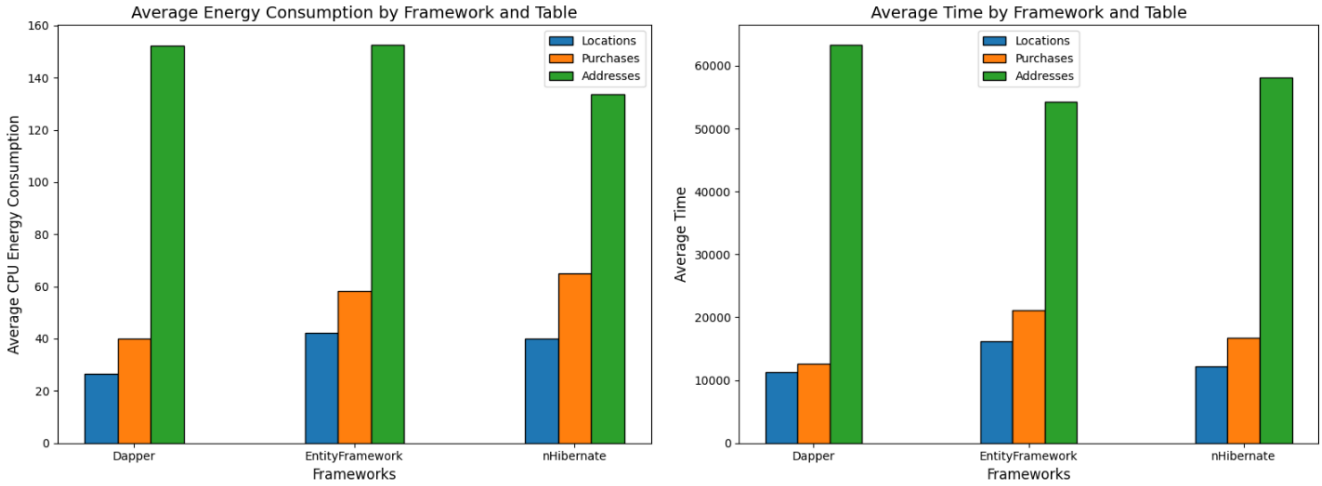

Figure 6: (a) Average Energy Consumption in J per framework and table (b) Average Execution time in ms per framework and table

remains relatively stable. This finding indicates that the amount of data stored in a table influences energy consumption, with larger tables consuming more energy during data retrieval. Moving on to the second plot Figure 7b, the trends align with the first graph Figure 7a, with one intriguing exception. The execution time of the "Delete" operation for the "Locations" table stands out as notably higher compared to the other tables, despite the similar execution times of other operations across all tables. This unexpected finding suggests that there may be specific factors or complexities associated with the "Delete" operation in the "Locations" table that resulted in longer execution time. These findings underscore the significance of considering table size and operation type when analyzing energy consumption and execution time in ORM frameworks. By optimizing data retrieval processes and addressing potential inefficiencies in specific operations and table combinations, developers can enhance energy efficiency and optimize performance in their applications.

In the domain of performance evaluation for .NET Object Relational Mappers (ORMs), previous studies have primarily focused on comparing time performance and memory consumption, providing insights into query execution, transaction management, and data processing. However, these studies have often overlooked the energy efficiency implications of ORM frameworks. In contrast, this research addresses this gap by conducting a comprehensive evaluation of both time performance and energy consumption, specifically focusing on the newer versions of .NET and selected ORM frameworks. By consid-

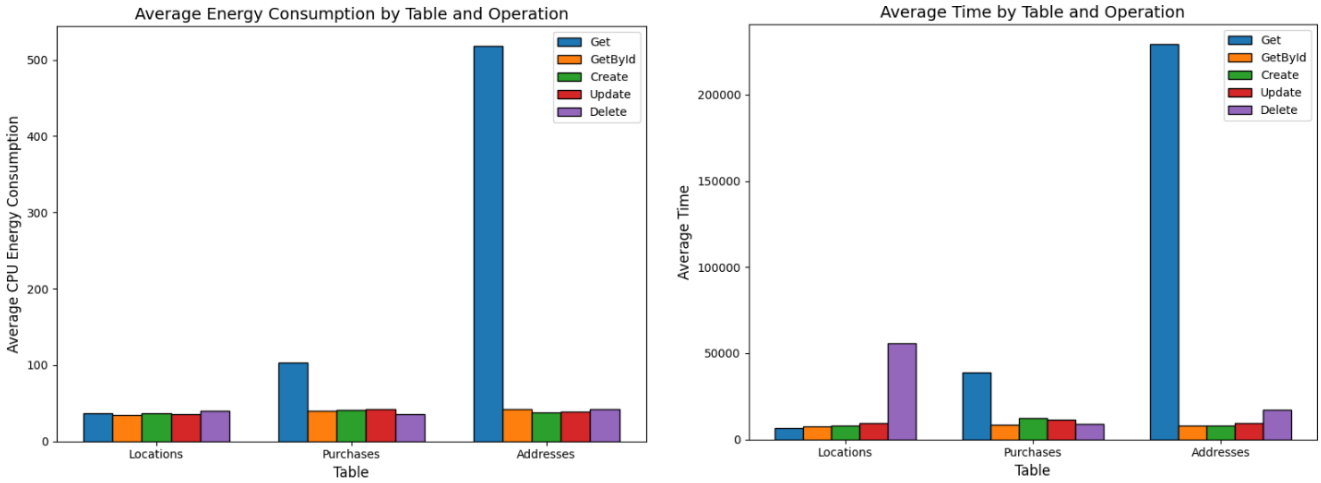

Figure 7: (a) Average Energy Consumption in J per operation and table (b) Average Execution time in ms per operation and table

ering the latest advancements in technology, this study provides valuable guidance for developers in selecting ORM frameworks that optimize both performance and energy efficiency, ensuring the use of efficient and up-to-date tools in software development.

The findings of this research have significant implications for both academia and industry. By meticulously examining the energy efficiency implications of widely embraced ORM frameworks within the .NET landscape, this study adds valuable insights to the existing knowledge base. The comprehensive analysis of energy consumption, execution time, and other relevant metrics sheds light on the performance characteristics and trade-offs associated with different frameworks, query types, and table sizes. These insights can be applied in practice by software engineers and developers to make informed decisions regarding ORM framework selection and data management practices, ultimately optimizing energy efficiency in software systems. Furthermore, the potential impact of these findings extends to relevant stakeholders, including software development companies, IT departments, and sustainability-focused organizations, who can leverage the knowledge gained from this study to enhance energy-efficient software development processes and contribute to broader sustainability goals in the field.

While this research provides valuable insights into the energy efficiency implications of ORM frameworks, there are several areas that warrant further investigation. Firstly, future studies could delve deeper into the specific factors influencing energy consumption and execution time,

such as the underlying hardware configurations, database optimizations, or network latency. Additionally, the impact of different database management systems (DBMS) on the performance and energy efficiency of ORM frameworks could be explored. Furthermore, considering the evolving nature of software development and the emergence of new frameworks, it would be valuable to conduct longitudinal studies to assess the long-term energy performance of ORM frameworks and track any changes or improvements over time. Finally, investigating the scalability of energy consumption and execution time across different system sizes and workloads could provide insights into the performance characteristics in real-world scenarios. These potential areas of investigation would build upon the findings of this study and contribute to a more comprehensive understanding of energy efficiency in software systems.

7 Threats to validity

Threats to validity are an important consideration in this study. The threats are classified according to validity types following the structure proposed by Wohlin et al. [16].

7.1 Construct validity

During the research, potential threats to construct validity were identified, particularly in relation to the energy consumption and execution time of database queries. One such threat involved the influence of internet connectivity, specifically in scenarios with remote data access or online resources. Unreliable or unstable internet connections could introduce variability in query execution time and data retrieval, potentially impacting the accuracy and consistency of the measured constructs.

To mitigate this threat, measures were taken to maintain a stable and reliable internet connection throughout the experiments, minimizing the potential impact of internet connectivity on the results. Additionally, the Running Average Energy Limit (RAPL) interface was utilized as a standardized mechanism for monitoring and controlling the energy consumption of the systems under test. Although a calibration of the RAPL interface was not conducted, it provided valuable insights into the energy efficiency of the ORM frameworks.

By addressing the potential influence of internet connectivity and leveraging the RAPL interface, the research aimed to enhance the accuracy and reliability of the measurements, strengthening the construct validity. These steps were taken to ensure valid and meaningful conclusions regarding the energy consumption and execution time of the ORM frameworks under investigation.

7.2 Internal validity

Threats to internal validity have been meticulously addressed in this study. Given the absence of randomization techniques, the analysis employed robust non-parametric tests that accounted for deviations from normality. These alternative tests ensured the reliability of the inferential analysis and bolstered the internal validity of the study.

In an effort to reduce the impact of any potential confounding variables, the experimental environment underwent close monitoring. Concerted efforts were made to identify and manage any elements that could distort the outcomes. This rigorous strategy enhances the internal validity of the study by asserting a clear correlation between the observed effects and the specific factors under investigation.

While the reuse of the same samples for assessing different factors and treatments may introduce a potential threat to internal validity, steps were taken to mitigate this concern. Each experimental object was meticulously restored to its initial state before applying each treatment, minimizing the influence of prior measurements. Additional measures, such as the disabling of caching, were implemented to ensure an impartial and equitable comparison between the different ORM systems.

Considering these rigorous measures, it is evident that the potential threats to internal validity have been thoroughly acknowledged and effectively addressed in this master thesis. This comprehensive approach bolsters the internal validity of the study and instills confidence in the validity and reliability of the findings.

7.3 Conclusion validity

The thorough examination of this research study highlights several important aspects that need to be considered when evaluating the conclusion validity of the findings. Despite the meticulous approach and robust methodology employed, it is crucial to acknowledge the potential threats that could impact the reliability and generalizability of the conclusions.

The potential for measurement error also needs to be acknowledged. Despite rigorous efforts to minimize measurement error through instrument configuration and repeated measurements, some degree of variability or measurement error may still exist. It is important to recognize and account for these factors to ensure the accuracy and validity of the conclusions drawn. The reliability and validity of the measurement instruments used in the study should be carefully evaluated to ascertain the robustness of the findings.

Another critical aspect to consider is the representativeness of the selected samples. In this study, convenience sampling was employed, which may introduce potential biases and limit the generalizability of the findings. It is vital to acknowledge the limitations associated with convenience sampling and recognize that the conclusions drawn from these samples may not fully capture the variability and diversity present in the larger population. Thus, the findings should

be cautiously applied to other populations or contexts, and the limitations of the sample representativeness should be transparently discussed.

The method used for assigning treatments or factors to the experimental units is an additional factor that could influence the conclusion validity. Randomization or systematic allocation methods were not utilized in this study. While this decision was made based on practical considerations, it is important to acknowledge the potential biases or limitations introduced by the chosen assignment method. The potential impact of the assignment method on the validity of the conclusions should be openly recognized and discussed.

By thoroughly addressing these potential threats to conclusion validity, this research study strives to provide a comprehensive evaluation of the limitations and potential biases that may affect the conclusions drawn. The openness and transparency in discussing these considerations contribute to the overall strength and reliability of the conclusions, allowing readers to critically evaluate the implications of the findings in different contexts.

7.4 External validity

External validity also played a role in this research, as it aimed to assess the generalizability of the findings beyond the specific research context. Several potential threats to external validity were acknowledged and addressed. One such threat was the need to ensure the representativeness of the sample used in the study. Careful consideration was given to selecting a sample that accurately reflected the characteristics and diversity of the target population, enhancing the external validity of the findings.

Another important consideration was the applicability of the treatments or interventions implemented in the research. It was recognized that the effectiveness and feasibility of these treatments could be influenced by the specific conditions and resources available in the chosen research setting. Thus, the research took into account the contextual factors and carefully assessed the transferability of the treatments to different settings, strengthening the external validity of the study.

By addressing these potential threats and diligently considering the representativeness of the sample and the applicability of the treatments or interventions, the research aimed to enhance the external validity of the findings. This approach ensured that the results could be more confidently generalized to broader contexts and provided meaningful insights beyond the specific research setting.

8 Conclusion

In conclusion, this master thesis examined the energy efficiency implications of three ORM frameworks in the .NET landscape: Dapper, Entity Framework, and nHibernate. The findings provide valuable insights for software engineers in selecting an ORM framework and optimizing energy efficiency in their applications.

Noticeable differences in energy consumption and execution time were observed among the studied frameworks. Dapper exhibited better energy consumption, while Entity Framework showed higher execution time and energy consumption, despite its popularity and extensive tooling support.

The analysis of query types and table sizes highlighted the impact on energy consumption and execution time. The "Get" operation demonstrated higher energy consumption and longer execution time, especially for larger tables, emphasizing the need for optimization in data retrieval processes.

Threats to validity were considered, and measures were taken to enhance the validity and reliability of the study. This research contributes to the existing knowledge base and offers practical insights for software engineers and development companies.

Further research could explore additional factors, such as hardware configurations and database management systems. Longitudinal studies and scalability analyses would provide a more comprehensive understanding of performance and energy efficiency.

Reflecting on the context of data centers as brought up in the Introduction, the energy efficiency considerations of ORM frameworks hold even more significance. With projections indicating a rise in power usage of data centers, improving energy efficiency at the software level becomes a vital contribution. Improved energy efficiency in ORM frameworks can help mitigate the growing power demands of data centers, subsequently leading to reduced operating costs and environmental impact. This study, thus, not only serves to guide the choice of ORM frameworks for software engineers but also provides a valuable perspective in the broader context of sustainable and energy-efficient practices in the digital sector.

In conclusion, this master thesis provides valuable guidance for selecting ORM frameworks and optimizing energy consumption in software systems. By applying these insights, software engineers can make informed decisions and contribute to sustainable software development practices.

References

[1] Adventureworks2019, 2023. Microsoft. (n.d.). AdventureWorks2019. Retrieved May 18, 2023, from https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16tabs=ssms.

[2] Dapper, 2023. (n.d.). NuGet. Retrieved May 18, 2023, from https://www.nuget.org/packages/Dapper.

[3] Dapper: Dapper, 2023. (n.d.). GitHub Repository. Retrieved May 18, 2023, from https://github.com/DapperLib/Dapper.

[4] Entity framework, 2023. (n.d.). NuGet. Retrieved May 18, 2023, from https://www.nuget.org/packages/EntityFramework.

[5] Entity framework: Microsoft, 2023. (n.d.). Entity Framework. Retrieved May 18, 2023, from https://learn.microsoft.com/en-us/aspnet/entity-framework.

[6] nhibernate, 2023. (n.d.). NuGet. Retrieved May 18, 2023, from https://www.nuget.org/packages/NHibernate.

[7] nhibernate: Nhibernate community, 2023. (n.d.). nHibernate. Retrieved May 18, 2023, from https://nhibernate.info/.

[8] Anders SG Andrae. Hypotheses for primary energy use, electricity use and co2 emissions of global computing and its shares of the total between 2020 and 2030. *WSEAS Transactions on Power Systems*, 15:4, 2020.

[9] A.S.G. Andrae. Prediction studies of the electricity use of global computing in 2030. *International Journal of Science and Engineering Investigations*, 8:27–33, 2019.

[10] A.S.G. Andrae and T. Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6:117–157, 2015.

[11] Irakli Basheleishvili, Avtandil Bardavelidze, and Khatuna Bardavelidze. Study and analysis of the. net platform-based technologies for working with the databases. In *Proceedings of the 33rd International Conference on Information Technologies (InfoTech-2019), Bulgaria*, 2019.

[12] V. Basili and G. Caldiera. Rombach. *The goal question metric approach*, volume 2. Wiley, 1994.

[13] Suhas Chatekar. *Learning NHibernate 4*. Packt Publishing Ltd, 2015.

[14] Cheng Chen et al. Green databases through integration of renewable energy. *CIDR*, 2013.

[15] Tse-Hsun Chen et al. Detecting performance anti-patterns for applications developed using object-relational mapping. In *Proceedings of the 36th International Conference on Software Engineering*, 2014.

[16] Tse-Hsun Chen et al. An empirical study on the practice of maintaining object-relational mapping code in java systems. In *Proceedings of the 13th International Conference on Mining Software Repositories*, 2016.

[17] Stevica Cvetković and Dragan Janković. A comparative study of the features and performance of orm tools in a. net environment. In *Objects and Databases: Third International Conference, ICOODB 2010, Frankfurt/Main, Germany, September 28-30, 2010. Proceedings 3. Springer Berlin Heidelberg*, 2010.

[18] Ronald Aylmer Fisher. Design of experiments. *British Medical Journal*, 1(3923):554, 1936.

[19] Phillip I. Good. *Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses*. Springer Science and Business Media, 2013.

[20] Aleksandra Gruca and Przemysław Podsiadło. Performance analysis of. net based object–relational mapping frameworks. In *Beyond Databases, Architectures, and Structures: 10th International Conference, BDAS 2014, Ustron, Poland, May 27-30, 2014. Proceedings 10. Springer International Publishing*, 2014.

[21] Kashif Nizam Khan et al. Rapl in action: Experiences in using rapl for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 3:1–26, 2018.

[22] Willis Lang and Jignesh Patel. Towards eco-friendly database management systems. *arXiv preprint arXiv:0909.1767*, 2009.

[23] R.B. Mitchell and R. York. Reducing the web's carbon footprint: Does improved electrical efficiency reduce webserver electricity use? *Energy Research and Social Science*, 65, 2020.

[24] Giuseppe Procaccianti, Héctor Fernández, and Patricia Lago. Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 117:185–198, 2016.

[25] Giuseppe Procaccianti, Patricia Lago, and Wouter Diesveld. Energy efficiency of orm approaches: an empirical evaluation. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2016.

[26] Nornadiah Mohd Razali and Yap Bee Wah. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2.1:21–33, 2011.

[27] K. Roebuck. *Object-Relational Mapping: High-impact Strategies - What You Need to Know*. Lightning Source Publisher, 2011.

[28] K. M. Smith, S. Wilson, P. Lant, and M. E. Hassall. How do we learn about drivers for industrial energy efficiency—current state of knowledge. *Energies*, 15:2642, 2022.

[29] J. Song, T. Li, X. Liu, and Z. Zhu. Comparing and analyzing the energy efficiency of cloud database and parallel database. In *Advances in Computer Science, Engineering Applications*, pages 989–997. Springer, 2012.

[30] Witoon Wiphusitphunpol and Thitiporn Lertrusdachakul. Fetch performance comparison of object relational mapper in. net platform. In *2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2017.

[31] Zichen Xu. Building a power-aware database management system. In *Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research*, 2010.

[32] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Exploring power-performance tradeoffs in database systems. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010.

[33] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Online energy estimation of relational operations in database systems. *IEEE transactions on computers*, 64:3223–3236, 2015.

[34] Doina Zmaranda et al. Performance comparison of crud methods using net object relational mappers: A case study. *International Journal of Advanced Computer Science and Applications*, 11:1, 2020.

A Appendix - Additional Statistics

This appendix includes graphs and statistics that are integral to the thorough understanding of the research but were not included in the main body of the thesis. The reasoning behind this decision pertains to the large volume of data and the aim to maintain a succinct and focused narrative in the main discussion. However, for those desiring a deeper dive into the data, these supplementary figures and statistics offer an expansive view of the collected data and a more detailed understanding of the experiment's outcomes.

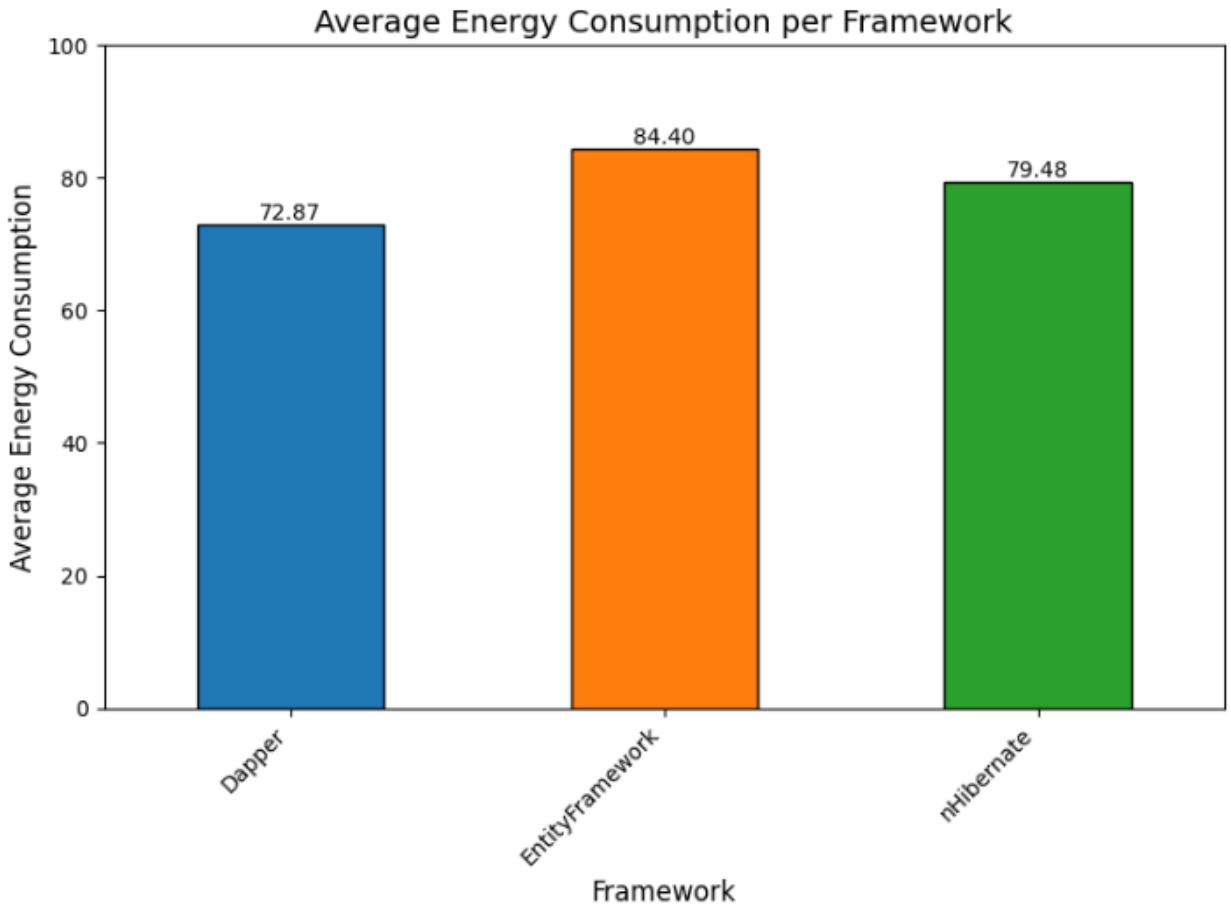

Figure 8: Average Energy Consumption per Framework

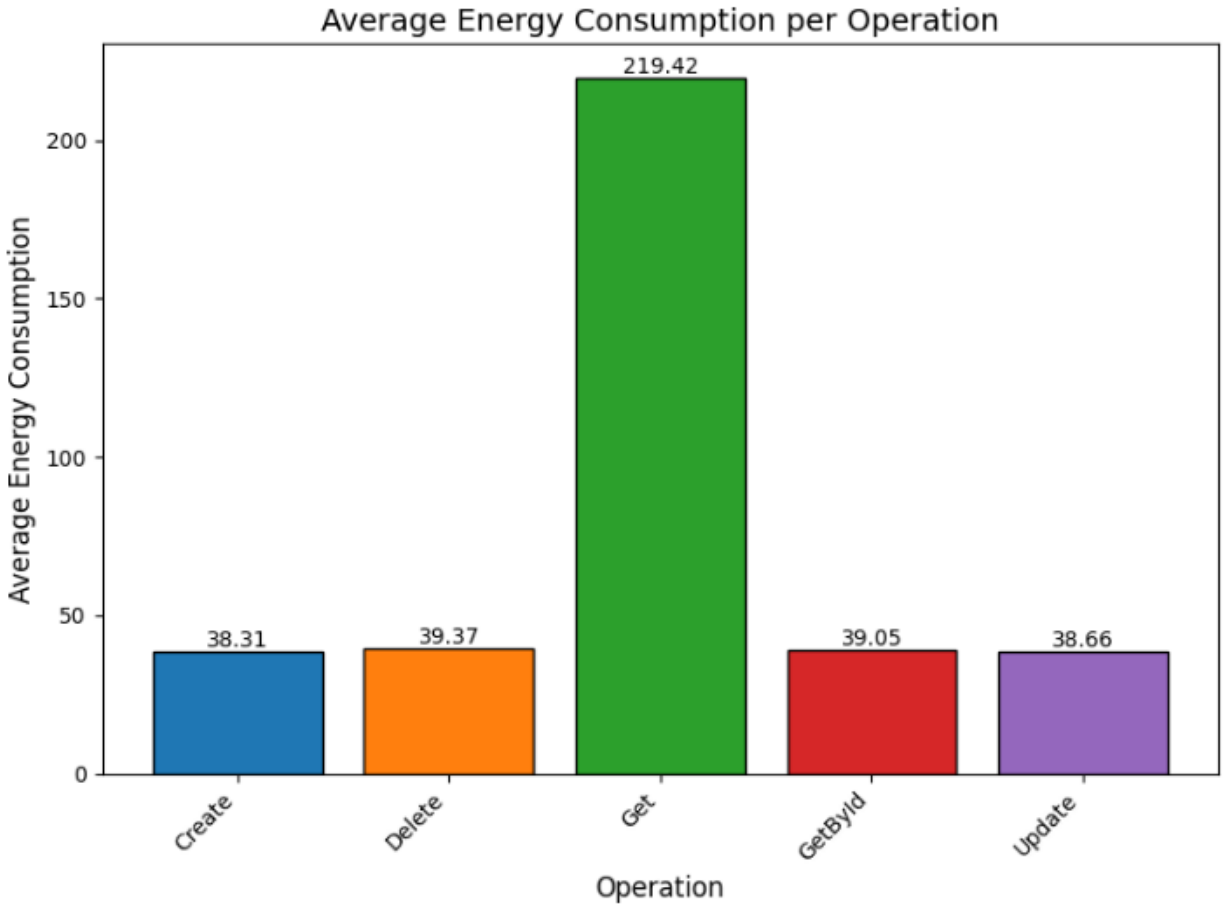

Figure 9: Average Energy Consumption per Operation

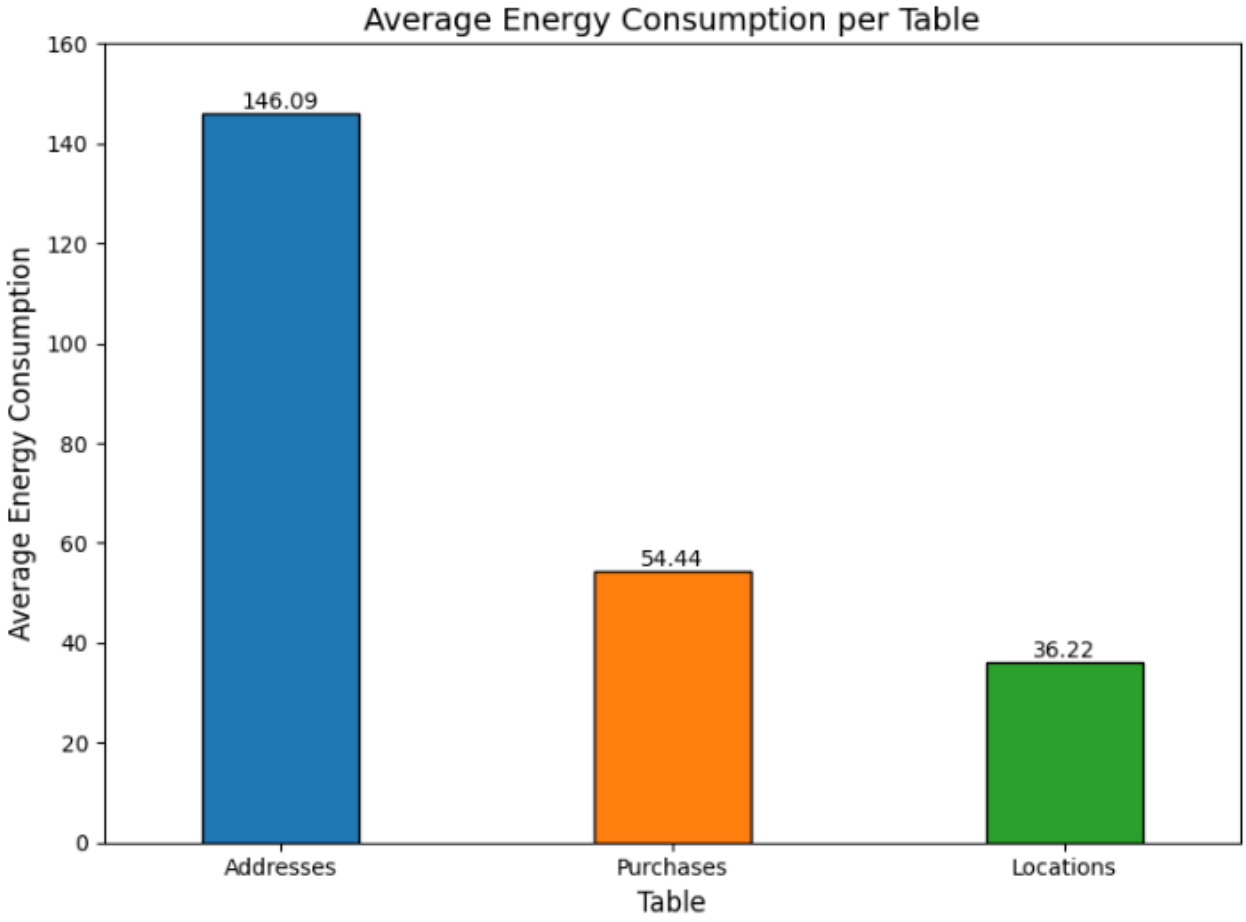

Figure 10: Average Energy Consumption per Table

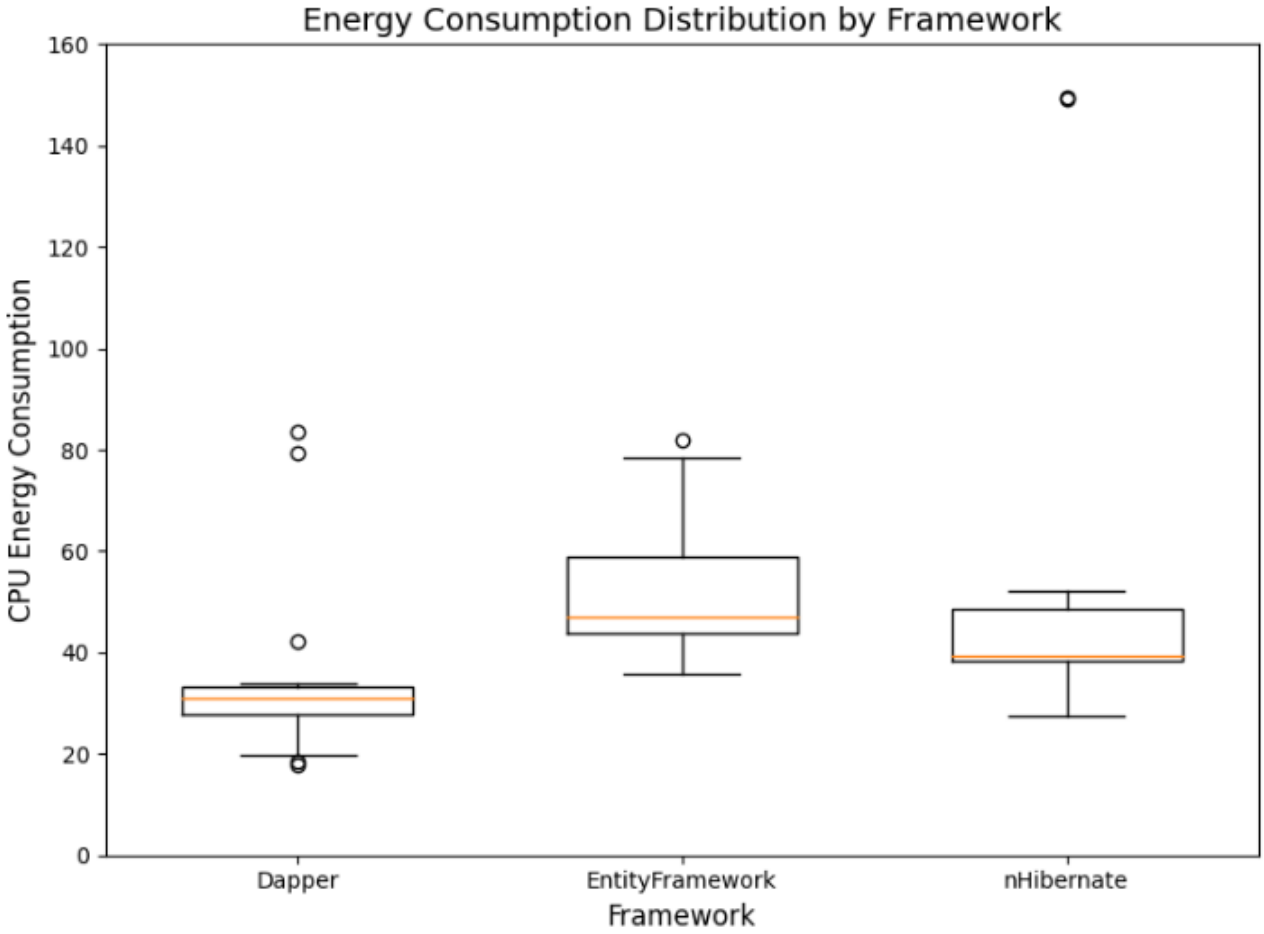

Figure 11: Energy Consumption Distribution By Framework

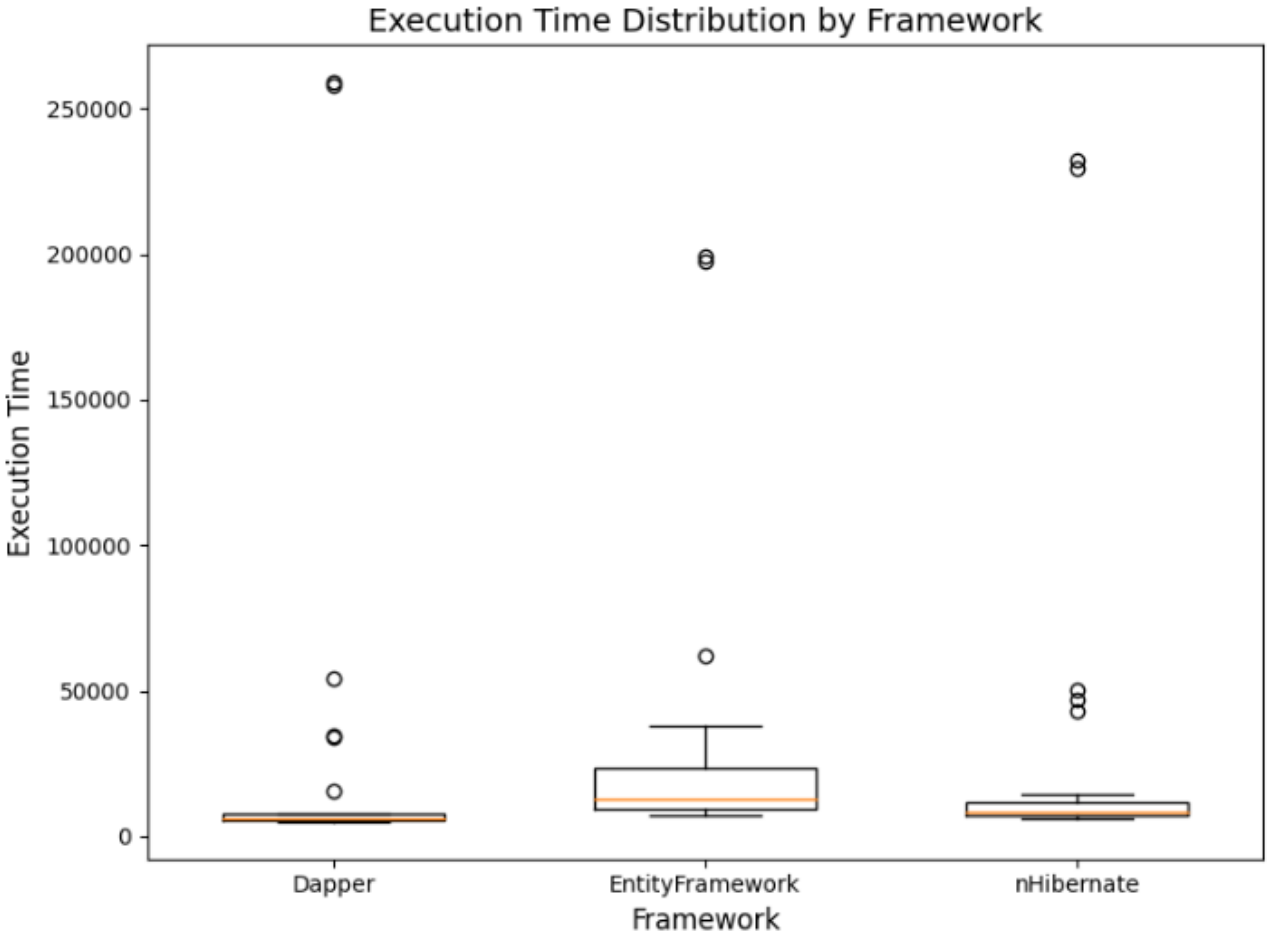

Figure 12: Execution Time Distribution By Framework

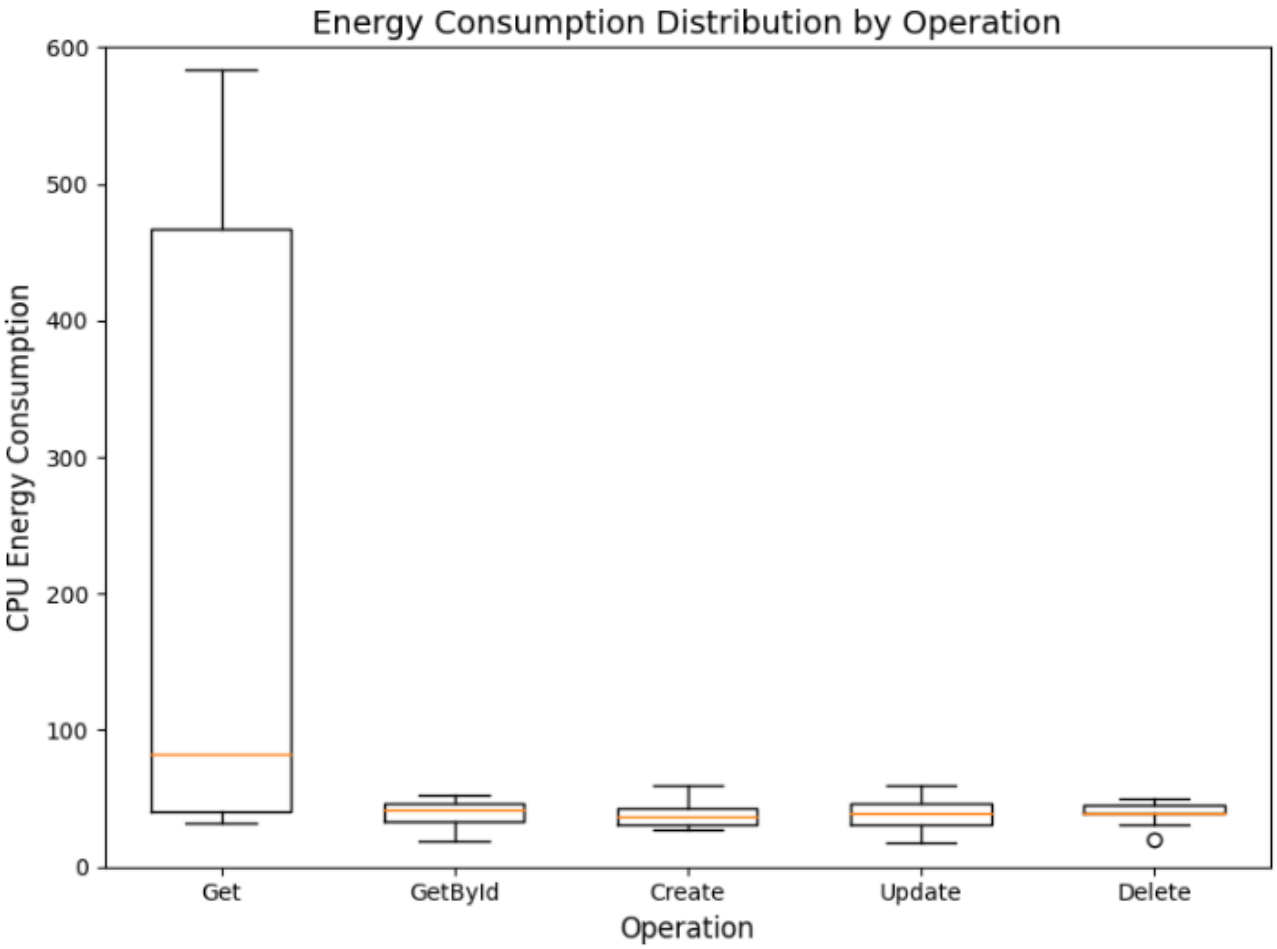

Figure 13: Energy Consumption Distribution By Operation

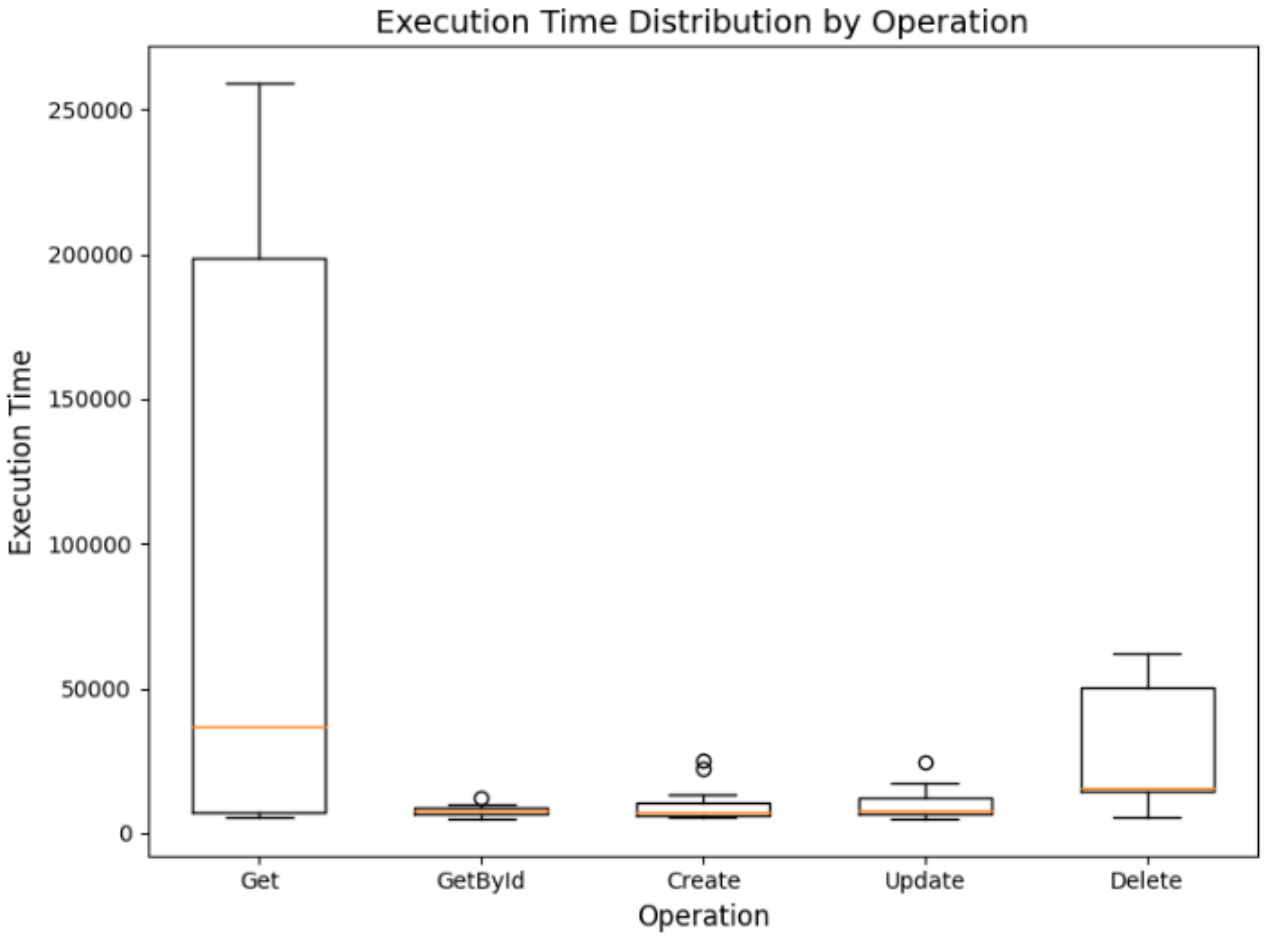

Figure 14: Execution Time Distribution By Operation

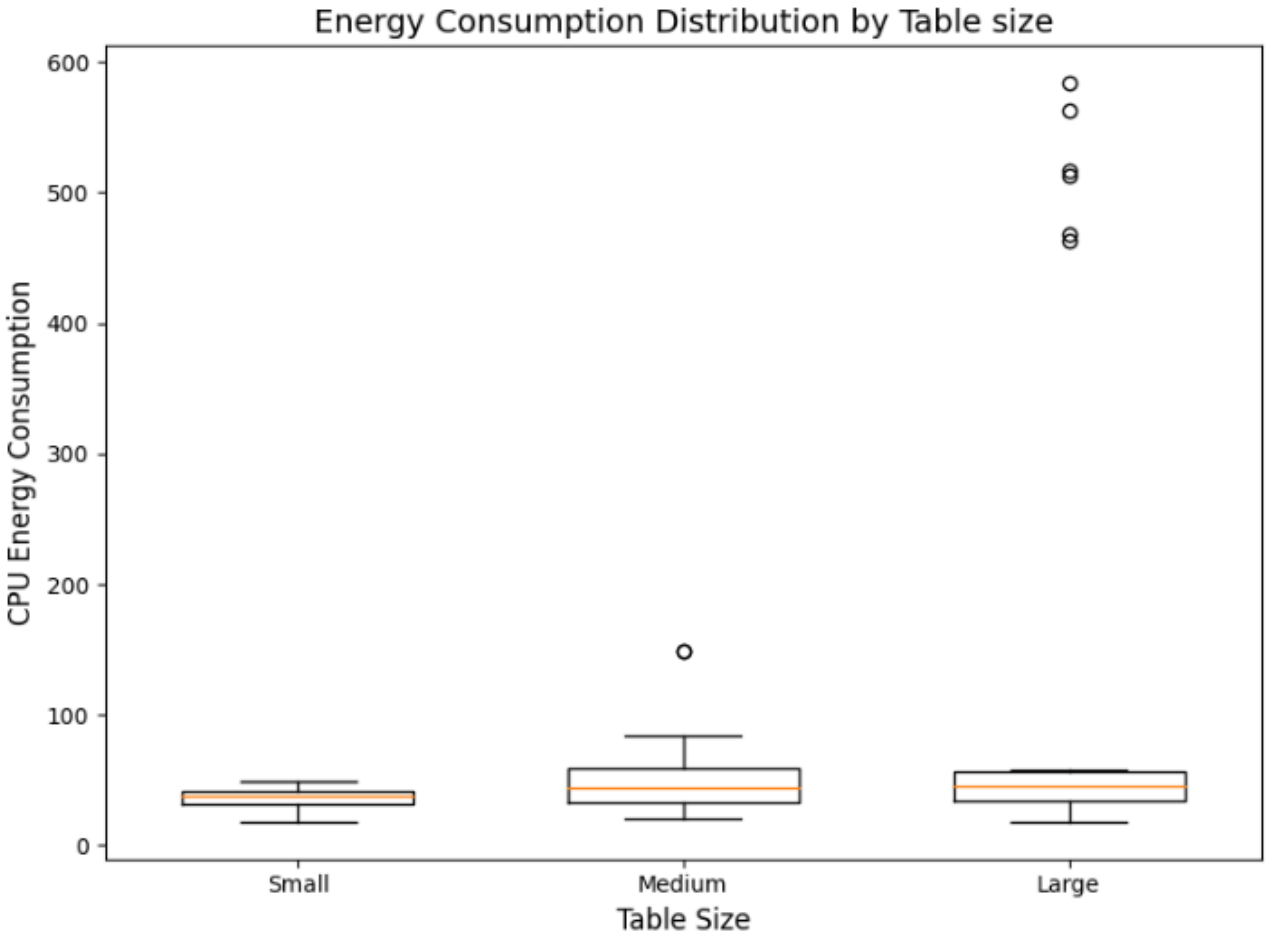

Figure 15: Energy Consumption Distribution By Table

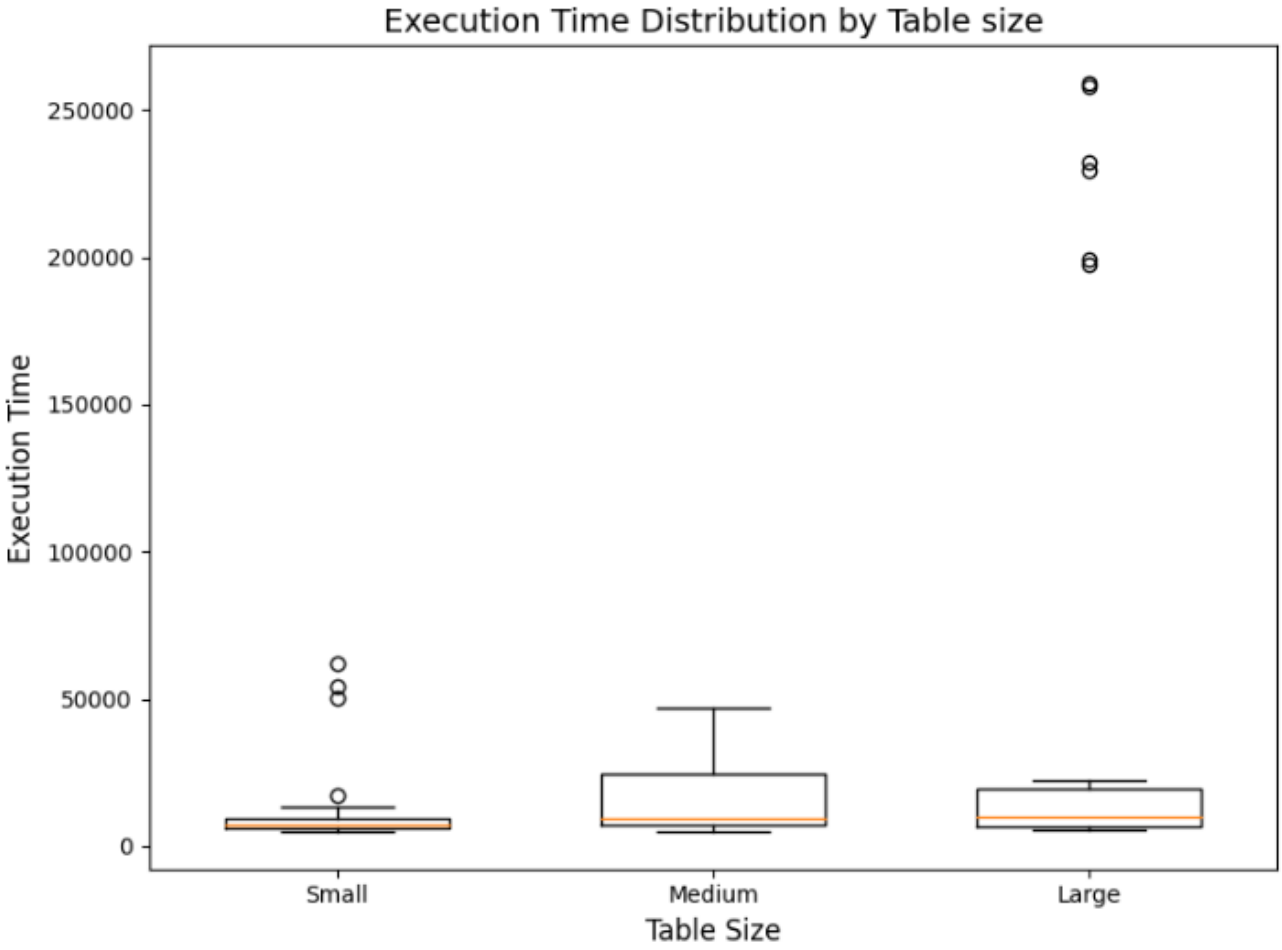

Figure 16: Execution Time Distribution By Table

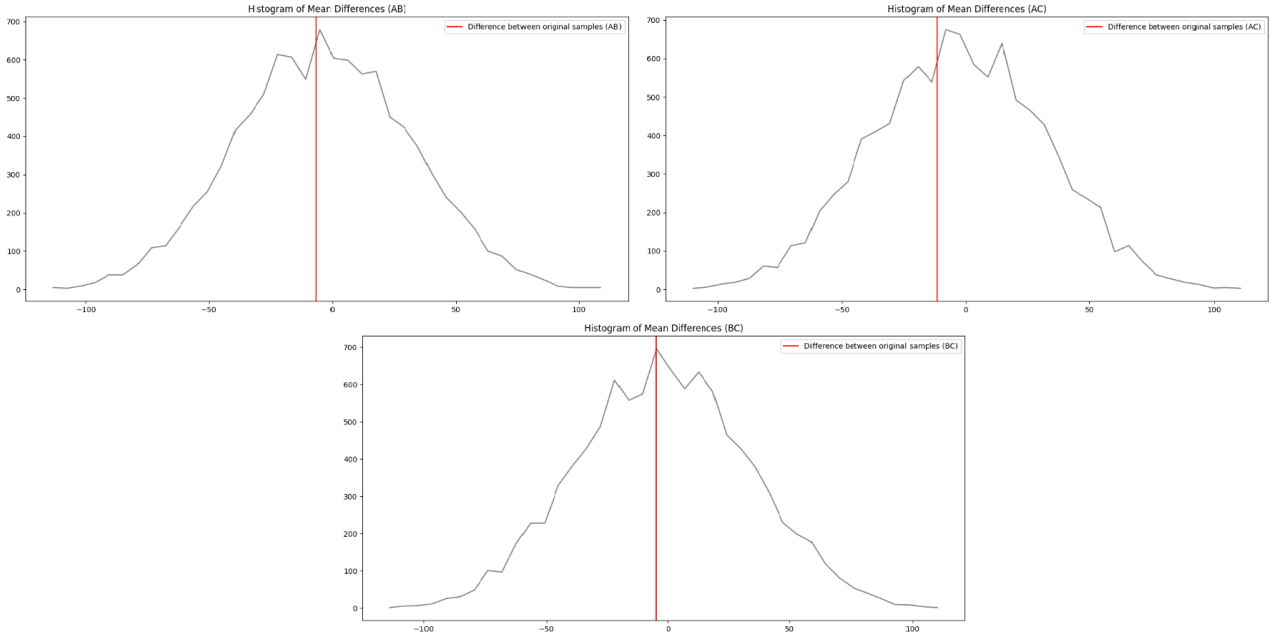

Figure 17: Mean Difference Energy Consumption By Framework

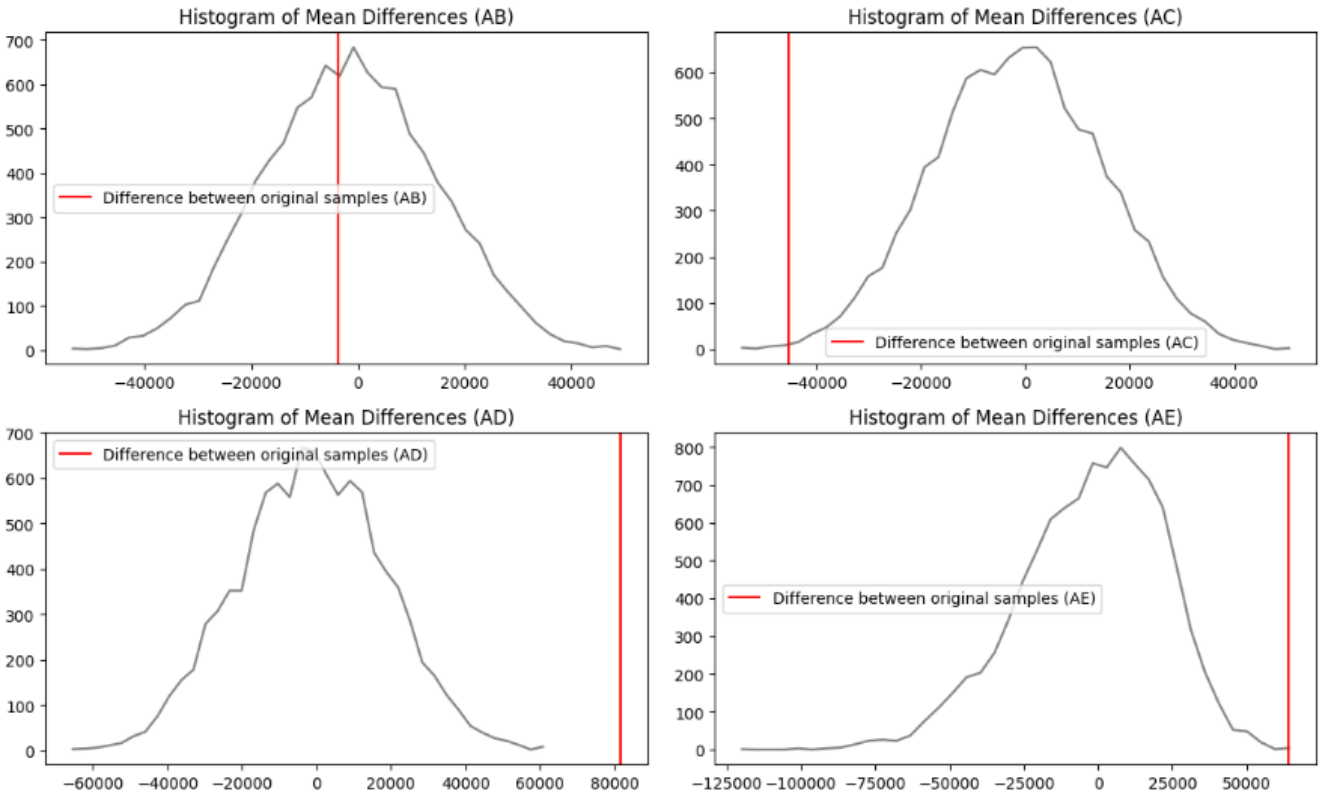

Figure 18: Mean Difference Energy Consumption By Operation

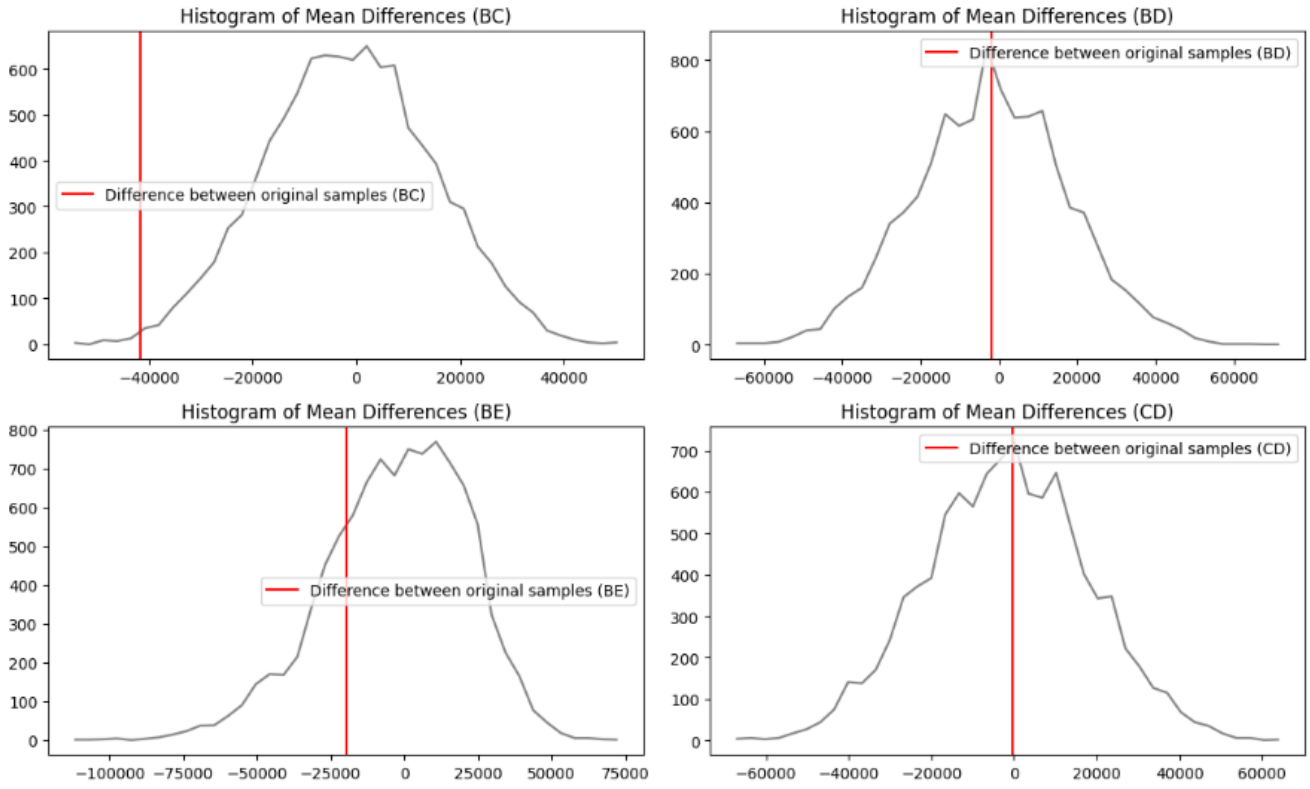

Figure 19: Mean Difference Energy Consumption By Operation

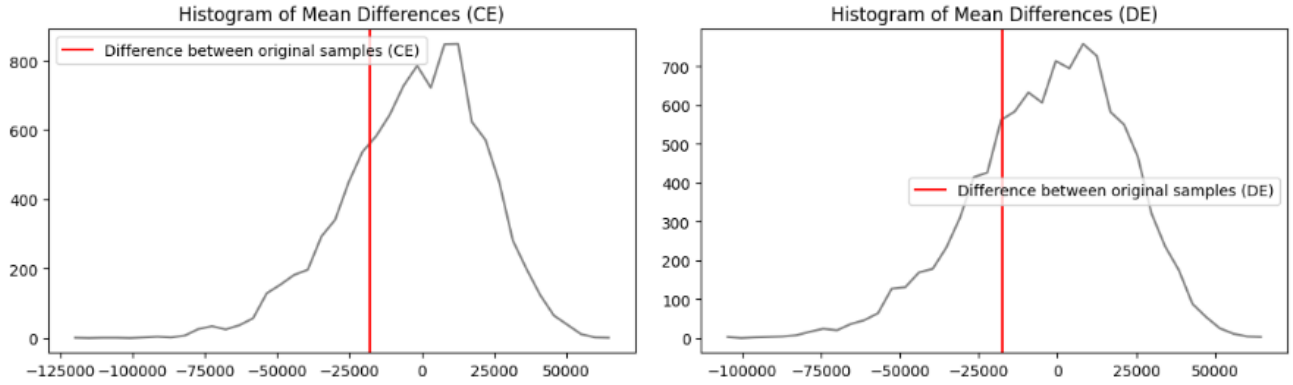

Figure 20: Mean Difference Energy Consumption By Operation

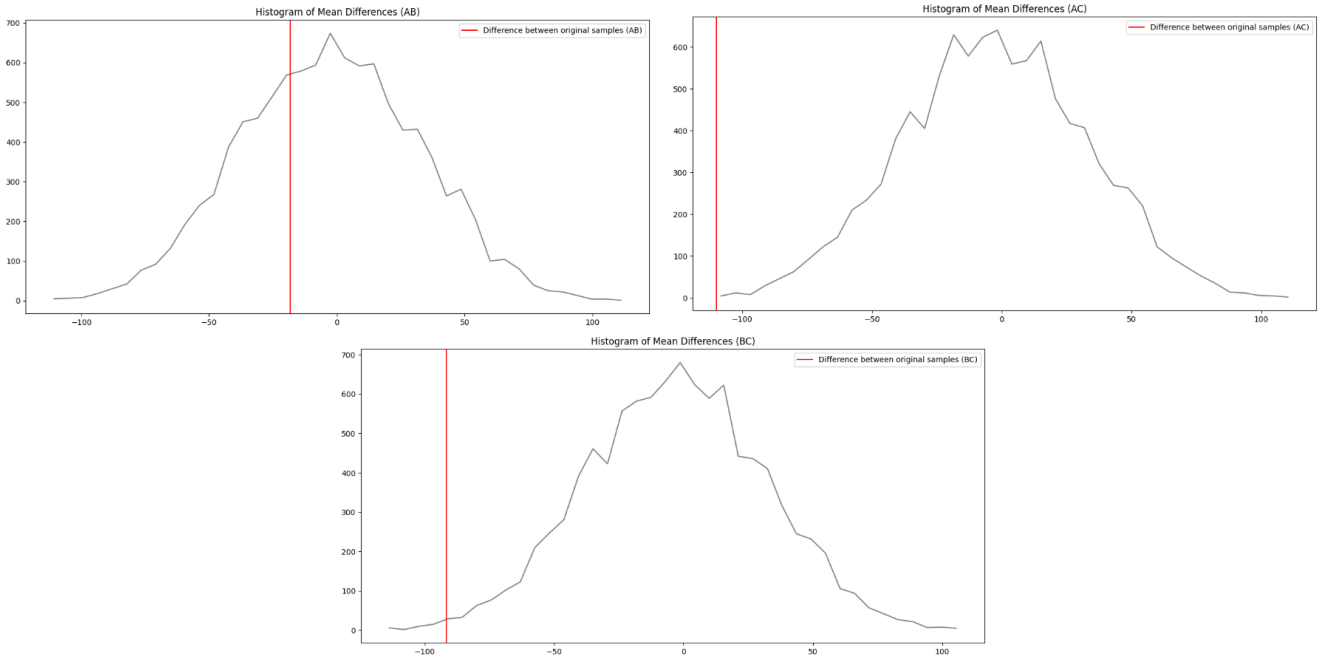

Figure 21: Mean Difference Energy Consumption By Table

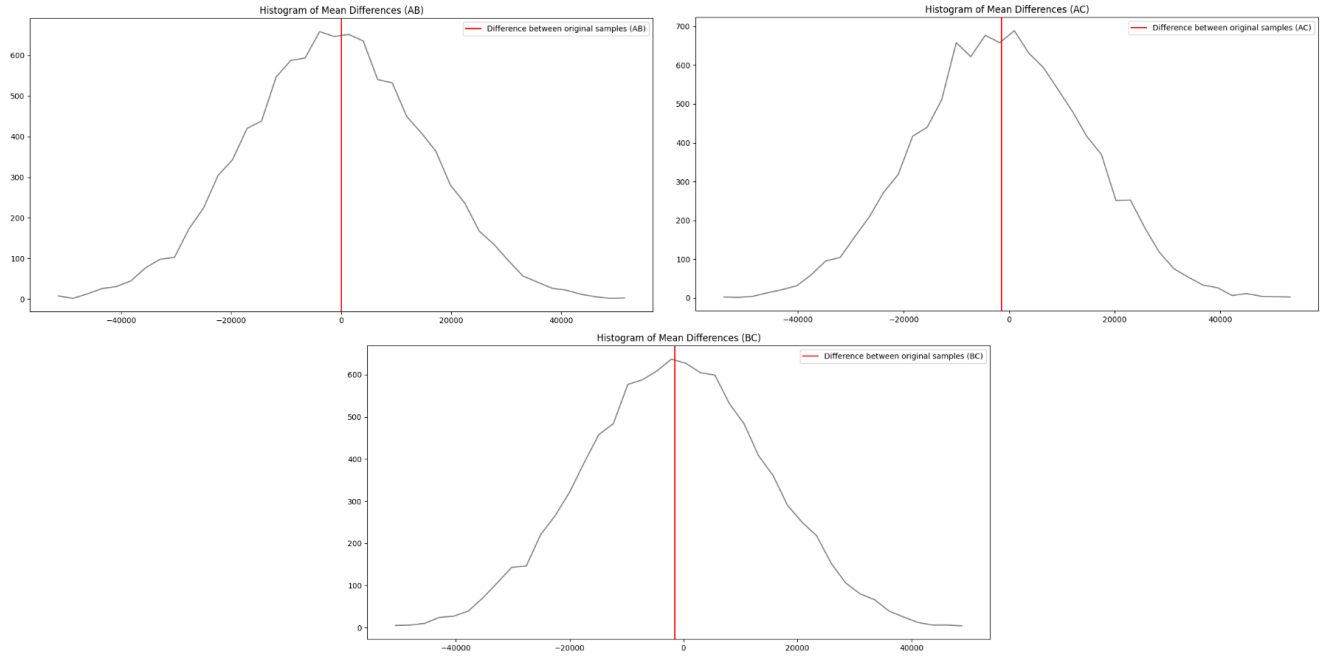

Figure 22: Mean Differences Execution Time By Framework

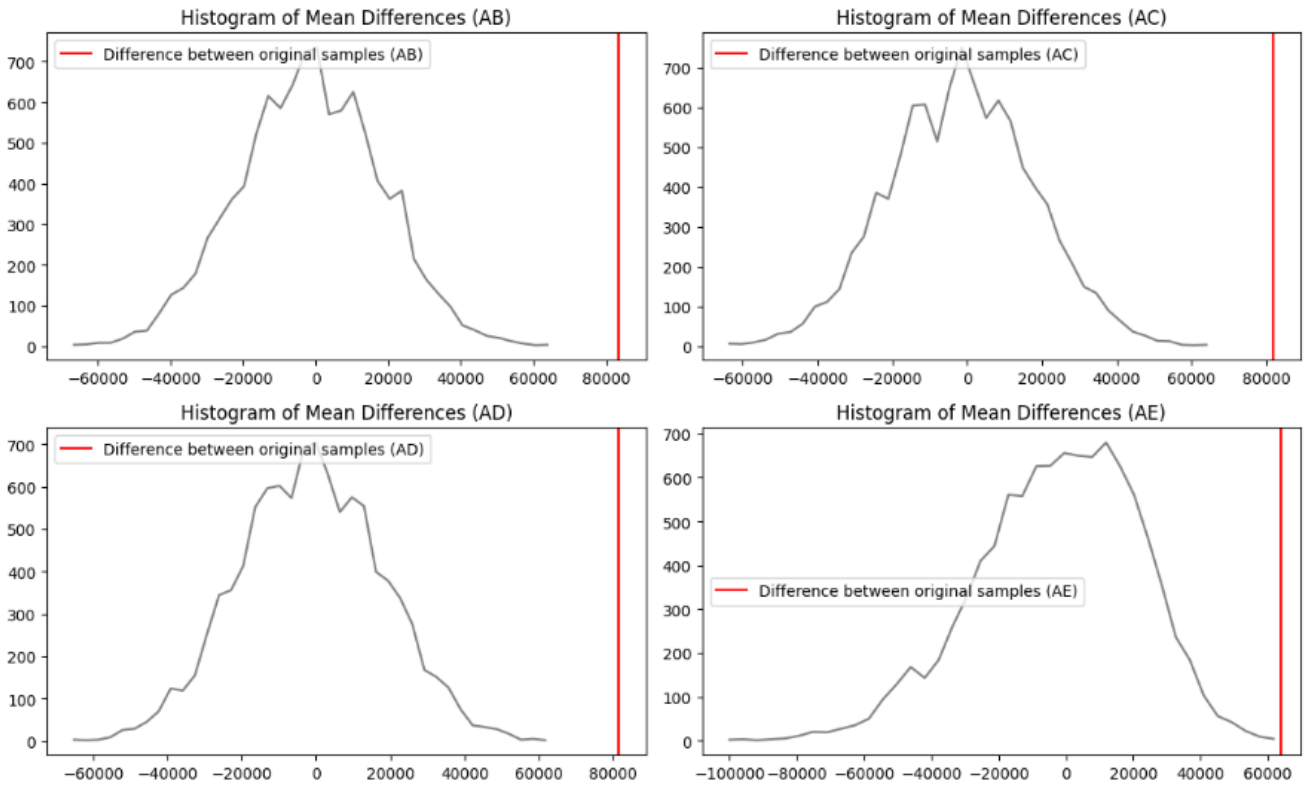

Figure 23: Mean Differences Execution Time By Operation

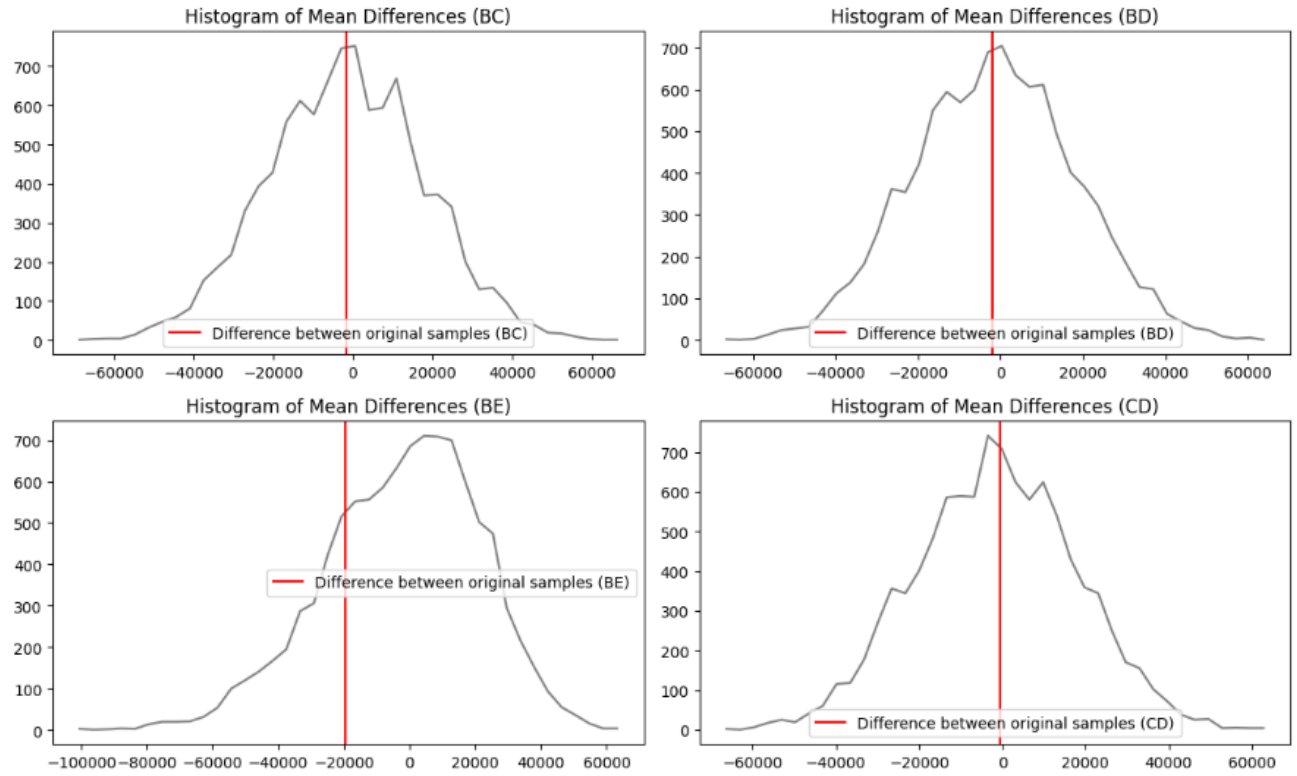

Figure 24: Mean Differences Execution Time By Operation

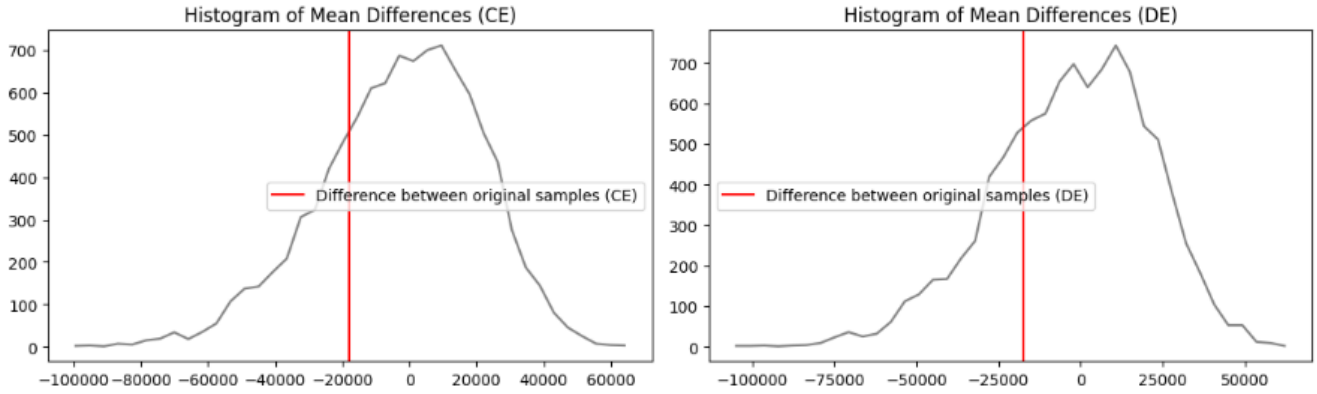

Figure 25: Mean Differences Execution Time By Operation

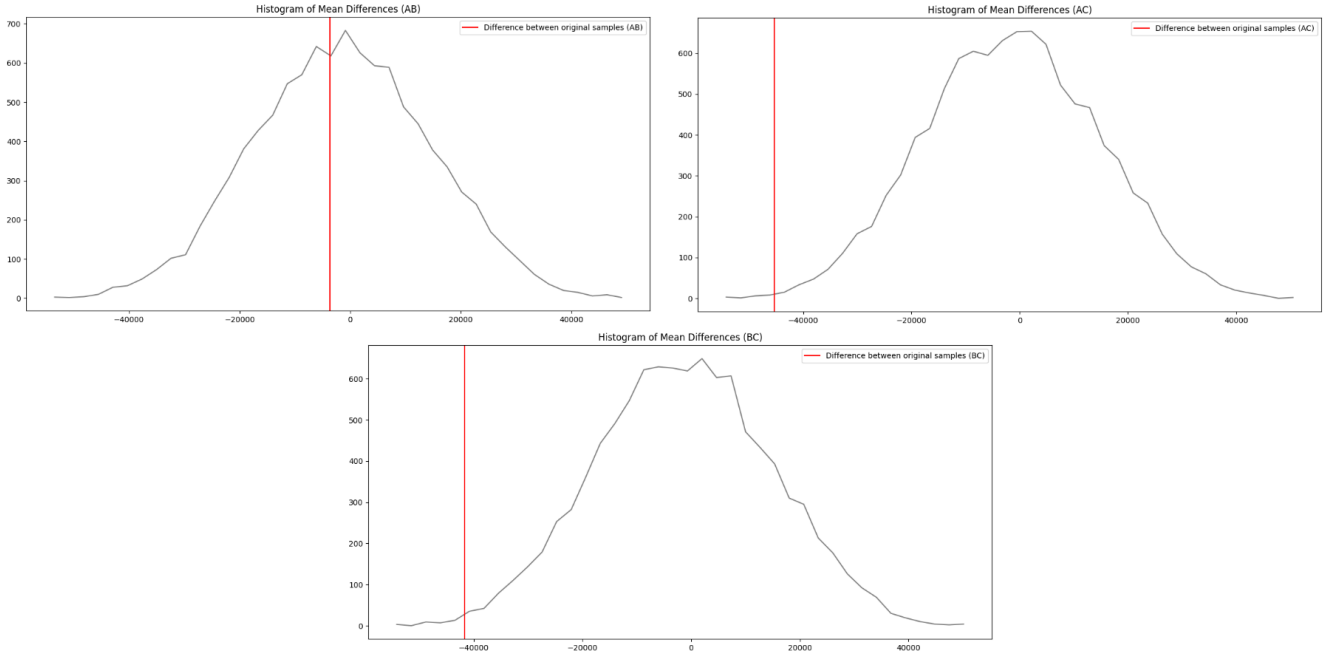

Figure 26: Mean Differences Execution Time By Table

B Appendix - Experiment code

The subsequent appendix presents a selection of code snippets taken from the practical component of this Master's thesis. These snippets are fundamental, demonstrating the implementation of three distinct object-relational mapping (ORM) frameworks: Dapper, Entity Framework, and nHibernate. Each code snippet highlights the specific methods used to perform various operations with these frameworks, thus providing an in-depth view of the research methodology. While the snippets cover significant portions of the implemented code, they encapsulate only parts of the entire codebase utilized during the research.

Listing 1: Dapper: Fetching All Locations

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    for (int i = 0; i < iterations; i++)
    {
        var results = connection.Query<Location>("SELECT * FROM Production.
            Location");
    }
    connection.Close();
}
```

Listing 2: Dapper: Fetching A Specific Location

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    for (int i = 1; i < iterations; i++)
    {
        var result = connection.QueryFirstOrDefault<Location>("SELECT * FROM
            Production.Location WHERE LocationID = @Id", new { Id = i });
    }
    connection.Close();
}
```

Listing 3: Dapper: Inserting Locations

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
```

```
    for (int i = 0; i < iterations; i++)
    {
        var insertSql = "INSERT INTO Production.Location (Name, CostRate,
            Availability, ModifiedDate) VALUES (@Name, @CostRate, @Availability,
             @ModifiedDate)";
        var insertParameters = new { Name = "Name", CostRate = 1, Availability =
             1, ModifiedDate = new DateTime(2023, 5, 18, 15, 30, 0) };
        var rowsAffected = connection.Execute(insertSql, insertParameters);
    }
    connection.Close();
}
```

Listing 4: Dapper: Updating Locations

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    for (int i = 1; i < iterations; i++)
    {
        var updateSql = "UPDATE Production.Location SET ModifiedDate =
            @ModifiedDate WHERE LocationID = @Id";
        var updateParameters = new { ModifiedDate = new DateTime(2023, 5, 19,
            15, 30, 0), Id = i };
        var rowsAffected = connection.Execute(updateSql, updateParameters);
    }
    connection.Close();
}
```

Listing 5: Dapper: Deleting Locations

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    for (int i = 1; i < iterations; i++)
    {
        var deleteSql = "DELETE FROM Production.Location WHERE LocationID = @Id"
            ;
        var deleteParameters = new { Id = i };
        var rowsAffected = connection.Execute(deleteSql, deleteParameters);
    }
    connection.Close();
}
```

Listing 6: Entity Framework Core: Fetching All Locations

```
for (int i = 0; i < iterations; i++)
{
    var results = _context.Locations.AsNoTracking().ToList();
}
```

Listing 7: Entity Framework Core: Fetching A Specific Location

```
for (int i = 1; i < iterations; i++)
{
    var result = _context.Locations.AsNoTracking().Single(x => x.LocationId ==
        id);
}
```

Listing 8: Entity Framework Core: Inserting Locations

```
for (int i = 0; i < iterations; i++)
{
    var l = new Location()
    {
        Availability = 1,
        CostRate = 1,
        ModifiedDate = new DateTime(2023, 5, 18, 15, 30, 0),
        Name = "Name"
    };
    _context.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.
        NoTracking;
    _context.Add(l);
    _context.SaveChanges();
}
```

Listing 9: Entity Framework Core: Updating Locations

```
for (int i = 1; i < iterations; i++)
{
    var row = _context.Locations.AsNoTracking().FirstOrDefault(r => r.LocationId
        == id);
    row.ModifiedDate = new DateTime(2023, 5, 19, 15, 30, 0);
    _context.SaveChanges();
}
```

Listing 10: Entity Framework Core: Deleting Locations

```
for (int i = 1; i < iterations; i++)
{
    var row = _context.Locations.AsNoTracking().FirstOrDefault(r => r.LocationId
        == id);
    _context.Locations.Remove(row);
    _context.SaveChanges();
}
```

Listing 11: NHibernate: Fetching All Locations

```
using (var session = sessionFactory.OpenSession())
using (var transaction = session.BeginTransaction())
{
    for (int i = 0; i < iterations; i++)
    {
        var locations = session.Query<Location>().ToList();
    }
    transaction.Commit();
}
```

Listing 12: NHibernate: Fetching A Specific Location

```
using (var session = sessionFactory.OpenSession())
using (var transaction = session.BeginTransaction())
{
    for (int i = 1; i < iterations; i++)
    {
        var query = session.Query<Location>().FirstOrDefault(l => l.LocationId
            == i);
    }
    transaction.Commit();
}
```

Listing 13: NHibernate: Inserting Locations

```
using (var session = sessionFactory.OpenSession())
using (var transaction = session.BeginTransaction())
{
    for (int i = 0; i < 3; i++)
    {
```

```
6           var location = new Location
7           {
8               Name = "Name",
9               CostRate = 1,
10              Availability = 1,
11              ModifiedDate = new DateTime(2023, 5, 18, 15, 30, 0)
12          };
13
14          session.Save(location);
15      }
16      transaction.Commit();
17  }
```

Listing 14: NHibernate: Updating Locations

```
1  using (var session = sessionFactory.OpenSession())
2  using (var transaction = session.BeginTransaction())
3  {
4      for (Int16 i = 1; i < iterations; i++)
5      {
6          var location = session.Get<Location>(i);
7
8          location.ModifiedDate = new DateTime(2023, 5, 19, 15, 30, 0);
9
10         session.Update(location);
11     }
12     transaction.Commit();
13 }
```

Listing 15: NHibernate: Deleting Locations

```
1  using (var session = sessionFactory.OpenSession())
2  using (var transaction = session.BeginTransaction())
3  {
4      for (Int16 i = 1; i < iterations; i++)
5      {
6          var location = session.Get<Location>(i);
7
8          session.Delete(location);
9      }
10     transaction.Commit();
11 }
```