



UNIVERSITY  
OF SKÖVDE

## Energy consumption of programming languages in machine learning

Comparing compiled and interpreted languages in the context  
of training machine learning models

Bachelor Degree Project in Information Technology  
Basic level 30 ECTS  
Spring 2025

Sol Milleding, Markus Carlsson

Supervisor: Simon Butler  
Examiner: Birgitta Lindström

# Abstract

As machine learning is seeing increased adoption, it is important to consider the sustainability of the technology. Training machine learning models can be resource-intensive, which may lead to high energy consumption. In order to achieve sustainable use, ways of reducing the energy consumption of machine learning is needed.

In this study, a quasi-experiment was conducted to compare the energy consumption of the interpreted language `Python`, and the compiled language `C++`, in the context of training machine learning models. The energy consumption of both execution and compilation was measured, while also considering the impact of compiler optimization levels.

The results showed that there were differences between interpreted and compiled languages in machine learning, however, the differences were smaller than found in previous research. There were also differences between compiler optimization levels, but some levels were more consistent than others. While certain patterns in energy consumption were seen, determining the most energy efficient programming language or optimization level was difficult. The study concluded that the energy consumption can be attributed to factors other than the programming language itself and varies between use-cases.

**keywords:** Machine learning, Programming languages, Compiled languages, Interpreted languages, Energy consumption

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	Machine Learning . . . . .	2
2.2	Programming Languages and Libraries . . . . .	2
2.3	Energy Consumption Measurement Tools . . . . .	3
2.4	Related Work . . . . .	3
<b>3</b>	<b>Problem</b>	<b>5</b>
3.1	Aim . . . . .	5
3.2	Motivation . . . . .	5
3.3	Research Questions . . . . .	5
3.4	Hypothesis . . . . .	6
3.5	Objectives . . . . .	7
<b>4</b>	<b>Method</b>	<b>8</b>
4.1	Alternative Method . . . . .	8
<b>5</b>	<b>Approach</b>	<b>9</b>
5.1	Programming Languages and Libraries . . . . .	9
5.2	Machine Learning Algorithms . . . . .	9
5.3	Machine Learning Implementations . . . . .	10
5.4	Measurement Tools . . . . .	10
5.5	Machine Learning Datasets . . . . .	10
5.6	Baseline Measurements . . . . .	11
5.7	Compiler Optimization . . . . .	12
5.8	Experiment Setup . . . . .	12
5.9	Statistical Significance . . . . .	14
<b>6</b>	<b>Result</b>	<b>15</b>
6.1	Data Processing . . . . .	15
6.2	Baseline Measurements . . . . .	15
6.3	Programming Languages . . . . .	17
6.4	Compiler Optimization Levels . . . . .	19

<b>7 Discussion</b>	<b>25</b>
7.1 Threats to Validity . . . . .	25
7.1.1 Construct Validity . . . . .	25
7.1.2 Internal Validity . . . . .	25
7.1.3 Conclusion Validity . . . . .	26
7.1.4 External Validity . . . . .	26
7.2 Programming Languages . . . . .	26
7.3 Compiler Optimization Levels . . . . .	27
7.4 Conclusion . . . . .	28
7.5 Sustainability . . . . .	29
7.6 Future Work . . . . .	29
<b>A Appendix - Optimization Level Tables</b>	<b>I</b>
<b>B Appendix - Optimization Level Figures</b>	<b>III</b>
<b>C Appendix - Programming Language Figures</b>	<b>X</b>

# 1 | Introduction

Software is deeply embedded in every aspect of society, at a scale we have never seen before. There is currently a surge of investment in new technologies like Artificial Intelligence (AI). Machine Learning (ML) is a field within AI used in both research and industry to accomplish tasks, such as, classifying information and predicting trends. ML has many applications and is seeing increased adoption in both industry and consumer products.

As ML becomes more popular, challenges with the technology are also becoming more apparent. Notably, training ML models can be resource-intensive. When considered at a scale, the energy consumption associated with ML can lead to high costs and environmental impact. As ML technology becomes more widely adopted, the issue of energy consumption is becoming increasingly important. Although there have been studies investigating the energy efficiency of ML, most research focus on performance metrics such as model accuracy (García-Martín et al., 2019).

In order to improve energy efficiency, it is important to explore different strategies that reduce energy consumption. One strategy is to compare the differences in energy efficiency between programming languages (Albonico et al., 2024; Georgiou et al., 2018; Gordillo et al., 2024; Koedijk and Oprescu, 2022; Camargo-Henríquez et al., 2024; Pereira et al., 2021). Another strategy is to compare compiler optimization levels (Abdulsalam et al., 2014). While both strategies have been explored in previous research, the energy consumption was only measured during execution, and not compilation.

Two languages types that have commonly been used in open-source ML development, are interpreted and compiled programming languages (Gonzalez et al., 2020). Interpreted languages have been shown to be less energy efficient compared to compiled languages outside the context of ML (Albonico et al., 2024; Georgiou et al., 2018; Gordillo et al., 2024; Koedijk and Oprescu, 2022; Pereira et al., 2021). However, interpreted languages were the most used in open-source ML development (Gonzalez et al., 2020). Therefore, a comparison of the two types of languages in the context of ML should be considered.

To fill the gap in research, a quasi-experiment was conducted to compare compiled and interpreted programming languages, in the context of training ML models. The comparison included the energy consumption associated with execution, compilation, and compiler optimizations. The results of this study could provide insight to developers, showing how the choice of programming language can affect their ML development process in terms of energy efficiency.

## 2 | Background

### 2.1 Machine Learning

Machine learning is a field within AI, commonly used in research and industries to process and analyse data. ML can itself be further divided into categories constituting different learning types. These learning types are utilized depending on the task and the nature of the data used for training. Two well-known learning types are supervised and unsupervised learning.

Supervised learning consists of algorithms used to train models on labelled data. Sarker (2021) provides an overview of ML techniques, algorithms, and their applications. The author divides supervised learning into classification and regression. The author states that the purpose of classification algorithms is to train models to predict classes, which is a set of pre-determined labels, while regression is used to predict continuous values.

There are a wide range of algorithms for both classification and regression, which use different approaches to make predictions. The classification algorithm decision tree, builds a tree of decision nodes by splitting the training data (Thai, 2022). Logistic regression instead predicts classifications based on probability (Sarker, 2021). A commonly used regression algorithm is linear regression, which uses the linear equation to fit a straight line through the training data, to predict continuous values (Sarker, 2021).

Unsupervised learning algorithms can be utilized to train models on unlabelled data. Sarker (2021) explains that they can be used for extracting features and discovering structures, trends, and groupings within the processed data. One common unsupervised method is clustering, which involves grouping related samples. Thai (2022) and Sarker (2021) identifies three subclassifications of clustering that group samples in different ways.

Centroid-based algorithms partition samples in clusters based on their distance to the centre point, called the centroid. Distribution-based algorithms assume that each cluster is of a certain shape, e.g a Gaussian distribution. Finally, density-based algorithms group samples into clusters based on their proximity to each other, allowing for clusters of irregular shapes.

### 2.2 Programming Languages and Libraries

Gonzalez et al. (2020) investigated commonly used programming languages for ML development from a large set of GitHub repositories. The authors showed that the interpreted programming language Python was the most used language. As for compiled languages, the study showed that C++ was commonly seen in projects developing ML libraries and frameworks.

While each language provides built-in features, there is often a need for external libraries to facilitate the development of complex applications. In order to create ML models, libraries can be utilized for their training algorithms and evaluation tools.

There are several libraries that enable ML development for both Python and C++. The `scikit-learn` library for Python provides an interface to develop ML models using supervised and unsupervised learning algorithms (Pedregosa et al., 2011). A similar library, enabling ML development in C++, is `mlpack` (Curtin et al., 2023). In comparison, `scikit-learn` provides a wider range of training algorithms and tools, but both libraries provide a similar interface to develop models.

## 2.3 Energy Consumption Measurement Tools

In order to investigate energy efficiency of software, the energy consumption needs to be measured.

One approach is to use external tools connected to the hardware. These tools can measure energy consumption of the system as a whole by measuring at the power plug, or in more detail by connecting to individual components. This approach is still somewhat lacking as components inside the CPU are inaccessible (Khan et al., 2018). Access to these specialized tools is also required, and the setup process can be difficult.

García-Martín et al. (2019) proposes power modelling as a more accessible approach, which instead estimates energy consumption. The authors outline power estimation models used in ML research. There are many estimation models depending on the level of detail needed, some of which allow for detailed estimates of hardware components or CPU instructions. One way to estimate the general energy consumption of an application is by using performance monitoring counters (PMCs), which count hardware events. The reliability of energy consumption estimates can however vary depending on the quality of the estimation model (Khan et al., 2018).

Running Average Power Limit (RAPL) is a feature of newer Intel processors allowing for direct energy consumption measurements of the CPU and RAM. Khan et al. (2018) suggests RAPL as an alternative to power modelling that can give accurate real time readings. Software tooling utilizing RAPL has been used extensively in research (Pereira et al., 2021; Gordillo et al., 2024; Albonico et al., 2024; Camargo-Henríquez et al., 2024; Nouredine, 2022).

## 2.4 Related Work

There are several papers where the authors have investigated the energy efficiency of programming languages. A few studies focused on comparing and ranking programming languages based on energy consumption during the execution of smaller algorithms.

In Pereira et al. (2021), the authors measured the energy consumption and performance of programming languages when executing a set of algorithms from the Computer Language Benchmarks Game (CLBG). Each algorithm was implemented in multiple languages, while trying to keep the implementations as similar as possible in all languages. The results revealed a trend where interpreted languages tended to be slower and less energy-efficient than compiled languages, although there was an exception noted in the case of string manipulation.

Another study investigating programming language energy efficiency is Gordillo et al. (2024). The authors extended the work of Pereira et al. (2021) by measuring the energy consumption of the CLBG algorithms using hardware tools. These measurements were then compared to software estimates in order to evaluate their accuracy. The results confirmed the findings of Pereira

et al. (2021) and demonstrated that software estimates were comparable to that of hardware measurements when ranking the energy efficiency of software.

Georgiou et al. (2018) investigated the energy efficiency and performance of specific programming language tasks, such as, IO and string manipulation. The algorithms were performed on different types of platforms and measured using a hardware tool. The study aimed to find the most efficient language for each task and platform. The results showed that the energy efficiency of a programming language varied based on the type of task performed.

In contrast to previously mentioned studies, Albonico et al. (2024) compared the energy consumption of Python and C++ in a larger context. The authors compared the energy consumption of nodes in a publish-subscribe robotic system implemented in Python and C++. The authors stated that both implementations consumed a similar amount of energy when handling one subscriber, while C++ was more efficient with higher numbers of subscribers. This shows that there are cases where C++ and Python are closer in energy consumption, compared to what other studies might imply.

Another aspect that has been considered is the use of compiler optimizations to reduce the energy consumption of software during execution.

Abdulsalam et al. (2014) investigated how programming languages and compiler optimization levels affect the energy consumption of programs during execution. Additionally, the authors investigated the impact of implementation details, such as the data-structures used. A key insight from the study is that the -O2 and -O3 optimization levels for the C and C++ compilers resulted in more energy efficient programs.

Similarly, Koedijk and Oprescu (2022) investigated the impact of optimization flags on energy efficiency, while also accounting for the idle power draw of the system. They conducted an experiment using hardware measurements of energy consumption in programs from the CLBG dataset across several programming languages. The authors concluded that C and C++ programs compiled using optimization flags were the most energy efficient.

While the studies presented in this section have provided insight into the energy consumption of programming languages and compiler optimizations, there are aspects that have not been considered. These studies only measured execution cost, and not compilation cost.

Furthermore, most studies use specialized algorithms in their benchmarks, which might not reflect how a language is used in other settings. Investigating programming languages in larger contexts, where they are commonly used, could provide further insight on their energy consumption in real-world settings.

# 3 | Problem

## 3.1 Aim

The aim of this research is to analyse the energy efficiency of programming languages in the context of training ML models. This involves comparing the execution of compiled and interpreted languages, while also considering the energy consumption of compilation and the impact of compiler optimizations. The comparison will use both supervised and unsupervised algorithms to represent a wide range of ML use-cases.

## 3.2 Motivation

Previous studies on energy efficiency have shown that there is a significant disparity in energy consumption between programming languages (Pereira et al., 2021; Gordillo et al., 2024). One key finding is that interpreted languages tend to be slower and less energy-efficient compared to compiled languages. Although Python is an interpreted language, it remains one of the most used languages for ML (Gonzalez et al., 2020).

If the choice of programming language has even a marginal effect on energy efficiency, it could lead to substantial cost savings. Most of the studies conducted so far benchmark small, optimized algorithms (Pereira et al., 2021; Gordillo et al., 2024; Georgiou et al., 2018; Abdulsalam et al., 2014; Koedijk and Oprescu, 2022). However, real-world usage of a programming language can differ significantly from these algorithms. One real-world application is ML development, which often involves complex algorithms and large libraries. Furthermore, there are several other factors that previous studies have not considered.

An important aspect is compilation and its associated energy consumption. The cost of compilation should not be overlooked, as it is required for execution in compiled languages.

Another consideration related to compilation is the use of optimization flags. Previous studies showed that optimization flags have an effect on the energy consumption during execution but did not consider the compilation cost (Abdulsalam et al., 2014; Koedijk and Oprescu, 2022). While optimizations may provide performance benefits, it is unclear if they also provide lower energy consumption when accounting for the cost of compilation.

## 3.3 Research Questions

To conduct the study, a set of research questions were defined. The purpose was to investigate energy efficiency of interpreted and compiled programming languages in the context of training ML models. To explore this area, an initial question was defined:

**RQ1** How does energy consumption during execution differ between interpreted and compiled programming languages, in the context of training machine learning models?

Since the study intended to account for the energy consumption associated with the compilation process, an additional research question was defined to investigate this aspect:

**RQ2** How does the total energy consumption when accounting for compilation differ between interpreted and compiled programming languages, in the context of training machine learning models?

To further explore how the energy consumption is affected by compilation, the following research questions were defined to investigate the impact of compiler optimizations:

**RQ3** How does the energy consumption during execution differ between compiler optimization levels, in the context of training machine learning models?

**RQ4** How does the total energy consumption when accounting for compilation differ between compiler optimization levels, in the context of training machine learning models?

### 3.4 Hypothesis

**H1<sub>0</sub>** *There is no significant difference in energy consumption during execution between interpreted and compiled programming languages, in the context of training machine learning models.*

**H1<sub>1</sub>** *There is a significant difference in energy consumption during execution between interpreted and compiled programming languages, in the context of training machine learning models.*

**H2<sub>0</sub>** *There is no significant difference in total energy consumption when accounting for compilation between interpreted and compiled programming languages, in the context of training machine learning models.*

**H2<sub>1</sub>** *There is a significant difference in total energy consumption when accounting for compilation between interpreted and compiled programming languages, in the context of training machine learning models.*

**H3<sub>0</sub>** *There is no significant difference in energy consumption during execution between compiler optimization levels, in the context of training machine learning models.*

**H3<sub>1</sub>** *There is a significant difference in energy consumption during execution between compiler optimization levels, in the context of training machine learning models.*

**H4<sub>0</sub>** *There is no significant difference in total energy consumption when accounting for compilation between compiler optimization levels, in the context of training machine learning models.*

**H4<sub>1</sub>** *There is a significant difference in total energy consumption when accounting for compilation between compiler optimization levels, in the context of training machine learning models.*

## 3.5 Objectives

A number of objectives that needed to be completed to conduct the study were defined:

1. Choose the programming languages and ML libraries
2. Choose the energy consumption measurement tool
3. Choose ML algorithms and implement models
  - 3.1. Supervised learning algorithms (Markus)
  - 3.2. Unsupervised learning algorithms (Sol)
4. Setup and configure hardware and software for the experiment environment
5. Prepare datasets for the training of ML algorithms
6. Implement a script to automate the measurements
  - 6.1. Configure the energy measurement software (Sol)
  - 6.2. Control measurements by accounting for CPU temperature (Markus)
7. Perform CLBG benchmarks and compare to Pereira et al. (2021) (Sol)
8. Measure and compare the energy consumption of compiler optimization levels (Markus)
9. Measure and compare the energy consumption of the chosen programming languages
10. Analyse the results and present the conclusions

## 4 | Method

This study aims to compare differences between programming languages, making it essential that the implementations are equivalent. Equivalence is important to ensure that the comparison is focused on the differences between programming languages and not the implementations. While achieving equivalence is difficult, one way to approach it is by writing the implementations for this purpose. A quasi-experiment enables full control over the implementations, allowing for the selection of programming languages, libraries, configurations, and hyperparameters. The selection of specific ML algorithms is also possible with this method, which is important to cover both supervised and unsupervised ML learning techniques. By maintaining control over these factors, a fairer comparison that is also relevant to previous research can be achieved. Therefore, the method used for this study is a quasi-experiment.

### 4.1 Alternative Method

One alternative method to the quasi-experiment is a case study. A case study could be conducted to analyse ML models developed by a company. An advantage of conducting a case study at a company is that the investigated models would represent applied ML implementations, used in real-world settings. Investigating models developed by companies could also allow for generalization of the findings to a larger population. However, it is unlikely that any company would implement similar models in multiple programming languages. The case study could involve several companies, allowing for the comparison of implementations across several companies. Although this approach could provide models in multiple languages, the challenge of finding similar implementations would still remain.

Another case study could involve ML implementations hosted on open-source platforms, providing a wider range of implementations for comparison. However, this approach shares the same challenges as the previous alternative.

# 5 | Approach

## 5.1 Programming Languages and Libraries

The choice of programming language by companies and researchers is likely to be influenced by the availability of ML libraries and their long-term support. Therefore, it was appropriate to choose programming languages with well-supported ML libraries for this study.

In order to answer the research questions, one interpreted language and one compiled language had to be selected. Gonzalez et al. (2020) showed that both Python and C++ are commonly used for ML development in open-source projects. Therefore, Python was chosen as the interpreted language, and C++ was chosen as the compiled language.

To enable ML development, the `sci-kit learn` (Pedregosa et al., 2011) and `mlpack` (Curtin et al., 2023) libraries were chosen for their respective programming language. These libraries were used as they provided many of the same algorithms, on a similar abstraction level.

## 5.2 Machine Learning Algorithms

Both `scikit-learn` and `mlpack` offer implementations of various ML algorithms. The choice of algorithms for the experiment was influenced by several factors. One factor was that any chosen algorithm had to be fully implemented in both libraries. Another important factor was how well-known and relevant the algorithms were for ML. Algorithms frequently described or used in literature and research were the most appropriate to include, as these are more likely to be used by companies and researchers. To ensure a comprehensive analysis, the algorithms were selected to encompass a range of types and use-cases (see Table 5.1). Most algorithms were selected from those reviewed by Sarker (2021) and Thai (2022).

Table 5.1: ML implementations by type and technique.

Learning type	Technique	Algorithm
Supervised	Classification	Decision Tree Logistic Regression
	Regression	Linear Regression Bayesian Linear Regression
Unsupervised	Clustering	K-means Gaussian Mixture Model DBSCAN

## 5.3 Machine Learning Implementations

Before implementing any model, the documentation for each library was inspected. Implementation details and hyperparameters were considered, to determine whether both libraries provided comparable implementations of an algorithm. In most cases, similar hyperparameters were available for an algorithm in both libraries, but often under different names. To determine whether hyperparameters served the same purpose in both libraries, their name and descriptions were taken into account. Whenever possible, corresponding hyperparameters were set to the same values before training. A difference between the libraries was that `mlpack` only provided a subset of the parameters provided by `scikit-learn`. In some cases, this was due to optional parameters that `scikit-learn` provided.

While most algorithms appeared similar in both libraries, some had differences that could affect their performance and energy consumption.

The library implementations of the decision tree algorithm had some disparities, due to the use of different underlying algorithms. The `mlpack` decision tree uses the ID3 algorithm and Gini gain as a metric for splitting (mlpack developers, 2024a). The `scikit-learn` decision tree classifier uses a variant of the CART algorithm and Gini impurity as a metric for splitting (scikit-learn developers, 2025a). Despite these differences, the decision tree algorithm is still included in the experiment due to its widespread use in ML.

There were also some differences between the library implementations of the Gaussian mixture models (GMM) algorithm. Because GMM uses K-means as a pre-processing step, it will also be affected by the differences between the K-means implementations (scikit-learn developers, 2025b). The hyperparameters adjusted to make the K-means implementations more similar were not accessible for the GMM algorithm. As a result, the default behaviour for each library was used in the pre-processing step.

## 5.4 Measurement Tools

Energy consumption measurements were collected with the Command-Line Interface (CLI) tool, `perf`. `perf` utilizes Performance Monitoring Unit (PMU) events to gather information provided by RAPL. The specific PMU events used for this analysis were "`power/energy-pkg/`" and "`power/energy-ram/`", which provide the energy consumption data for the CPU cores, CPU cache, and RAM.

`perf` can be used to run and measure a single process, ensuring that measurements are taken only during the lifetime of that specific process. It is important to note that the obtained measurements are not exclusive to individual processes, rather, they reflect the components and their total energy consumption.

## 5.5 Machine Learning Datasets

In order to train the models, appropriate datasets were required. Since different ML training algorithms are used for different tasks, they often require certain dataset characteristics. For instance, when logistic regression is used to perform binary classification, the dataset should only contain two classes.

The sizes of the datasets were also an important factor, as they impact the time and energy consumption required to train models. The datasets are loaded during runtime, which means

that the size only impacts the energy consumption of execution, and not compilation. To emphasize the difference in energy consumption between compilation and execution, a smaller and a larger dataset was used for each ML algorithm. It is important to note that 'small' and 'large' simply describes the relative size difference between the datasets, and is primarily used to denote the associated execution time.

To ensure accurate measurements, the execution time of each algorithm needed to be controlled. The reason being that the measurement tool `perf` utilizes RAPL to gather energy measurements. According to Intel (2024), RAPL retrieves its energy measurement roughly once each millisecond. A minimum execution time was needed to account for this frequency. The dataset sizes were therefore set based on their effect on execution time. The small datasets were based on an execution time of roughly one second, while the large datasets were based on an execution time of 10 seconds. Overall, the C++ implementations were the fastest in terms of execution time. Therefore, the dataset sizes were selected based on the execution time of those implementations.

The datasets were generated using the `scikit-learn` library (scikit-learn developers, 2025c). A reason for generating, instead of using existing datasets, was due to the difficulty in finding datasets with appropriate sizes and data types. Generating the datasets allowed for control over these factors.

The datasets, presented in Table 5.2 were generated to accommodate the tasks performed by the algorithms. The regression algorithms do not use classes, which is why the number of classes are not specified in the table for these algorithms.

Table 5.2: Generated datasets for each ML algorithm.

Type	Technique	Algorithm	Dataset	Samples	Features	Classes
Supervised	Classification	Decision Tree	Small	52,000	20	5
			Large	350,000	20	5
		Logistic Regression	Small	200,000	14	2
			Large	800,000	25	2
	Regression	Bayesian Ridge Regression	Small	250,000	17	-
			Large	1,500,000	30	-
		Linear Regression	Small	250,000	17	-
			Large	1,500,000	30	-
Unsupervised	Clustering	K-means	Small	159,500	10	10
			Large	700,000	14	14
		DBSCAN	Small	65,000	8	8
			Large	240,000	10	10
		Gaussian Mixture Model	Small	3,420	6	6
			Large	22,000	6	6

## 5.6 Baseline Measurements

Previous studies evaluating the energy efficiency of programming languages utilized algorithms from the CLBG dataset (Pereira et al., 2021; Gordillo et al., 2024; Koedijk and Oprescu, 2022). To

identify the effects of changes in hardware, software, and other variables, the CLBG benchmarks (see Table 5.3) from Pereira et al. (2021) were repeated to obtain a baseline measurement. The same optimization level, `-O3`, and flags were used in order to get accurate results. Repeating the study ensured that the results of this study were anchored to those of Pereira et al. (2021). If the findings of earlier studies can be confirmed, the results from the ML algorithms can be compared to them with greater confidence.

Table 5.3: CLBG algorithms.

Algorithm
Binary Trees
Fannkuch Redux
Fasta
K-nucleotide
Mandelbrot
N-body
Pidigits
Regex Redux
Reverse Complement
Spectral Norm

## 5.7 Compiler Optimization

In this study, the GNU `g++` compiler and four optimization levels were included based on the study by Abdulsalam et al. (2014). The authors investigated how different optimization levels affected the energy consumption of software during execution. The `g++` compiler provides a wide range of flags, as well as predefined optimization levels that includes sets of optimization flags. These levels may be used instead of providing individual flags to the compiler. Abdulsalam et al. (2014) used `g++` without optimization and with the levels `-O1`, `-O2` and, `-O3`. The same levels were included in this experiment. The `-O0` level was also included, since it is the default level and includes no additional optimizations (Free Software Foundation, 2025).

The documentation explains each optimization level, stating that each level adds additional optimization flags (Free Software Foundation, 2025). The `-O1` and `-O2` level provides optimizations to reduce the execution time. According to Oracle (2021), the `-O3` level includes flags to optimize for speed by compromising space.

To investigate the energy consumption associated with these optimizations, measurements were taken during compilation and execution of the ML implementations, using both dataset sizes. These measurements should provide sufficient data to compare and discuss the impact that these optimizations have on energy consumption during both execution and compilation.

## 5.8 Experiment Setup

To ensure that all measurements were taken under the same conditions, a number of factors were controlled. Georgiou et al. (2018) described steps to mitigate external factors that could affect the outcome of the measurements. In their experiment, certain background processes such as

cronjobs were disabled. Additionally, the authors specified certain conditions, such as system temperature, that had to be met before proceeding with a measurement. In this study, similar steps were taken. All `systemd` timers (see Table 5.4) and scheduled cron jobs were disabled to prevent them from running concurrently with the measurements.

Table 5.4: Disabled `systemd` timers.

Unit
<code>sysstat-collect.timer</code>
<code>fwupd-refresh.timer</code>
<code>apt-daily.timer</code>
<code>man-db.timer</code>
<code>motd-news.timer</code>
<code>dpkg-db-backup.timer</code>
<code>logrotate.timer</code>
<code>sysstat-summary.timer</code>
<code>apt-daily-upgrade.timer</code>
<code>update-notifier-download.timer</code>
<code>e2scrub_all.timer</code>
<code>fstrim.timer</code>
<code>update-notifier-motd.timer</code>
<code>systemd-tmpfiles-cleaner.timer</code>

A script was implemented to automate the measurement process. After each measurement, the CPU temperature needed to fall to a threshold temperature before proceeding with the next measurement. This step ensures that each measurement starts at the same temperature. The threshold temperature was set to 35 Celsius, a few degrees above idle temperature. To ensure that the first measurement also starts at the threshold temperature, the dataset for the baseline measurements is generated first, which raises the temperature.

The script queues a set of commands, where each command represents an individual implementation in `Python` or `C++`, with a specific dataset. The script was set to run 100 iterations, in order to gather sufficient data for statistical analysis. A single iteration in the script measures every implementation once. At the start of each iteration, the order of the commands is randomized to eliminate any potential bias in the results due to the order of execution. When executing the measurement of a `Python` implementation, `perf` simply executes and measures the execution of the program. The measurement of a single `C++` implementation involves several steps. A `C++` implementation needs to be compiled and executed using four different optimization levels, `-O0`, `-O1`, `-O2` and `-O3`. For each optimization level, both the compilation and the execution of the compiled binary is measured. These four levels were measured in order to answer RQ3 and RQ4, while RQ1 and RQ2 only used results with the `-O3` level, which is the recommended optimization level for `mlpack` (mlpack developers, 2024b).

The measurements were performed on a machine running `Ubuntu Server 24.04 LTS` with the `Linux 6.8.0-54-generic` kernel. The machine was equipped with an `Intel i5-6600` CPU and 8 GB RAM. System packages and libraries used with `C++` were installed with `apt`. For `Python` a virtual environment was used and packages were installed with `pip`. All installed packages are shown in Table 5.5.

Table 5.5: Packages and versions.

Use	Package	Version
System	build-essential	12.10ubuntu1
	linux-tools-generic	6.8.0-54.56
	python3	3.12.3-0ubuntu2
	python3-pip	24.0+dfsg-1ubuntu1.1
	libc6	2.39-0ubuntu8.4
	g++	4:13.2.0-7ubuntu1
C++	libmklpack-dev	4.3.0-2build1
	libapr1-dev	1.7.2-3.1ubuntu0.1
	libaprutil-dev	1.6.3-1.1ubuntu7
	libgmp-dev	2:6.3.0+dfsg-2ubuntu6.1
	libboost-regex-dev	1.83.0.1ubuntu2
Python	scikit-learn	1.6.1
	pandas	2.2.3
	numpy	2.2.2
	scipy	1.15.1
	gmpy2	2.2.1

## 5.9 Statistical Significance

Each research question addresses the difference between two or more groups. To answer a research question, it is necessary to either confirm or reject its null hypothesis, which asserts that no significant difference exists between the groups. A statistically significant difference can be identified through several steps. Initially, the confidence level was utilized to calculate the confidence intervals for each group. A confidence level of 95% was chosen for this experiment. If the intervals were found to be disjunct, a significant difference existed. If the intervals overlapped, further investigation in the form of a one-way ANOVA test would have been needed.

# 6 | Result

## 6.1 Data Processing

Before analysing the raw data, potential outliers were considered. The only notable cases were found in the `Python` implementation of the GMM algorithm. In one measurement using the large dataset, the value was 60 times larger than the mean. Additionally, two measurements using the small dataset were twice the mean. These outliers were removed, as they were considered anomalous. With these outliers removed, the variance was low for every algorithm. All diagrams include error bars, but the bars are not visible due to the low standard error.

## 6.2 Baseline Measurements

In this section, the results for the baseline measurements are presented. The groups compared are the two programming languages, `C++` and `Python`. First, the energy consumption during execution of CLBG algorithms is compared, followed by a discussion on the statistical significance.

The results seen in Figure 6.1 show that `C++` tended to consume less energy compared to `Python`. An exception is `regex-redux`, which was the only algorithm where `Python` consumed less energy than `C++`. These findings are consistent with Pereira et al. (2021) who note that interpreted languages were among the most energy efficient in the `regex-redux` benchmark, while performing worse in other scenarios.

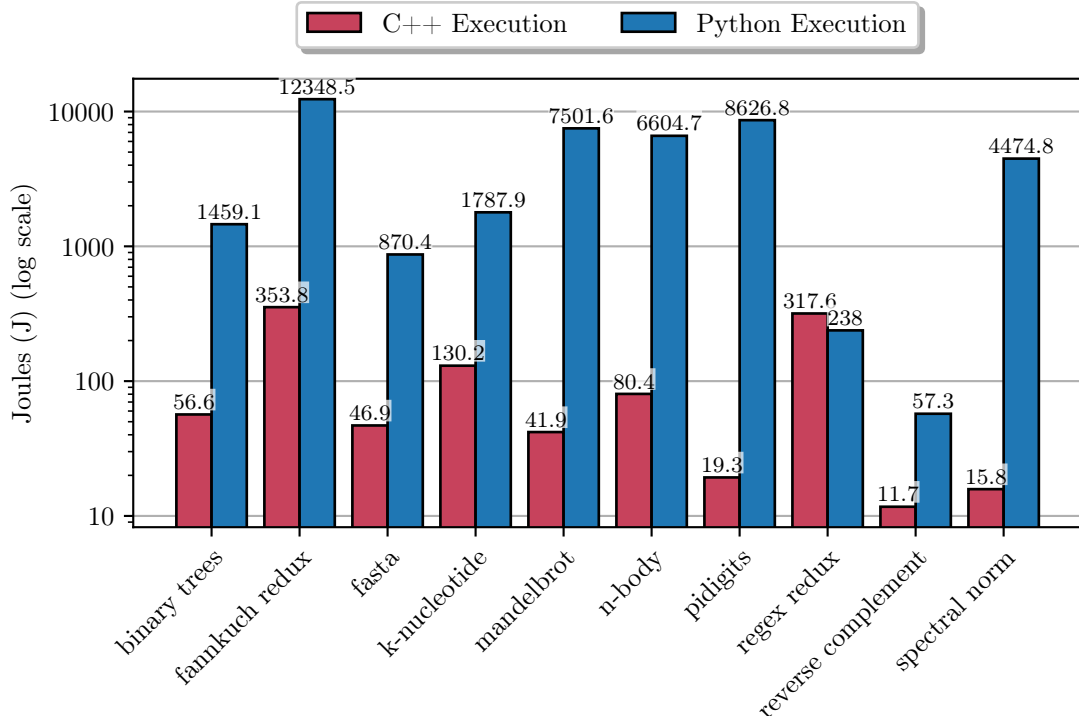


Figure 6.1: Mean energy consumption during execution of algorithms in the CLBG dataset.

The confidence interval for each algorithm and language is presented in Table 6.1. Because the difference in energy consumption between the two languages was quite large there were no overlapping intervals. This was true even for the regex-redux algorithm, where the difference between C++ and Python was the smallest. There is therefore a statistically significant difference between the languages for every algorithm.

Table 6.1: Energy consumption (Joules) of all CLBG algorithms.

Algorithms	C++			Python		
	Mean	SE	CI	Mean	SE	CI
binary trees	56.6	0.048	0.094	1459.1	0.696	1.365
fannkuch redux	353.8	0.144	0.282	12348.5	13.699	26.850
fasta	46.9	0.034	0.068	870.4	1.251	2.452
knucleotide	130.2	0.097	0.190	1787.9	1.384	2.713
mandelbrot	41.9	0.014	0.026	7501.6	7.487	14.674
nbody	80.4	0.042	0.083	6604.7	8.496	16.653
pidigits	19.3	0.017	0.033	8626.8	3.924	7.690
regex redux	317.6	0.389	0.763	238.0	0.110	0.216
revcomp	11.7	0.012	0.024	57.3	0.036	0.071
spectral norm	15.8	0.010	0.020	4474.8	3.227	6.325

### 6.3 Programming Languages

In this section, the results for the energy consumption of ML algorithms are presented. The groups compared are the two programming languages `C++` and `Python`. First, the difference in execution cost between the languages using the small datasets is shown, followed by the difference in the large datasets. Then the total energy consumption when accounting for compilation is presented, followed by the statistical significance of all measurements.

In order to answer RQ1, the execution costs of `C++` and `Python` were compared. The results using the small datasets, displayed in Figure 6.2, show that the difference in energy consumption between programming languages varied. For certain algorithms, the execution cost of `Python` was twice that of `C++`, and vice versa. `C++` demonstrated lower energy consumption in five out of seven algorithms, which somewhat aligns with the baseline measurements shown in Figure 6.1. However, the difference between the languages were much smaller than those observed in the baseline measurements.

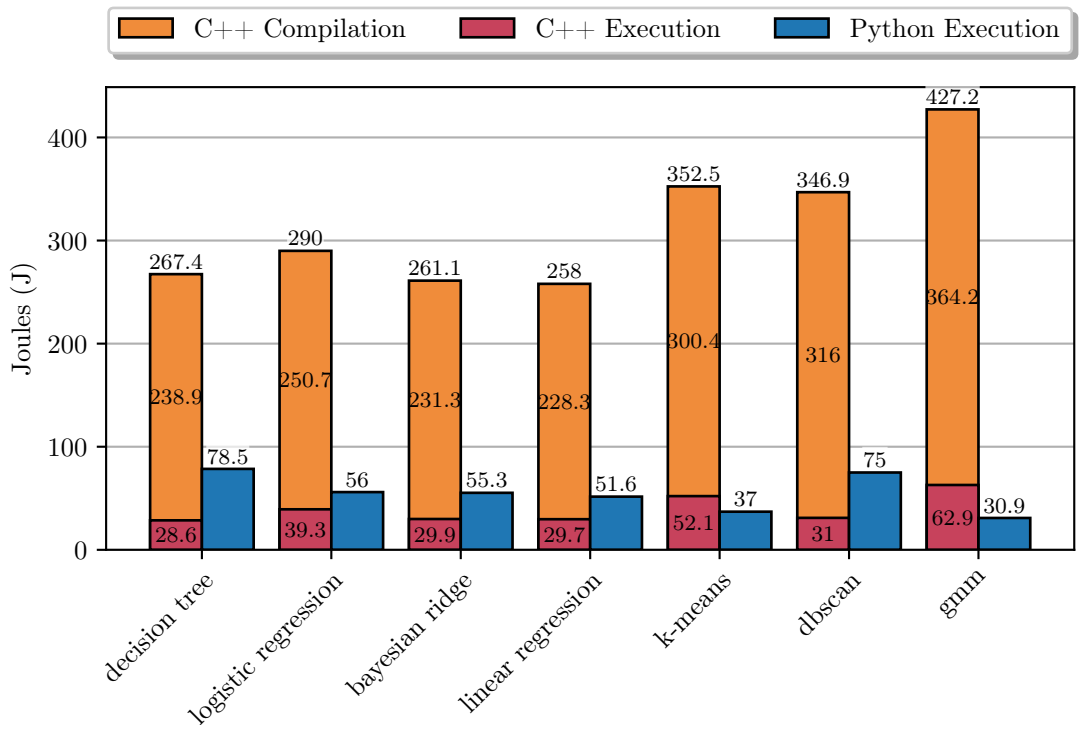


Figure 6.2: Mean energy consumption of programming languages, using the small datasets and -O3 optimization level.

The findings in Figure 6.3 reveal that `Python` consumed less energy in four out of seven algorithms while using the large datasets, with greater differences between the languages. The notably low energy consumption of `Python` in the K-means and GMM algorithms using the small datasets, seen in Figure 6.2, was even more pronounced here.

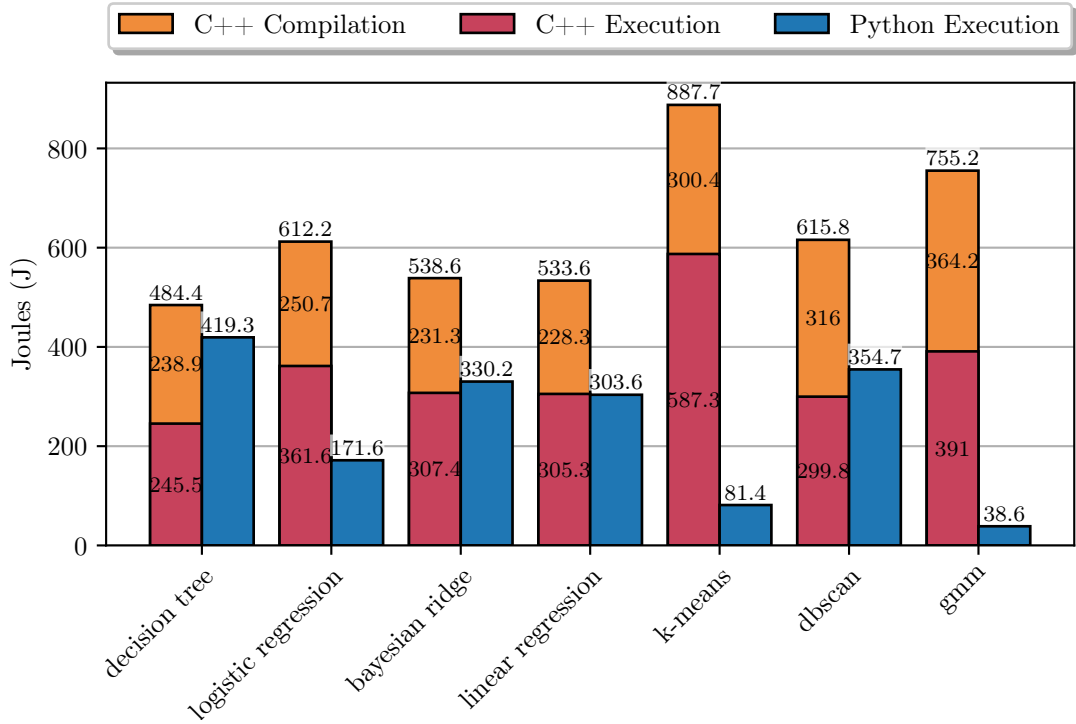


Figure 6.3: Mean energy consumption of programming languages, using the large datasets and -O3 optimization level.

For RQ2, the combined cost of compilation and execution of C++ were compared to Python. When accounting for compilation costs, C++ consumed more energy than Python in every measurement. Furthermore, the compilation cost represented a larger portion of the total energy consumption for C++ when using the small datasets compared to the large datasets. Specifically, the compilation cost was about ten times the execution cost when using the small datasets, compared to only two times for the large datasets. As previously mentioned, this difference can be explained by the fact that the compilation cost remains constant. The execution, compilation, and total combined costs for each algorithm and dataset can be seen in Table 6.2 and 6.3. Individual plots for each algorithm can be seen in Appendix C.

The results, presented in Table 6.2 and 6.3, show that the confidence intervals are very small. This means that even though the mean values for some algorithms are close, there is still a statistically significant difference between languages for each algorithm.

Table 6.2: Energy consumption (Joules) of ML algorithms, using the small datasets and -O3 optimization level.

Algorithms	C++ Execution			C++ Compilation			C++ Total			Python		
	Mean	SE	CI	Mean	SE	CI	Mean	SE	CI	Mean	SE	CI
decision tree	28.6	0.015	0.030	238.9	0.131	0.257	267.4	0.135	0.264	78.5	0.050	0.097
logistic regression	39.3	0.023	0.046	250.7	0.147	0.289	290.0	0.152	0.298	56.0	0.090	0.176
bayesian ridge	29.9	0.017	0.034	231.3	0.147	0.289	261.1	0.154	0.303	55.3	0.036	0.070
linear regression	29.7	0.017	0.033	228.3	0.126	0.248	258.0	0.133	0.261	51.6	0.035	0.068
kmeans	52.1	0.037	0.073	300.4	0.189	0.370	352.5	0.199	0.391	37.0	0.023	0.045
dbscan	31.0	0.018	0.036	316.0	0.198	0.388	346.9	0.204	0.400	75.0	0.052	0.102
gmm	62.9	0.047	0.093	364.2	0.222	0.436	427.2	0.233	0.458	30.9	0.019	0.038

Table 6.3: Energy consumption (Joules) of ML algorithms, using the large datasets and -O3 optimization level.

Algorithms	C++ Execution			C++ Compilation			C++ Total			Python		
	Mean	SE	CI	Mean	SE	CI	Mean	SE	CI	Mean	SE	CI
decision tree	245.5	0.137	0.268	238.9	0.131	0.257	484.4	0.223	0.436	419.3	0.488	0.956
logistic regression	361.6	0.149	0.291	250.7	0.147	0.289	612.2	0.241	0.472	171.6	0.211	0.413
bayesian ridge	307.4	0.230	0.450	231.3	0.147	0.289	538.6	0.283	0.554	330.2	0.232	0.455
linear regression	305.3	0.266	0.521	228.3	0.126	0.248	533.6	0.322	0.631	303.6	0.229	0.448
kmeans	587.3	0.128	0.251	300.4	0.189	0.370	887.7	0.264	0.517	81.4	0.045	0.088
dbscan	299.8	0.148	0.291	316.0	0.198	0.388	615.8	0.271	0.531	354.7	0.226	0.442
gmm	391.0	0.181	0.354	364.2	0.222	0.436	755.2	0.304	0.595	38.6	0.232	0.455

## 6.4 Compiler Optimization Levels

In this section, the results for compiler optimization levels are presented. First, the measurements taken during execution are presented, and the statistical significance is shown. Then the measurements for the total energy consumption is presented. The statistical significance for execution and total energy consumption is shown by comparing all four optimization levels per algorithm. Individual plots for each algorithm, showing execution and compilation costs associated with each optimization level, can be seen in Appendix B.

Figure 6.4 shows the mean energy consumption of the ML implementations during execution with the small datasets, when compiled using each optimization level. A notable trend was that algorithms compiled with -O0 consumed the most energy of all levels, by a large margin. Energy consumption decreased for each level until -O2, where an improvement from -O2 to -O3 was observed only for the decision tree algorithm. For four of the seven algorithms, the lowest energy consumption was achieved when -O2 was used. However, no change in average energy consumption was seen between -O2 and -O3 for Bayesian ridge regression and linear regression.

These observations in energy consumption may be caused by the optimizations to execution time added by each level, as previously described (see Section 5.7). The high consumption of

-00 could be explained by the lack of any optimizations. The lack of an improvement in -03 over -02 may be due to the compromise between speed and space.

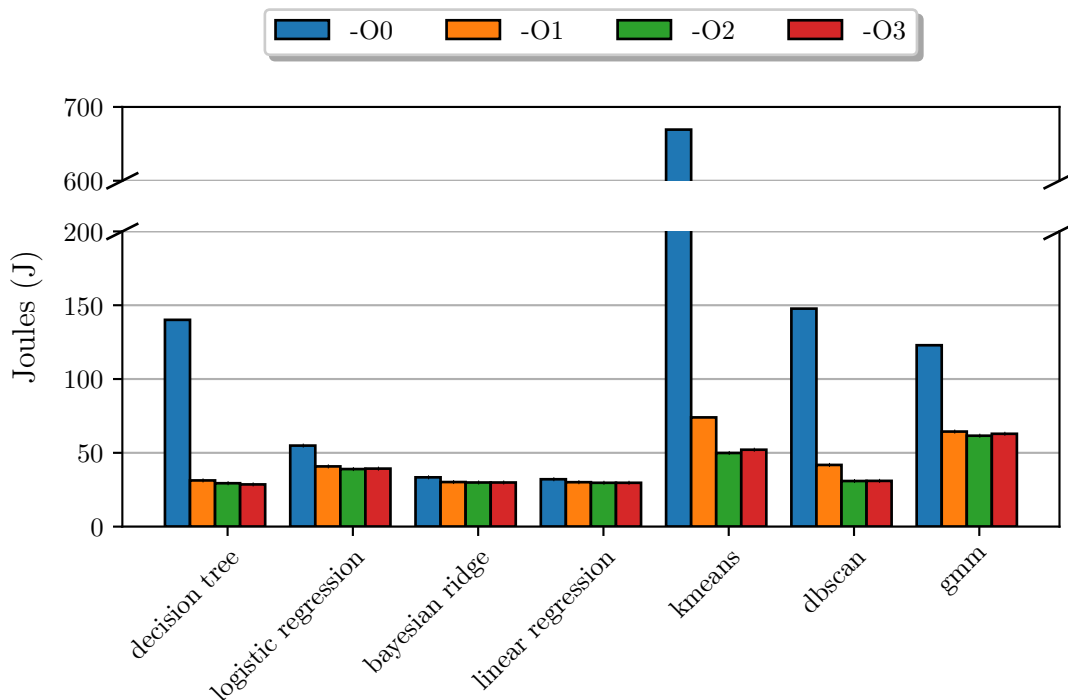


Figure 6.4: Mean energy consumption during execution for each optimization level, using the small datasets.

The table in Appendix A.1 presents the means and confidence intervals for the measurements taken during execution using the small datasets. Most algorithms had no overlapping intervals between the optimization levels. However, the difference in execution cost was not as prominent for Bayesian ridge regression and linear regression. Both algorithms had no change in average energy consumption between -02 and -03, resulting in overlapping confidence intervals between these levels. For this reason, the one-way ANOVA test was used to determine the statistical significance of optimization levels during execution. All ANOVA tests resulted in p-values less than  $1 \times 10^{-277}$ , well below the significance level. Therefore, there is a statistically significant difference.

Measurements taken during execution using the large datasets showed similar relations between the optimization levels as when using the small datasets (See Figure 6.5). An overlap was found between -02 and -03 for Bayesian ridge regression and linear regression using the large datasets (See Appendix A.2), therefore the one-way ANOVA test was performed on these measurement groups as well. The tests showed a statistical significance for all the algorithms, since the highest p-value taken was less than  $1 \times 10^{-307}$ .

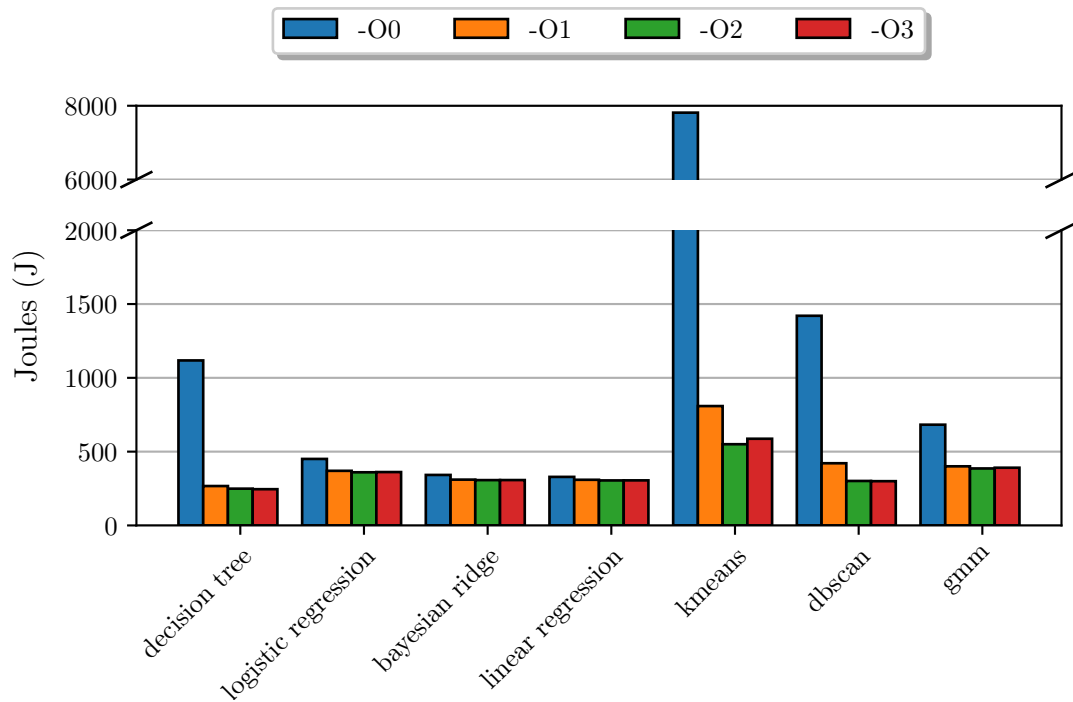


Figure 6.5: Mean energy consumption during execution for each optimization level, using the large datasets.

For compilation, an increase in energy consumption was observed for each consecutive level. As seen in Figure 6.6, the lowest energy consumption during compilation was achieved when using -00, and the highest when using -03. Contrary to the energy consumption of execution, which decreased with each level, the cost of compilation instead increased. However, this trend is more consistent, showing an increase for -03 as well.

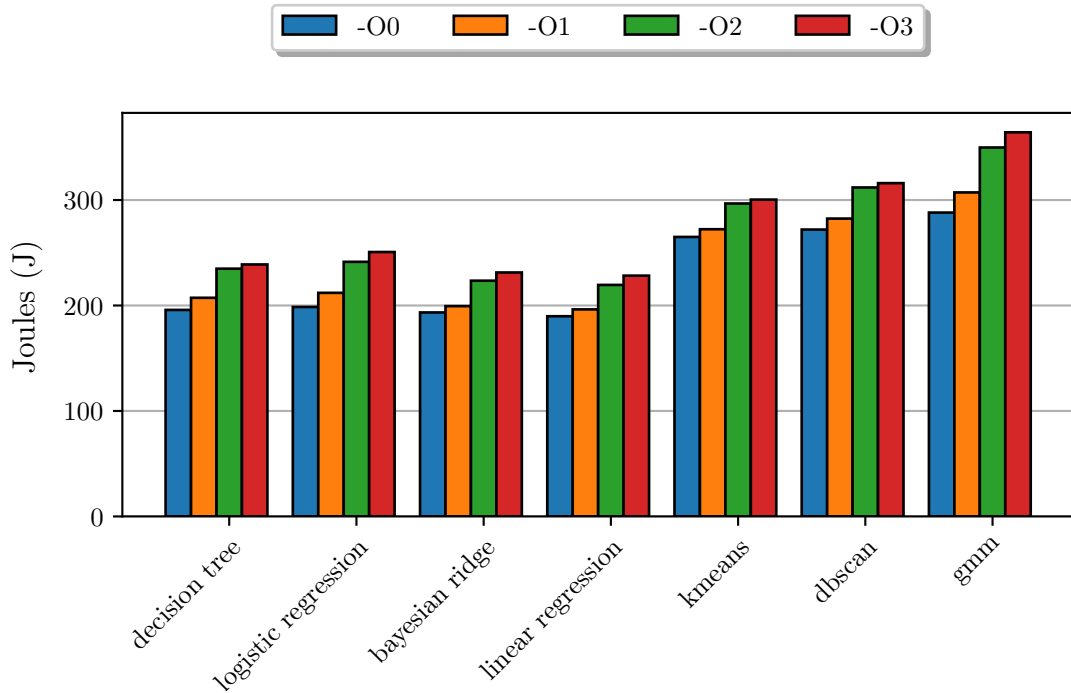


Figure 6.6: Mean energy consumption during compilation for each optimization level.

The table in Appendix A.1 presents the means and confidence intervals for the measurements taken during compilation. None of the optimization levels had overlapping confidence intervals for any of the algorithms. Therefore, there is a statistically significant difference in energy consumption between optimization levels during compilation.

Another aspect to consider is the total energy cost when compilation and execution is combined. During compilation -00 consumed the least amount of energy, while being the worst during execution. In contrast, -02 and -03 consumed the least during execution, but were the highest during compilation. Figure 6.7, shows the total energy consumption for each level when the small datasets were used. When comparing the total energy consumption of all the levels, -01 performed notably better compared to previous results. The -00 level showed mixed results. For a majority of algorithms, -00 consumed the most, but was also the least energy consuming level for two of the algorithms.

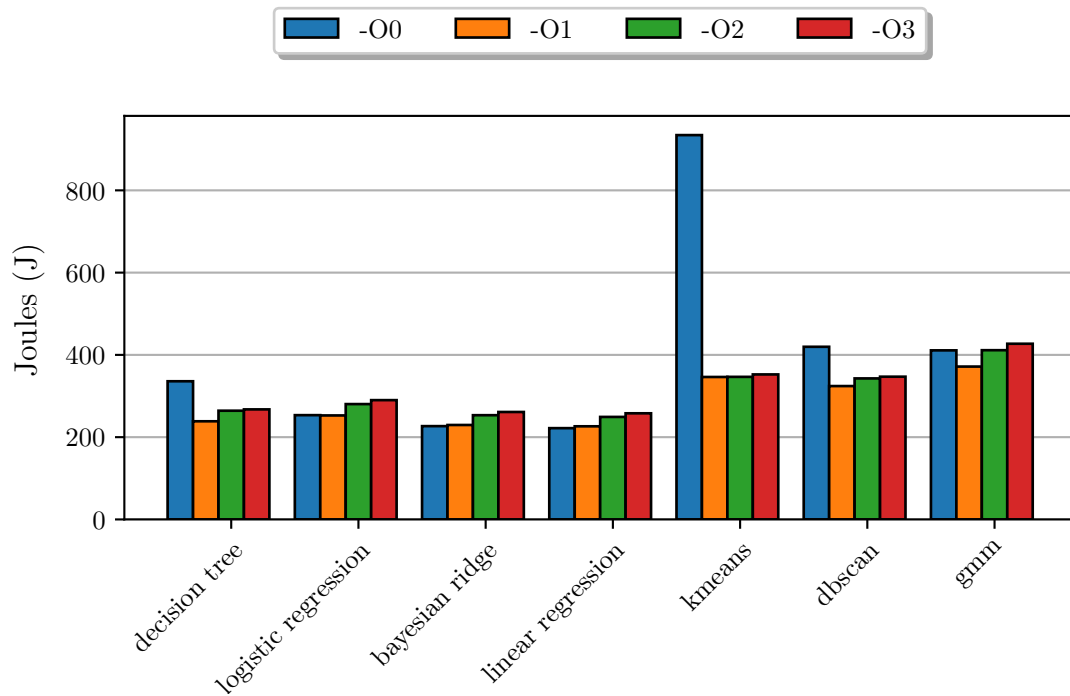


Figure 6.7: Mean total energy consumption for each optimization level, using the small datasets.

When comparing the total energy cost of the levels using the large datasets (see Figure 6.8), a few differences were seen in comparison to the small datasets. Notably, -00 was no longer the least energy consuming level for any of the algorithms, but was still the most energy consuming level by a large margin for most algorithms. Furthermore, -01 was still the least energy consuming level for five out of seven algorithms, but seemed to perform worse for the remaining two in comparison to -02 and -03.

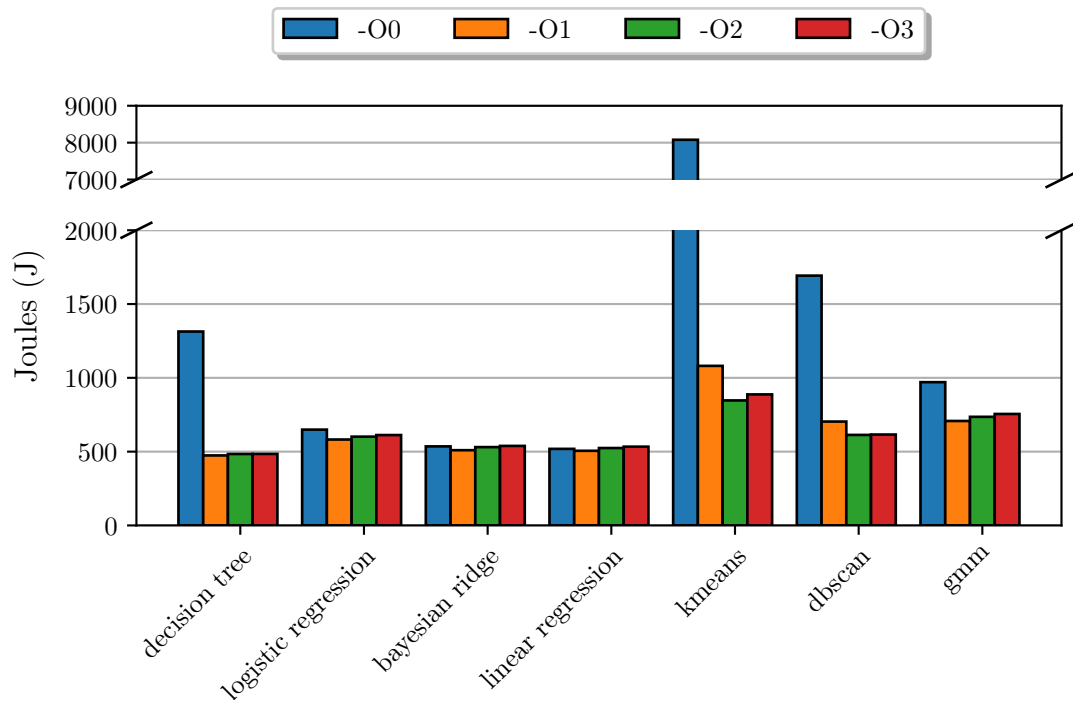


Figure 6.8: Mean total energy consumption for each optimization level, using the large datasets.

No overlaps were found between the groups when using the small or the large datasets, as seen in the corresponding tables in Appendices A.1 and A.2. Therefore, there is a statistically significant difference between the total energy consumption of the optimization levels, when using both dataset sizes.

# 7 | Discussion

## 7.1 Threats to Validity

In this section, threats to the validity of the study are presented. The threats are addressed by validity type, based on risks and issues described by Wohlin et al. (2012).

### 7.1.1 Construct Validity

There are several threats to the construct validity of the study that should be considered. Because the experiment only includes one measurement tool for the energy consumption of the ML implementations, one threat is mono-method bias (Wohlin et al., 2012). In the experiment, the `perf` tool is used to collect the energy consumption of the system during the lifetimes of the processes being measured. `perf` uses RAPL which is frequently used in studies on energy consumption. CLBG benchmarks by Pereira et al. (2021) were repeated to validate the chosen measurement tool. The results showed that measurements collected with `perf` aligned with the results of the previous study. However, including multiple measurement tools would enable the comparison of measurements, reducing the risk of misleading results caused by individual tools.

Another threat to the construct validity of the study involves restricted generalizability across constructs (Wohlin et al., 2012). Optimizations in software can often come at a trade-off. In this study, only energy consumption is considered, leaving its effects on other factors unknown. While low energy consumption during the training of models has a positive effect on energy efficiency, there could be negative effects on other factors, such as, model accuracy, execution and compilation time, or memory usage. If a particular language or optimization results in low energy consumption during model training, but has a detrimental effect on model accuracy, it would likely be considered an undesirable compromise.

### 7.1.2 Internal Validity

One threat to internal validity is instrumentation (Wohlin et al., 2012), involving the quality of the ML implementations. Each algorithm was implemented using both `mlpack` and `scikit-learn`. An important aspect was the similarity of the algorithm implementations, since differences between implementations could affect the results. Misconceptions concerning library functions or hyperparameters, could result in mistakes, leading to even greater differences. The documentation of each algorithm was inspected to gain a better understanding of implementation details in both libraries. While this step was essential, it did not guarantee similar implementations.

### 7.1.3 Conclusion Validity

A possible threat to the conclusion validity of the study, involves random irrelevancies in experimental setting (Wohlin et al., 2012). The experiment was performed in an unsupervised computer lab, frequently accessed by students. Supervision was not possible due to the long runtime of the script, meaning that possible interference from external factors could not be observed. The results contained a few anomalies with an unknown cause. The possibility that these anomalies were caused by external factors cannot be ruled out.

### 7.1.4 External Validity

The research questions focus on compiled and interpreted languages. However, only a single language from each language type was selected. A single language might not represent all languages within its type. While this can limit the generalizability of the languages to their respective type, the languages were chosen based on their use within the ML field. Because the languages were commonly used for ML development, they were considered suitable candidates. However, only a single compiler was used with C++, possibly limiting the generalizability of the results to that specific compiler.

It is also possible that the libraries chosen may not fully represent their respective programming languages. There are many ML libraries available, but only a single one is used for each language. It is likely that other libraries would perform differently from the ones chosen, meaning that the ability to generalize the findings to all libraries in a language is diminished.

Measurements were performed on a Linux-based 64-bit system, which could limit the applicability of the results to that specific architecture and operating system. While different architectures and operating systems may produce different results, 64-bit Linux is well-known and widely used. Therefore, the results can likely be generalized to a large part of ML and software development industries.

Another threat is the interaction of setting and treatment, which limits the ability to generalize the findings to industrial practices (Wohlin et al., 2012). The libraries used in this experiment might not represent those used in production. Both `mlpack` and `scikit-learn` are made for ease of use and are limited to only using the CPU. There are more complex libraries intended for use in production that could represent the use case of companies better. Ultimately, time constraints limited the type and number of libraries included in this study.

## 7.2 Programming Languages

When considering the energy consumption of the ML algorithms in execution, the differences between `Python` and `C++` were much smaller than in the baseline measurements. Because of this, determining what language had better energy consumption during execution was not straightforward. Two algorithms were always more energy efficient in execution with `Python`, while three were always more efficient with `C++`. The remaining two algorithms changed from being more efficient with `C++` using the small datasets, to `Python` using the large datasets. While this might indicate that `C++` performs better when the execution time is low, the change seems to depend on the specific algorithm.

When considering the total energy consumption of execution and compilation, `Python` was significantly more energy efficient in all measurements. However, there are two factors that can influence which language should be considered more energy efficient.

One factor is the size of the dataset. The relative cost of compilation decreased as the dataset grew because the energy consumption of compilation was independent of the dataset. This means that the cost of compilation will play less of a role when using larger datasets.

Another factor to consider is the number of executions. An algorithm only needs to be compiled once but can then be executed multiple times. There were many algorithms where `Python` had a higher execution cost compared to `C++`, but where `C++` had a higher total cost. However, if such an algorithm were to be executed multiple times, `C++` would eventually become more energy efficient.

The energy consumption of a programming language will not be the same for all use cases. Because `C++` requires compilation, it is likely to consume more energy compared to `Python` in settings where an algorithm is recompiled often, like during the development process. However, when an algorithm is executed more frequently than compiled, `C++` could have an edge, at least for those algorithms where its execution cost is lower than that of `Python`.

The algorithms in the CLBG dataset, used for the baseline measurements, were written to be as similar as possible. For some algorithms, this was achieved by using the same external `C` library with all languages. Compiled languages were shown to be more energy efficient compared to interpreted languages while running native code, but also while using an external library written in `C` (Pereira et al., 2021; Gordillo et al., 2024). Unlike the CLBG dataset, this study used libraries written mostly in their respective language when comparing ML algorithms. Although `scikit-learn` is written mostly in `Python` the energy efficiency was much closer to `C++` compared to previous studies. This difference is likely to depend on the libraries, rather than the languages. Previous studies show that `Python` is more energy efficient than `C++` in string manipulation, while this study shows that `Python` can match or surpass `C++` in other areas. Interpreted languages might be less energy efficient for the type of low-level operations performed in many benchmarks. However, the use of optimized and well-supported libraries could put them on equal footing with compiled languages.

### 7.3 Compiler Optimization Levels

When selecting which optimization level to use in regard to energy consumption, an aspect to consider involves the compilation frequency in relation to the expected number of executions. If an algorithm is expected to be compiled only once and executed many times, compilation cost can probably be disregarded, since the relative cost of compilation decreases as the number of executions increases. In such cases, an optimization level with low energy consumption during execution should be used.

In reality, determining a single optimization level that achieves low execution cost in all cases is not that simple. As seen in the results (see Section 6.4), `-O2` consumed the least amount of energy for many of the algorithms during execution. However, `-O2` and `-O3` were still quite similar, which aligns with Abdulsalam et al. (2014), that concluded that both `-O2` and `-O3` are appropriate to use when low energy cost during execution is a priority.

When frequent compilation is expected, compilation cost also becomes important. During development, the frequency of compilation and execution can often be expected to occur on a one-to-one ratio, meaning one execution per compilation. It is therefore important to evaluate the total energy consumption of execution and compilation for these use-cases. As shown in the results (see Section 6.4), each consecutive optimization level increased the energy consumption of the compiler, indicating that energy consumption increases as more optimizations are added

to the compilation process. Since the relationship between the optimization levels were similar for all the algorithms, the choice of optimization level to accommodate low compilation cost becomes easier compared to low execution cost.

For all algorithms, -00 consumed the least amount of energy during compilation. However, when looking at the total energy consumption of both compilation and execution, -00 consumed the most amount of energy for many of the algorithms, when evaluated with the small datasets. In this comparison, -01 became the least energy consuming level overall. When using the large datasets, -01 showed increased energy consumption in two of the algorithms. In these cases, -02 was the least energy consuming level.

Since both compilation and execution cost were taken into account when evaluating the total energy consumption of optimization levels, there were also greater variation between the optimization levels. Consequently, identifying a single optimization level as the most energy efficient in regard to total energy consumption became difficult. However, the -02 and -03 optimization levels showed more consistent measurements across both dataset sizes, which indicates that these are more reliable due to their predictability in the results of this study.

## 7.4 Conclusion

### Research question 1

The results (see Section 6.3) show that there was a statistically significant difference between the programming languages in execution, as the confidence intervals did not overlap for any algorithm. Therefore, **H1<sub>0</sub>**, which states that “*There is no significant difference in energy consumption during execution between interpreted and compiled programming languages, in the context of training machine learning models.*”, could be rejected.

Although a statistically significant difference between the languages existed, which language was more energy efficient changed depending on the algorithm and dataset size. While some trends in execution cost could be seen, not enough data was collected to determine any correlations. However, the energy consumption of Python was much closer to C++ in the ML algorithms compared to the baseline measurements. This difference indicates that other factors, such as external libraries, can bridge the gap in energy efficiency between languages.

### Research question 2

The results (see Section 6.3) show that there was a statistically significant difference in the total energy consumption between programming languages, as the confidence intervals did not overlap for any algorithm. Therefore, **H2<sub>0</sub>**, which states that “*There is no significant difference in total energy consumption when accounting for compilation between interpreted and compiled programming languages, in the context of training machine learning models.*”, could be rejected.

While C++ consumed more energy in every measurement and the difference was statistically significant, there are factors affecting how these results should be interpreted. Both the size of the dataset and the number of times an algorithm is executed play a role in which language should be seen as the most energy efficient. The specific use-case will therefore determine when the energy consumption of compilation needs to be considered.

### Research question 3

The results showed that there were overlapping confidence intervals between optimization levels

during execution (see Section 6.4). Since overlapping confidence intervals were found in two cases, a one-way ANOVA test was used, which showed that there was a significant difference in energy consumption during execution between all four optimization levels. Therefore, **H3<sub>0</sub>**, which stated that “*There is no significant difference in energy consumption during execution between compiler optimization levels, in the context of training machine learning models.*”, could be rejected.

A decrease in execution cost was seen for each level. Although, many of the algorithms achieved the lowest execution cost when -02 was used, the results were often similar to -03. It is therefore difficult to identify a single level as the best in terms of energy consumption during execution.

#### Research question 4

There were no overlapping confidence intervals between the optimization levels in total energy consumption, which means that there is a statistically significant difference (see Section 6.4). Therefore, **H4<sub>0</sub>**, which stated that “*There is no significant difference in total energy consumption when accounting for compilation between compiler optimization levels, in the context of training machine learning models.*”, could be rejected.

The results showed that no single optimization level achieved the lowest energy consumption overall. Notably, -00 and -01 seemed to have a larger variance in energy consumption compared to -02 and -03. While, -02 and -03 did not always achieve the lowest energy consumption, these levels showed more consistent results, making them more predictable.

## 7.5 Sustainability

To reduce energy consumption in ML, it is essential to explore strategies that support this goal. This study investigated how interpreted and compiled programming languages affect energy consumption within the context of ML. One potential benefit of this research is that it can increase awareness of strategies for improving energy efficiency in ML. Improving energy efficiency supports sustainable use of ML and similar technologies, which are seeing more widespread use. Reducing energy consumption of modern technologies aligns with UN goal 7 “Ensure access to affordable, reliable, sustainable and modern energy for all”, which has a subgoal of increasing energy efficiency (United Nations, 2024). Energy efficient ML development can lead to reduced energy consumption of the field as a whole, possibly resulting in less strain on energy infrastructure and reduced environmental footprint.

Energy efficient ML models may also benefit companies and researchers economically, as the energy consumption and its associated cost is reduced. Reduced cost could enable new companies to enter the market as ML development becomes more economically viable. Already established actors could also scale current solutions without increased costs. Furthermore, the strain on required equipment and infrastructure could also be lessened.

## 7.6 Future Work

The findings of this study can be complemented by conducting further research, addressing threats to the validity of this study.

There are several ways to improve generalizability. One way would be to include more languages and libraries. Investigating different types of algorithms would also give broader perspec-

tive on ML as a whole. Another way would involve using hardware or libraries more relevant for the industry.

This study only used a single measurement tool and metric. In order to reduce the risk of bias from the measurement tool, additional tools could be included. This study utilized `perf` to measure energy consumption, but there are several other software tools available. Another alternative is to include the use of hardware tools, similarly to Gordillo et al. (2024).

The relation between energy consumption and other factors, such as accuracy, execution time, compilation time and memory, should also be considered. Investigating these relations could reveal correlations that would otherwise remain hidden.

This study used two languages and libraries. Both libraries were implemented in their respective languages, which means there will be differences between them. To provide a more complete comparison, a library providing bindings for multiple languages could be used.

Another area of investigation is the differences between library implementations. While this study considered high-level differences, such as hyperparameters, future work could investigate the source code of the libraries. Knowledge about these differences could possibly provide explanations for the difference in energy consumption.

# Bibliography

- Abdulsalam, S., Lakowski, D., Gu, Q., Jin, T. and Zong, Z. (2014), Program energy efficiency: The impact of language, compiler and implementation choices, *in* ‘International Green Computing Conference’, pp. 1–6.
- Albonico, M., Varela, P. J., Rohling, A. J. and Wortmann, A. (2024), Energy efficiency of ROS nodes in different languages: Publisher/subscriber case studies, *in* ‘Proceedings of the 2024 ACM/IEEE 6th International Workshop on Robotics Software Engineering’, RoSE ’24, Association for Computing Machinery, New York, NY, USA, p. 1–8.
- Camargo-Henríquez, I., Martínez-Rojas, A. and Castillo-Sánchez, G. (2024), Energy optimization in software: A comparative analysis of programming languages and code-execution strategies, *in* ‘2024 9th International Engineering, Sciences and Technology Conference (IESTEC)’, pp. 507–511.
- Curtin, R. R., Edel, M., Shrit, O., Agrawal, S., Basak, S., Balamuta, J. J., Birmingham, R., Dutt, K., Eddelbuettel, D., Garg, R., Jaiswal, S., Kaushik, A., Kim, S., Mukherjee, A., Sai, N. G., Sharma, N., Parihar, Y. S., Swain, R. and Sanderson, C. (2023), ‘mlpack 4: a fast, header-only C++ machine learning library’, *Journal of Open Source Software* **8**(82), 5026.
- Free Software Foundation (2025), ‘Optimize options (using the GNU compiler collection (GCC))’, <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>. Accessed: 2025-04-03.
- García-Martín, E., Rodrigues, C. F., Riley, G. and Grahn, H. (2019), ‘Estimation of energy consumption in machine learning’, *Journal of Parallel and Distributed Computing* **134**, 75–88.
- Georgiou, S., Kechagia, M., Louridas, P. and Spinellis, D. (2018), What are your programming language’s energy-delay implications?, *in* ‘Proceedings of the 15th International Conference on Mining Software Repositories’, MSR ’18, Association for Computing Machinery, New York, NY, USA, p. 303–313.
- Gonzalez, D., Zimmermann, T. and Nagappan, N. (2020), The state of the ML-universe: 10 years of artificial intelligence & machine learning software development on GitHub, *in* ‘Proceedings of the 17th International Conference on Mining Software Repositories’, MSR ’20, Association for Computing Machinery, New York, NY, USA, p. 431–442.
- Gordillo, A., Calero, C., Moraga, M. Á., García, F., Fernandes, J. P., Abreu, R. and Saraiva, J. (2024), ‘Programming languages ranking based on energy measurements’, *Software Quality Journal* **32**(4), 1539–1580.
- Intel (2024), ‘Running average power limit energy reporting / CVE-2020-8694 , CVE-2020-8695 / INTEL-SA-00389/’, <https://www.intel.com/content/www/us/en/>

- developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html. Accessed: 2025-03-18.
- Khan, K. N., Hirki, M., Niemi, T., Nurminen, J. K. and Ou, Z. (2018), ‘RAPL in action: Experiences in using RAPL for power measurements’, *ACM Trans. Model. Perform. Eval. Comput. Syst.* **3**(2).
- Koedijk, L. and Opreescu, A. (2022), Finding significant differences in the energy consumption when comparing programming languages and programs, in ‘2022 International Conference on ICT for Sustainability (ICT4S)’, pp. 1–12.
- mlpack developers (2024a), ‘Decisiontree’, [https://www.mlpac.org/doc/user/methods/decision\\_tree.html](https://www.mlpac.org/doc/user/methods/decision_tree.html). Accessed: 2025-04-09.
- mlpack developers (2024b), ‘mlpack in C++ quickstart’, <https://www.mlpac.org/doc/quickstart/cpp.html>. Accessed: 2025-04-11.
- Noureddine, A. (2022), PowerJoular and JoularJX: Multi-platform software power monitoring tools, in ‘18th International Conference on Intelligent Environments (IE2022)’, Biarritz, France.
- Oracle (2021), ‘Oracle Linux 6 porting guide’, <https://docs.oracle.com/en/operating-systems/oracle-linux/6/porting/ch04s03.html>. Accessed: 2025-04-03.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011), ‘scikit-learn: Machine learning in Python’, *Journal of Machine Learning Research* **12**, 2825–2830.
- Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P. and Saraiva, J. (2021), ‘Ranking programming languages by energy efficiency’, *Science of Computer Programming* **205**, 102609.
- Sarker, I. H. (2021), ‘Machine learning: Algorithms, real-world applications and research directions’, *SN computer science* **2**(3), 160.
- scikit-learn developers (2025a), ‘1.10. decision trees’, <https://scikit-learn.org/stable/modules/tree.html>. Accessed: 2025-04-09.
- scikit-learn developers (2025b), ‘2.1. gaussian mixture models’, <https://scikit-learn.org/stable/modules/mixture.html>. Accessed: 2025-04-10.
- scikit-learn developers (2025c), ‘7. dataset loading utilities’, <https://scikit-learn.org/stable/datasets.html>. Accessed: 2025-03-10.
- Thai, H.-T. (2022), ‘Machine learning for structural engineering: A state-of-the-art review’, *Structures* **38**, 448–491.
- United Nations (2024), ‘Ensure access to affordable, reliable, sustainable and modern energy for all’, <https://sdgs.un.org/goals/goal7>. Accessed: 2025-04-15.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. and Wesslén, A. (2012), *Planning*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 89–116.

# A | Appendix - Optimization Level Tables

Table A.1: Energy consumption (Joules) of ML algorithms for each optimization flag, using the small datasets.

Algorithm	Flag	Compilation			Execution			Total		
		Mean	SE	CI	Mean	SE	CI	Mean	SE	CI
decision tree	-O0	195.8	0.114	0.224	140.1	0.078	0.152	335.9	0.165	0.323
	-O1	207.3	0.142	0.278	31.3	0.017	0.033	238.6	0.146	0.287
	-O2	234.9	0.135	0.265	29.4	0.016	0.032	264.3	0.141	0.276
	-O3	238.9	0.131	0.257	28.6	0.015	0.030	267.4	0.135	0.264
logistic regression	-O0	198.6	0.113	0.222	54.9	0.032	0.062	253.5	0.126	0.246
	-O1	212.0	0.133	0.260	40.8	0.021	0.041	252.8	0.138	0.270
	-O2	241.4	0.129	0.253	39.0	0.027	0.052	280.4	0.133	0.261
	-O3	250.7	0.147	0.289	39.3	0.023	0.046	290.0	0.152	0.298
bayesian ridge	-O0	193.4	0.123	0.241	33.4	0.028	0.054	226.8	0.131	0.256
	-O1	199.4	0.120	0.236	30.2	0.017	0.034	229.6	0.125	0.245
	-O2	223.5	0.144	0.283	29.9	0.017	0.034	253.3	0.150	0.294
	-O3	231.3	0.147	0.289	29.9	0.017	0.034	261.1	0.154	0.303
linear regression	-O0	189.8	0.122	0.239	32.1	0.017	0.033	221.9	0.126	0.247
	-O1	196.3	0.123	0.242	30.1	0.019	0.038	226.4	0.130	0.254
	-O2	219.5	0.146	0.286	29.7	0.025	0.048	249.2	0.153	0.300
	-O3	228.3	0.126	0.248	29.7	0.017	0.033	258.0	0.133	0.261
kmeans	-O0	265.0	0.145	0.284	669.3	0.198	0.387	934.3	0.257	0.504
	-O1	272.3	0.162	0.317	74.0	0.154	0.303	346.3	0.235	0.462
	-O2	296.7	0.191	0.374	49.9	0.035	0.069	346.6	0.200	0.393
	-O3	300.4	0.189	0.370	52.1	0.037	0.073	352.5	0.199	0.391
dbscan	-O0	272.0	0.154	0.302	147.7	0.090	0.176	419.6	0.194	0.379
	-O1	282.4	0.158	0.310	41.8	0.046	0.090	324.2	0.167	0.327
	-O2	311.9	0.175	0.344	30.9	0.020	0.040	342.8	0.183	0.358
	-O3	316.0	0.198	0.388	31.0	0.018	0.036	346.9	0.204	0.400
gmm	-O0	288.1	0.194	0.381	122.9	0.059	0.116	411.0	0.222	0.434
	-O1	307.2	0.187	0.367	64.4	0.036	0.071	371.5	0.196	0.383
	-O2	349.8	0.204	0.400	61.6	0.034	0.067	411.4	0.217	0.426
	-O3	364.2	0.222	0.436	62.9	0.047	0.093	427.2	0.233	0.458

Table A.2: Energy consumption (Joules) of ML algorithms for each optimization flag, using the large datasets.

Algorithm	Flag	Compilation			Execution			Total		
		Mean	SE	CI	Mean	SE	CI	Mean	SE	CI
decision tree	-O0	195.8	0.114	0.224	1117.9	0.562	1.102	1313.7	0.621	1.217
	-O1	207.3	0.142	0.278	266.9	0.156	0.305	474.2	0.247	0.485
	-O2	234.9	0.135	0.265	249.1	0.148	0.290	483.9	0.231	0.452
	-O3	238.9	0.131	0.257	245.5	0.137	0.268	484.4	0.223	0.436
logistic regression	-O0	198.6	0.113	0.222	450.5	0.203	0.399	649.1	0.247	0.485
	-O1	212.0	0.133	0.260	370.0	0.145	0.284	581.9	0.214	0.420
	-O2	241.4	0.129	0.253	360.1	0.148	0.290	601.5	0.221	0.433
	-O3	250.7	0.147	0.289	361.6	0.149	0.291	612.2	0.241	0.472
bayesian ridge	-O0	193.4	0.123	0.241	342.1	0.286	0.560	535.5	0.328	0.642
	-O1	199.4	0.120	0.236	310.3	0.153	0.300	509.7	0.223	0.438
	-O2	223.5	0.144	0.283	307.1	0.175	0.343	530.5	0.264	0.517
	-O3	231.3	0.147	0.289	307.4	0.230	0.450	538.6	0.283	0.554
linear regression	-O0	189.8	0.122	0.239	328.9	0.169	0.332	518.7	0.233	0.457
	-O1	196.3	0.123	0.242	309.3	0.159	0.312	505.6	0.232	0.455
	-O2	219.5	0.146	0.286	304.9	0.161	0.315	524.3	0.252	0.494
	-O3	228.3	0.126	0.248	305.3	0.266	0.521	533.6	0.322	0.631
kmeans	-O0	265.0	0.145	0.284	7813.2	1.459	2.859	8078.2	1.500	2.940
	-O1	272.3	0.162	0.317	808.7	1.485	2.910	1081.0	1.501	2.942
	-O2	296.7	0.191	0.374	550.2	0.130	0.255	846.9	0.261	0.511
	-O3	300.4	0.189	0.370	587.3	0.128	0.251	887.7	0.264	0.517
dbscan	-O0	272.0	0.154	0.302	1420.5	0.705	1.382	1692.5	0.757	1.485
	-O1	282.4	0.158	0.310	421.3	0.229	0.449	703.7	0.304	0.596
	-O2	311.9	0.175	0.344	301.3	0.168	0.329	613.2	0.278	0.546
	-O3	316.0	0.198	0.388	299.8	0.148	0.291	615.8	0.271	0.531
gmm	-O0	288.1	0.194	0.381	682.5	0.314	0.616	970.6	0.405	0.793
	-O1	307.2	0.187	0.367	400.5	0.183	0.359	707.6	0.292	0.572
	-O2	349.8	0.204	0.400	386.0	0.184	0.362	735.8	0.309	0.606
	-O3	364.2	0.222	0.436	391.0	0.181	0.354	755.2	0.304	0.595

## B | Appendix - Optimization Level Figures

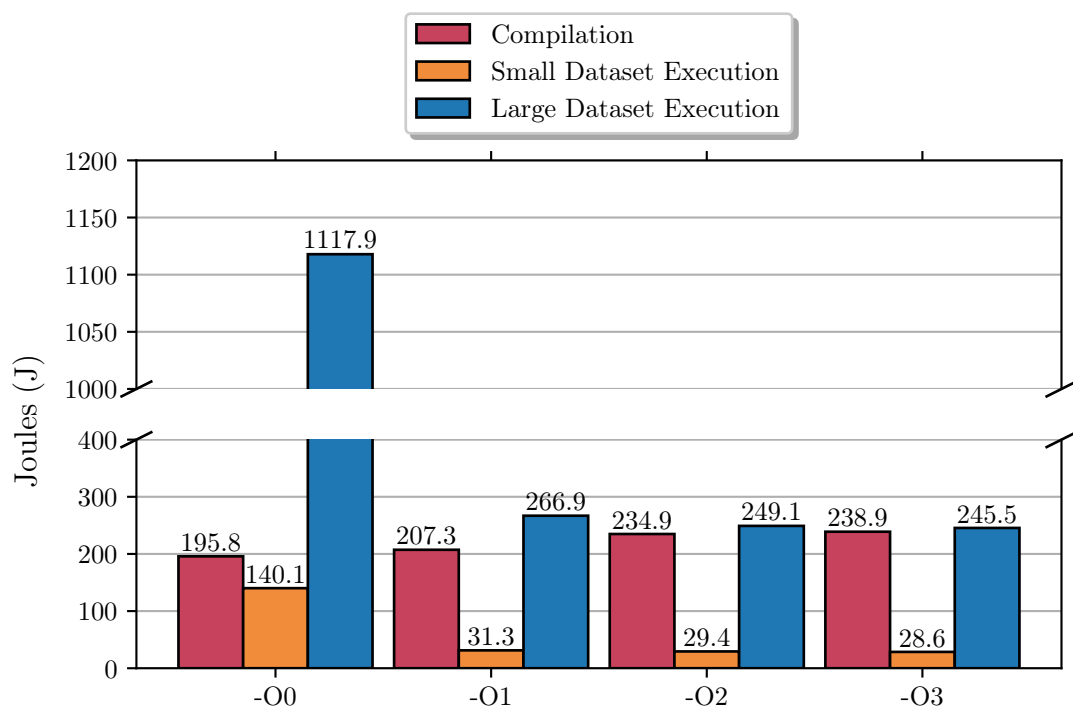


Figure B.1: Decision Tree mean energy consumption by optimization level.

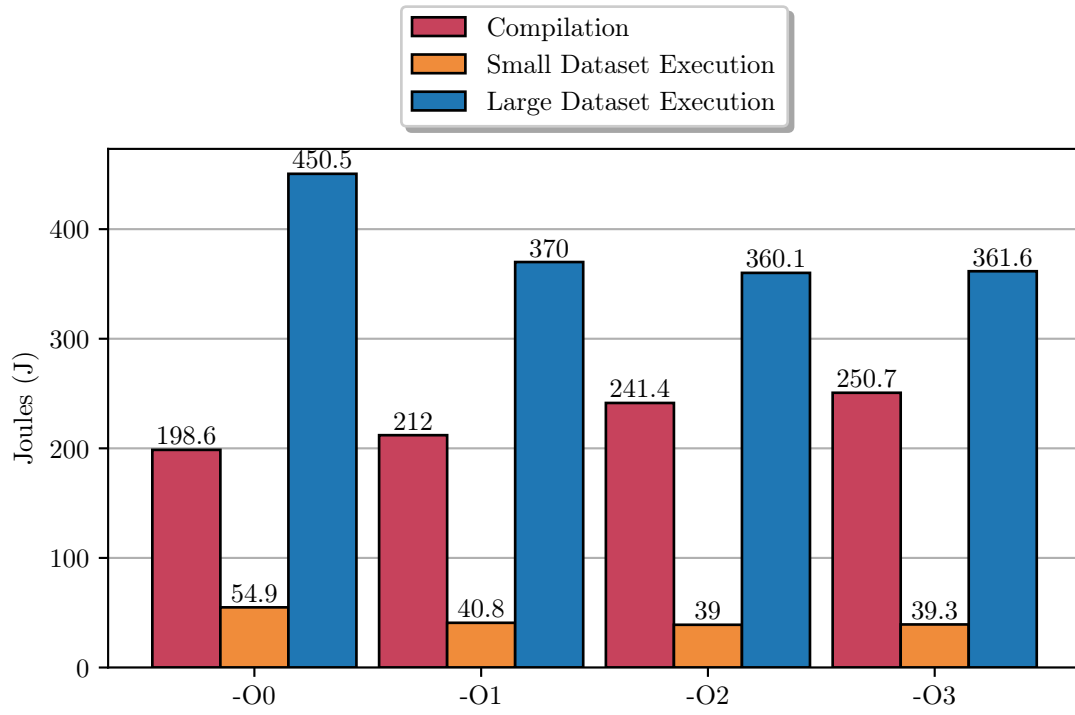


Figure B.2: Logistic Regression mean energy consumption by optimization level.

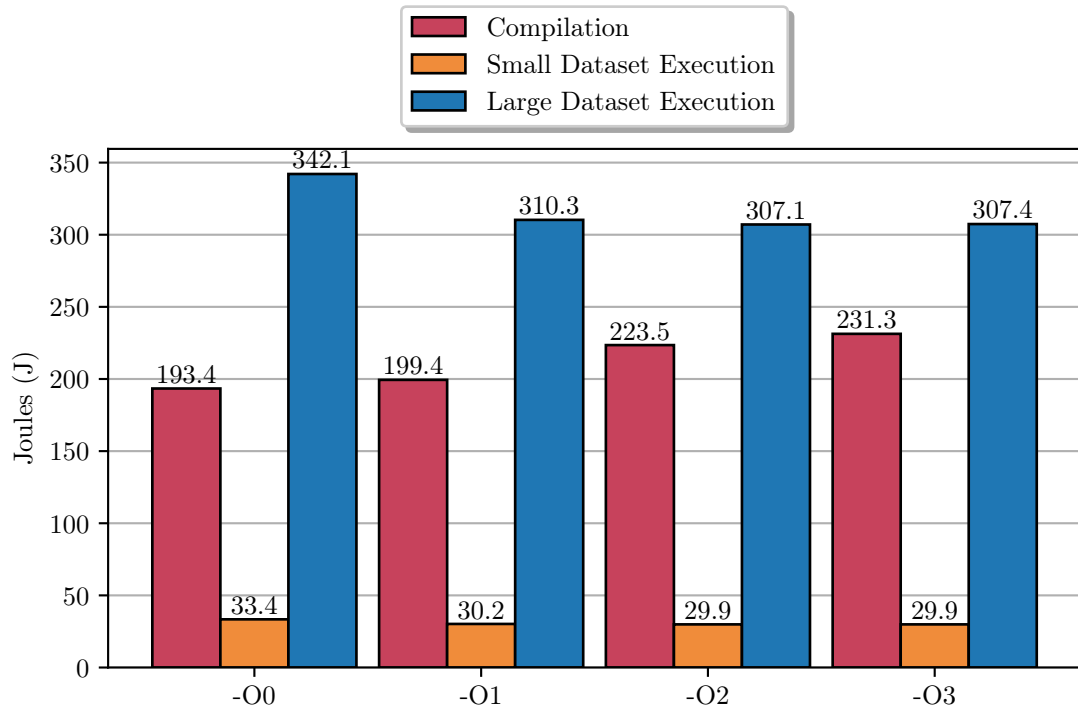


Figure B.3: Bayesian Ridge Regression mean energy consumption by optimization level.

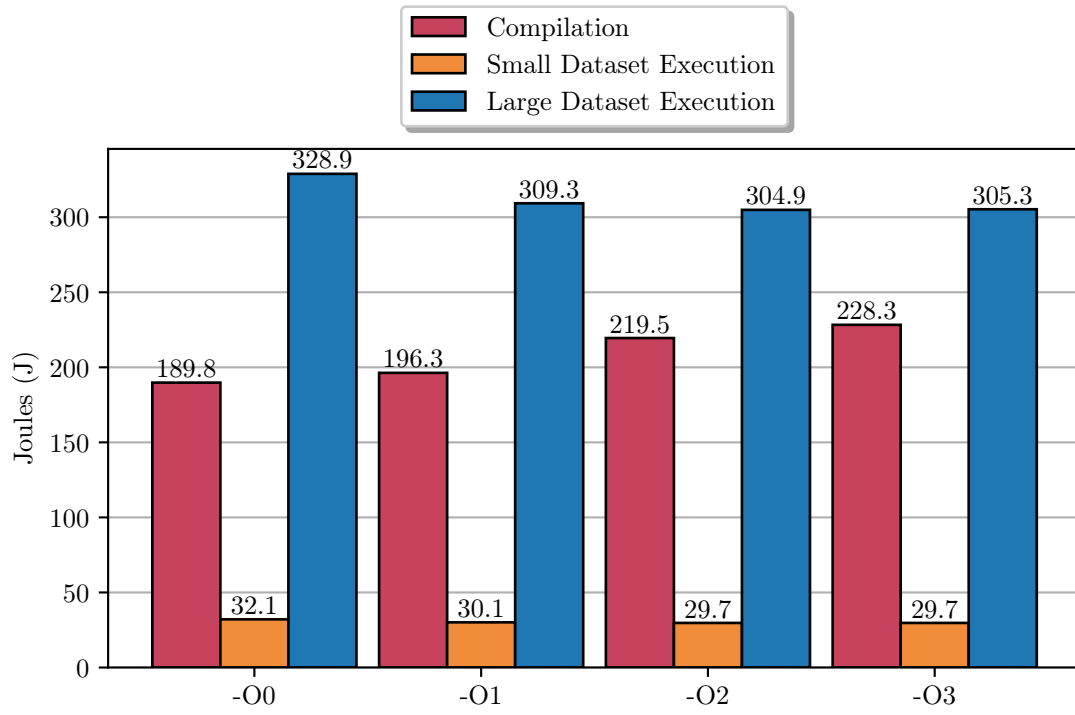


Figure B.4: Linear Regression mean energy consumption by optimization level.

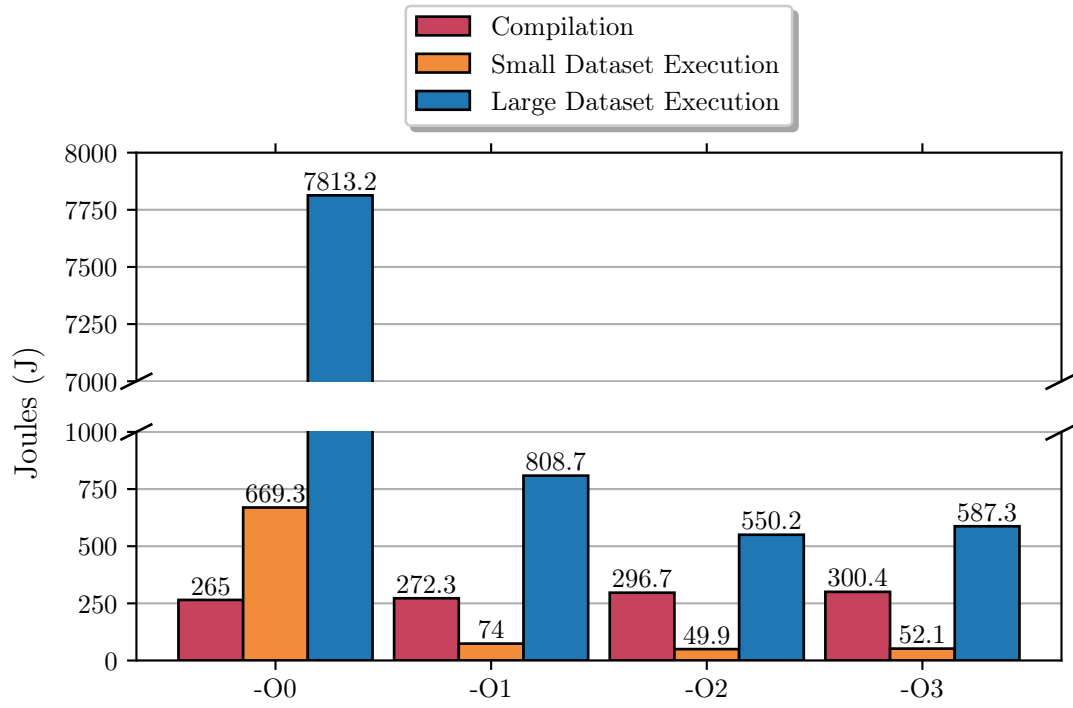


Figure B.5: K-means mean energy consumption by optimization level.

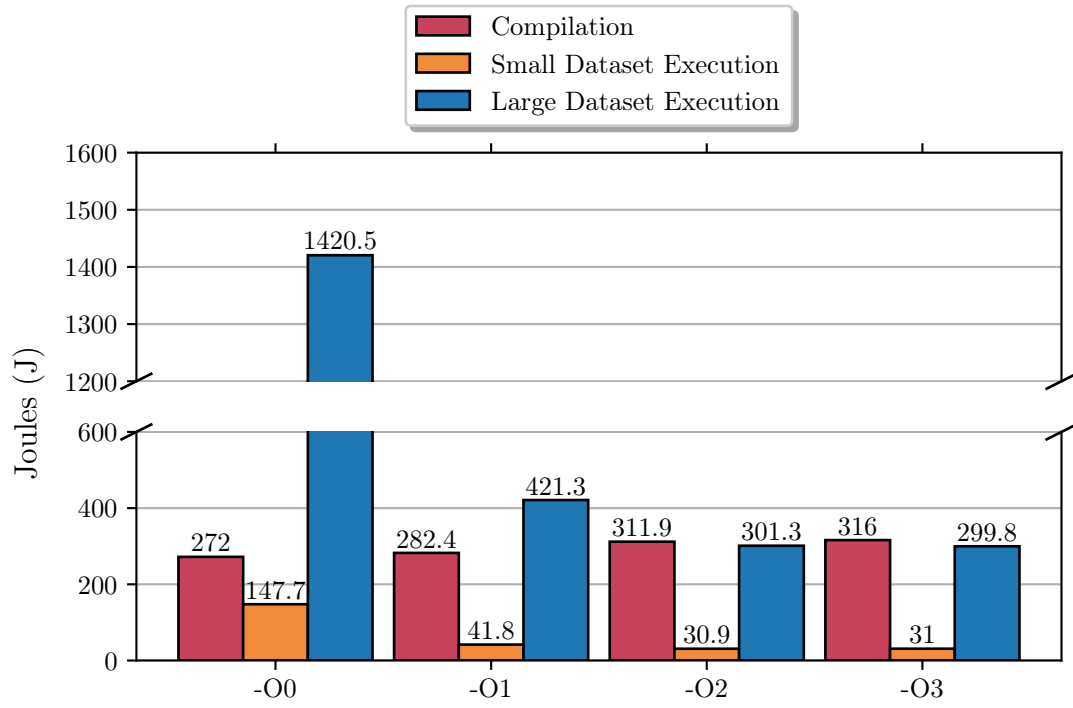


Figure B.6: DBSCAN mean energy consumption by optimization level.

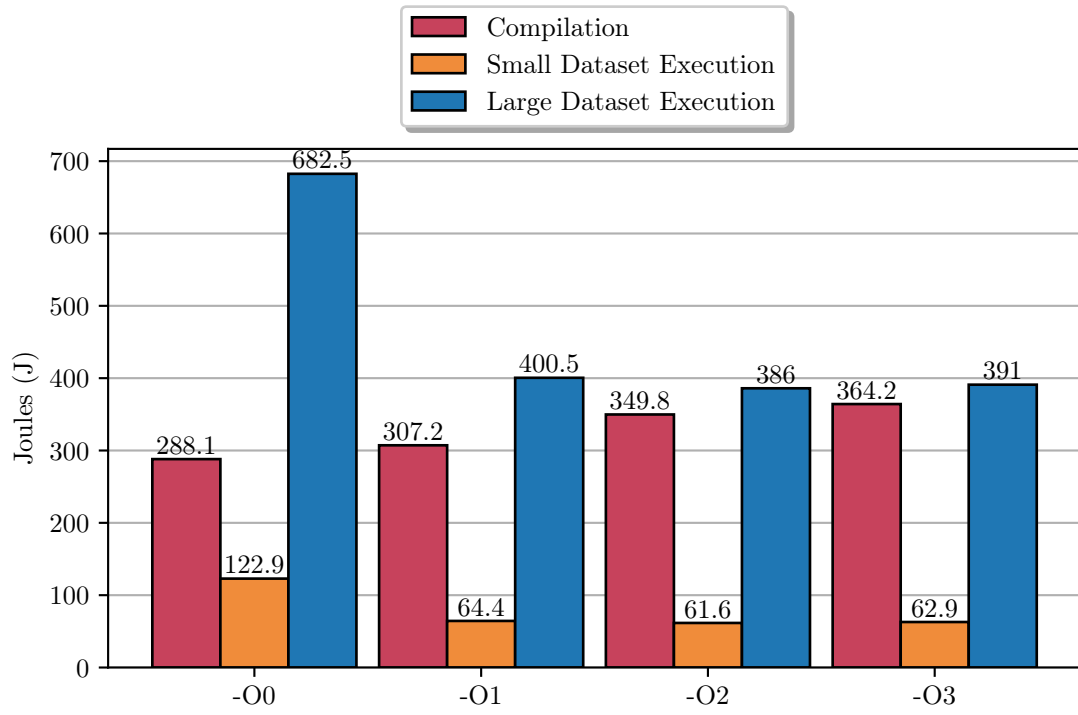


Figure B.7: GMM mean energy consumption by optimization level.

# C | Appendix - Programming Language Figures

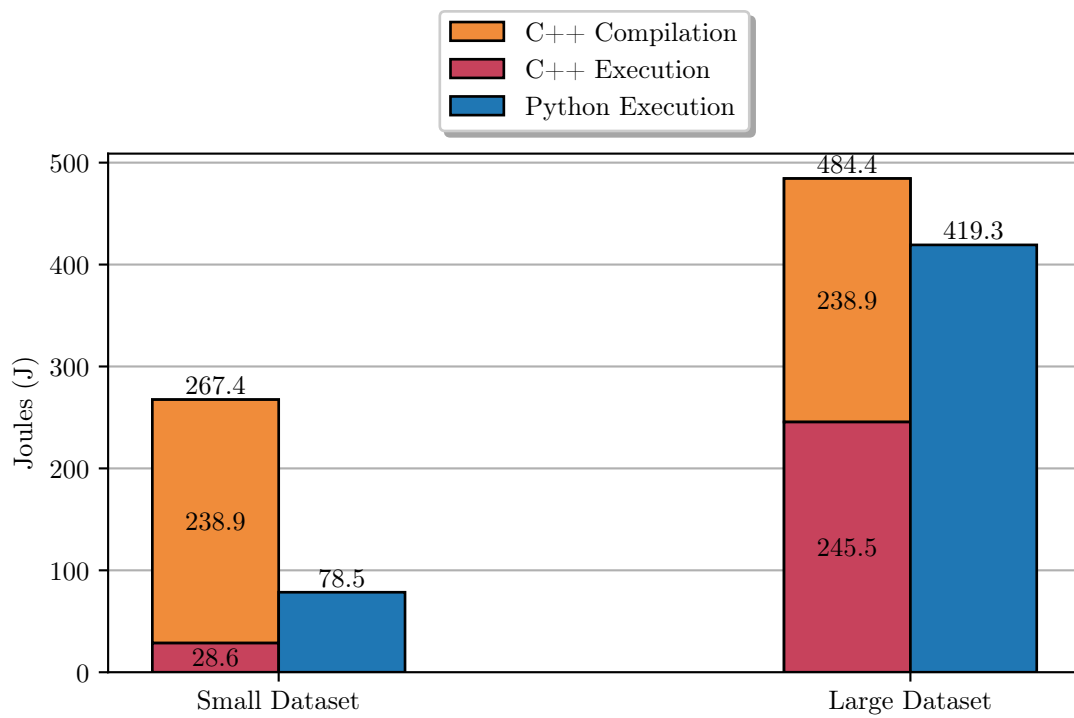


Figure C.1: Decision Tree mean energy consumption, using the -O3 optimization level.

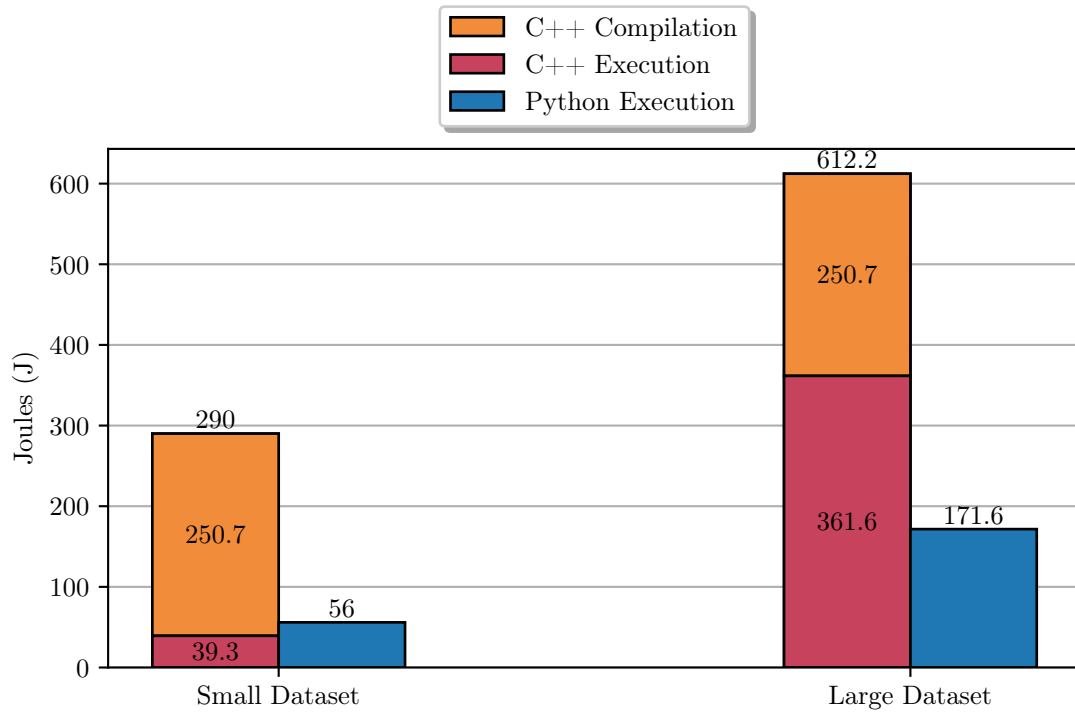


Figure C.2: Logistic Regression mean energy consumption, using the -O3 optimization level.

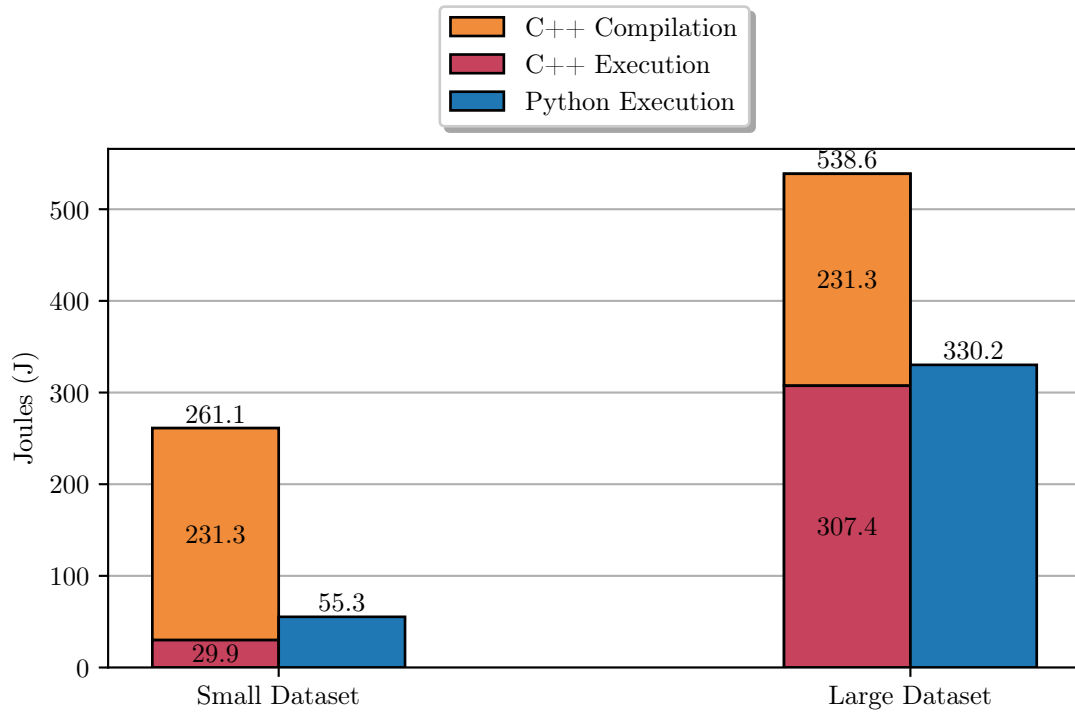


Figure C.3: Bayesian Ridge Regression mean energy consumption, using the -O3 optimization level.

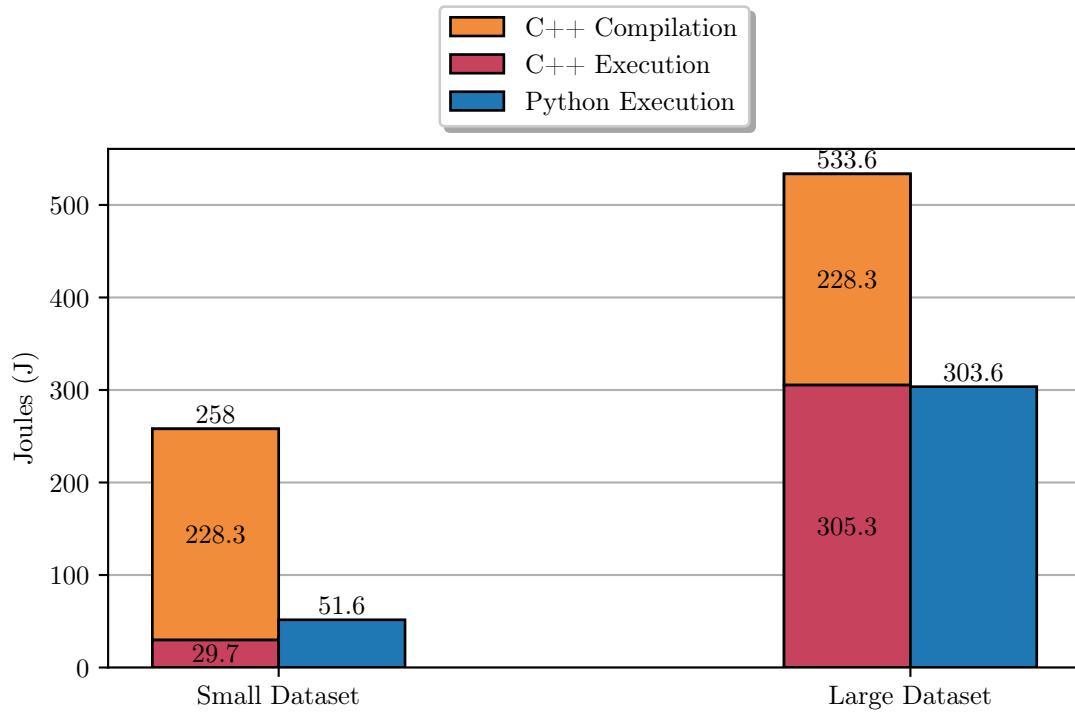


Figure C.4: Linear Regression mean energy consumption, using the -O3 optimization level.

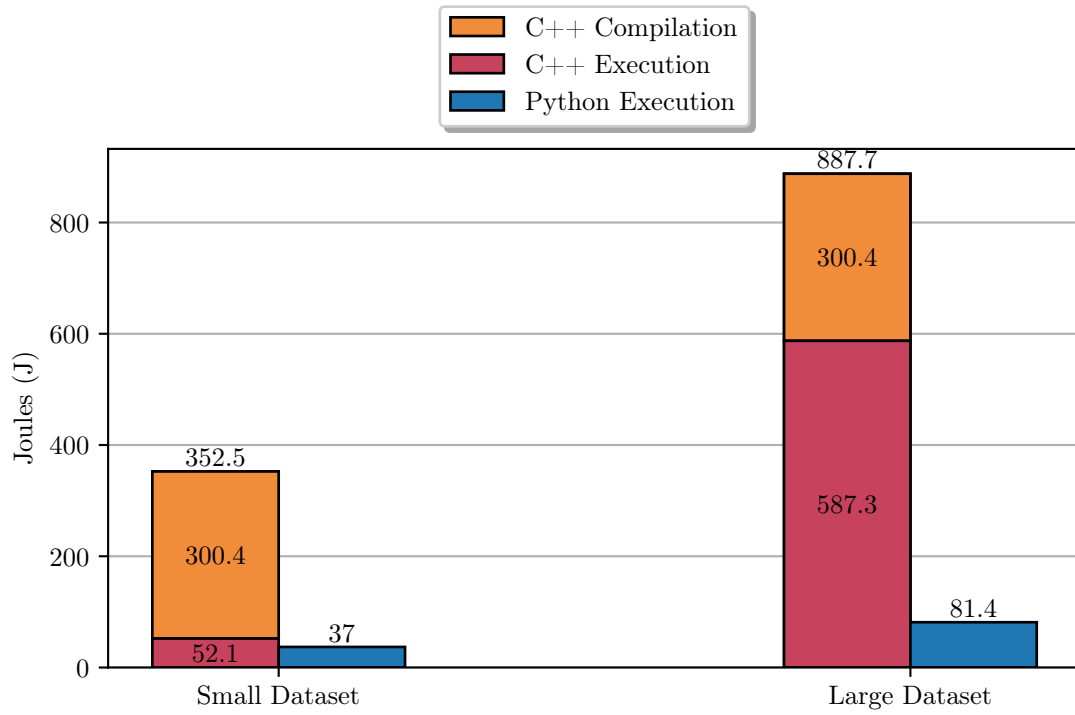


Figure C.5: K-means mean energy consumption, using the -O3 optimization level.

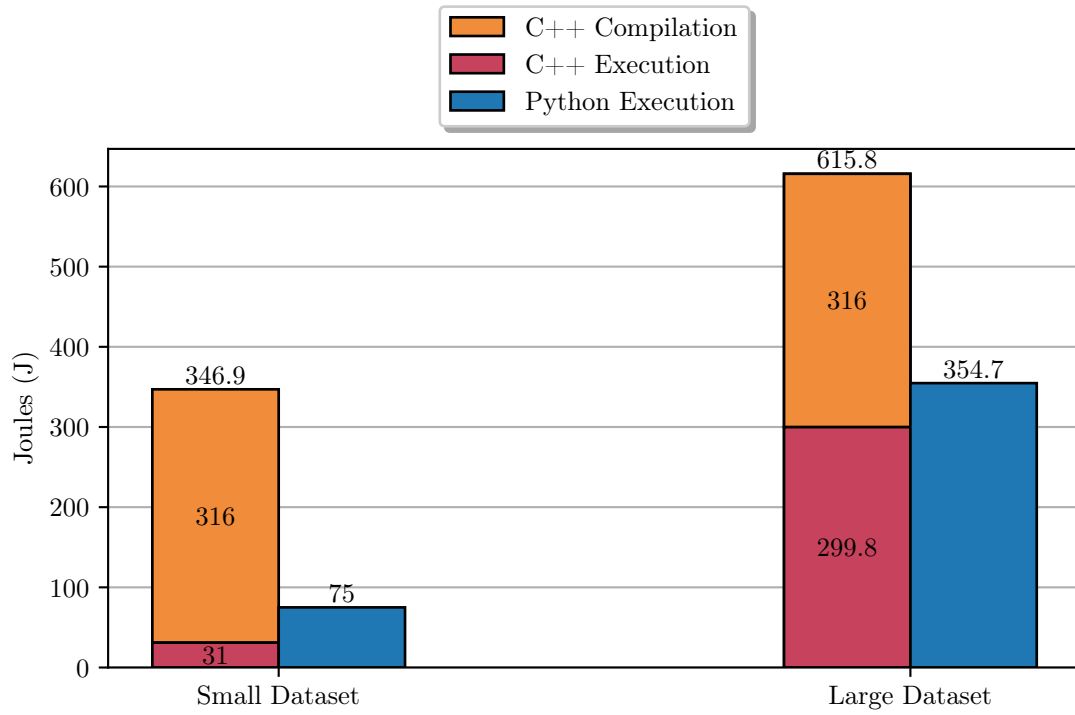


Figure C.6: DBSCAN mean energy consumption, using the -O3 optimization level.

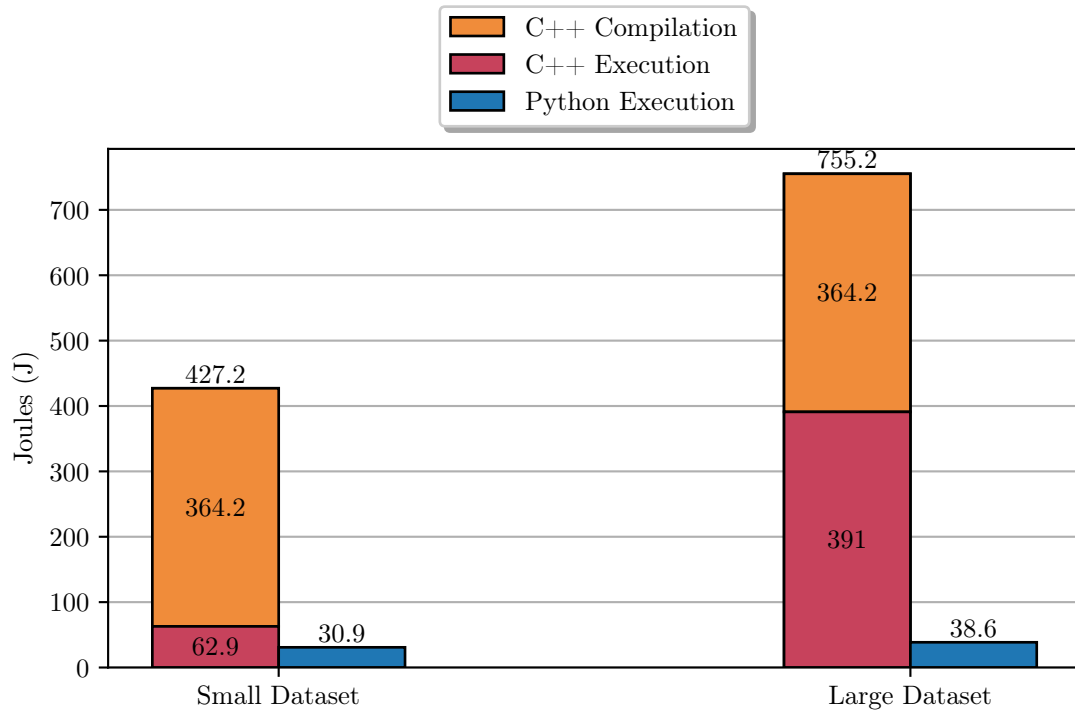


Figure C.7: GMM mean energy consumption, using the -O3 optimization level.