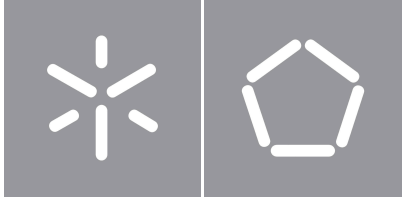


University of Minho
School of Engineering

Nuno Filipe Quintela Teixeira Carvalho

Performance and Carbon Footprint Modelling of Irregular Tasks

february 2025



University of Minho
School of Engineering

Nuno Filipe Quintela Teixeira Carvalho

Performance and Carbon Footprint Modelling of Irregular Tasks

Master's Dissertation in Physics Engineering

Dissertation supervised by
André Martins Pereira

february 2025

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to users of this work:



CC BY

https://creativecommons.org/licenses/by/4.0/ *[Esta licença permite que outros distribuam, remixem, adaptem e criem a partir do seu trabalho, mesmo para fins comerciais, desde que lhe atribuam o devido crédito pela criação original. É a licença mais flexível de todas as licenças disponíveis. É recomendada para maximizar a disseminação e uso dos materiais licenciados.]*

Acknowledgements

As enjoyable as it has been to explore the fields of machine learning and High-Performance Computing Systems in this thesis, the journey was not without its ups and downs—moments of both fulfillment and failure. Throughout it all, there are people to whom I owe my deepest gratitude, and I will forever be thankful for the pleasure of having their unwavering support.

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor André Martins Pereira, for giving me the opportunity to work under his guidance. He has always been available, offering his support with professionalism and a friendly demeanor.

I am also profoundly grateful to my family for providing me with the opportunity to pursue a path of righteousness and success. My mother, Rosa, and my father, Manuel, made the most of what they had to give me a chance at success. My siblings—Rute, Cristiano, Tânia, and Ana—have always believed in my potential, supporting me in countless ways throughout this journey. I would also like to thank their respective partners, Joel, Davide, and Tânia R., who, though not blood-related, are an integral part of our family and offered their help and support in the same way. I can't forget the little ones—my nephews, Rui, Rodrigo, and Bernardo, as well as my goddaughter Francisca— who grew up too quickly alongside me and inspired me to strive to be a better role model.

A heartfelt thank you goes out to my friends, Bernardo and Hélder, who shared this journey with me. Their camaraderie, mutual help, and the unforgettable memories we created together made this experience much easier and more enjoyable.

Finally, I would like to extend my deepest thanks to my girlfriend, Maria, whose unwavering support, patience, and kindness helped me stay grounded and focused on achieving my goals. I will never forget the late nights and early mornings, and all the things you could have done but chose instead to stay by my side, keeping me company and offering your support.

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, february 2025

Nuno Filipe Quintela Teixeira Carvalho

Abstract

High-Performance Computing (HPC) systems face increasing challenges in energy efficiency and resource management as computational demands grow. A central concern is the carbon footprint of irregular tasks, which exhibit unpredictable runtime and power consumption patterns, complicating job scheduling. Efficient scheduling requires differentiating between regular, predictable jobs and irregular ones, allowing for customized scheduling that reduces energy consumption and enhances performance. Furthermore, there is a notable lack of real-world HPC data, which necessitates the creation of synthetic datasets. These datasets are essential for enabling predictive modeling of job characteristics, as machine learning models require substantial data volumes for optimal accuracy.

To address these challenges, this study explores machine learning-based techniques for predicting job characteristics in HPC systems. By generating and analyzing a synthetic dataset derived from existing job metrics, predictive models capable of classifying regular versus irregular jobs and estimating key performance metrics was developed. Specifically, an ExtraTreesClassifier model, achieving 96.7% accuracy in job classification. Additionally, subsequent regression models predict runtime and power consumption, attaining a Mean Absolute Error (MAE) of 495 seconds and 280 watts, respectively. These predictions support more efficient scheduling strategies, optimizing both resource utilization and energy demands in HPC systems.

Keywords High-Performance Computing, Power Consumption Prediction, Runtime Prediction, Machine Learning, Supervised Learning, Synthetic Data.

Resumo

Os sistemas de Computação de Alto Desempenho (HPC) enfrentam desafios crescentes em eficiência energética e gestão de recursos à medida que as exigências computacionais aumentam. Uma preocupação central é a pegada de carbono de tarefas irregulares, que apresentam padrões imprevisíveis de tempo de execução e consumo de energia, dificultando o agendamento de tarefas. O agendamento eficiente requer a distinção entre tarefas regulares e previsíveis e tarefas irregulares, permitindo um agendamento personalizado que reduz o consumo de energia e melhora o desempenho. Além disso, existe uma escassez significativa de dados reais de HPC disponíveis, o que torna essencial a criação de conjuntos de dados sintéticos. Estes conjuntos de dados são cruciais para possibilitar a modelagem preditiva das características das tarefas, pois os modelos de aprendizagem automática exigem grandes quantidades de dados para uma performance ótima.

Para abordar estes desafios, este estudo investiga técnicas baseadas em aprendizagem automática para prever as características das tarefas em sistemas de HPC. Através da criação e análise de um conjunto de dados sintético derivado de métricas de tarefas existentes, desenvolveram-se modelos preditivos capazes de classificar tarefas regulares versus irregulares e de estimar métricas de desempenho chave. Especificamente, utilizaram-se um ExtraTreesClassifier, atingindo uma precisão de 96,7% na classificação das tarefas. Adicionalmente, modelos de regressão subsequentes prevêem o tempo de execução e o consumo de energia, alcançando um Erro Médio Absoluto (MAE) de 495 segundos e 280 watts, respetivamente. Estas previsões apoiam estratégias de agendamento mais eficientes, otimizando tanto a utilização de recursos quanto as necessidades energéticas em sistemas de HPC.

Palavras-chave Computação de Alto Desempenho, Previsão de Consumo de Energia, Previsão de Tempo de Execução, Aprendizagem Automática, Aprendizagem Supervisionada, Dados Sintéticos.

Contents

I Introductory material 1

1 Introduction 2

1.1 Motivation . . . 3
1.2 Challenges and Goals . . . 4
1.3 Block Diagram of the Goals . . . 5
1.4 Document Structure . . . 5

2 State of the Art 7

2.1 Computational tasks . . . 7
2.1.1 Regular Tasks . . . 9
2.1.2 Execution Time and Power Consumption in Regular Tasks . . . 9
2.1.3 Irregular Tasks in Computational Systems . . . 10
2.1.4 Impact of Irregular Tasks on Execution Time and Power Consumption . . . 10
2.2 Heterogeneous Systems . . . 11
2.3 The Scheduling Problem . . . 13
2.3.1 Static vs. Dynamic Task Scheduling . . . 14
2.3.2 Energy Considerations in Task Scheduling . . . 15
2.3.3 Scheduling with Machine Learning Approaches . . . 15
2.4 Significance of Runtime and Power Consumption Prediction in Scheduling . . . 16
2.4.1 The Role of Runtime Prediction . . . 16
2.4.2 The Role of Power Consumption Prediction . . . 17
2.4.3 Advances in Runtime and Power Consumption Prediction . . . 17
2.5 Machine Learning and Artificial Intelligence . . . 18
2.5.1 Types of Machine Learning . . . 18

2.5.2 Supervised Learning . . . 19
2.5.3 Deep Learning . . . 21
2.6 Dataset Collection . . . 22
2.7 Synthetic Data Generation . . . 23
2.7.1 Kernel Density Estimation (KDE) . . . 24
2.7.2 Generative Adversarial Networks (GANs) . . . 24
2.7.3 Variational Autoencoders (VAEs) . . . 24

II Core of the Dissertation 26

3 Assembling the Dataset 27
3.1 Base Datasets. . . . 27
3.1.1 Data Distribution . . . 28
3.1.2 The Power Consumption metric. . . . 29
3.2 Data Analysis . . . 29
3.3 Synthetic Data . . . 32
3.3.1 Dataset Blending . . . 33
3.3.2 Labeling Process . . . 33
3.3.3 Data Augmentation . . . 34
3.3.4 Dataset Summary . . . 36

4 Models for job classification 37
4.0.1 Performance Metrics: . . . 37
4.1 Different classifiers performance: . . . 38
4.1.1 Scaling Data. . . . 38
4.1.2 The `ExtraTreesClassifier` Model . . . 41
4.1.3 Model Complexity Performance . . . 43
4.1.4 Feature Importance analysis. . . . 44
4.1.5 Losing accuracy for better situational performance. . . . 46
4.1.6 Case Study: . . . 47
4.2 Discussion: Explanation of the Results . . . 49

5 Prediction Models 53

5.1 Regression Performance Metrics . . . 53
5.2 Run Time Prediction . . . 54
5.2.1 Different Regression models performance . . . 55
5.2.2 Random Forest Performance Analysis . . . 56
5.2.3 Model Pipeline . . . 58
5.2.4 The Models in Stacking . . . 60
5.2.5 Regression Models For the Classes . . . 61
5.2.6 The Models in the Regression Process . . . 65
5.2.7 Run Time Predictions Result Discussion . . . 66
5.3 Power Consumption Prediction . . . 68
5.3.1 Regression Models predicting Power consumption . . . 69
5.3.2 Understanding Power consumption . . . 69
5.3.3 Introducing Mean Power Consumption in the dataset (Power Consumption per sample) . . . 70
5.3.4 Repeating the Power Consumption prediction with different models . . . 70
5.3.5 Complexity Analysis . . . 71
5.3.6 Feature Importance . . . 74
5.3.7 Further Analyzing the model . . . 74
5.3.8 Contextual Analysis of MAE and R^2 . . . 75
5.3.9 Power Consumption Predictions Result Discussion . . . 78

6 Conclusions and Future Work **82**
6.1 Prospects for Future Work . . . 84

III Appendices **93**

A Details of results **94**

B Listings **97**

C Tooling **101**

List of Figures

1 Block Diagram with step by step workflow in this thesis. 5

2 Heterogeneous system architecture. 12

3 Distributions of the different variables in the HPC-ODA dataset. 28

4 Distributions of the different variables in the PM100 dataset. 29

5 `HPC-ODA` dataset analysis. 31

6 `PM100` dataset analysis. 32

7 Distribution of the relationships between power and other variables in the HPC-ODA dataset. 33

8 Correlation matrix of the blended dataset. 34

9 Correlation matrix of the extended dataset. 35

10 Supervised Learning classifier models performance in job classification. 39

11 Supervised Learning classifier models performance in job classification (Data Scaled). . 40

12 Different `scalers` performance in job classification with `ExtraTreesClassifier` model. 41

13 Tree 0 of the Classification Model 43

14 `ExtraTreesClassifier` model performance with increasing complexity. 44

15 `ExtraTreesClassifier` model performance across different number of variables chose by `RFE`. 45

16 Feature importance in the job classification model. 46

17 Feature importance in the classification model with polynomial features introduced. . . . 47

18 ROC curve and Confusion Matrix of the model with standard weights and threshold. . . . 48

19 ROC curve and Confusion Matrix with changed weights and threshold. 49

20 Neural Network Shape. 50

21 Simple Neural Network performance in job classification. 51

22 More complex Neural Network performance in job classification. 51

23 Classifier model performance across training, testing and validation splits and overall accuracy. . . . 52
24 Data analysis for the dataset containing only irregular jobs. . . . 53
25 Supervised Learning Regression models performance in Time prediction. . . . 55
26 Different `scalers` performance in run time regression predictions with `RandomForestRegressor` model. . . . 56
27 `RandomForestRegressor` model performance with increasing complexity. . . . 57
28 Run time regressor pipeline model architecture. . . . 59
29 Scatter plot of the run time regressor pipeline model. . . . 63
30 Run time prediction pipeline model diagram with additional class. . . . 63
31 Confusion Matrix for the Final Multi-Classifier model in run time prediction. . . . 64
32 Scatter Plot of the Final Model. . . . 65
33 Scatter plot of the run time regressor final model across all data. . . . 67
34 Residual error plot of the run time regressor final model across all data. . . . 67
35 Initial power prediction supervised learning regression models using the total power feature. 69
36 Mean Power relations with existent dataset features. . . . 71
37 Initial power prediction supervised learning regression models using the power consumption per sample feature. . . . 72
38 `RandomForestRegressor` model performance with increasing complexity. . . . 75
39 Feature Importance in power consumption per sample prediction with the `RandomForestRegressor` model. . . . 76
40 Different `scalers` Performance in the power consumption prediction `RandomForestRegressor` model. . . . 77
41 Cumulative Distribution Function of the power consumption per sample metric. . . . 78
42 Power consumption distribution across different percentile and their respective associated prediction error. . . . 79
43 Actual values vs. Predicted values of power consumption, bar and scatter plots. . . . 80
44 Scatter plot of the power consumption regressor final model across all data. . . . 81
45 Residual error plot of the power consumption regressor final model across all data. . . . 81
46 Overall developed model performance across all steps. . . . 84
47 Data Augmented with KDE. . . . 94

48 Data Augmented with GAN. 94
49 Classifier Models Performance in Multi-Classification of the run time data. 95
50 Stacking Multi-Classifier of run time data Confusion Matrix for the whole dataset. 95
51 Regressor performance within each class of run time prediction. 96

List of Tables

1 Table of job attributes within the synthetic generated dataset. 36

2 Table of Run Time data statistics. 54

3 Performance metrics for `RandomForestRegressor` in run time prediction with different `n_estimators` 58

4 Mean Absolute Error (MAE) across different percentiles of run time data distribution. . . 58

5 Features utilized for different class of run time regression models. 61

6 Mean Absolute Error (MAE) for the run time regressor pipeline model across different percentiles. 62

7 Mean Absolute Error (`MAE`) for the run time pipeline model across different percentiles (extra split). 64

8 Table of Power Consumption data statistics. 70

9 Table of Mean Power Consumption data statistics. 71

10 Table of power consumption prediction model results analysis. 77

11 Summary of `ExtraTreesClassifier` Parameters used in 4 97

12 Summary of `RandomForestRegressor` Parameters used for power prediction in 5.3 98

13 Summary of `StackingClassifier` and Base Learners Parameters used in run time prediction 5.2.3. 99

14 Summary of `RandomForestRegressor` Parameters for Classes 0, 1 used in run time prediction 5.2.3 100

15 `ExtraTreesRegressor` Parameters for Classes 2,3,4 and 5 used in run time prediction 5.2.3 100

16 Summary of `HuberRegressor` Parameters for Class 6 used in run time prediction 5.2.3. 100

17 Versions of Software and Libraries 101

Part I

Introductory material

Chapter 1

Introduction

Data centers, the backbone of modern computing infrastructure, consist of servers that serve as the fundamental building blocks for processing and storing vast amounts of data. The architecture of these servers can be broadly categorized into two types: homogeneous and heterogeneous. A homogeneous server environment is characterized by uniformity in server hardware and software configurations. All servers in the infrastructure share similar or identical specifications, such as CPUs architecture, memory capacity, storage devices, and operating systems. This uniformity simplifies management and maintenance tasks and is often implemented in data centers where simplicity and ease of administration are critical. Homogeneous servers are suitable for a wide range of workloads without over-specializing in a specific problem.

Meanwhile, a heterogeneous server environment is characterized by diversity in server hardware and software configurations. Heterogeneous servers have the flexibility to employ various processor types and cores, incorporating accelerators such as GPUs alongside conventional processing units. This flexibility is advantageous when dealing with varied computational requirements or when integrating existing systems with different specifications. Therefore, heterogeneous environments are common in scenarios where different applications are usually handled by specialized hardware configurations.

It is crucial to acknowledge that energy consumption is a concern, as computational power is often directly related to energy consumption. The projected estimation that data centers and High-Performance Computing (HPC) systems (which encompass both homogeneous and heterogeneous systems) could contribute up to 8% of global carbon dioxide emissions by 2030 raises significant environmental concerns Baolin et al. [2023]. This emphasizes the need for sustainable practices and energy-efficient technologies in the ever-expanding landscape of server infrastructures. Thus, energy-related challenges associated with HPC systems arise, prompting the exploration of solutions to mitigate their environmental impact Roblex et al. [2023]. Despite these concerns, there is a growing recognition of the potential for energy savings, especially when leveraging the distinct advantages of heterogeneous servers for specific workloads Da-

yarathna et al. [2016]. As organizations navigate the intricate balance between performance and energy efficiency, effective job scheduling emerges as a critical factor not only in optimizing energy consumption but also in reducing job run times within heterogeneous server environments Pawel Czarnul [2019], Silva et al. [2024].

1.1 Motivation

Job scheduling is a critical aspect of computer systems that involves deciding the order in which various jobs are executed on a computer's resources Pinedo [2012]. Imagine a scheduler as a busy manager constantly assigning jobs to different employees; job scheduling is like the manager prioritizing and assigning time slots for each job, ensuring that they are processed efficiently and in a balanced manner. The goal is to maximize the use of computing resources, such as CPUs and GPUs, to enhance overall system performance. Schedulers make decisions based on factors like job priority, expected execution times on each different worker, and the current state of the system. Efficient job scheduling is crucial for maintaining system responsiveness, fairness, and optimal utilization of computing resources, contributing to the smooth and effective operation of computer systems.

Jobs are computational workloads submitted for execution by the processing units in a computer system and can encompass a wide range of activities, from user-initiated commands or applications to background processes and system-level operations. Each job represents a specific set of instructions or computations that the CPUs or GPU must perform. For example, a job could involve running a specific program, processing data, rendering graphics, handling input/output operations, or managing system resources.

These jobs can be categorized into two main types: regular and irregular. Regular jobs exhibit well-defined deterministic behavior with predictable periodicity, characterized by fixed timing requirements. Conversely, irregular jobs exhibit variability in runtime and power consumption that depends on the workload distribution, with a consistent processing time for each individual dataset element, even if total runtimes fluctuate (discussed further in Section 2.1).

This distinction between regular and irregular jobs is fundamental in job scheduling strategies Pinedo [2012], where regular jobs typically adhere to static scheduling algorithms, while irregular jobs require dynamic scheduling algorithms capable of accommodating their non-uniform temporal characteristics. Effective scheduling aims to minimize a specific objective function—often the total execution time or energy usage of a workload—rather than simply the execution time of individual jobs. Furthermore, the capability

to predict the characteristics of irregular jobs enables a scheduler to allocate them more effectively Bielecki and Śmiałek [2023].

Regular jobs are generally managed by recurrent scheduling algorithms, which are strategies used in computer systems to manage and execute jobs at regular intervals or according to fixed patterns. For instance, in a system where jobs represent various computing operations, recurrent scheduling algorithms ensure that these jobs are performed predictably and consistently, with known timing intervals. However, using dynamic scheduling algorithms for regular jobs can introduce unnecessary overhead, as regular jobs typically do not require frequent adjustments.

In contrast, dynamic scheduling algorithms are used to schedule irregular jobs and are designed as flexible strategies that adapt job execution based on changing conditions. Unlike fixed or periodic schedules, dynamic scheduling allows the system to adjust job priorities and allocations in real-time, making it valuable for handling unpredictable workloads, varying job priorities, or dynamically changing environments. However, static scheduling approaches are often inadequate for irregular tasks, as they lack the flexibility to adapt to varying resource demands. Therefore, distinguishing between job types is essential for optimal scheduling, which requires information about the job characteristics before allocation. If a scheduler can understand the job's behavior beforehand, it will be better equipped to handle it effectively and align with overall optimization goals (see Section 2.3).

1.2 Challenges and Goals

To optimize job scheduling in heterogeneous HPC systems, this work aims to develop a model that can predict job characteristics before execution. Understanding whether a job is regular or irregular, and identifying its specific resource and runtime needs, allows for better scheduling decisions. For regular jobs, historical data can provide insights into execution patterns. For irregular jobs, predictive models are necessary to estimate runtime and energy consumption, ensuring that resources are allocated efficiently and system performance is maximized.

Efficient job characterization is crucial for minimizing execution delays and energy consumption. The characterization process must be swift to avoid adding overhead that outweighs its benefits. Predictive modeling and low-power algorithms can provide rapid and accurate job insights, enhancing scheduling strategies without incurring significant computational or energy costs.

In summary, this research investigates job characterization and scheduling strategies in HPC environments to enhance resource allocation, reduce execution times, and minimize energy usage. By predicting

job behavior and understanding resource demands, HPC systems can achieve better performance and cost-efficiency, ensuring optimal management of complex and resource-intensive computations.

1.3 Block Diagram of the Goals

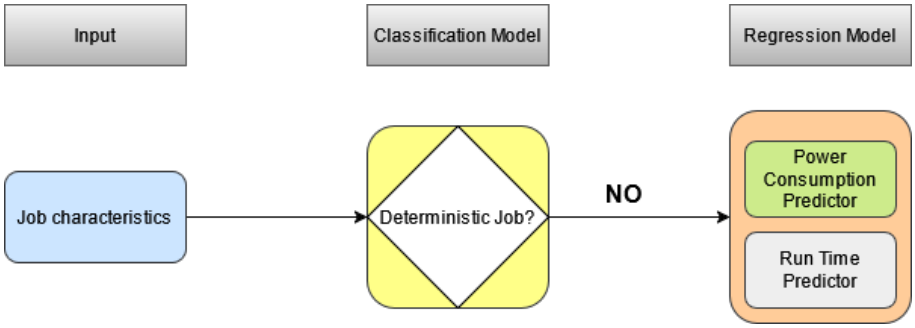


Figure 1: Block Diagram with step by step workflow in this thesis.

As illustrated in the block diagram in Figure 1, the first phase involved developing a Machine Learning (ML) algorithm designed to predict job characteristics within the computational framework. This algorithm was trained to distinguish between regular and irregular jobs, informing subsequent scheduling decisions. For regular jobs, the algorithm facilitated straightforward planning and resource allocation using standard recurrent scheduling techniques. In contrast, irregular jobs require the algorithm to predict execution times and power consumption, enabling adaptive scheduling strategies tailored to these dynamic workloads.

The ML algorithm was rigorously trained using job-related metrics to accurately predict the runtime and power consumption associated with each job. This predictive modeling serves as a crucial tool for preemptively assessing the resource requirements of jobs. By integrating Machine Learning techniques, this approach can optimize scheduling methods and resource allocation within HPC systems, leading to a more efficient and streamlined computational workflow.

1.4 Document Structure

This thesis is organized as follows:

- **Chapter 2** provides essential background information on the key topics explored in this work, including HPC system architectures, the characteristics and distinctions of computational jobs, job scheduling algorithms, as well as an overview of Machine Learning techniques and synthetic data generation.

- **Chapter 3** describes the processes involved in dataset creation, focusing on the synthetic data generation and augmentation methods employed to develop a robust dataset suitable for Machine Learning applications in HPC job prediction.

- **Chapter 4** introduces the Machine Learning models designed to classify jobs as regular or irregular, along with the metrics utilized for evaluating model performance.

- **Chapter 5** outlines the methodology and models used to predict job runtime and power consumption. This chapter explores the implications of these predictions on resource allocation and energy efficiency in job scheduling, providing analysis and discussion of the obtained results.

- **Chapter 6** concludes the thesis by summarizing the contributions of this research, highlighting its potential impact on HPC scheduling optimization, and proposing directions for future research, including enhancements to the models and exploration of additional datasets.

This structured approach provides a comprehensive understanding of how predictive modeling can contribute to the efficiency of HPC systems by enabling optimized job scheduling tailored to specific resource and energy requirements.

Chapter 2

State of the Art

To achieve the objectives set for this thesis—characterizing job performance and carbon footprint in heterogeneous (HPC) systems— it is essential to contextualize and explore several interrelated topics. This chapter aims to provide an overview of these subjects to establish the foundational knowledge required for the study.

First, it is crucial to define what constitutes a computational job or task and understand what makes it regular or irregular. Next, the distinction between the two primaryHPC architectures, homogeneous and heterogeneous systems, must be clarified, along with how these architectures influence job behavior and performance. Understanding these factors is key to recognizing the role of scheduling in computational frameworks.

Following this, the importance of job scheduling within HPC systems is examined, illustrating how efficient scheduling can impact overall system performance and energy consumption. Then the chapter delves into the processes of runtime and power consumption prediction, emphasizing their relevance to improving efficiency in HPC environments.

Finally, the discussion will cover the necessity of data collection and analysis to support prediction models, laying the groundwork for understanding the approaches used to optimize both performance and sustainability in modern HPC systems.

2.1 Computational tasks

To begin discussing computational tasks or jobs, it's important to understand that in the context of HPC and computer systems in general, a task or job refers to a unit of work assigned to a worker for execution. In simpler terms, a job is a specific program or function that the system's hardware and software must execute, often as part of a larger workload. Jobs may vary significantly in their size and complexity, from simple calculations to vast simulations spanning numerous processors. The system's architecture,

available resources, and the type of job influence how effectively a job is executed.

A job can be characterized by two distinct sets of features: those that are **inherent to the job itself** and those that are **inherent to the job's interaction with the system**. This distinction is critical for understanding how computational tasks behave in different environments, particularly in HPC systems.

Inherent Job Characteristics: These features are intrinsic to the job's logic and algorithmic structure and remain constant regardless of the hardware on which the job is executed.

- **Time complexity** reflects the computational complexity of the job's algorithm. It measures how the runtime scales with the size of the input, independent of the system on which it is executed. For example, an algorithm with $O(n^2)$ time complexity will retain this property regardless of the hardware Cormen et al. [2009].

- **Branching behavior** refers to the control flow of the job, including how decisions are made through if-else statements or loops. This behavior is dictated by the program's logic and design, remaining fixed across different systems. Branching behavior may introduce irregularity into the workload, as varying execution paths can lead to fluctuations in computational demand and performance. Although the speed of branch execution may vary based on system architecture, the control flow remains an inherent aspect of the job's structure Aho et al. [2006].

System-Dependent Job Characteristics: In contrast, some features of a job depend on the system on which it runs and can vary based on the hardware and software environment.

- **Instructions** refer to the specific operations executed by the processor, such as arithmetic calculations, data movements, and logic operations. While the job's high-level operations are consistent, the exact instruction set varies with the hardware architecture (e.g., x86 for CPUs, CUDA for GPUs) Hennessy and Patterson [2017]. Therefore, instructions are system-dependent as they adapt to the hardware on which the job is executed.

- **Runtime** is the total time it takes to complete a job. While influenced by the inherent complexity of the task, it also depends on system-specific factors such as processor speed, memory bandwidth, and I/O performance. For instance, a job may complete faster on a high-performance system with more cores and faster memory compared to a less capable system Herault and Robert [2015].

- **Power consumption** refers to the energy consumed while executing a job, which varies depending on the system architecture, processor type (e.g., CPU vs GPU), and power management policies.

Running a job on energy-efficient hardware, for example, will typically consume less power than on less efficient systems Rountree et al. [2012].

- **Resource utilization** describes how a job uses system resources such as CPU cycles, memory, and I/O bandwidth. This utilization can vary significantly based on the architecture and current system load, but also partially on the instructions of the workload. For instance, a job may fully leverage the resources of a multi-core processor in one system but may encounter constraints due to limited memory bandwidth or disk I/O on another Kirk and mei W. Hwu [2016].

The inherent characteristics of a job, such as its time complexity and branching behavior, are fundamental to how the job is defined and structured. These properties contribute to determining whether a job exhibits regular or irregular behavior. Conversely, system-dependent features like instructions, runtime, power consumption, and resource utilization are influenced by the interaction between the job and the system environment. Recognizing these distinctions is critical to create a Machine Learning model approach for categorizing jobs as regular or irregular based on their behavior and interaction with the system.

2.1.1 Regular Tasks

Regular computational tasks are distinguished by uniform workload distribution and predictable resource utilization across processing units. Unlike irregular tasks, which are characterized by variability and unpredictability, regular tasks maintain consistency in their computational requirements and data access patterns. This consistency is advantageous in parallel and distributed systems, as it allows for even workload balancing and optimal resource management Hennessy and Patterson [2017].

Regular tasks typically involve operations where the workload is predictable and can be divided evenly across cores or processors, such as matrix multiplication or simple loop-based calculations with uniform data Dongarra et al. [2005]. These tasks avoid the complexities of dynamic partitioning and data sparsity, making them well-suited for environments where stability and efficiency in resource utilization are priorities. Because of their regular structure, these tasks enable straightforward scheduling and load balancing, minimizing idle times and maximizing system throughput Borkar [2007].

2.1.2 Execution Time and Power Consumption in Regular Tasks

The predictability of regular tasks translates into consistent execution times and power consumption. With uniform workload distributions, regular tasks are less likely to experience delays caused by processing imbalances, which are common in irregular tasks Kumar et al. [1994]. This consistency makes regular

tasks ideal for systems with strict performance requirements, such as real-time applications and embedded systems, where tasks must complete within precise time frames to meet system specifications Kopetz [2011].

In terms of power consumption, the even resource utilization characteristic of regular tasks leads to stable power demands and predictable thermal profiles. This stability is beneficial in systems where energy efficiency is a priority, as power requirements can be accurately estimated and managed without the need for adaptive power management techniques. Consequently, regular tasks contribute to an efficient and reliable operational model, particularly in power-sensitive applications such as battery-operated devices and IoT systems Brooks et al. [2000].

2.1.3 Irregular Tasks in Computational Systems

Irregular computational tasks are characterized by non-uniform or unpredictable workload distributions, often leading to variability in resource utilization across cores or processors. Unlike regular tasks, which have consistent behavior, irregular tasks exhibit variability due to intrinsic workload characteristics like data sparsity, dynamic workload partitioning, or heterogeneous computational requirements.

This irregularity creates significant challenges in resource management, especially in parallel and distributed systems. Tasks involving sparse matrices, unstructured grids, or dynamic data structures such as graphs often present irregular access patterns, which can be challenging to balance across processing units. These imbalances can result in underutilization of some processors while others are overloaded, reducing overall system efficiency and complicating workload distribution Karypis and Kumar [1998].

In high-performance computing (HPC) and large-scale data processing, understanding the nature of irregular tasks is crucial for designing systems that can handle non-uniform workloads effectively. Maintaining a balanced workload across processing units is key to optimizing system performance, even when tasks inherently differ in complexity and resource requirements Williams et al. [2009].

2.1.4 Impact of Irregular Tasks on Execution Time and Power Consumption

The unpredictable workload distribution of irregular tasks significantly impacts both execution time and power consumption. Due to the uneven allocation of work across processing units, the system often experiences delays as some cores complete their work faster, while others are left handling larger or more complex workloads. This imbalance leads to increased execution times, as the overall task completion is limited by the slowest or most heavily loaded processing units. The disparity in workload distribution can cause processors to wait on each other, further extending the total execution time and potentially leading

to bottlenecks in system throughput Hsu and Feng [2005].

For instance, in systems processing large graphs or sparse matrices, certain operations may take disproportionately longer to complete due to the irregular structure of the data. This unpredictability in execution time makes it challenging to optimize task scheduling, as traditional load balancing techniques often fall short in handling the dynamic load variations inherent in irregular tasks.

In terms of power consumption, the uneven workload distribution can lead to fluctuations in power demands and create uneven thermal profiles across the system, which can result in thermal hotspots that degrade both system performance and reliability. Adaptive power management techniques, such as dynamic voltage and frequency scaling DVFS and adaptive scheduling algorithms, become essential to handle these fluctuations. These techniques allow the system to adjust power consumption dynamically based on current workload demands, reducing energy waste and managing thermal output more effectively Hsu and Feng [2005].

Overall, the inherent variability in execution time and power consumption associated with irregular tasks requires sophisticated scheduling, load balancing, and power-aware computing techniques. These methods aim to optimize performance, control power usage, and mitigate the unpredictable execution times that make irregular tasks challenging in diverse computing environments Williams et al. [2009].

2.2 Heterogeneous Systems

Task behavior depends heavily on the system on which the job is executed. Although some features of a job, such as the number of instructions, remain unchanged, many others, like execution time and power consumption, depend significantly on the devices and their architecture.

The architecture of computational systems can be broadly categorized into two types: **homogeneous** and **heterogeneous**.

Homogeneous computing systems consist of identical processing units (PUs), which can be organized into single-socket or multi-socket configurations, with each PU designed as a single-core or multi-core processor. In contrast, heterogeneous servers are systems that use more than one kind of processor or core. They include accelerators like GPUs in addition to traditional PUs on the same bus, allowing for shared memory. This diversity enables heterogeneous systems to leverage the unique strengths of each processor type to optimize performance Gao and Zhang [2016], which makes them well suited to handle more complex workloads. This work focuses on the characterization of job behavior in heterogeneous systems.

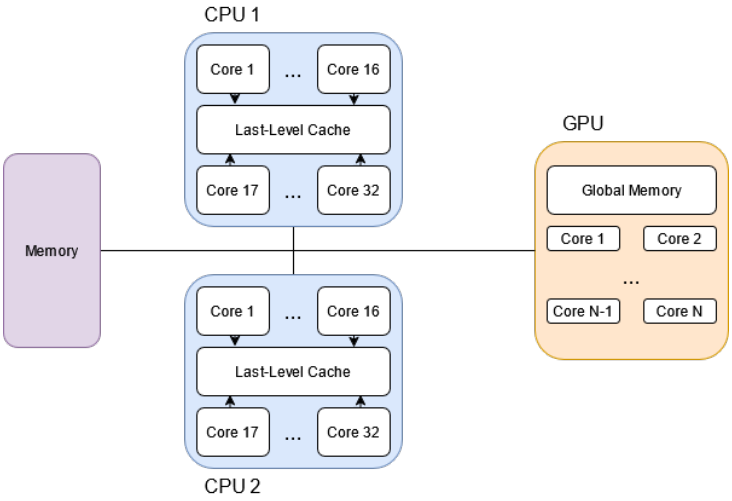


Figure 2: Heterogeneous system architecture.

Figure 2 provides a simplified illustration of a heterogeneous system, which typically includes both Central Processing Units (CPUs), Graphics Processing Units (GPUs), and occasionally other accelerators like Field-Programmable Gate Arrays (FPGAs) or Tensor Processing Units (TPUs), connected to either shared or separate memory units.

This diversity in architecture allows for greater flexibility in handling a wide range of computational tasks. Heterogeneous systems are designed to leverage the strengths of different types of processors, where CPUs handle tasks and complex control logic, while GPUs are optimized for parallel processing of large volumes of data simultaneously.

Heterogeneous computing systems are essential due to the evolving landscape of computational demands. Modern applications, ranging from artificial intelligence and Machine Learning to scientific simulations, exhibit diverse workloads that can be optimally handled by specialized processing units like GPUs and other accelerators. These systems offer energy efficiency, cost-effectiveness, and enhanced performance by strategically combining various processors based on their strengths. Moreover, the scalability and adaptability of heterogeneous architectures cater to the growing complexity of computing tasks, enabling advancements in fields such as deep learning, data analytics, and parallel processing. As technology continues to progress, heterogeneous systems provide a versatile solution to address the specific requirements of emerging applications and optimize overall computational efficiency Abdallah et al. [2017].

To take full advantage of a heterogeneous environment, efficient utilization of diverse processing units is essential. Intelligent workload distribution is required to ensure tasks are assigned to the most suitable processors based on their specific strengths. A given task may run on either a CPU or GPU, but performance and energy consumption will differ depending on the processing unit chosen. This requires

sophisticated scheduling algorithms that can dynamically adapt to the characteristics of each processor, balancing the computational load and minimizing idle times. Additionally, it is necessary to design applications with parallelism in mind to fully exploit the capabilities of the heterogeneous architecture. When these considerations are taken into account, heterogeneous systems can significantly enhance energy efficiency, providing a balanced and sustainable approach to high-performance computing Rodriguez et al. [2021].

In homogeneous systems, all processing units have similar architectures and capabilities. Therefore, scheduling tasks is generally simpler due to the uniformity in processing units. Load balancing can also be more straightforward, as tasks can be distributed evenly among the homogeneous processors. Task scheduling algorithms benefit from this uniformity, as they can be designed with the assumption of similar processing speeds and memory access times across all units.

2.3 The Scheduling Problem

Task scheduling is a critical area of study in computer science, particularly in operating systems, real-time systems, and distributed computing environments. It involves the allocation of computational resources (such as CPU cores) to various tasks to optimize performance metrics like response time, throughput, energy consumption, or fairness. Task scheduling determines the order and timing with which tasks are executed on which computing resources.

A plethora of scheduling algorithms is employed to ascertain the sequencing of job execution, each serving distinct purposes and selected based on specific system requirements Uddin et al. [2017] Putera and Siahaan [2016] and the sought-after equilibrium between fairness, responsiveness, and efficiency. The scheduling problem is central to optimizing the performance of computational systems, balancing factors such as CPU utilization, response time, and power consumption. Accurate task scheduling is closely linked to predicting task behavior, as understanding how tasks perform can help make better decisions on how they are scheduled.

Regular jobs can be scheduled using algorithms like First-Come-First-Served (FCFS) Silberschatz et al. [2018], where processes are executed in the order of arrival in the ready queue, or Round Robin (RR) Tanenbaum [2007], assigning each process a fixed time slice for CPU usage in a circular order. These are examples of **static scheduling** approaches where the order and timing of task execution are predictable and predefined. Deterministic scheduling strategies are well-suited for systems that require guaranteed response times, such as real-time systems, where predictability is essential for system correctness Kopetz

[2011].

For irregular jobs, where duration or resource requirements can vary, dynamic scheduling algorithms such as Shortest Job Next (SJN) or Shortest Job First (SJF) Samal et al. [2022] prioritize processes based on the shortest expected burst time. Additionally, Priority Scheduling assigns priorities to processes, allowing the one with the highest priority to be selected for execution. Multilevel Queue Scheduling categorizes processes into multiple priority queues, each managed by its own distinct scheduling algorithm. However, all of these algorithms depend on having prior knowledge of job characteristics to effectively schedule them. This reliance necessitates mechanisms to obtain job characteristics before scheduling attempts can be made.

Nevertheless, challenges persist in accurately predicting job execution times, managing dynamic workloads, and mitigating resource contention. Furthermore, the inclusion of considerations for real-time constraints, fairness, and energy efficiency elevates the complexity of the job scheduling process Silberschatz et al. [2018]. Deterministic scheduling is ideal for predictable environments but may be inefficient for dynamic workloads where task characteristics change frequently.

2.3.1 Static vs. Dynamic Task Scheduling

In contrast to static scheduling, **dynamic scheduling** involves approaches where the scheduling decisions are not fixed and may vary depending on the state of the system, external events, or probabilistic choices. This type of scheduling is useful in environments where workloads are unpredictable or where system performance depends on adaptability to varying conditions, such as cloud computing platforms and multi-agent systems Dean and Ghemawat [2004]. Dynamic scheduling can lead to better average-case performance by dynamically adapting to changing conditions, reducing waiting times, and improving resource utilization. However, its unpredictability makes it less suitable for applications that require strict timing guarantees, such as real-time systems Buttazzo [2005].

Algorithms like Randomized Algorithms, Stochastic Scheduling, and Opportunistic Load Balancing are widely used in dynamic scheduling. Additionally, the Heterogeneous Earliest Finish Time (HEFT) algorithm is a popular choice in heterogeneous computing environments. HEFT assigns tasks to processors based on estimated completion times, achieving efficient scheduling by prioritizing tasks and mapping them to resources that minimize their finish time Topcuoglu et al. [2002]. Another example is the Min-Min algorithm, which assigns tasks by selecting the task with the minimum completion time on any available resource and mapping it to the corresponding processor, thereby improving load balancing and reducing idle times Maheswaran et al. [1999].

The choice between static and dynamic scheduling strategies depends on the specific requirements of the system. Static scheduling provides predictability and guarantees that are crucial for real-time and safety-critical systems, while dynamic scheduling offers flexibility and better average-case performance for dynamic and unpredictable environments.

2.3.2 Energy Considerations in Task Scheduling

With the increasing complexity and scale of HPC systems, task scheduling has become a critical area of research. Energy-efficient scheduling relies on accurate predictions of task behavior to minimize energy consumption and maximize resource utilization. Sadasivan et al. Shankar and Reuther [2022] emphasized the need for integrating energy considerations into system design, arguing that as computational demands grow, energy becomes a vital design variable. This perspective is crucial because understanding task behavior and execution time directly impacts energy-efficient scheduling.

For non-deterministic jobs, Ashikahmed Bhuiyan et al. Bhuiyan et al. [2020] proposed a method for energy optimization in non-preemptive mixed-criticality scheduling using probabilistic information. Their approach integrates probabilistic energy prediction with DVFS, aiming to minimize energy consumption while ensuring task schedulability. The integration of probabilistic models in scheduling illustrates the importance of accurate task behavior predictions in achieving effective scheduling outcomes.

2.3.3 Scheduling with Machine Learning Approaches

Reinforcement Learning (RL) Shyalika et al. [2020] and Neural Networks Melnik and Nasonov [2019] have shown substantial impact in tackling issues related to environmental uncertainty, self-learning, computational efficiency, and adaptability in dynamic environments. Reinforcement learning, in particular, has proven to be a powerful tool for optimizing task scheduling by adapting based on both historical and real-time data. For instance, Xiao et al. Xiao et al. [2009] developed a Q-learning-based algorithm for real-time dynamic task allocation across multiple flows. Their approach demonstrated a improvement of task throughput while decreasing average execution time per task over traditional static scheduling methods, underscoring reinforcement learning's effectiveness in complex scheduling scenarios.

Cheng et al. Cheng et al. [2023] proposed an enhanced deep Q-learning algorithm Mnih et al. [2013] for cloud task scheduling, specifically designed to tackle challenges such as unstable response times and extensive computations. Their approach optimizes the objective function by defining the state space, action space, and reward function, significantly improving scheduling efficiency. The proposed algorithm was validated through simulations conducted with Pycloudsim, showing superior performance over traditional

algorithms such as random, earliest, and Round Robin (RR) in terms of instruction-to-response time ratio and CPU utilization.

Both reinforcement learning and neural networks, through their capacity for learning optimal decision-making strategies based on interactions with the environment, have proven successful in real-time scheduling applications. They offer promising avenues for both burst time and energy consumption prediction, enhancing the adaptability and efficiency of scheduling processes in complex and changing scenarios.

Overall, analyzing jobs before scheduling enables a more informed selection of existing algorithms and potential optimizations of the scheduling process, enhancing overall efficiency. Understanding task behavior is integral to developing effective scheduling strategies, and accurate predictions enable more informed scheduling decisions.

2.4 Significance of Runtime and Power Consumption Prediction in Scheduling

Accurate predictions of runtime and power consumption are essential for efficient task scheduling in modern computing systems. As computational workloads grow in complexity and scale across larger, more heterogeneous architectures, precise estimates of task execution times and energy usage become critical to achieving optimal system performance and energy efficiency. These predictions enable schedulers to make informed decisions that balance resource utilization, workload distribution, and system sustainability.

2.4.1 The Role of Runtime Prediction

Runtime prediction refers to the estimation of the time required for a task to complete. Reliable predictions allow task schedulers to allocate resources efficiently, ensuring that deadlines are met and system performance is maximized. Additionally, accurate runtime estimates improve load balancing and resource allocation by reducing idle times.

For example, runtime prediction algorithms that leverage statistical models or Machine Learning techniques significantly enhance scheduling efficiency. Machine Learning models can be trained on historical data to predict task runtimes under varying workloads and system conditions, resulting in more adaptive and accurate scheduling Shankar and Reuther [2022]. These models have the ability to handle the dynamic nature of modern computing systems, improving the overall reliability of scheduling decisions.

2.4.2 The Role of Power Consumption Prediction

Power consumption prediction is equally important, as it directly impacts the operational costs and environmental sustainability of computing systems. In environments like HPC clusters and data centers, managing energy consumption is a critical challenge due to rising power demands and thermal constraints. Predicting power consumption enables systems to optimize energy usage without compromising performance, which is vital for both cost savings and reducing carbon footprints Shankar and Reuther [2022].

Energy-aware scheduling strategies, which incorporate power consumption predictions, optimize the trade-off between energy efficiency and system performance. Techniques like DVFS allow systems to adjust power consumption based on predicted workloads, dynamically scaling down power usage during low-demand periods to conserve energy Bhuiyan et al. [2020]. Such techniques help reduce the overall energy footprint of computing systems while maintaining high computational performance.

2.4.3 Advances in Runtime and Power Consumption Prediction

Recent advancements in Machine Learning (ML) and artificial intelligence (AI) have significantly improved the accuracy of runtime and power consumption predictions. Techniques like reinforcement learning (RL) have shown the potential to dynamically adapt scheduling strategies based on real-time feedback and historical performance data. RL algorithms can continuously learn optimal scheduling policies that balance system performance and energy consumption by interacting with the environment Shyalika et al. [2020].

In the field of runtime prediction, notable works include *CPU Burst-Time Estimation using Machine Learning* Samal et al. [2022], where Machine Learning models were evaluated, with `RandomForest` outperforming other models in runtime estimation accuracy. In *Estimation of Execution Time for Computing Tasks* Bielecki and Śmiałek [2023], various Machine Learning techniques were combined in a black-box model to improve runtime prediction. Additionally, *Multiple Resource Management and Burst Time Prediction using Deep Reinforcement Learning* Shambharkar et al. [2019] explored burst time predictions using job characteristics, with scheduling decisions made through a Deep Q-Learning model. These studies illustrate the effectiveness of ML and RL techniques in refining runtime prediction and resource management, enabling more responsive and efficient scheduling.

While burst time prediction has received substantial attention, power consumption prediction remains a critical but less explored area. Energy-aware scheduling strategies have contributed the most to advancements in power consumption prediction. For instance, Bhuiyan et al. Bhuiyan et al. [2020] developed

a probabilistic-based energy prediction model for use in energy-aware scheduling frameworks. Similarly, Berral et al. Berral et al. [2010] utilized Machine Learning models to predict power consumption based on historical system behavior, allowing schedulers to optimize energy use in computing systems.

2.5 Machine Learning and Artificial Intelligence

This works employed the fast emerging field of AI and Machine Learning to efficiently characterise jobs in an HPC cluster environment. The term **Artificial Intelligence (AI)** refers to the field of computer science that focuses on creating systems capable of performing tasks that typically require human intelligence. These tasks include recognizing speech, making decisions, understanding natural language, and even playing complex games.

AI encompasses a wide range of approaches and techniques, one of the most prominent being **Machine Learning (ML)**. Machine Learning is a subset of AI that enables systems to learn from data, improve performance, and make decisions with minimal human intervention. Instead of being explicitly programmed with a set of rules to perform a task, ML models are designed to identify patterns in data and make predictions or decisions based on those patterns.

$$\text{Machine Learning} = \text{Data} + \text{Algorithms} + \text{Training} \rightarrow \text{Prediction or Decision}$$

2.5.1 Types of Machine Learning

This work employs different types of Machine Learning models. Machine Learning can be broadly categorized into four types:

1. **Supervised Learning**: In supervised learning, the model is trained on labeled data, where each input is paired with the correct output. The goal is to learn a mapping from inputs to outputs so that the model can predict the output for new, unseen inputs. This approach is commonly used for tasks like classification and regression Bishop [2006].

   $$\text{Supervised Learning:} \quad \mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \quad \rightarrow \quad \text{Model} \quad f(\mathbf{x}; \theta) \quad \text{predicts} \quad \hat{y} \text{ for new } \mathbf{x}$$

2. **Unsupervised Learning**: In unsupervised learning, the model is trained on data without explicit labels. The objective is to discover the underlying structure or distribution in the data. Common tasks in this category include clustering and dimensionality reduction James et al. [2013].

   $$\text{Unsupervised Learning:} \quad \mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{N} \quad \rightarrow \quad \text{Model} \quad f(\mathbf{x}; \theta) \quad \text{finds patterns or clusters}$$

3. **Reinforcement Learning**: In reinforcement learning, an agent interacts with an environment and learns to make decisions by receiving rewards or penalties. The agent's goal is to maximize the cumulative reward over time by learning a policy that maps states of the environment to actions Sutton and Barto [2018].

$$\text{Reinforcement Learning:} \quad \text{State} \quad s_t \rightarrow \text{Action} \quad a_t \rightarrow \text{Reward} \quad r_t \rightarrow \text{Updated Policy}$$

4. **Deep Learning**: Deep learning is a subset of Machine Learning that employs neural networks with many layers (deep neural networks) to model complex patterns in large datasets. It has been particularly successful in tasks such as image and speech recognition, natural language processing, and more. Deep learning models are characterized by their use of multiple layers of non-linear transformations to learn hierarchical representations of data Goodfellow et al. [2016].

$$\text{Deep Learning:} \quad \text{Model} \quad f(\mathbf{x}; \theta) \text{ with multiple layers learns hierarchical features from data}$$

This work focuses on models capable of performing classification based on task regularity and regression for predicting task runtime and power consumption using a labeled dataset. Therefore, the most suitable Machine Learning approaches for this problem are Supervised Learning and Deep Learning.

2.5.2 Supervised Learning

The effectiveness of this subfield of Machine Learning in classification and regression stems from a combination of key factors, including:

1. **Labeled Data**: The hallmark of supervised learning is the presence of labeled data. Each training example consists of an input-output pair, where the input $\mathbf{x}$ is a vector of features, and the output y is the corresponding target label. For instance, in a house price dataset, $\mathbf{x}$ could represent features such as the size of the house, the number of bedrooms, and other relevant factors, while y would denote the price of the house James et al. [2013].

2. **Model**: A supervised learning model attempts to approximate the true function that maps inputs to outputs. This can be mathematically expressed as $f(\mathbf{x}; \theta)$, where θ represents the model parameters. Models can vary widely in complexity, ranging from simple linear regressions to more complex neural networks Bishop [2006].

3. **Objective Function**: The performance of the model is evaluated by an objective (or loss) function, which quantifies the difference between the predicted output $\hat{y}$ and the actual output y. For

regression tasks, common loss functions include Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where N is the number of examples, y_i is the true output, and $\hat{y}_i = f(\mathbf{x}_i; \theta)$ is the predicted output James et al. [2013].

4. **Training Process**: During training, the model parameters θ are adjusted to minimize the loss function. This process typically utilizes optimization algorithms such as Gradient Descent, in which the model's parameters are iteratively updated by moving in the direction opposite to the loss gradient.

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$$

Here, η is the learning rate, and $\mathcal{L}(\theta)$ represents the loss function James et al. [2013].

5. **Evaluation**: After training, the model is evaluated on a separate test set to gauge its generalization performance on unseen data. Common evaluation metrics include accuracy, precision, recall, and F1-score for classification tasks, and MSE or MAE (Mean Absolute Error) for regression tasks Bishop [2006].

Common Models in Supervised Learning

Supervised learning encompasses a range of models, each suited for different tasks and data types:

- **Linear Regression**: A simple model for predicting continuous outputs by fitting a linear relationship between input features and the target Friedman et al. [2001].
- **Gradient Boosting Models**: Such as `XGBoost`, `LightGBM`, and `CatBoost`, which sequentially build models that correct the errors of previous models, achieving state-of-the-art performance in many tasks Chen and Guestrin [2016], Ke et al. [2017], Prokhorenkova et al. [2018].
- **K-Nearest Neighbors (`KNN`)**: A distance-based model where the prediction is made based on the majority label (for classification) or the average output (for regression) of the k-closest data points Friedman et al. [2001].
- **`ExtraTrees` (Extremely Randomized Trees)**: An ensemble method similar to Random Forests but using more randomization in selecting splits, resulting in faster training at the cost of slightly higher variance Geurts et al. [2006].

- **Random Forests**: An ensemble method that builds multiple decision trees and averages their predictions, reducing overfitting and improving accuracy Breiman [2001].
- **Decision Trees**: A non-parametric model that splits the data into subsets based on feature values, creating a tree-like structure of decision rules Quinlan [1986].

Each model comes with its strengths. For instance, linear models are interpretable and efficient for small datasets, while ensemble methods like Random Forests and gradient boosting are powerful for handling complex data and large-scale problems Friedman et al. [2001].

2.5.3 Deep Learning

In addition to Supervised Learning, Deep Learning is another subfield employed to tackle complex classification and regression problems, driven by several key components, including:

1. **Neural Networks**: The core of deep learning models is the neural network, which consists of layers of interconnected neurons. These layers include an input layer, multiple hidden layers, and an output layer. Each neuron applies a nonlinear transformation to the data, enabling the network to learn intricate patterns Goodfellow et al. [2016].
2. **Activation Functions**: Activation functions introduce non-linearity into the network, allowing it to model complex relationships. Common activation functions include:
    - **ReLU (Rectified Linear Unit)**: $\text{ReLU}(x) = \max(0, x)$ Nair and Hinton [2010].
    - **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$ Hinton and Salakhutdinov [2006].
    - **Tanh**: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ Hinton and Salakhutdinov [2006].
3. **Loss Function**: In deep learning, the loss function is crucial for measuring how well the model's predictions match the true outputs. For classification, cross-entropy loss is widely used, while regression tasks often rely on Mean Squared Error Goodfellow et al. [2016].
4. **Training Process**: Training a deep learning model involves adjusting the weights of the network to minimize the loss function. This is typically achieved through backpropagation combined with optimization algorithms like Stochastic Gradient Descent (SGD) or Adam Goodfellow et al. [2016].
5. **Evaluation**: Deep learning models are evaluated on validation and test datasets using task-specific metrics such as accuracy for classification or MSE for regression. Cross-validation is often used to ensure the model generalizes well Goodfellow et al. [2016].

Popular Architectures in Deep Learning

Deep learning encompasses various neural network architectures tailored for specific tasks:

- **Feedforward Neural Networks (FNNs)**: The most basic architecture where information moves in one direction from input to output Goodfellow et al. [2016].
- **Convolutional Neural Networks (CNNs)**: Designed for image-related tasks, CNNs apply convolutional filters to capture spatial hierarchies in data Krizhevsky et al. [2012].
- **Recurrent Neural Networks (RNNs)**: Used for sequential data, such as time series or natural language, RNNs maintain information across timesteps using recurrent connections Hochreiter and Schmidhuber [1997].
- **Transformer Networks**: Currently state-of-the-art in natural language processing, transformers excel in capturing long-range dependencies in sequential data Vaswani et al. [2017].

2.6 Dataset Collection

Several datasets relevant to this work are referenced in the literature. The work of Francesco Antici et all in Antici et al. [2023], aims to fill the lack of resources for job power prediction and provide the HPC community with techniques to create a job power consumption dataset from workload manager data and node power metrics logs, offering a dasaset already created with around 230 thousand jobs. This dataset consists of data derived from the M100 dataset Borghesi et al. [2023], a holistic dataset extracted from a production supercomputer hosted at the HPC centre CINECA in Italy.

The original M100 dataset is part of an extensive collection gathered from Marconi100, a Tier-0 supercomputer hosted at CINECA. It covers the entire system, including data from over 980 computing nodes, such as core loads, temperatures, memory operations, and GPU usage, as well as system-wide metrics related to cooling systems, power supply, workload management, and job statistics. This comprehensive dataset includes hundreds of metrics from each node and various sensors across the system, all organized into a structured format to enhance accessibility. Stored as a partitioned Parquet dataset, the data is divided by year, month, plugin, and metric. Accompanying software modules facilitate data access and analysis. The data is not ready to be used and needs treatment therefore leading to the work developed in Antici et al. [2023].

Other dataset developed with intent to be used in the power consumption prediction field are those

from the HPC-ODA dataset collection Netti [2021], designed for Operational Data Analytics (ODA) in High-Performance Computing (HPC) systems, with a focus on improving reliability and energy efficiency.

The collection includes six segments, each representing different ODA tasks with varying granularities:

- **Power Consumption Prediction**: Fine-granularity dataset from a single HPC compute node, including node-level and per-core metrics. Suitable for power consumption regression tasks.
- **Fault Detection**: Medium-granularity dataset from a single node under fault injection, with node-level data and labeled faults/applications. Used for fault classification.
- **Application Classification**: Medium-granularity dataset from 16 HPC nodes running MPI applications, with node-level data and application labels. Used for application classification.
- **Infrastructure Management**: Coarse-granularity dataset with cluster-wide HPC data on warm water cooling and power consumption. Suitable for regression tasks like temperature or heat prediction.
- **Cross-architecture**: Medium-granularity dataset similar to Application Classification but collected from three compute node types with different CPU architectures. Enables cross-architecture classification and performance comparison.
- **DEEP-EST Dataset**: Medium-granularity dataset from 16 compute nodes in the DEEP-EST HPC system, collected under varying cooling conditions. Useful for CPU/GPU temperature prediction and thermal characterization.

There are also other options like the dataset from Helmy et al. [2015] used for runtime prediction and Patel et al. [2020] that even though not built with the intent of power consumption prediction it could be used in this area.

2.7 Synthetic Data Generation

Synthetic data generation addresses the shortage of available data for training Machine Learning models by creating artificial datasets that mimic the statistical properties of real data. This technique is vital in fields such as Machine Learning and data science, where high-quality datasets are essential. Synthetic data helps overcome challenges like data scarcity and privacy concerns, particularly in supervised learning tasks Tarockey et al. [2024]. Advancements like Generative Adversarial Networks (GANs) Goodfellow et al.

[2014], Variational Autoencoders (VAEs) Kingma and Welling [2013], and Kernel Density Estimation (KDE) Silverman [1986] have significantly enhanced its effectiveness, enabling the creation of highly realistic datasets.

2.7.1 Kernel Density Estimation (KDE)

KDE is a non-parametric method for estimating the probability density function of a dataset Silverman [1986]. It smoothens data distributions and generates new data points by sampling from the estimated density. The KDE formula is given by:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where $K(u)$ is typically a Gaussian kernel:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}.$$

2.7.2 Generative Adversarial Networks (GANs)

GANs consist of two competing neural networks: the generator, which creates synthetic data samples, and the discriminator, which evaluates their authenticity Goodfellow et al. [2014]. The objective is to find a Nash equilibrium through a minimax optimization problem:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))].$$

This adversarial training improves both networks until the discriminator can no longer distinguish between real and generated data.

2.7.3 Variational Autoencoders (VAEs)

VAEs combine variational inference and deep learning to generate synthetic data. They consist of an encoder that maps input data to a lower-dimensional latent space and a decoder that reconstructs the original data from this latent representation Kingma and Welling [2013]. The objective function is to maximize the evidence lower bound (ELBO):

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) \,||\, p(z)).$$

Synthetic data generation finds applications in data augmentation, privacy preservation, and testing. Benefits include increased data availability and improved model accuracy. However, challenges remain, such as ensuring data quality, managing model complexity, and avoiding bias in generated datasets.

Part II

Core of the Dissertation

Chapter 3

Assembling the Dataset

Developing a robust machine learning model requires large amounts of meaningful data. However, there is a recognized shortage of HPC system metrics data, as highlighted by Antici et al. [2023] (see section 2.6).

To address this, in addition to developing the machine learning method, this work will also propose a technique to synthesize a dataset by leveraging data from multiple available sources. This data, containing various metrics related to cluster system workloads from state-of-the-art (SOA) work, is initially collected. The relevant metrics are then selected and blended into a consolidated dataset. The data undergoes a labeling and augmentation phase to generate sufficient data points for a robust model.

A synthetic dataset is a viable alternative when limited real-world data is available. This approach not only helps to avoid bias toward a specific machine (HPC system) but also establishes a general methodology for studying HPC system behavior.

3.1 Base Datasets.

The datasets used in this thesis are state-of-the-art but were not specifically tailored for the thesis's objectives. Therefore, a comprehensive analysis and filtering process is necessary to ensure that the data achieves the required level of structure and relevance for the context of this research.

This study employs two primary datasets: the PM100 dataset Antici et al. [2023] and the HPC-ODA dataset Netti [2021]. The characteristics and relevant aspects of these datasets are detailed below.

The PM100 dataset is an aggregation of data from the M100 workload dataset Borghesi et al. [2023], which was extracted from Marconi100, a tier-0 production supercomputer hosted at CINECA, Italy. The PM100 dataset contains approximately 230,000 job records; however, only around 180,000 records are used in this study, as only "COMPLETED" jobs are considered. This dataset includes multiple data entries; for this investigation, it is important to use metrics that could be obtained before job execution, except for

the metrics intended to be predicted. The chosen variables include power consumption metrics, runtime measurements, the number of cores and GPUs requested per job, memory requested, the quality of service (QoSODA) requested, the priority level assigned, the number of tasks (parallelization level) of each job and eligible at which the job was ready to be ran.

The HPC-ODA dataset represents a collection of datasets acquired from production HPC systems, reflecting various real-world use cases in Operational Data Analytics (ODA). This research focuses on the segment related to power consumption.

The power consumption segment from the HPC-ODA dataset is a high-resolution dataset collected from a single compute node in the CooLMUC-3 HPC system at the Leibniz Supercomputing Centre (LRZ). This dataset contains both node-level data and per-CPU core metrics. It provides measurements over time, including metrics such as the number of instructions executed, the number of branch instructions, cache misses, branch misses, and power consumption. For this study, only the number of instructions executed, the number of branch instructions, and power consumption will be utilized.

3.1.1 Data Distribution

Histograms of the different parameters are shown below.

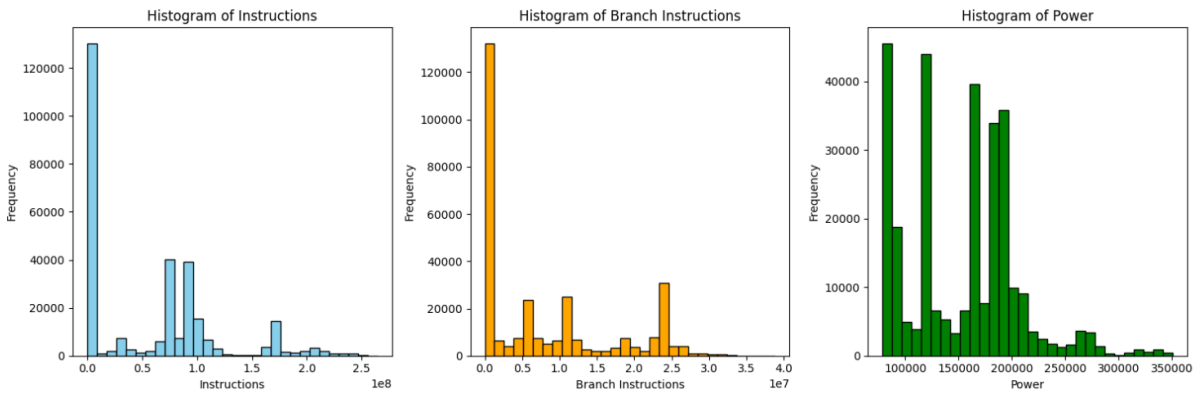


Figure 3: Distributions of the different variables in the HPC-ODA dataset.

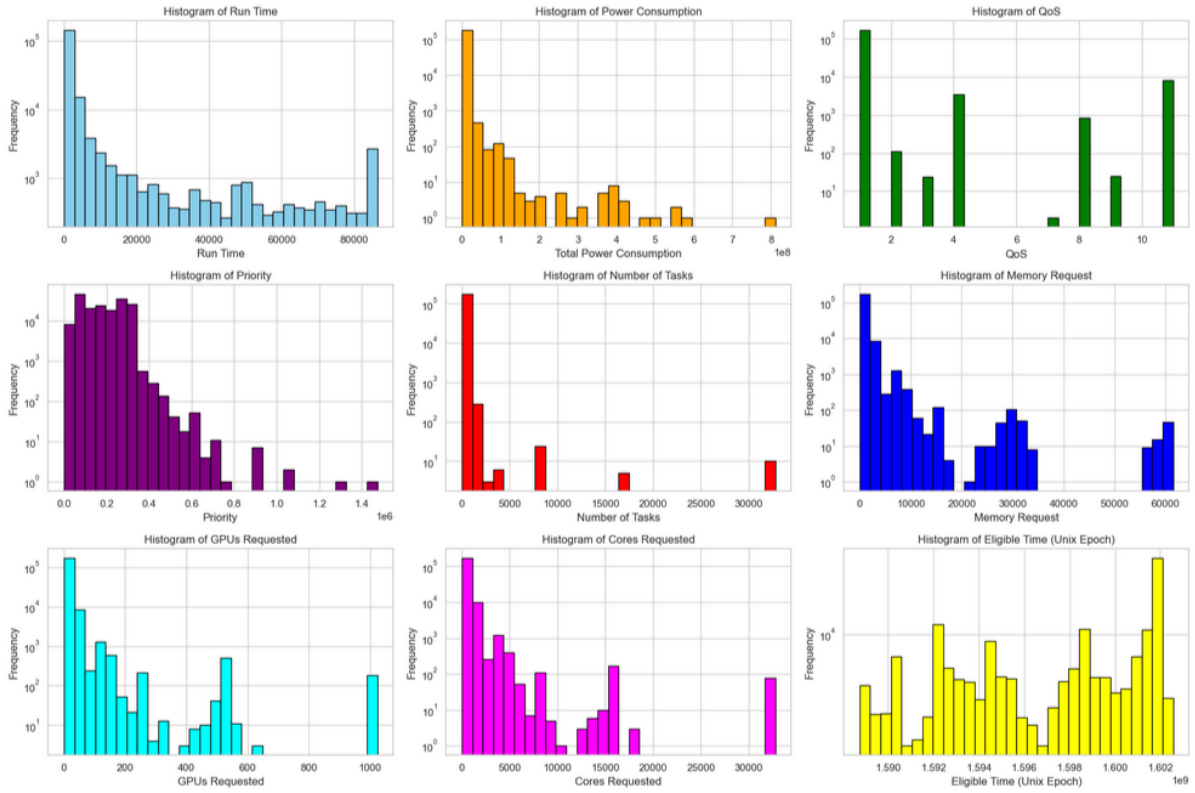


Figure 4: Distributions of the different variables in the PM100 dataset.

3.1.2 The Power Consumption metric.

Within an HPC cluster system, assigning a specific value of power consumption to an individual job is challenging because power consumption is measured at the hardware level. Instead, power consumption is typically assessed at the node level, taking into account the power usage of CPUs , GPUs , and memory. This metric is sampled at regular intervals over time. In this thesis, the power consumption metric used is the sum of power consumption at the node level considering that for each job in the PM100 dataset there are multiple values of power consumption sampled across its execution.

3.2 Data Analysis

In this section, data relations are analyzed within the dataset using different methods: Pearson correlation, Spearman correlation, and Mutual Information (MI) methods. These metrics were computed to quantify the relationships between variables under different assumptions and data characteristics.

Pearson Correlation The Pearson correlation coefficient r measures the linear relationship between two continuous variables X and Y. It is defined as the covariance of the two variables normalized by the

product of their standard deviations. The formula for the Pearson correlation coefficient is given by:

$$r_{XY} = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y} \tag{3.1}$$

where $\text{Cov}(X, Y)$ is the covariance between X and Y, and σ_X and σ_Y are the standard deviations of X and Y, respectively. The value of r_{XY} ranges from -1 to 1, where:

$$r_{XY} = \begin{cases} 1, & \text{if there is a perfect positive linear relationship,} \\ 0, & \text{if there is no linear relationship,} \\ -1, & \text{if there is a perfect negative linear relationship.} \end{cases}$$

This measure assumes that the variables are normally distributed and only captures linear dependencies Pearson [1896].

Spearman Correlation The Spearman correlation ρ is a non-parametric measure of the strength and direction of a consistently increasing or decreasing relationship between two variables. It assesses how well the relationship between the variables can be described by a monotonic function. The Spearman correlation is computed as the Pearson correlation between the rank variables of X and Y:

$$\rho_{XY} = \frac{\text{Cov}(\text{rank}(X), \text{rank}(Y))}{\sigma_{\text{rank}(X)} \sigma_{\text{rank}(Y)}} \tag{3.2}$$

Alternatively, for a sample of size n, Spearman correlation can be expressed using the difference in ranks:

$$\rho_{XY} = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)} \tag{3.3}$$

where d_i is the difference between the ranks of the corresponding values of X and Y. Spearman correlation does not assume normality and is robust to outliers, making it suitable for both linear and non-linear relationships Spearman [1904].

Mutual Information Mutual Information (`MI`) quantifies the amount of information obtained about one variable by observing another. Unlike correlation, `MI` is non-parametric and can capture both linear and non-linear dependencies. For two random variables X and Y, the mutual information is defined as:

$$I(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \left(\frac{p(x, y)}{p(x) p(y)} \right) \tag{3.4}$$

where $p(x, y)$ is the joint probability distribution of X and Y, and $p(x)$ and $p(y)$ are the marginal probability distributions of X and Y, respectively. The value of MI is always non-negative, with $I(X; Y) =$

0 indicating that X and Y are independent, and higher values indicating stronger dependencies Cover and Thomas [1991].

The mutual information of a variable with itself, $I(X;X)$, represents the total uncertainty contained within the variable. Since a variable is perfectly dependent on itself, the mutual information is maximized. In fact, for any random variable X, the mutual information with itself equals its entropy:

$$I(X;X) = H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{3.5}$$

where $H(X)$ is the entropy of X, which quantifies the uncertainty or the amount of information contained in X. This property implies that the mutual information of a variable with itself is a measure of the total information within that variable, and it is always non-negative. Therefore, $I(X;X)$ provides a useful baseline for understanding the extent of shared information between X and any other variable Y since $I(X;X) \geq I(X;Y)$, with equality holding only when $X = Y$.

Unlike Pearson and Spearman correlations, which are bounded between -1 and 1, `MI` has no upper bound. Therefore, higher `MI` values indicate stronger dependencies but are not restricted to specific ranges.

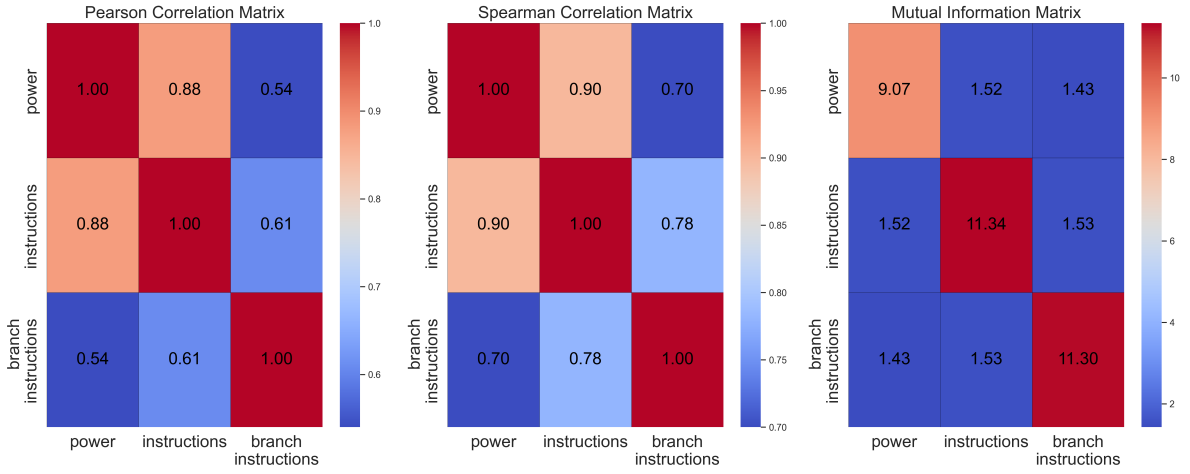


Figure 5: `HPC-ODA` dataset analysis.

The values observed in 5 and 6 will be used moving forward as reference points for the construction of the final dataset. For the purpose of keeping data integrity the final dataset should keep similar relations between the variables.

A notable phenomenon observed is the lack of correlation between runtime and all other variables (except power consumption) in the `PM100` dataset. This lack of correlation, despite high correlations among the other variables, can be explained by several factors:

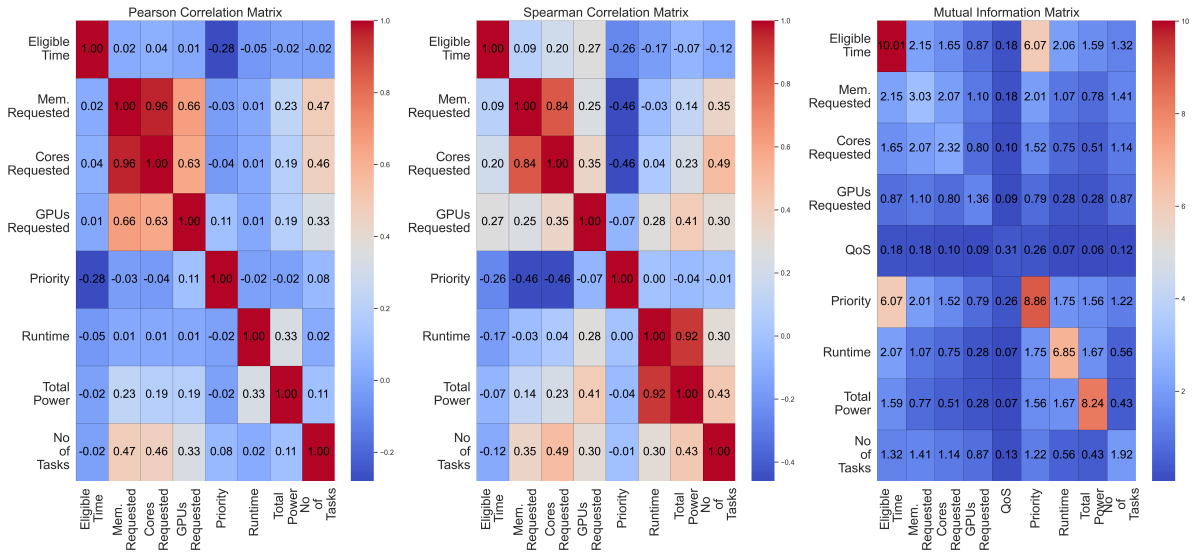


Figure 6: `PM100` dataset analysis.

- **Diverse Workloads:** `HPC` clusters run a variety of jobs (`CPU-bound`, `GPU-heavy`, memory-intensive, etc.), each with different resource requirements and runtimes, leading to weak correlation between runtime and specific resource usage.
- **Efficiency and Parallelization Differences:** Different jobs have varying levels of code optimization and parallel efficiency, resulting in different runtimes even with similar resource usage.
- **Variable Resource Utilization:** Resource usage (e.g., power, memory) can fluctuate during a job's execution, which may not directly align with total runtime.
- **Resource Request vs. Utilization:** Jobs often request more resources than they use to avoid limitations, leading to discrepancies between runtime and resource metrics.
- **High Correlation Among Other Metrics:** Power, memory, and `GPUs` are more directly linked to computational load, explaining their higher correlations with each other.

Overall, the lack of correlation between runtime and other variables arises from the diverse nature of jobs and their varying resource utilization patterns.

3.3 Synthetic Data

To correlate the datasets, considering one is structured temporally and the other job-centric, it is crucial to create a unified dataset that combines all metrics. This blending involves merging datasets to form a job-oriented dataset with metrics such as the number of instructions, number of branch instructions, memory

requested, number of GPUs requested, number of cores requested, priority level assigned, QoS (quality of service), number of tasks per job (level of parallelization), and the metrics intended for prediction: runtime and power consumption.

3.3.1 Dataset Blending

The blending method relied on analyzing the distribution of relationships between power and other variables in the HPC-ODA dataset Netti [2021]. By calculating the probability of values falling within specific intervals across all distribution, new values were generated for these relations and associated to the power values in the PM-100 dataset Antici et al. [2023]. The correlation matrices and mutual information of the blended dataset are shown in Figure 8.

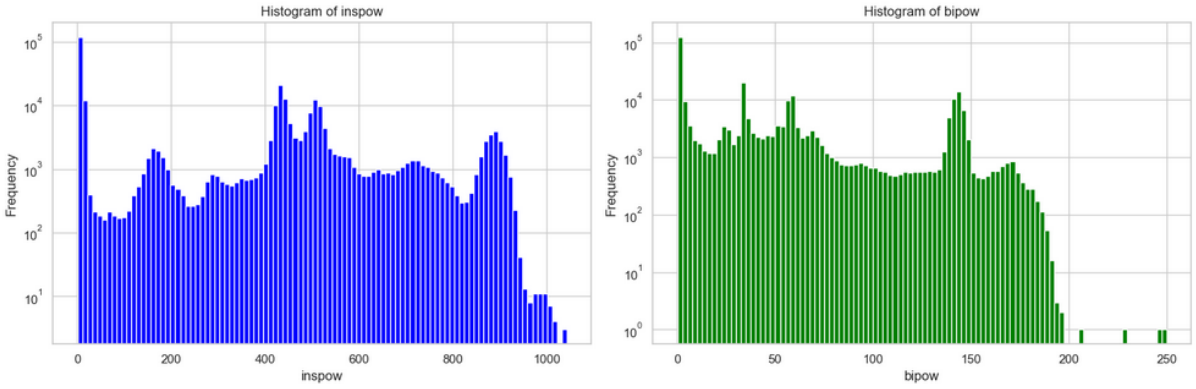


Figure 7: Distribution of the relationships between power and other variables in the HPC-ODA dataset.

It is important to analyze the correlation between the existing attributes in the PM100 dataset and the new synthetic set of instructions and branch instructions to ensure that the correlations are closely consistent with those in the original HPC-ODA dataset Netti [2021].

3.3.2 Labeling Process

Before the dataset can be used, it was necessary to classify jobs as *regular* or *irregular*. The job characteristics were compared to determine which jobs could potentially be same, allowing them to be grouped. Jobs with equal numbers of instructions, branch instructions, memory requested and run time were grouped together (within a 1% margin of error to account for cluster tool errors Weaver and Dongarra [2010]). Groups meeting the criteria were considered regular and labeled as '1', while all other groups or independent jobs were considered irregular and labeled as '0'.

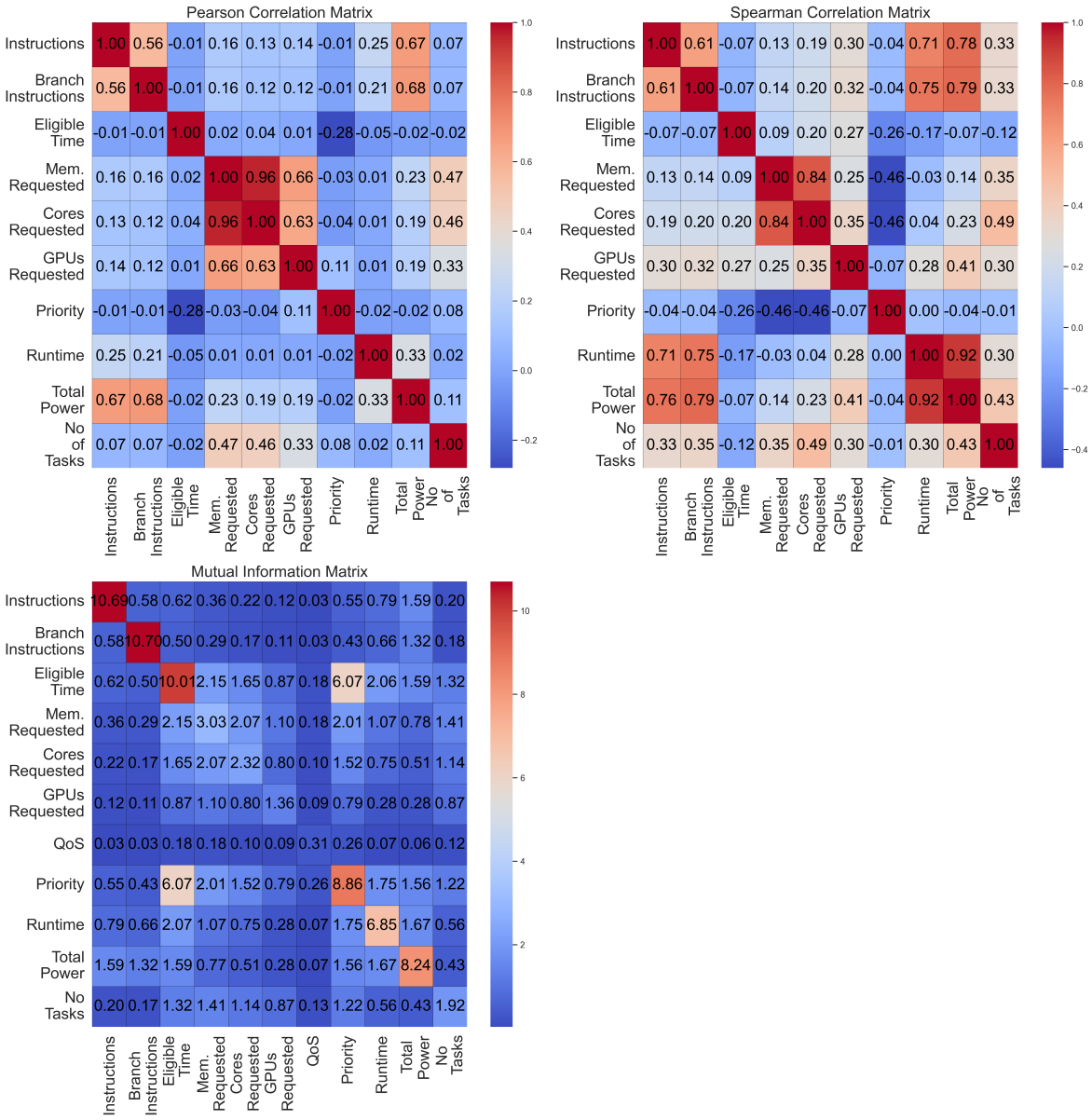


Figure 8: Correlation matrix of the blended dataset.

3.3.3 Data Augmentation

The blended dataset, had a total length of 172575 jobs where only about 7% where regular. In order to create a classifier model able to distinguish between regular and irregular tasks it is necessary to have a much higher percentage of regular jobs, luckily the fact that the main characteristic of this type of jobs is that their features are predictable knowing the history of the job it is possible to re-sample jobs within matched groups allowing them the same error interval of 1%.

Other techniques for data augmentation, such as KDE sampling or ML based methods were experimented with. However, these approaches either caused a loss of correlation or distorted the original

distributions. Some results obtained with these processes are presented in Appendices A showing the loss of data integrity.

The regular task groups were augmented leading a dataset of 276277 jobs with 43% of jobs labeled as regular '1' and the other 57% labeled as irregular '0'. The correlation matrix of the dataset, after augmentation, is shown in Figure 9. The linear correlation coefficients were affected by the augmentation of regular data but the overall relations (Spearman matrix) were not so affected and most importantly the mutual information between variables was kept. Considering that only the regular jobs were resampled, the correlations and information of irregular jobs, which will go through the regression models, remains unchanged.

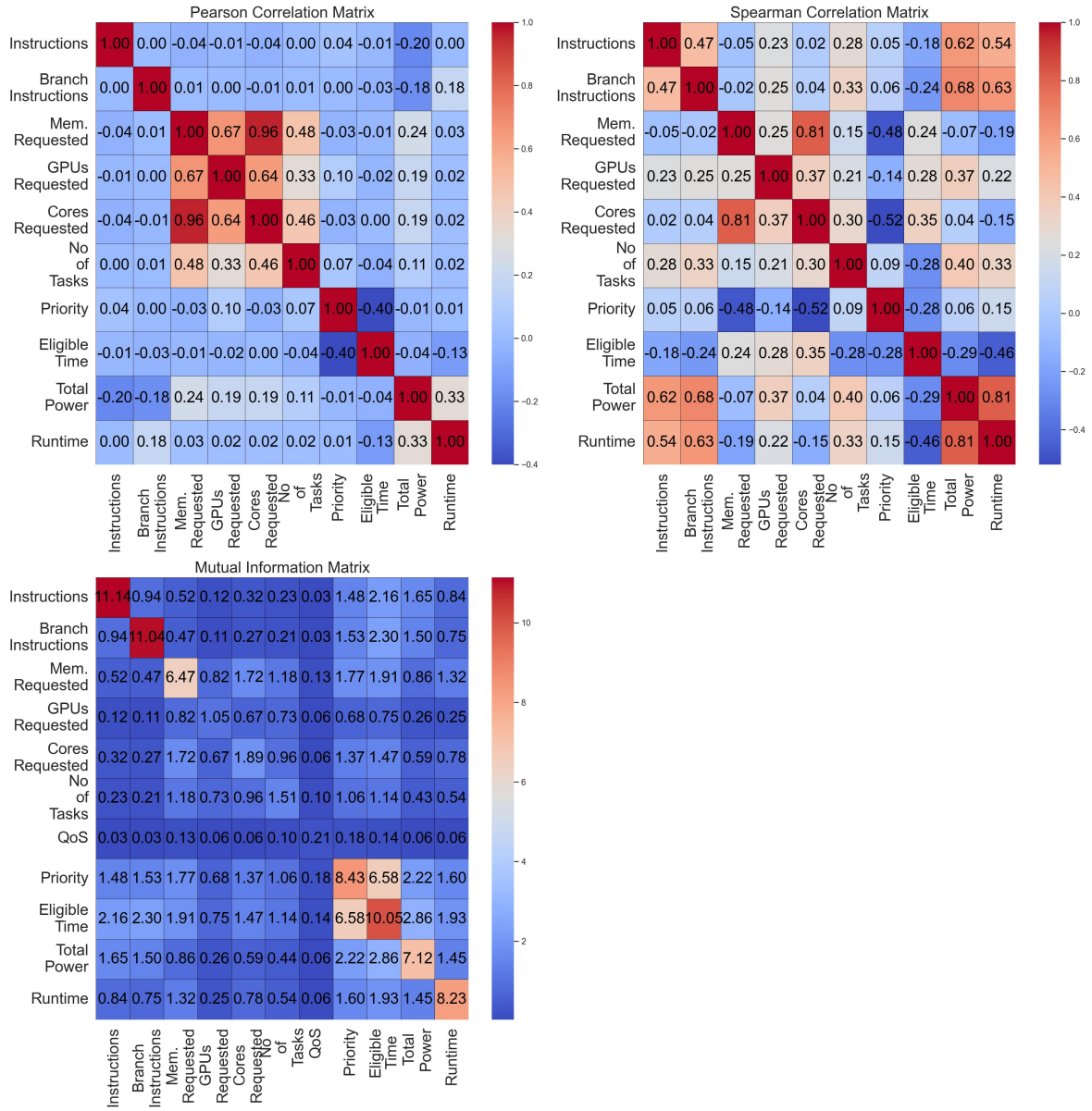


Figure 9: Correlation matrix of the extended dataset.

3.3.4 Dataset Summary

The dataset is available at Carvalho [2024] named *HPCsyntheticdata* with parquet as file format. It contains a total of 276277 jobs, with 57% labeled as *irregular* or '0' and 43% labeled as *regular* or '1'. There are 12 columns in this dataset corresponding to the following features:

Feature	**Description**	**Dataset ID**
Number of Instructions	Total count of instructions.	inst
Number of Branch Instructions	Count of branch instructions.	bi
Amount of Memory Requested	Memory requested for the job.	mem_req
Number of Cores Requested	Number of CPU cores requested for the job.	num_cores_req
Number of GPUs Requested	Number of GPUs requested for the job.	num_gpus_req
Quality of Service Requested	Level of service quality requested.	qos
Priority Level Assigned	Priority level assigned to the job.	priority
Number of Tasks	Total number of tasks within a job.	num_tasks
Eligible Time	Time at which the job was eligible to run	eligible_time_epoch
Power Consumption	Power consumption (Node level).	total_power
Runtime	Total runtime of the job, measured in seconds.	run_time
Label	Labeling as irregular '0' and regular '1'	label

Table 1: Table of job attributes within the synthetic generated dataset.

Chapter 4

Models for job classification

This chapter explores the classification of jobs into regular and irregular categories. The dataset used in this study (refer to chapter 3) is labeled as '0' for irregular tasks and '1' for regular tasks. Since the dataset is already labeled, supervised learning, a branch of machine learning specifically designed for the study of labeled data, was chosen as the field of study for the construction of the model to classify the data.

4.0.1 Performance Metrics:

In the classification process, two metrics used to evaluate the performance of a classification model were: **Accuracy** and **AUC-ROC**.

Accuracy is defined as the ratio of the number of correct predictions made by the model to the total number of predictions. Mathematically, it is simply expressed as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Accuracy is a straightforward measure that indicates how often the classifier is correct. It is useful when the classes in the dataset are balanced (i.e., each class is equally represented). However, accuracy can be misleading in cases where the dataset is imbalanced (i.e., one class is much more frequent than the other) reason why it was important in chapter 3 to approximate the number of regular and irregular jobs.

AUC-ROC stands for **Area Under the Receiver Operating Characteristic Curve**. It is a performance metric used to evaluate the quality of binary classification models. The **ROC curve** is a graphical representation that plots the *True Positive Rate (TPR)* against the *False Positive Rate (FPR)* at various threshold settings. The *True Positive Rate (TPR)*, also known as sensitivity or recall, measures the proportion of actual positives correctly identified by the model:

$$\text{TPR} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

The *False Positive Rate (FPR)* measures the proportion of actual negatives incorrectly identified as positives by the model:

$$\text{FPR} = \frac{\text{False Positives (FP)}}{\text{False Positives (FP)} + \text{True Negatives (TN)}}$$

The **AUC** (Area Under the Curve) represents the degree of separability between the two classes. It provides a single value summary of the ROC curve's performance. The AUC value ranges from 0 to 1:

- **AUC = 0.5**: The model has no discriminative ability and performs no better than random guessing.
- **AUC < 0.5**: The model performs worse than random guessing.
- **AUC = 1**: The model perfectly distinguishes between positive and negative classes.

A higher AUC indicates a better-performing model that is more capable of distinguishing between the positive and negative classes. **AUC-ROC** is particularly useful when the dataset is imbalanced because it takes into account both the sensitivity (TPR) and the specificity (1 - FPR) of the model across different thresholds.

4.1 Different classifiers performance:

Initially the data was just fed to multiple supervised learning classifiers in order to gauge their performance. The data was not scaled and the models were made to be quite simple overall. Looking in Figure 10 it is possible to see the models achieved already a very high accuracy and near perfect fitting of the model, but still further investigation and slight changes needed to be made in order to fine tune the model for its needs.

4.1.1 Scaling Data.

The models in Figure 10 were trained with data that was not scaled prior to being fed to the classifier, which is typically considered non-standard in machine learning. Scaling is an important preprocessing step in many machine learning models because it normalizes the range of the features, ensuring that each feature contributes equally to the model's learning process. In some models, especially those that rely on distance measurements or gradient-based methods (e.g., Support Vector Machines, K-Nearest Neighbors,

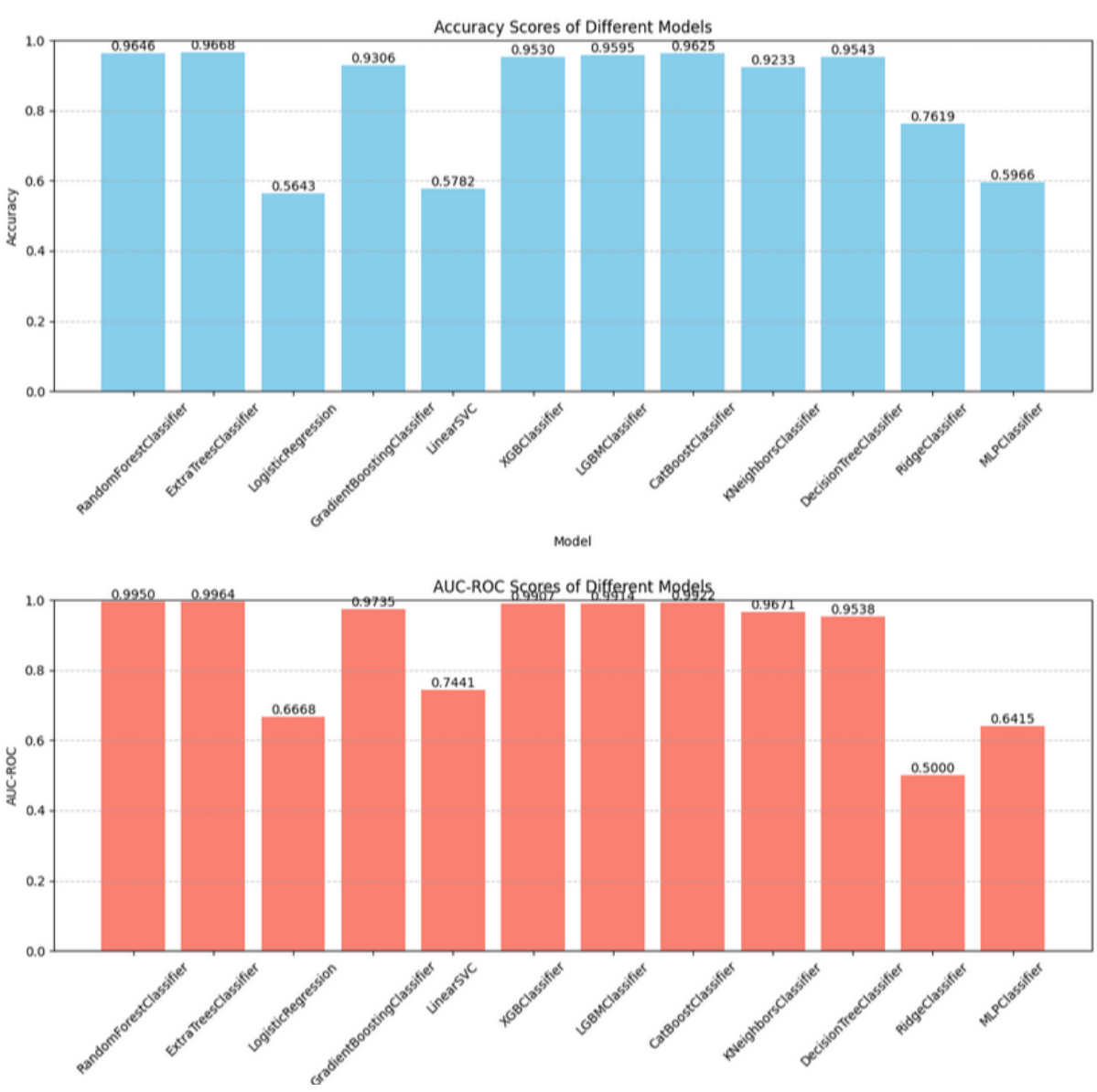


Figure 10: Supervised Learning classifier models performance in job classification.

and neural networks), scaling can significantly impact model performance by improving convergence speed and accuracy. However, for tree-based models like Random Forests, scaling may not provide the same level of benefit because these models are not sensitive to the magnitude of the feature values. Despite this, scaling is still crucial for future data because it ensures consistency between the training and new data, allowing the model to make accurate predictions. When data is scaled during training, the model learns patterns based on normalized feature ranges. If future data is not scaled in the same way, the model may receive values outside its expected range, leading to poor performance or inaccurate predictions. Consistent scaling helps maintain model stability, improves generalization, and ensures that the model responds reliably when deployed in real-world scenarios with unseen data.

Considering this, the models were once again evaluated with a Standard Scaling in order to test if tree-based models were only performing better because of their low sensitivity to magnitude of data. Figure 11 shows that the `RandomForest` and `ExtraTrees` models still outperform the others, maintaining their previous accuracy. In contrast, `CatBoost` which showed close performance to the other models did not benefit from scaling; its accuracy decreased. Notably, `ExtraTrees` slightly outperforms `RandomForest`.

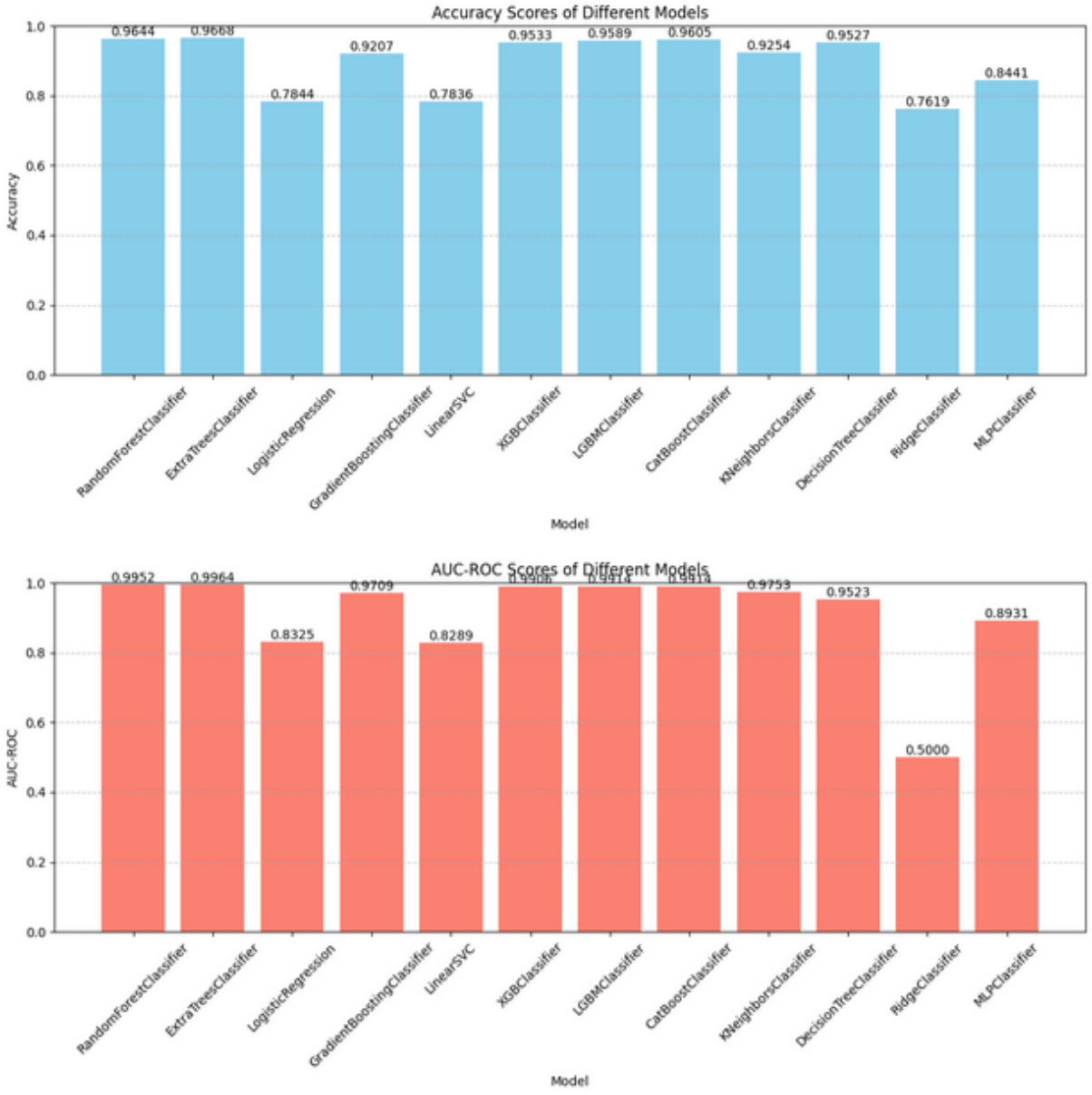


Figure 11: Supervised Learning classifier models performance in job classification (Data Scaled).

The model `ExtraTreesClassifier` (with 50 trees) showed the best overall results in the initial draft, achieving higher accuracy results and also a better **AUC-ROC** indicating a better distribution between regular and irregular jobs. Therefore it was used to evaluate the performance of the different scaling

techniques. In this case the `scalers` perform very similarly with this data as Figure 12 shows, and moving forward the model was scaled with a MaxAbsScaler() as it just slightly edged other `scalers` in accuracy.

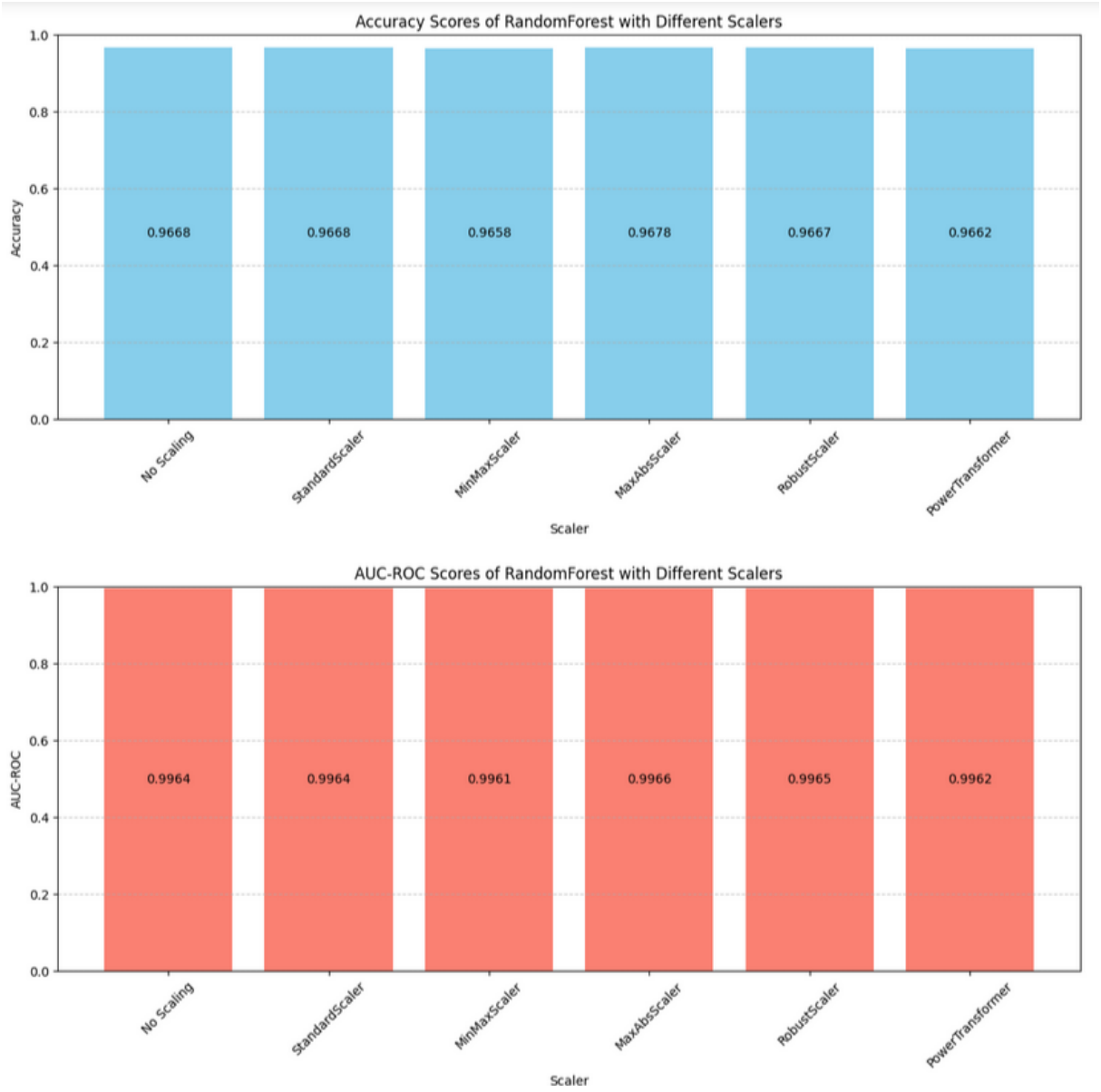


Figure 12: Different `scalers` performance in job classification with `ExtraTreesClassifier` model.

4.1.2 The `ExtraTreesClassifier` Model

The `ExtraTreesClassifier`, or Extremely Randomized Trees classifier, is an ensemble learning method that operates by constructing a large number of uncorrelated decision trees during training and using their collective predictions to classify new data points. It is a variant of the `RandomForestClassifier` but introduces additional randomization into the tree-building process to reduce variance and improve gen-

eralization.

The `ExtraTrees` algorithm differs from Random Forest in two key ways:

- **Random splits**: Instead of searching for the best possible split at each node, `ExtraTrees` randomly selects the splitting thresholds from a uniform distribution. This increased randomness helps reduce overfitting.
- **Using the entire dataset**: While Random Forest uses bootstrap aggregation (sampling with replacement), `ExtraTrees` uses the whole dataset to build each tree.

The `ExtraTreesClassifier` works by averaging the predictions of multiple trees to make a final classification decision. Mathematically, let the training dataset be represented as $D = \{(x_i, y_i)\}_{i=1}^{n}$, where x_i is a feature vector, and y_i is the corresponding label. The `ExtraTreesClassifier` constructs M trees $T_m(x)$, each trained on a subset of the data.

For a new input x, the prediction of the model is obtained by aggregating the predictions of all trees, typically through majority voting in classification tasks:

$$\hat{y} = \text{argmax}_k \sum_{m=1}^{M} \mathbb{1}(T_m(x) = k)$$

where $\mathbb{1}$ is the indicator function, and k represents the possible class labels.

Each tree T_m is built using the following steps:

1. For each node, randomly select a subset of the features from the total feature space.
2. For each selected feature, a random threshold is chosen to split the data.
3. The best split is chosen based on a splitting criterion such as Gini impurity or entropy.
4. Repeat recursively for each child node until the stopping criterion is met (e.g., a maximum tree depth or minimum samples per leaf).

This approach allows `ExtraTrees` to build more diverse trees, leading to better generalization compared to traditional decision trees or even Random Forest. The increased randomness reduces variance at the cost of a slight increase in bias, but in practice, this often results in improved performance, especially for noisy datasets.

4.1.3 Model Complexity Performance

Considering that the models were quite simple in terms of complexity, it was necessary to study how the model chosen (`ExtraTreesClassifier`) behaved with increasing complexity. In Figure 14, it is evident that the classification process does not benefit from increasing model complexity. This occurs when the model becomes overly intricate, potentially leading to overfitting, where it captures noise and specific patterns in the training data that do not generalize well to unseen data. As a result, the model performance plateaus or even declines with higher complexity, suggesting that simpler models are often more effective for this particular classification task. Moving forward, the model chosen was still the `ExtraTreesClassifier` with 50 trees.

Call of Attention: It is important to note that 50 trees (the model in Figure 10) might not seem like much, but to put it in perspective, Figure 13 shows how a single tree (1/50) can get complex:

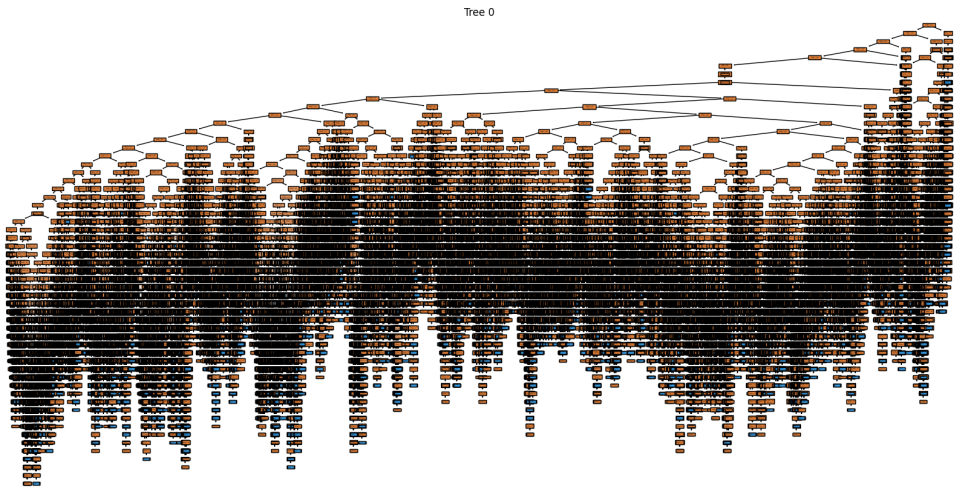


Figure 13: Tree 0 of the Classification Model

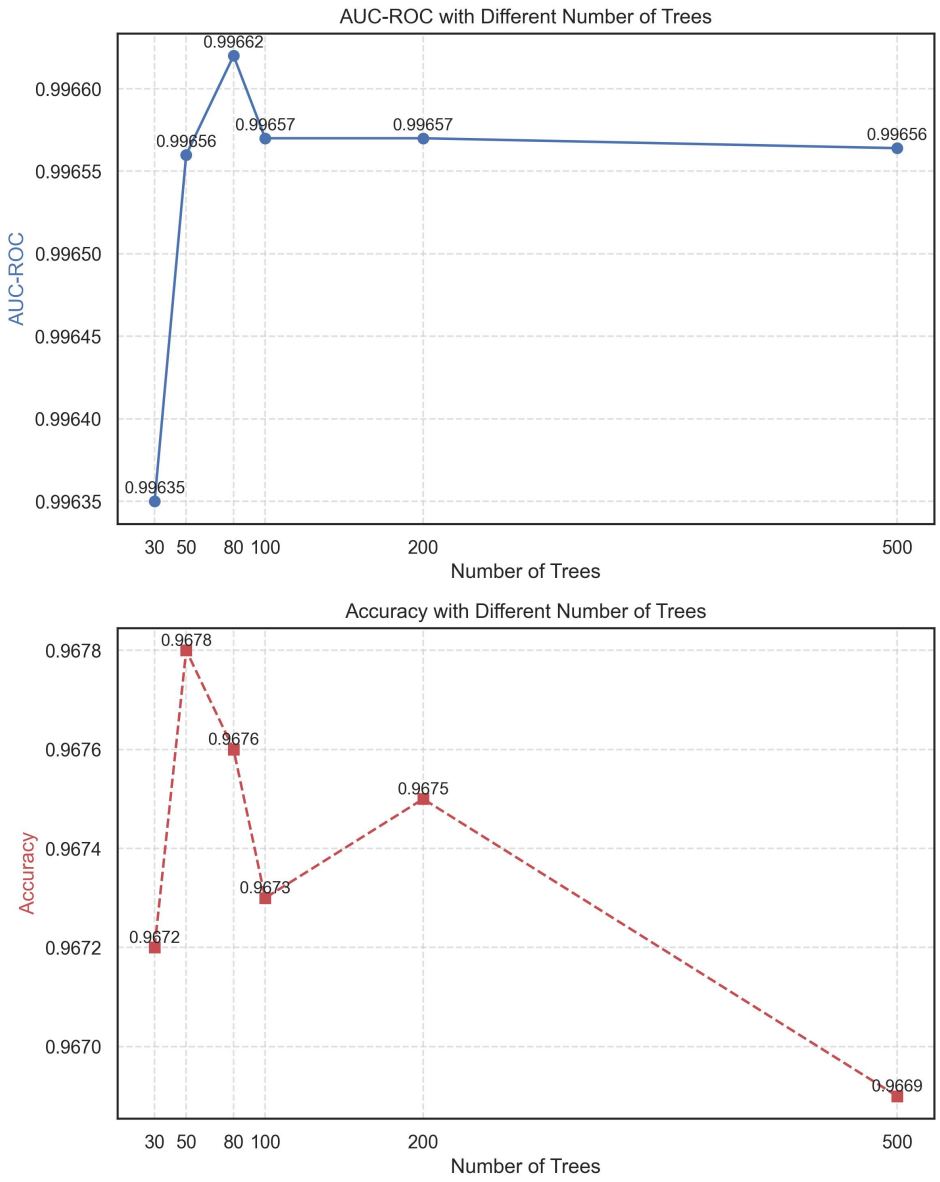


Figure 14: `ExtraTreesClassifier` model performance with increasing complexity.

4.1.4 Feature Importance analysis.

The next parameter to evaluate was feature importance. It was introduced RFE (Recursive Feature Elimination) to understand what features does the model benefit from using and what importance does each feature hold in the prediction framework. In Figure 15 it is possible to see that the best performance comes with 8 features.

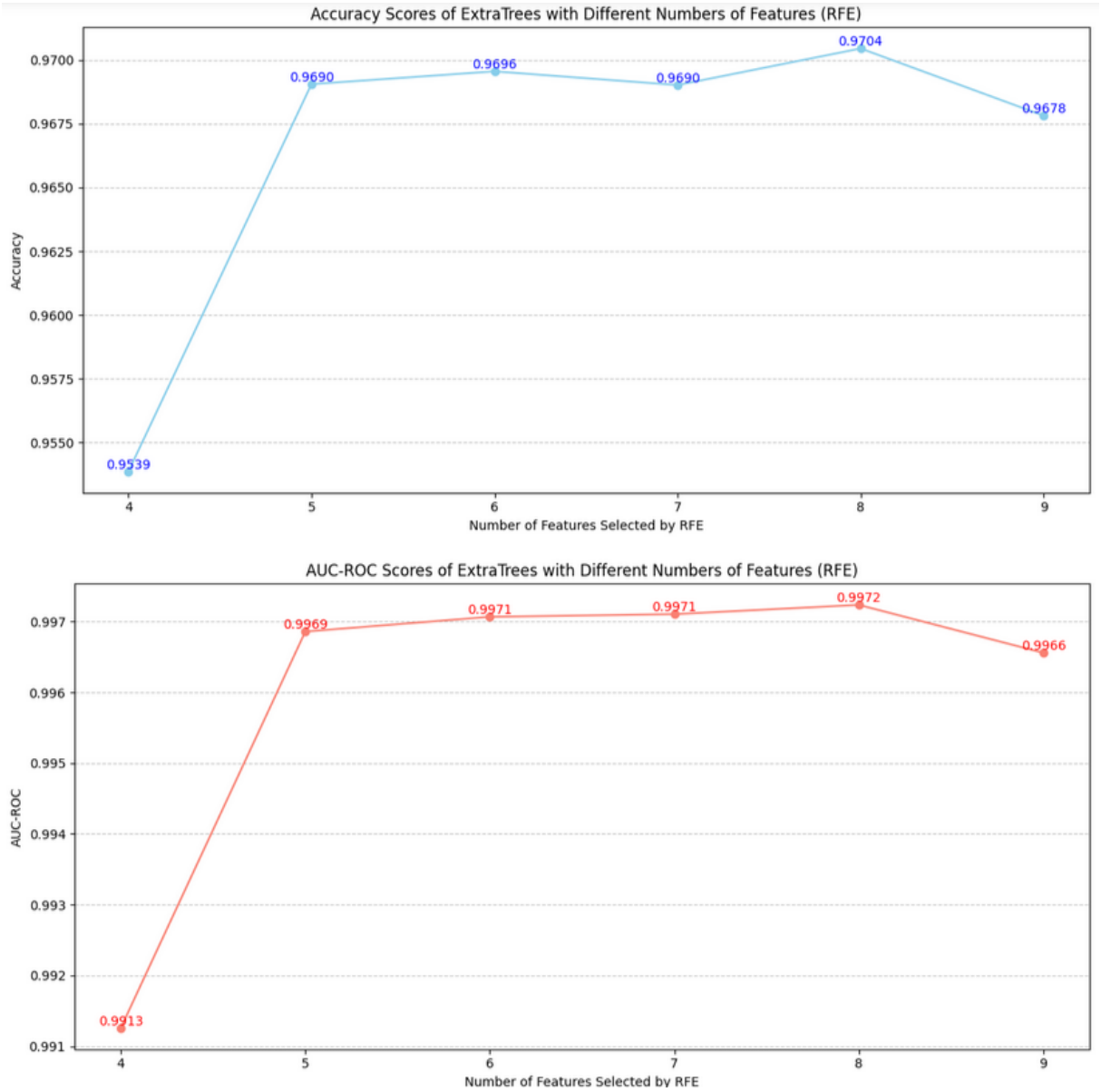


Figure 15: `ExtraTreesClassifier` model performance across different number of variables chose by RFE.

The exact metrics chosen for the classification and their respective importance are presented in Figure 16. Observing the feature importance in this figure, it is visible that the feature removed to slightly improve accuracy is the number of tasks. This feature, which represents the parallelism of the job, is prone to variation and can lead to classification errors because the number of tasks is subject to user-defined settings and does not always correlate with the job's actual behavior. On the other hand, the features with the highest importance are memory required and eligible time. This can be attributed to the deterministic nature of memory access in regular jobs and the fact that many regular jobs, which are system-related, tend to occur at consistent times of the day or follow periodic patterns.

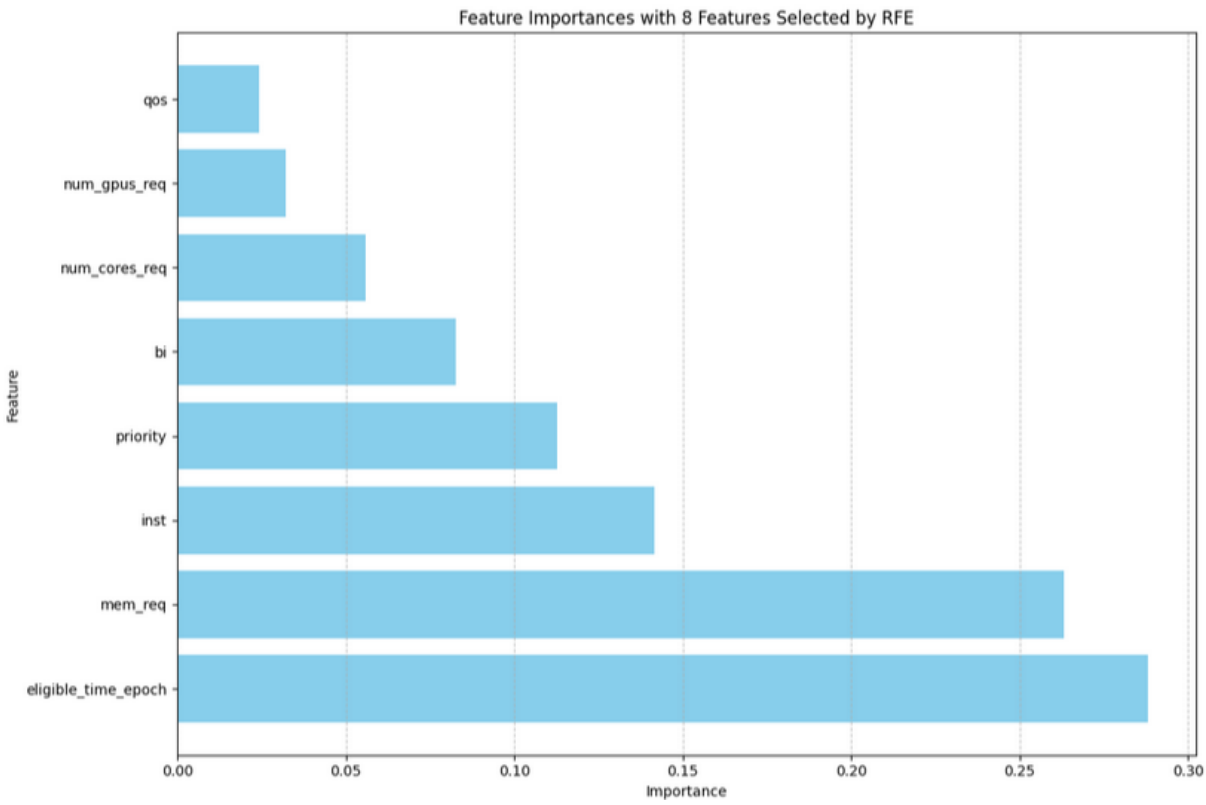


Figure 16: Feature importance in the job classification model.

Introducing Polynomial Features in the second degree and studying the interactions between variables, the model never seems to be able to achieve the same level of performance as seen in Figure 17. This likely happens because the introduction of polynomial features increases the dimensionality of the feature space, which can lead to overfitting, especially if the interactions between variables are not truly meaningful or if the data does not support complex relationships. Additionally, polynomial features can introduce multicollinearity, where features become highly correlated, further complicating the learning process and reducing the model's ability to generalize effectively. Thus, while polynomial features can capture complex interactions, they may also introduce noise and redundancy that hinder overall model performance.

4.1.5 Losing accuracy for better situational performance.

In the context of HPC scheduling, scheduling a regular job with a dynamic scheduling algorithm is less detrimental than scheduling a irregular job with a recurrent algorithm (see section 1.2). To address this issue, the model's penalty weights were adjusted so that a false positive—an irregular job incorrectly labeled as regular— was penalized 100 times more heavily than a false negative. This adjustment reflects the higher cost of misclassifying irregular tasks as regular ones. Furthermore, the prediction threshold was slightly decreased from the standard 0.5 to 0.4, in order to decrease the number of '1's predicted as '0's.

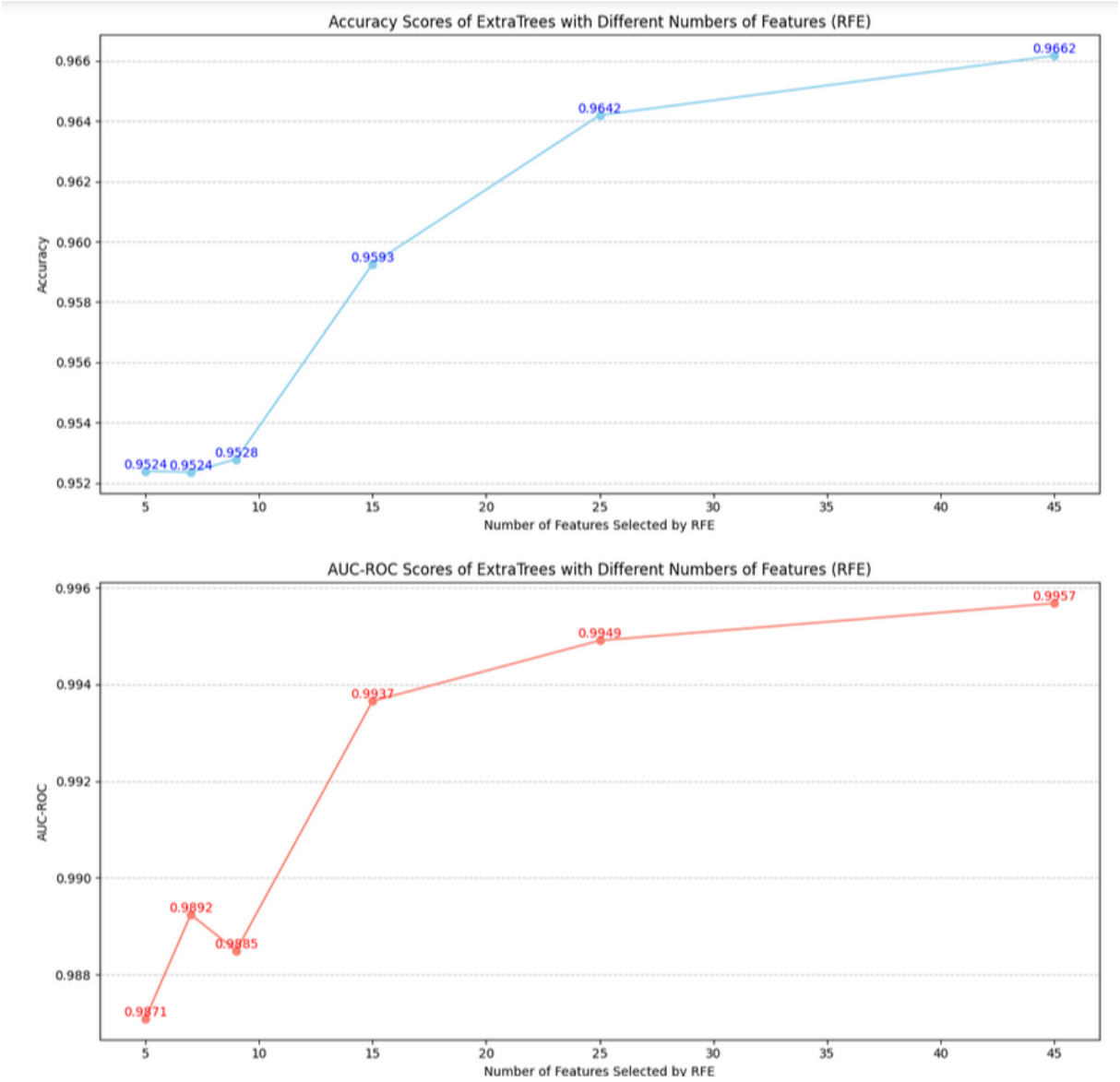


Figure 17: Feature importance in the classification model with polynomial features introduced.

As a result of these changes, the overall accuracy decreased slightly to 96.7%. However, the number of worst case classification (irregular '0's predicted as regular '1's) was reduced to about half.

4.1.6 Case Study:

Before discussing the results of the classification model it is important to analyze a specific situation. Considering the length of the dataset a deep learning model should be able to perform better than it is shown in Figure 10, where is possible to see that the `MLPClassifier` was not able to outperform the other models. However, given the prowess of deep learning models on larger datasets, it is worth further analyzing this case.

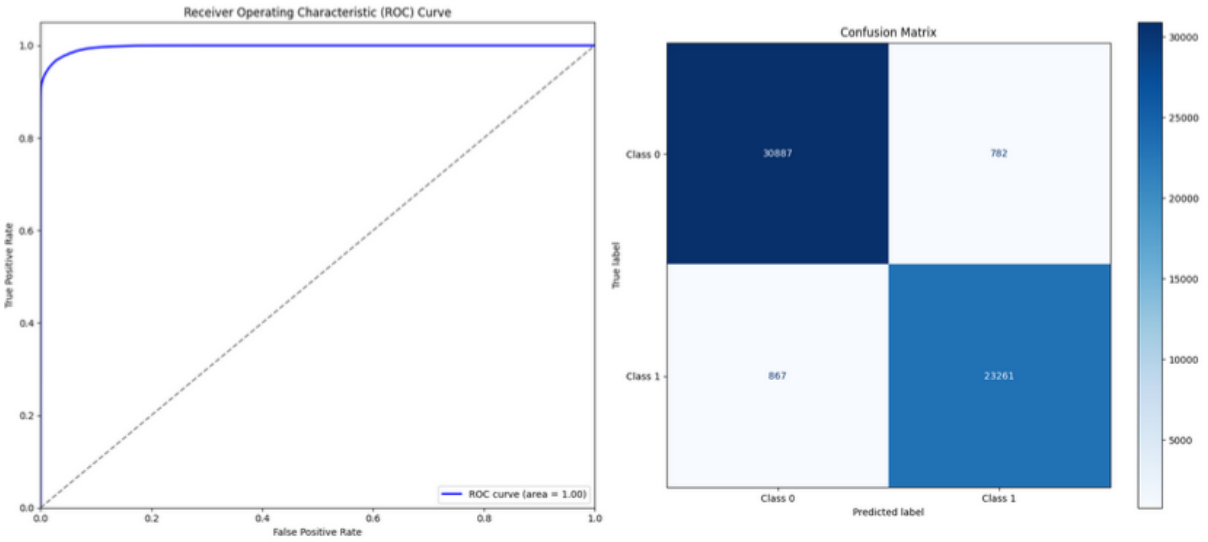


Figure 18: ROC curve and Confusion Matrix of the model with standard weights and threshold.

Among the various sub-genres of neural networks, Artificial Neural Networks (ANNs), Recurrent Neural Networks (RNNs), and Convolutional Neural Networks (CNNs) each have their specific strengths and applications. ANNs are highly versatile and can be effectively used for a wide range of tasks, including basic classification and regression problems. RNNs excel in handling sequential data due to their ability to maintain temporal dependencies and process inputs over time. They are particularly adept at tasks such as time-series prediction and natural language processing. CNNs, on the other hand, are specialized for spatial data, making them ideal for image processing tasks where they can capture spatial hierarchies and patterns. For tasks involving sequential or time-dependent data, RNNs are often the most appropriate choice due to their design, which allows them to retain and utilize information over time. For spatial data and tasks involving visual information, CNNs are more suitable LeCun et al. [2015]Hochreiter and Schmidhuber [1997]Krizhevsky et al. [2012]. Considering this the model built for the classification process was a Feedforward Neural Network

The models performance is enhanced by employing a *funnel-shaped architecture* (see Figure 20), where the number of neurons decreases progressively with each subsequent layer. This approach effectively reduces the complexity of the model while still capturing important patterns in the data.

Immediately by introducing a scaler, removing the feature that was slightly hindering the model before and introducing polynomial features which this time improve performance, likely because deep learning favors from complexity, the Neural Network with just a total of 96 neurons and 50 training steps is able to achieve a accuracy of 84.7% and a **AUC-ROC** of 90.7% which is much higher than the `MLPClassifier` in Figure 10.

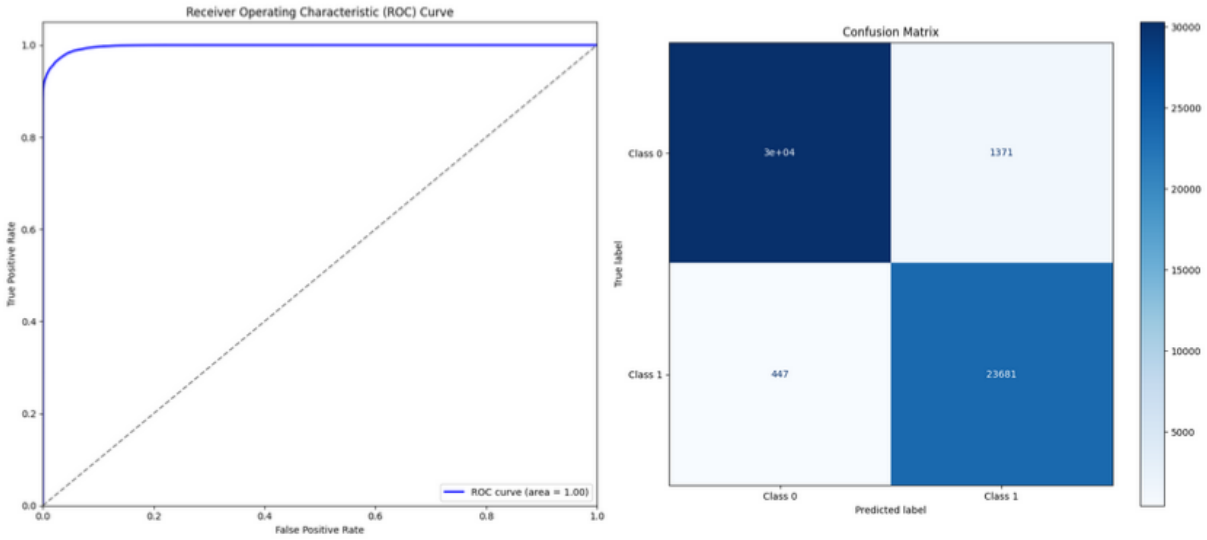


Figure 19: ROC curve and Confusion Matrix with changed weights and threshold.

Although the model demonstrated some improvement after the initial feature tuning, as shown in Figure 21, further increases in complexity and additional feature tuning did not lead to substantial gains. In Figure 22, the model features a total of 502 neurons arranged in a funnel-shaped architecture. Despite this increased complexity, extensive training with more steps and a lower batch size did not significantly enhance the classification performance. The final results achieved were an accuracy of 85.2% and an **AUC-ROC** of 90.9%. Given the substantial number of training steps, the model eventually reached a point where Early Stopping was triggered, indicating that it was no longer improving. This suggests that the added complexity and extended training did not yield further benefits.

Understanding that the model does not benefit from increased complexity and considering that Neural Networks are often referred to as *black boxes* due to the difficulty in interpreting their internal mechanisms it was determined to conclude that the deep learning methods do not fit the model in the most efficient way and not being able to out perform the previous model studied.

4.2 Discussion: Explanation of the Results

The model's performance, with an overall accuracy of 96.7%, indicates that it correctly predicted the classification of jobs in 53979 out of 55797 cases. With 3.3% error rate which is already a very small error in classification, a deeper look into the nature of the misclassifications shows that the model's results are highly favorable given the context of HPC scheduling. Out of the 1818 incorrect predictions, 1371 (75.4%) were regular jobs mistakenly classified as irregular. Meaning that of the 55797 predicted jobs only 0.8 % led to a worst case scenario prediction. This outcome is considered a satisfactory result for several

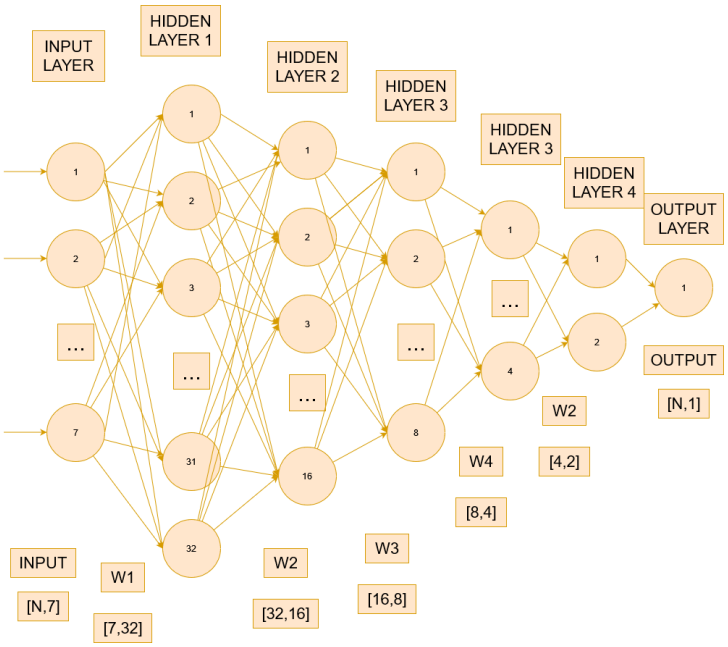


Figure 20: Neural Network Shape.

reasons:

- Prioritization of Critical Errors: In HPC scheduling, misclassifying a irregular job as regular is far more detrimental than the opposite scenario. Irregular jobs often require more dynamic handling to avoid inefficiencies or conflicts in resource allocation. By contrast, incorrectly labeling a regular job as irregular might lead to more conservative scheduling but does not pose a risk of significant resource contention or scheduling failures.

- Effective Penalization Strategy: The model was specifically designed to penalize false positives (irregular jobs labeled as regular) more heavily. This strategy ensured that the more costly type of error—treating a irregular job as regular—was minimized. The results show that this objective was achieved, as only a small fraction of errors involved this kind of misclassification.

- Acceptable Trade-off Between Accuracy and Error Distribution: Although the overall accuracy decreased slightly to 96.7%, this trade-off is acceptable because the distribution of errors is skewed towards the less harmful type (regular jobs predicted as irregular). This distribution is more aligned with practical needs, as it maintains a robust safety margin in scheduling operations.

Further more, with an **AUC-ROC** (Area Under the Receiver Operating Characteristic Curve) score of 0.9979, the model is very close to perfect classification, suggesting that it can distinguish between 'regular' and 'irregular' jobs with very high accuracy across all possible decision thresholds. This high

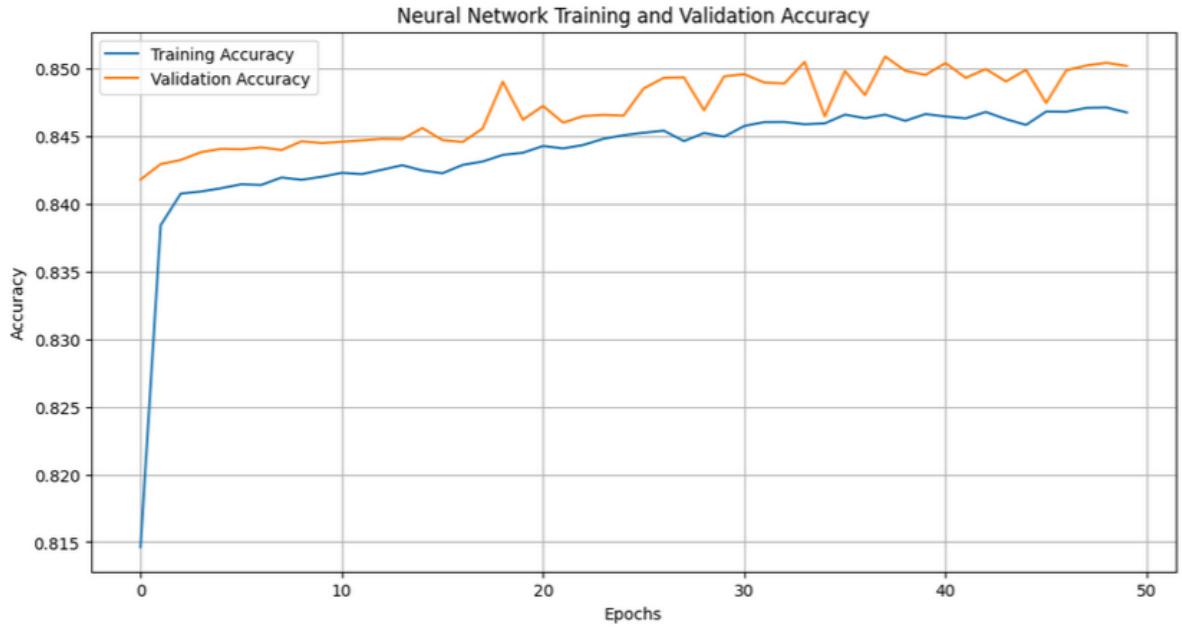


Figure 21: Simple Neural Network performance in job classification.

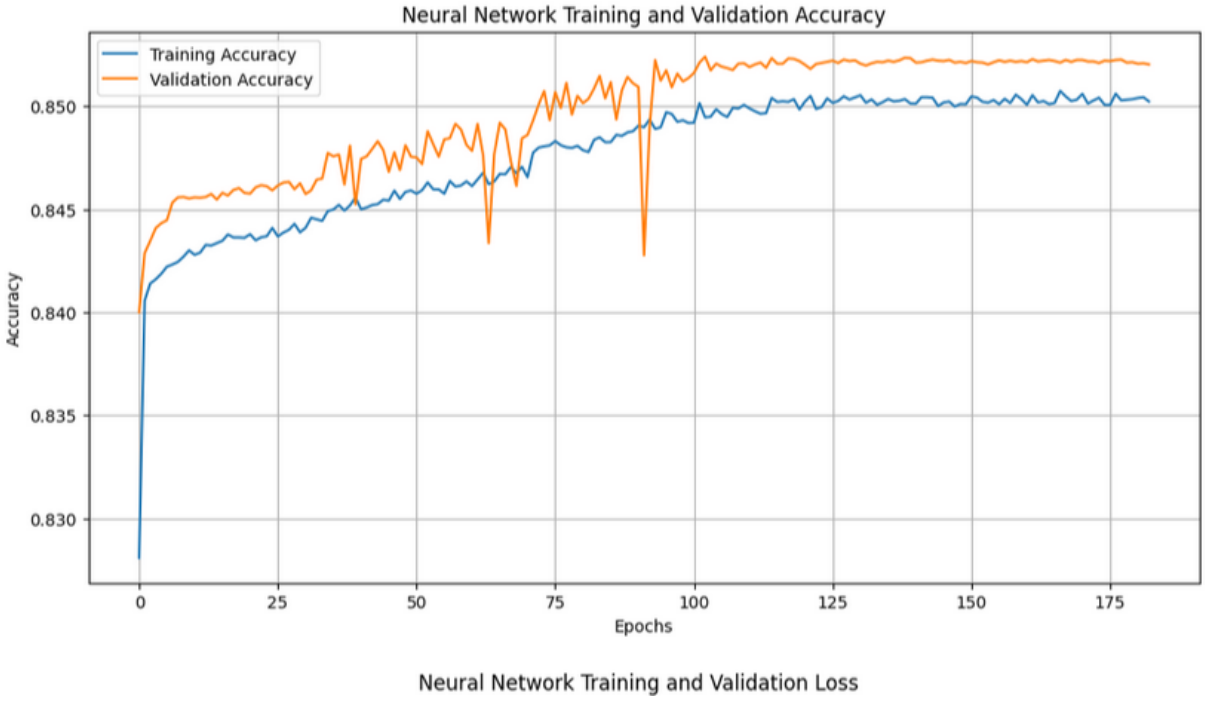


Figure 22: More complex Neural Network performance in job classification.

AUC-ROC value reflects the model's robustness, especially in scenarios where it is critical to balance between detecting irregular jobs and avoiding false positives. It also confirms that the model is highly effective in prioritizing the correct classification of jobs, which is crucial for optimizing HPC scheduling performance. Overall, an **AUC-ROC** of 0.9979 demonstrates that the model is not only accurate but also

reliable in differentiating between the two classes, providing confidence in its deployment for scheduling tasks where precision and error management are essential.

Combining the model's simplicity—which allows for faster predictions and reduced computational load—with its high accuracy rate, bias towards more favorable types of misclassification, and very high **AUC-ROC** score, the model was considered to be highly satisfactory.

Figure 23 shows the results obtained across all data splits. It is visible in this image the consistent great performance of the model achieving in the entire dataset (considering Training, Test and Validation splits) an overall accuracy of 99.34% (this accuracy is although boosted through the near perfect results obtained in model training and should be given less importance than the test and validation split results).

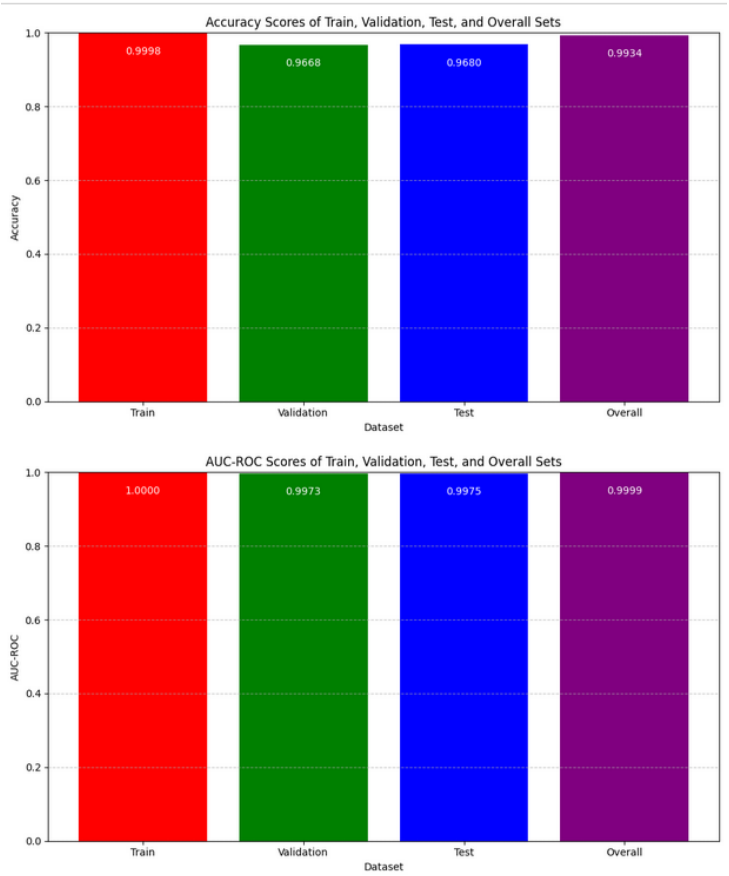


Figure 23: Classifier model performance across training, testing and validation splits and overall accuracy.

Chapter 5

Prediction Models

This chapter studies the prediction of the runtime and power consumption metric in an HPC system environment using supervised learning regression models. It focuses not only on achieving a good model but also studying and analyzing what affects this predictions.

All jobs labeled as '1' were removed from the original dataset in order to have a dataset with only jobs labeled as irregular leaving the dataset left with 158777 jobs.

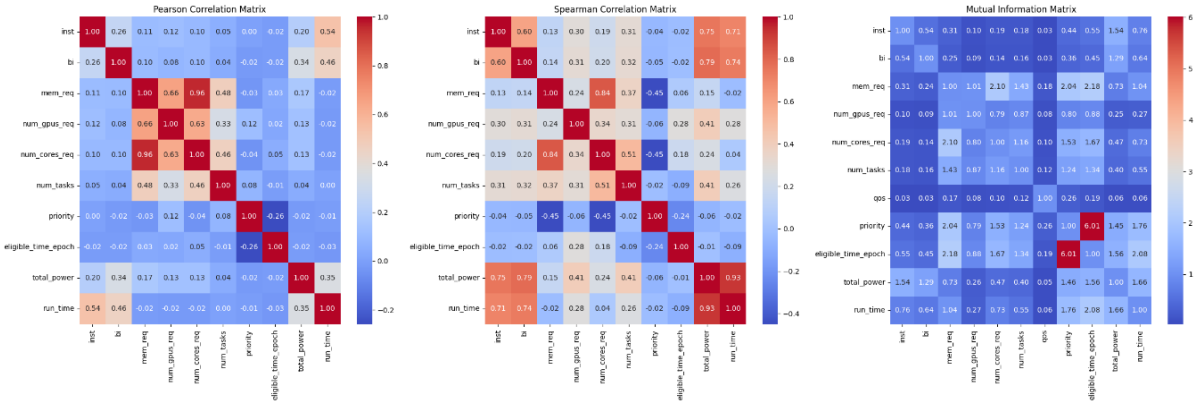


Figure 24: Data analysis for the dataset containing only irregular jobs.

5.1 Regression Performance Metrics

The metrics in analyses for performance comparison are:

1. **R-squared (R^2)**: R^2, also known as the coefficient of determination, is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variables. In regression, R^2 provides an indication of how well the data fits the model. It is calculated as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the actual values. An

R^2 value closer to 1 indicates a better fit, meaning that a greater proportion of variance is accounted for by the model. However, an R^2 value close to 0 suggests that the model does not explain much of the variance.

2. **Mean Absolute Error (MAE)**: MAE is a measure of errors between paired observations expressing the same phenomenon. In regression, it represents the average absolute difference between the actual values and the predicted values. The formula for MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i is the actual value and $\hat{y}_i$ is the predicted value. MAE provides a linear score which means that all the individual differences are weighted equally. A lower MAE indicates a better model, as it implies that the predictions are closer to the actual values.

Both R^2 and MAE were selected as they offer complementary insights into model performance. R^2 reveals how much of the data's variance is captured by the model, indicating overall fit, while MAE directly measures the average prediction error, providing a straightforward metric of accuracy. This combination allows for a robust evaluation, balancing interpretability with precision, especially in assessing models with varying complexity.

5.2 Run Time Prediction

In table 2 are present the general statistics associated to the Run time data in the new dataset. This data shows a very high standard deviation and furthermore a very large interval between the 75th percentile (75% of the values are lower than 1819 seconds) and the maximum/100th percentile (86395 seconds).

Metric	**Value**
Mean	4657.34
Standard Deviation	14049.40
Minimum	0.00
25th Percentile	16.00
50th Percentile (Median)	212.00
75th Percentile	1819.00
Maximum	86395.00

Table 2: Table of Run Time data statistics.

5.2.1 Different Regression models performance

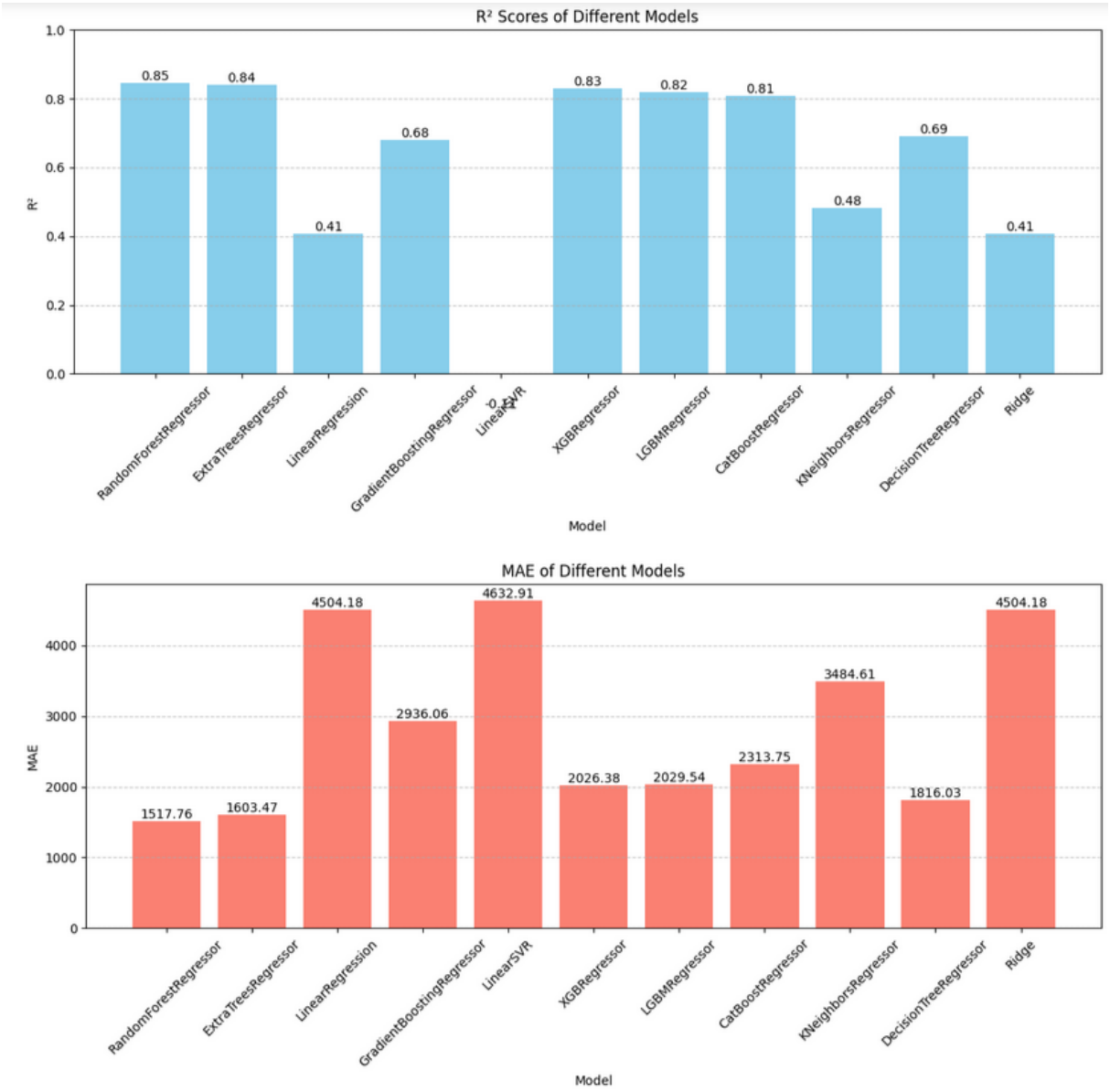


Figure 25: Supervised Learning Regression models performance in Time prediction.

Just like in the classification process, the data was fed to the regression models without any prior preparations and the models were default to gauge their performance. As illustrated in Figure 25, the `RandomForestRegressor` slightly outperforms the other models, even though there are no good results just yet, and so it was taken through a study process to see if would be possible to use it if fine tuned.

5.2.2 Random Forest Performance Analysis

Before testing what scaler perform best the data went through the process of creating polynomial features to study interactions between variables and then the selection of which variables should go to scale was done by using the factor Mutual information between the features and the target variables. In figure 26 it is visible the general improvement of the model by reducing the MAE in a considerable manner. The scalers perform similarly and just by a small margin the one chosen was the `MaxAbsScaler`.

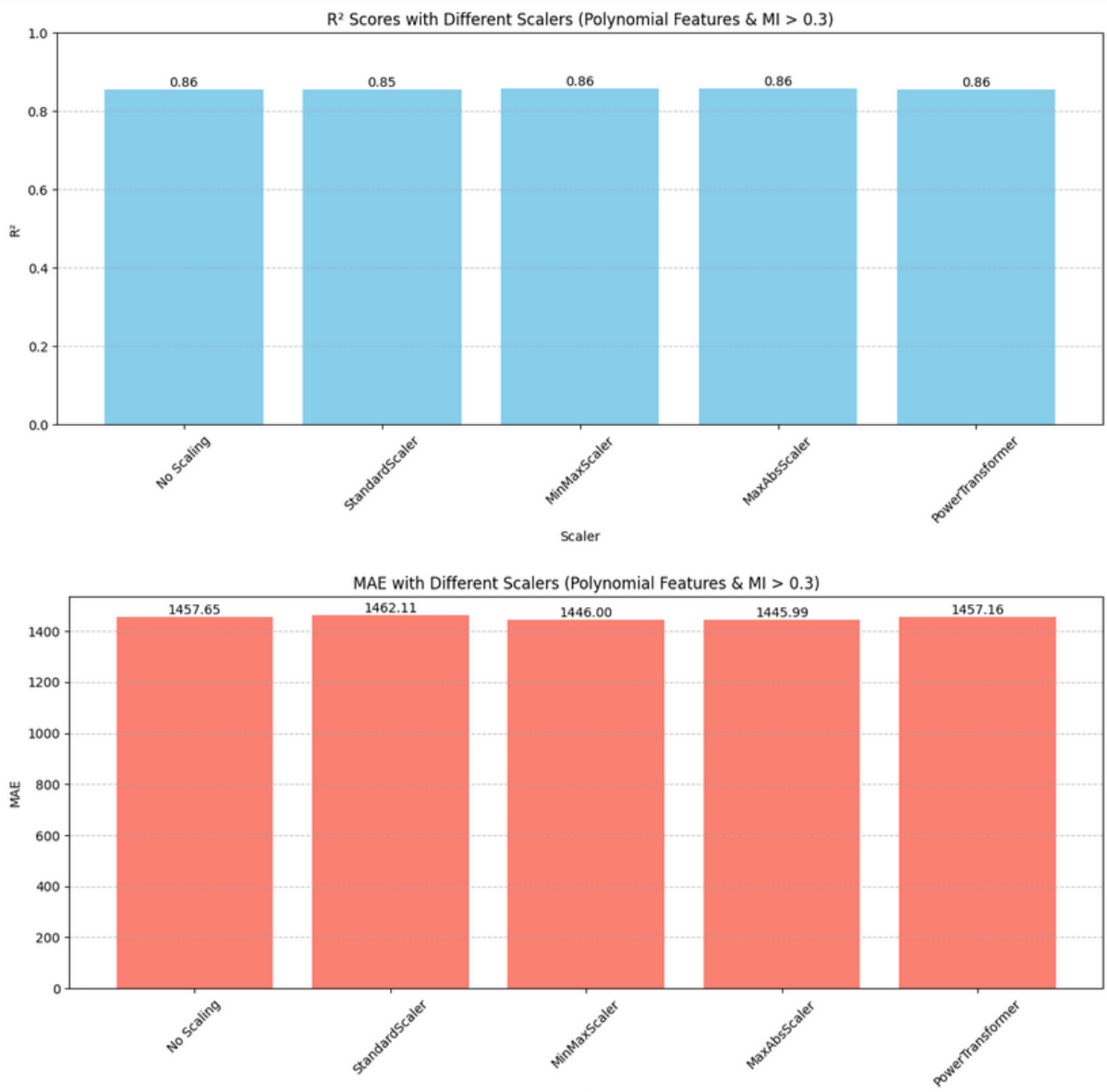


Figure 26: Different `scalers` performance in run time regression predictions with `RandomForestRegressor` model.

With regard to the response to complexity, Figure 27 demonstrates that the model does not exhibit significant improvements as the number of trees increases. Instead, the model appears to overfit the

training data, as evidenced by Table 3. This overfitting is apparent from the substantial drop in performance when comparing the training split to the validation and test splits.

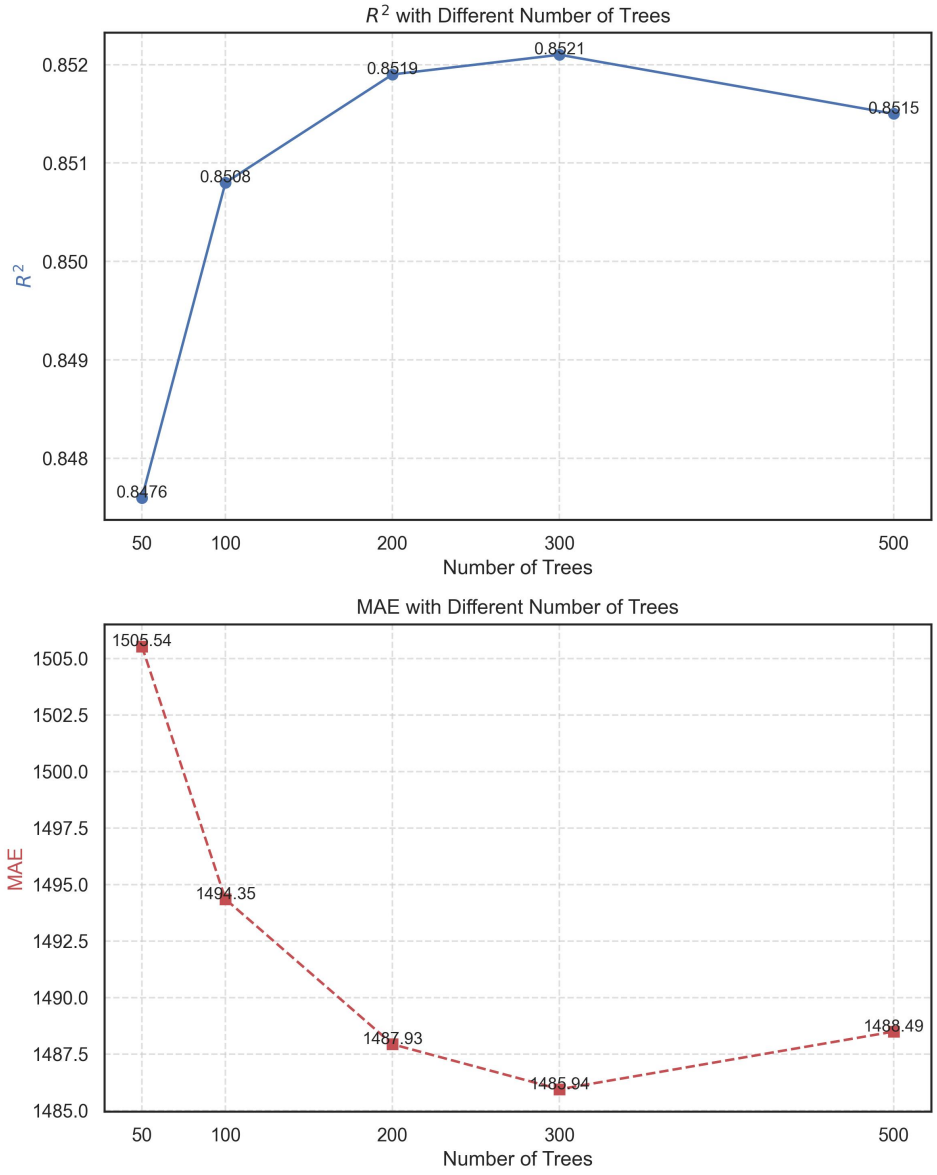


Figure 27: `RandomForestRegressor` model performance with increasing complexity.

In order to understand the error associated with the model it is important to break it down considering the range of values in Table 2 considering that a Mean Absolute Error of 1487 seconds (for the model with 200 trees) is not a good result but it needs context. At this point the Mean Absolute Error in the various intervals is very high being even higher than the biggest value in that interval and so a single model regression (in this case `RandomForestRegressor`) does not provide good results for the features at study.

n_estimators	Train R^2	Validation R^2	Test R^2	Train MAE	Validation MAE	Test MAE
50	0.978049	0.847610	0.851281	571.335699	1505.541291	1531.945708
100	0.979222	0.850782	0.852570	561.267501	1494.348206	1522.225357
200	0.979595	0.851918	0.852844	558.834781	1487.934187	1521.027210
300	0.979900	0.852088	0.853220	556.468781	1485.944170	1517.336383
500	0.979939	0.851506	0.853360	555.582966	1488.487008	1517.254675

Table 3: Performance metrics for `RandomForestRegressor` in run time prediction with different `n_estimators`

Metric	Value
MAE (Overall)	1438.27
MAE at 25th percentile	27.11
MAE at 25th-50th percentile	278.01
MAE at 50th-75th percentile	961.04
MAE at 75th-90th percentile	1525.17
MAE at 90th-95th percentile	5808.44
MAE at 95th-99th percentile	9688.74
MAE over 99th percentile	21684.50

Table 4: Mean Absolute Error (MAE) across different percentiles of run time data distribution.

5.2.3 Model Pipeline

The target variable, run time, spans a wide range, from 0 seconds to 86,395 seconds. This extensive range makes it difficult for a single model to effectively learn patterns, particularly at the extremes. To address this, a model pipeline that segments the data into smaller intervals was developed, allowing for more precise regression analysis within each range.

As illustrated in Figure 28, the data is first divided into intervals using a multi-class classification model. This model identifies key segments of data based on their importance. The first interval includes run times below the 25th percentile (0–16 seconds), representing fast-executing jobs. The second interval corresponds to the 25th–50th percentile (16–212 seconds), which still covers relatively short jobs given the overall distribution.

The third interval captures jobs with run times between the median (50th percentile) and the 75th

percentile (212–1,819 seconds). The fourth interval covers data between the 75th percentile and the mean (1,819–4,657 seconds), marking the beginning of longer jobs.

To handle the higher range of values, the next interval extends from the mean (4,657 seconds) to the 95th percentile (up to 30,905 seconds). The final interval captures the top 5% of run times, ranging from 30,905 to 86,395 seconds.

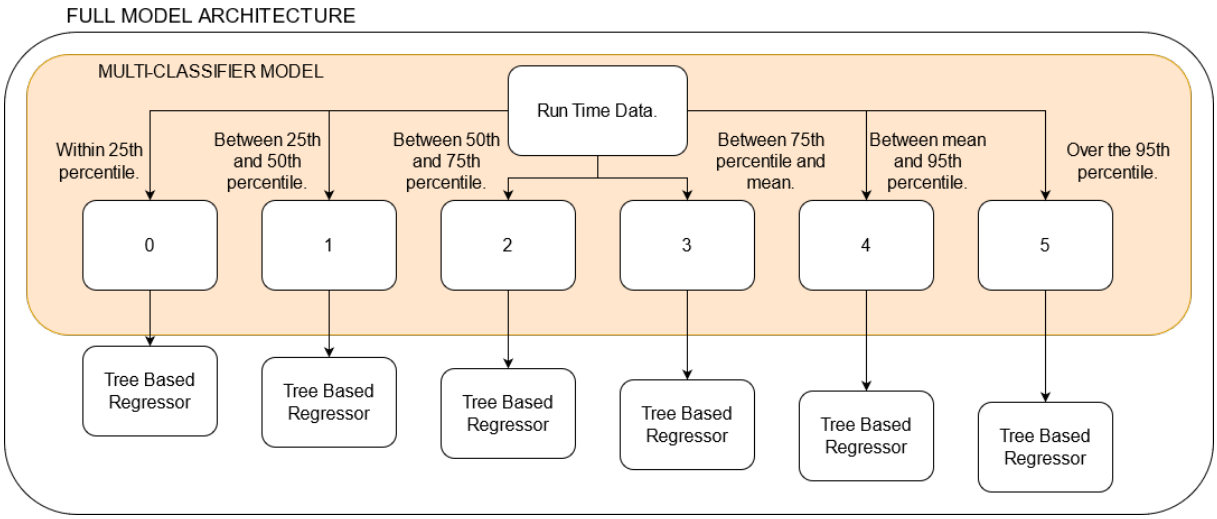


Figure 28: Run time regressor pipeline model architecture.

The multiclassification model was developed using a technique called stacking. Classifier stacking is an ensemble learning method in which multiple classifiers (referred to as base models) are trained on the same dataset. The predictions of these base models are then combined using a meta-classifier, which is trained on the outputs of the base models. This approach enables the meta-classifier to learn from the strengths and weaknesses of the individual classifiers, thereby improving overall accuracy and robustness in making predictions. In this model, tree-based classifiers (`ExtraTrees`), gradient-boosted classifiers (`XGBoost`), and linear classifiers (`SGD`) were used as base models, while a Logistic Regression model served as the meta-classifier.

While the Stacking Classifier achieved only a slight improvement in accuracy over models like `ExtraTreesClassifier` —90% compared to 89% —its primary advantage lay in the reduced impact of misclassifications. Specifically, if a value in the under 25th percentile is misclassified into the 25th to 50th percentile range, the consequences are less severe than if it were misclassified into the range of values between the mean and the 95th percentile. This characteristic enhances the model's utility in applications where the cost of misclassification varies significantly across classes (results of different standard Classifier models in Appendice A).

5.2.4 The Models in Stacking

In this stacking ensemble, three distinct types of classifiers—tree-based classifiers (`ExtraTrees`), gradient-boosted classifiers (XGBoost), and linear classifiers (SGD)—were used as base models. These models capture different aspects of the data, and their outputs are combined using a Logistic Regression model as the meta-classifier. Below is a brief mathematical explanation of each model:

ExtraTrees Classifier The `ExtraTreesClassifier` is described mathematically in subsection 4.1.2.

XGBoost Classifier `XGBoost` (eXtreme Gradient Boosting) is a gradient-boosted decision tree algorithm. It builds an ensemble of weak learners (decision trees) sequentially, with each new tree attempting to correct the residual errors of the previous trees. The model minimizes a loss function $L(\theta)$, typically the cross-entropy for classification tasks, by applying gradient descent.

Given the current model's prediction $y_i^{(t)}$ at iteration t, `XGBoost` updates the model by adding a new tree to minimize the residual:

$$L(\theta) = \sum_{i=1}^{n} \ell(y_i, y_i^{(t)} + f(x_i; \theta_t)) + \Omega(f_t)$$

where ℓ is the loss function, $f(x_i; \theta_t)$ is the newly added tree, and $\Omega(f_t)$ is the regularization term to prevent overfitting.

SGD Classifier The Stochastic Gradient Descent (SGD) classifier is a linear classifier that optimizes a loss function using stochastic gradient descent. For binary classification, it minimizes the hinge loss (for support vector machines) or the log loss (for logistic regression).

Given a training dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, the SGD algorithm updates the model's weights θ iteratively:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t)$$

where η is the learning rate, and $\nabla L(\theta_t)$ is the gradient of the loss function $L(\theta_t)$ with respect to the weights.

Logistic Regression (Meta-Classifier) The meta-classifier used in this stacking model is a Logistic Regression model, which combines the outputs of the base models. Logistic Regression models the probability of the binary outcome using the logistic function:

$$P(y = 1 \mid X) = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n))}$$

where $X_1, \ldots, X_n$ are the predictions of the base models, and $\beta_0, \beta_1, \ldots, \beta_n$ are the parameters

learned during training. The final prediction is made by applying a threshold (typically 0.5) to the predicted probability.

5.2.5 Regression Models For the Classes

After the multiclass classification process, each predicted class was assigned to its respective regressor model. Each regression model was evaluated using different scalers, as presented in Appendix A. The regression models for all classes were either `RandomForest` or `ExtraTreesregressors`. The first four classes showed a slight performance improvement with `MinMax` scaling, while the last two classes, representing the higher range of the data, benefited more from Standard scaling. The features utilized by each model and their contribution to the Mean Absolute Error (`MAE`) were tested manually to identify which features enhanced model performance. Table 5 presents the final set of features selected for each class model.

Class	**Features**
0	No. of Cores and `GPUs` Req., Memory Req., No. of Tasks (in a job), `QoS`, Priority, Eligible Time.
1	No. of Instructions, No. of Cores and `GPUs` Req., Memory Req., `QoS`, Eligible Time.
2	No. of Instructions and Branch Instructions, No. of Cores and `GPUs` Req., Memory Req., No. of Tasks (in a job), Priority, Eligible Time.
3	No. of Instructions and Branch Instructions, No. of Cores and `GPUs` Req., No. of Tasks (in a job), `QoS`, Priority, Eligible Time.
4	All Features Available.
5	No. of Cores and `GPUs` Req., Memory Req., No. of Tasks (in a job), `QoS`, Priority, Eligible Time.

Table 5: Features utilized for different class of run time regression models.

The results obtained with this model (Figure 28) are presented in Table 6. These results demonstrate

a significant improvement compared to the standard model results shown in Table 4. Along with these results the R^2 (coefficient of determination- see section 5.1) obtained for the model was 0.96 indicating that the full model is able to fit the data quite well. Figure 29 visualizes the scatter plot of predicted vs. actual values, where each point represents a prediction. The red line indicates a perfect prediction, and the closer a point is to this line, the more accurate the prediction. However, the data shows the highest levels of scattering after the 95th percentile, which is expected since this portion represents data within a range of 50,000 seconds, where higher scattering is common.

Although allowing this level of scatter may not be highly detrimental, two measures were implemented to improve accuracy. First, as seen in Figure 29, the far-right values exhibit significant scatter and under-prediction. These values likely correspond to jobs scheduled for daily execution. To address this, an additional classifier was developed specifically to identify and predict these values using a linear regression model (see Figure 30).

The second measure introduced a flagging mechanism. When a model predicts a value that falls outside its assigned class range, that value is flagged with the predicted class label. This allows a potential scheduler to treat it as a cautionary flag, indicating the possibility of an inaccurate prediction. Additionally, the flag provides context about the classifier's original class prediction, offering an interval reference for further decision-making.

Metric	**Value**
MAE (Overall)	578.82
MAE under 25th percentile	1.35
MAE at 25th-50th percentile	35.65
MAE at 50-75th percentile	246.63
MAE at 75th percentile to Mean	447.86
MAE at Mean-95th percentile	2393.49
MAE at over the 95th percentile	4621.12

Table 6: Mean Absolute Error (MAE) for the run time regressor pipeline model across different percentiles.

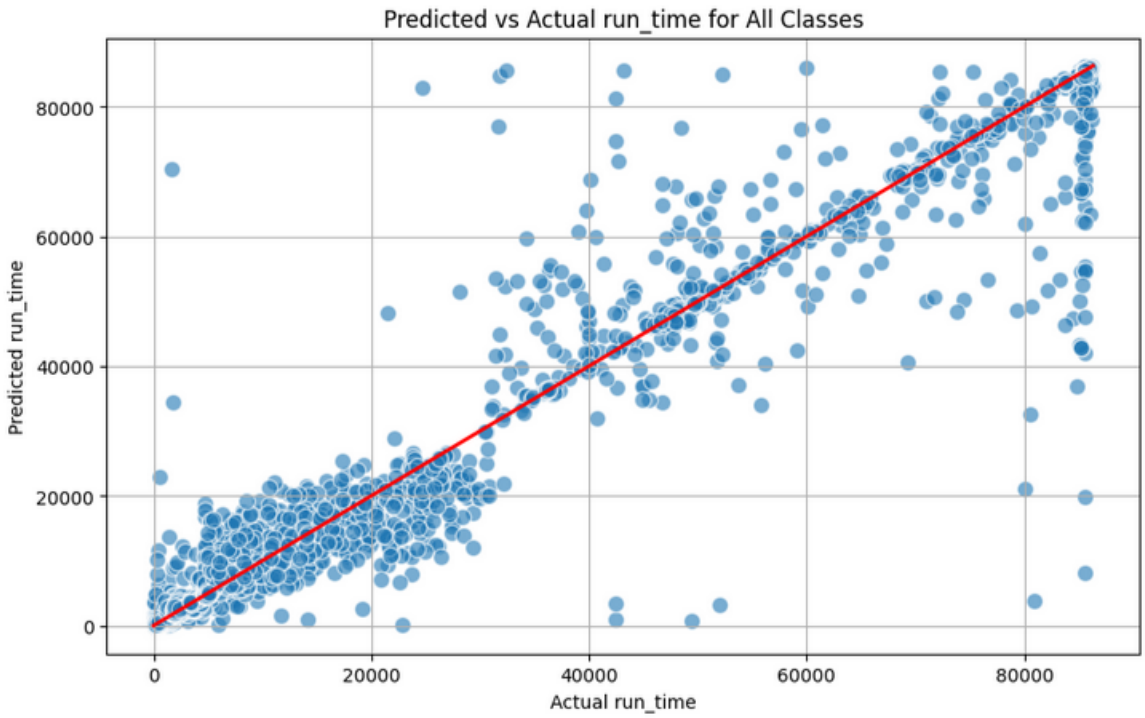


Figure 29: Scatter plot of the run time regressor pipeline model.

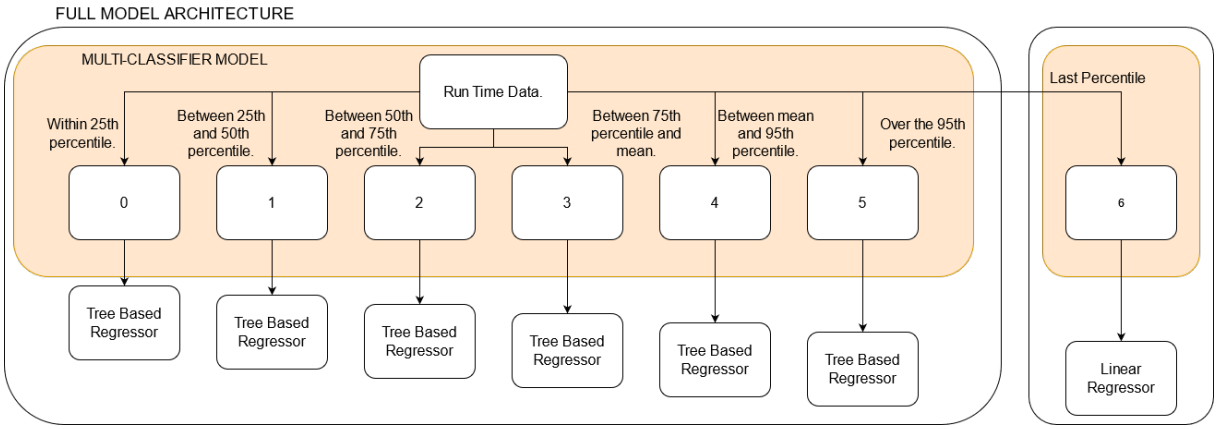


Figure 30: Run time prediction pipeline model diagram with additional class.

Figure 31 shows the confusion matrix related to the new multi-classifier with an extra class. Figure 32 presents the updated scatter plot, demonstrating that the corrective measure applied to the far-right data has effectively resolved the issue of severe under-prediction. The data in this region is now more accurately predicted. Table 7 shows the updated results for the model, highlighting an overall improvement in Mean Absolute Error (MAE), with a particularly significant reduction in the MAE for the last 5th percentile of the data. The R^2 (coefficient of determination 5.1) obtained for the model also improved slightly to 0.97 as expected considering the extra step to fit the data better.

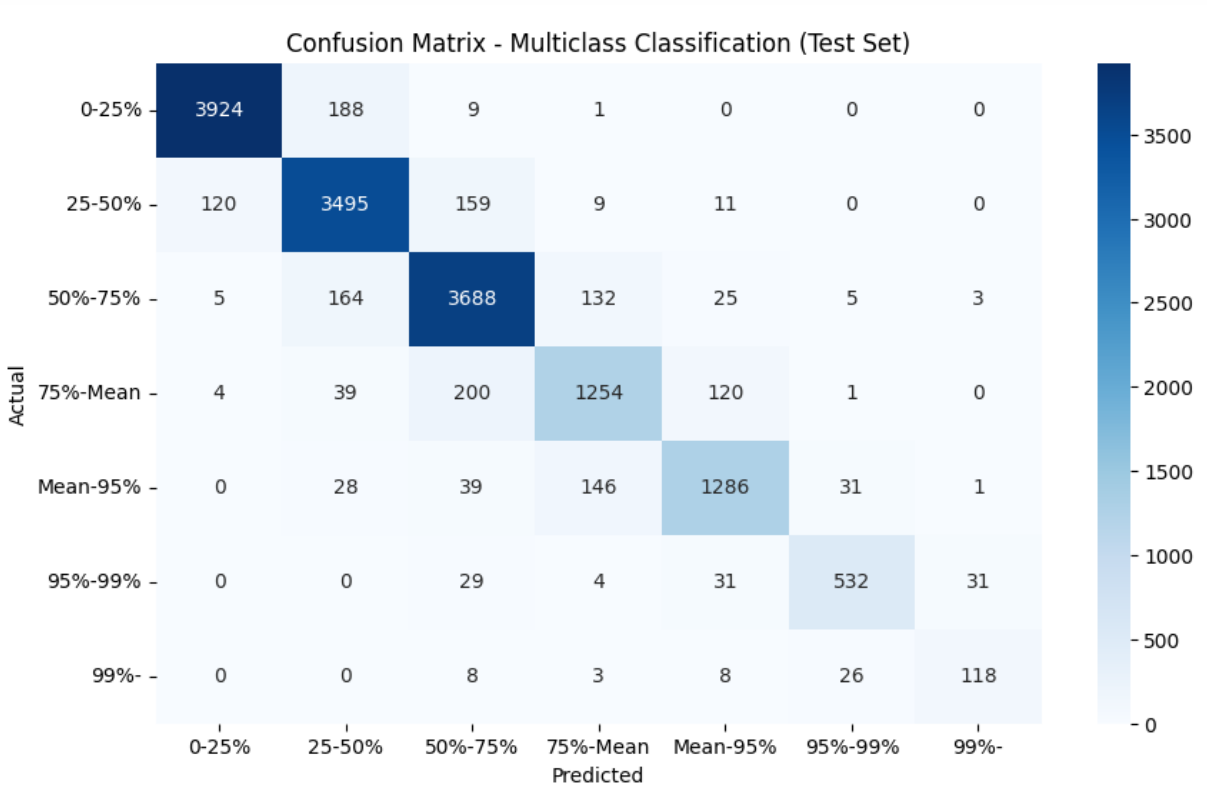


Figure 31: Confusion Matrix for the Final Multi-Classifier model in run time prediction.

Metric	Value
MAE (Overall)	495.47
MAE under 25th percentile	1.32
MAE at 25th-50th percentile	28.29
MAE at 50th-75th percentile	271.67
MAE at 75th percentile - Mean	397.49
MAE at Mean-95th percentile	2413.43
MAE at 95th-99th percentile	3554.1
MAE at the last percentile of Data	686.48

Table 7: Mean Absolute Error (MAE) for the run time pipeline model across different percentiles (extra split).

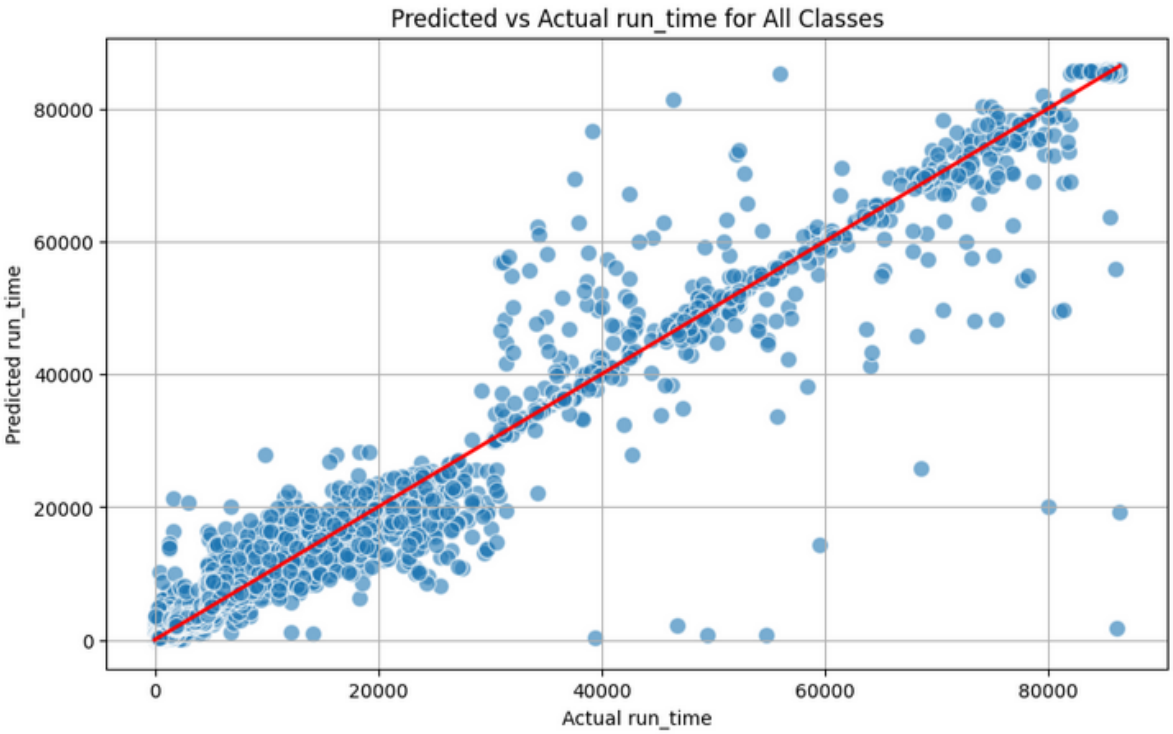


Figure 32: Scatter Plot of the Final Model.

5.2.6 The Models in the Regression Process

In the regression process, three models were employed: `ExtraTreesRegressor`, `RandomForestRegressor`, and `HuberRegressor`. Each of these models utilizes distinct approaches to predict continuous outcomes based on the input features. Below is a mathematical overview of each model.

ExtraTreesRegressor The `ExtraTreesRegressor` is an ensemble method that constructs multiple decision trees and averages their predictions to improve predictive performance. Unlike traditional decision trees, ExtraTrees introduces additional randomness in both feature selection and threshold splitting at each node.

Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, where x_i represents the feature vector and y_i is the continuous target variable, the `ExtraTreesRegressor` builds M trees $T_m(x)$, each trained on a bootstrap sample of the data. The prediction for a new input x is obtained by averaging the predictions of all M trees:

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} T_m(x)$$

RandomForestRegressor Similar to `ExtraTreesRegressor`, the `RandomForestRegressor` is an ensemble method that combines the predictions of multiple decision trees to enhance accuracy and reduce overfitting. However, it differs in that it uses a more structured approach for creating trees,

specifically bootstrap sampling and feature randomness.

The prediction for a new input x is calculated as follows:

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} T_m(x)$$

where each tree $T_m(x)$ is constructed from a different bootstrap sample of the training data. The trees in a Random Forest are typically grown to their maximum depth, and the split points are determined based on the best possible reduction in a criterion such as mean squared error (MSE).

HuberRegressor The `HuberRegressor` combines the principles of linear regression with a robust loss function to mitigate the influence of outliers in the dataset. It employs the Huber loss, which is defined piece wise:

$$L(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{if } |y - f(x)| \leq \delta \\ \delta \cdot (|y - f(x)| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

where δ is a threshold parameter that determines the point at which the loss transitions from quadratic to linear.

The `HuberRegressor` aims to minimize the total loss across all samples, defined as:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} L(y_i, f(x_i; \beta))$$

where $f(x_i; \beta)$ is the linear model given by $f(x_i; \beta) = \beta^T x_i$.

This approach allows `HuberRegressor` to remain sensitive to small errors while being robust to large errors, making it particularly effective in the presence of outliers in the dataset.

5.2.7 Run Time Predictions Result Discussion

In order to avoid job slowdowns the prediction time was studied with the python package **time** which allowed to understand that the predictions were able to be computed in the dozens of milliseconds which is an acceptable time response considering that the run time of the data is in the seconds (up to almost a day).

The final model achieved an overall mean absolute error of 495.47 seconds and an R^2 of 0.97 on the test data. Although the MAE value might initially seem large, Table 7 provides context showing that this figure appears inflated due to the wide range of values present in the data. When analyzed by interval, the model actually produces highly satisfactory results. Furthermore, in the scatter plots shown in Figure 32, it is visually apparent that the data points are closely clustered around the line representing perfect

prediction, further indicating the model's overall strong performance. Finally, the R^2 value near 1 indicates that the model understands the data very well.

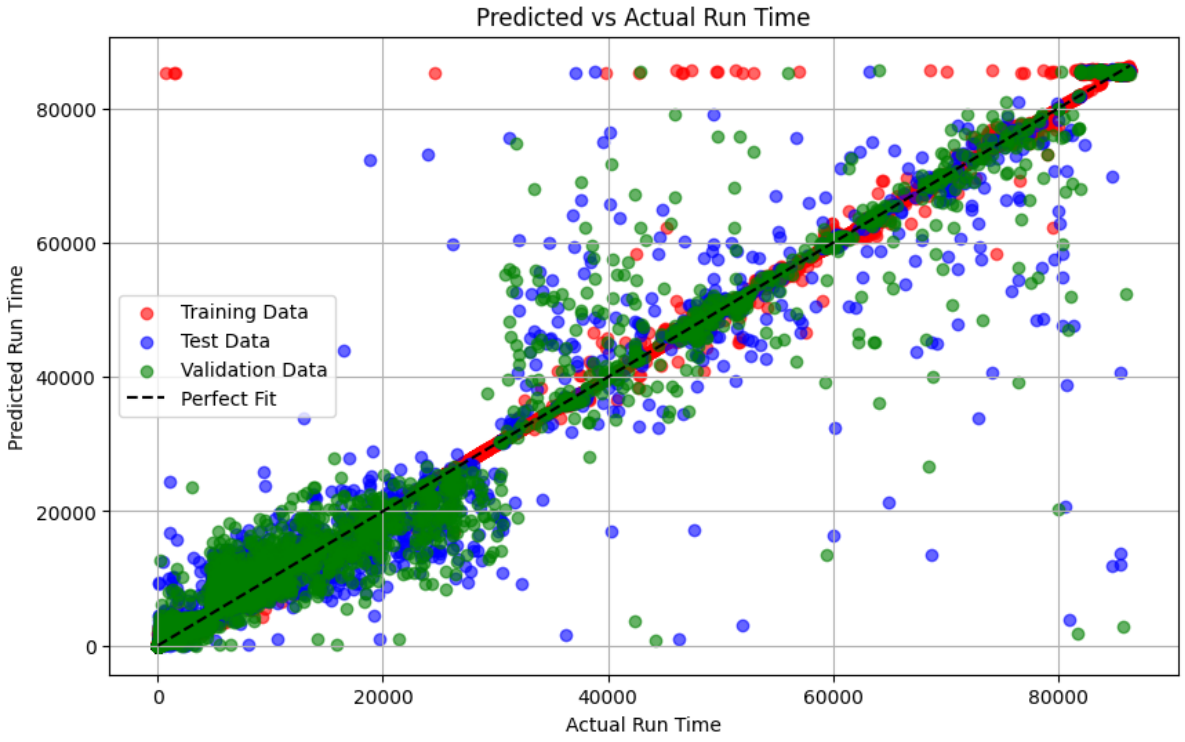


Figure 33: Scatter plot of the run time regressor final model across all data.

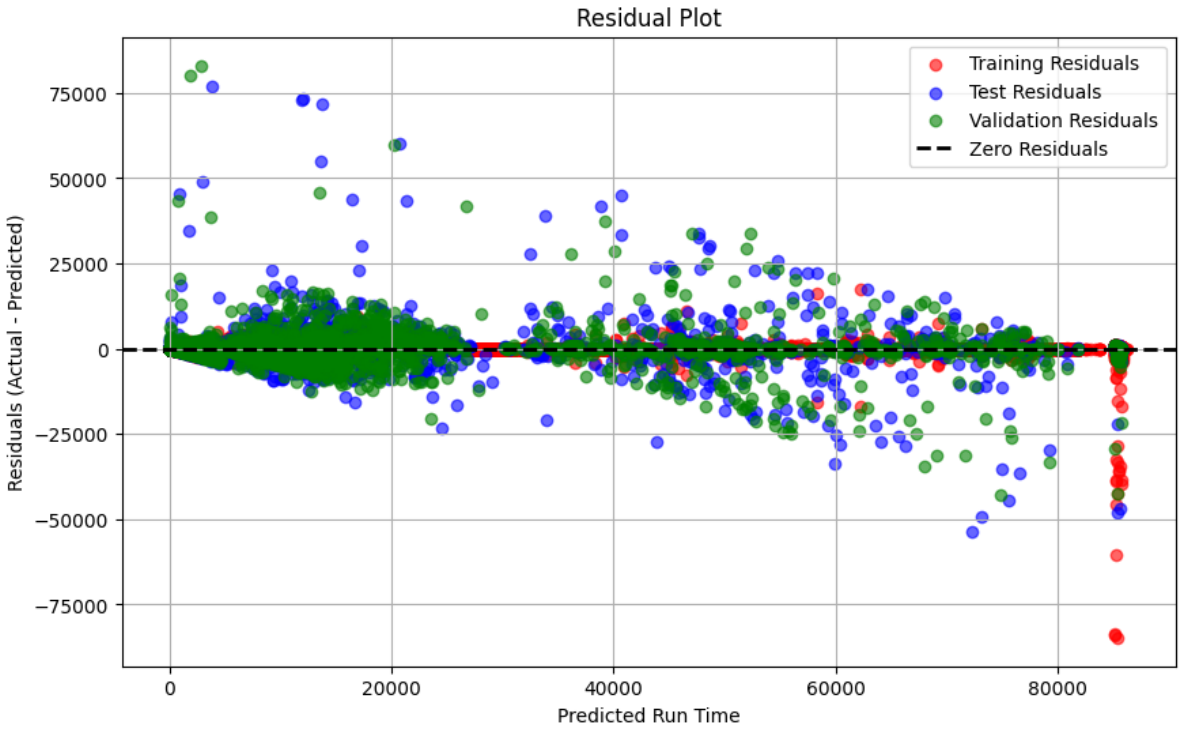


Figure 34: Residual error plot of the run time regressor final model across all data.

The scatter plots in Figure 33 and the residual plots in Figure 34 illustrate the distribution of data points across the Training, Test, and Validation datasets, as well as their corresponding residual errors. The overall Mean Absolute Error (MAE) for the combined dataset is 124 seconds, which is considerably lower than the training MAE. This discrepancy is expected, as training datasets often exhibit a close fit, with only a few outliers present.

These figures reveal a high density of points near the reference lines, suggesting that the model makes accurate predictions. However, there are notable outliers, represented as individual dots significantly distanced from the reference lines. These outliers stand out more prominently compared to the denser clusters of points, indicating areas where the model's predictions diverge from actual values.

In HPC environments, predicting job runtimes can be challenging due to several factors. Variability in system load, fluctuations in resource availability, and differing execution conditions can all significantly affect how long a job actually takes to complete. As a result, while the model may provide an accurate prediction based on historical data, the actual runtime can deviate considerably, leading to outliers in the results. These outliers are often accepted in HPC contexts because they reflect the inherent unpredictability of job execution under varying operational conditions. For instance, the contention for shared resources, such as CPU, memory, and I/O, can lead to unpredictable delays.

The complexity of the model, which leads to these good results, is largely necessitated by the nature of the available data. When the dataset lacks features that directly correlate with the target variable, the model must explore more intricate patterns and relationships. For instance, data such as the amount of memory being used, the number of jobs running concurrently, the clock speed, and other system-related metrics could significantly enhance the model's ability to generate accurate predictions. However, in the absence of such information, model complexity compensates by extracting deeper insights from the available features.

5.3 Power Consumption Prediction

This section focuses on analyzing the process of developing a regression model for power consumption prediction in a HPC system. The dataset used is still the dataset in Figure 24 meaning the dataset with only irregular jobs and the metrics used for gauging the performance of the model are the same as used in Section 5.2 (R^2 and Mean Absolute Error).

5.3.1 Regression Models predicting Power consumption

Identically to sections 5.2.1 and 4.1 the data was fed to the regression models without any prior preparations and the models were default to gauge their performance. Figure 35 presents the initial results for the power consumption regression models. In these results it is shown a poor performance by the R^2 metric but, more importantly, it is visible a very poor performance of the Mean Absolute Error. Values that high do not motivate a fine tuning attempt. Instead it is necessary to take a different approach to the problem.

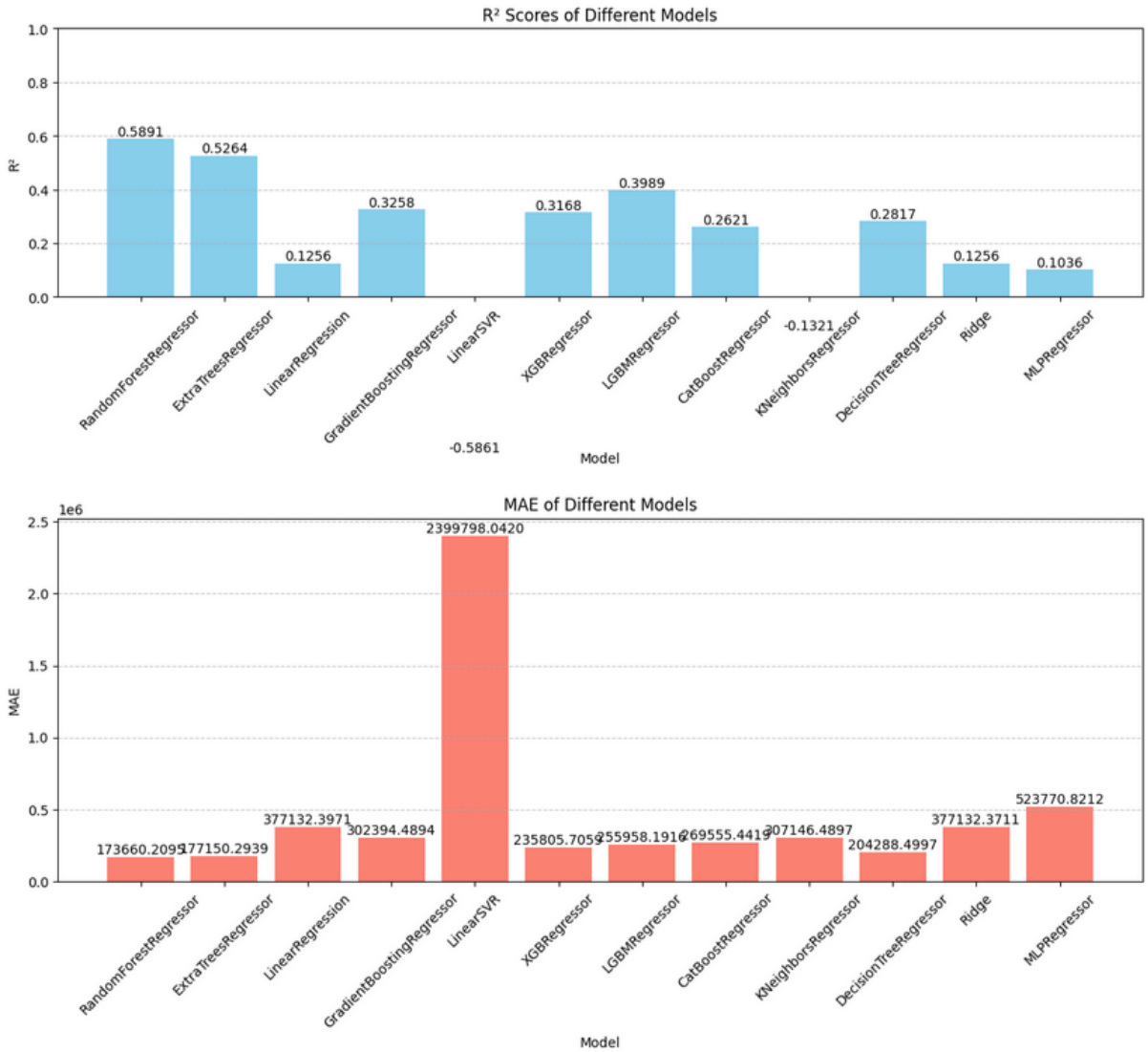


Figure 35: Initial power prediction supervised learning regression models using the total power feature.

5.3.2 Understanding Power consumption

Below, the statistics associated with the actual data for the Power Consumption metric are presented (Table 8). The range of values for power consumption in an HPC system is notably wide, ranging from 20

to 385,317,450 Watts, which reflects the variability in job durations. Jobs can run from under a second up to a day as seen in Section 5.2, leading to power consumption measurements that represent the usage of an HPC system node over extended periods. Consequently, understanding the power consumption of a job becomes challenging without considering the associated job duration and there is where the full model in this investigation fits the necessities in characterizing jobs in HPC systems.

The prediction of job runtime has been studied in this investigation (Chapter 5.2). In HPC systems, the sampling interval for power consumption (and other metrics) is predetermined by the system or the system administrator. Since this is a system-level metric rather than a job-specific one, predicting the runtime of a job and the power consumption per sample, or simply the mean power consumption, allows the system to understand the overall power consumption associated with that job.

Metric	**Value (Watts)**
Mean	392317.86
Standard Deviation	3183244.81
Minimum	20.00
25th Percentile	1020.00
50th Percentile (Median)	10360.00
75th Percentile	123530.00
Maximum	385317450.00

Table 8: Table of Power Consumption data statistics.

5.3.3 Introducing Mean Power Consumption in the dataset (Power Consumption per sample)

The metric power consumption per sample was introduced to the dataset present in Section 3.3.4. Table 9 contains the numerical statistics that describe the Power Consumption per sample metric, observing a drastic difference in range of values compared to Table 8.

5.3.4 Repeating the Power Consumption prediction with different models

Now, the power consumption prediction, in Figure 37, instantly achieved promising results. This happens because the power consumption per sample can be easily described by the amount of hardware it utilizes. A job utilizing/requesting more GPUs, more memory and more CPU cores is a job with a higher amount of

Metric	Value (Watts)
Mean	2289.80
Standard Deviation	8556.27
Minimum	20.00
25th Percentile	570.00
50th Percentile (Median)	780.00
75th Percentile	1096.01
Maximum	313297.50

Table 9: Table of Mean Power Consumption data statistics.

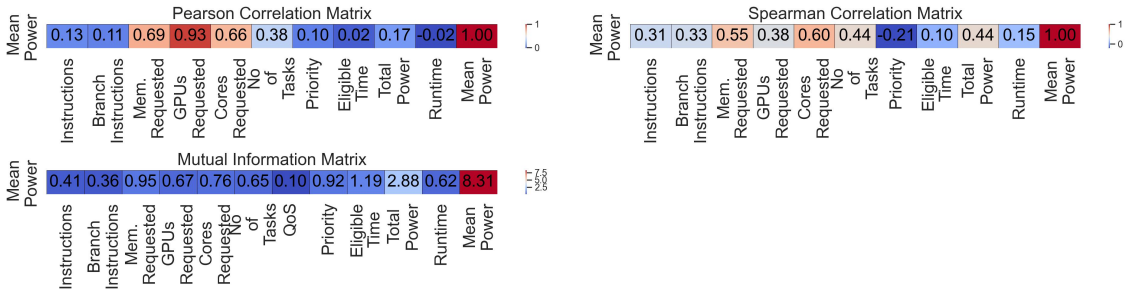


Figure 36: Mean Power relations with existent dataset features.

power consumption because in this case power consumption per sample is measuring exactly the power being utilized by those pieces of hardware. In figure 36 this is observable in the Pearson correlation matrix (left) which indicates a heavy linear relation between these hardware components and the Power Consumption per sample.

In evaluating the various regression models in Figure 37, the `RandomForestRegressor` model (mathematically expressed in Section 5.2.6) was selected based on the slight better performance compared to the other models in both metrics. It demonstrated robust generalization capabilities and effectively handled the complexities inherent in the dataset.

5.3.5 Complexity Analysis

In the correlation matrix Figure 36, it is possible to see that the correlation factors between the metrics used for prediction and power are, in general, quite high. It is fair to say that if two variables have a high correlation with each other, simpler Machine Learning models may perform well for predictions involving these variables for several reasons:

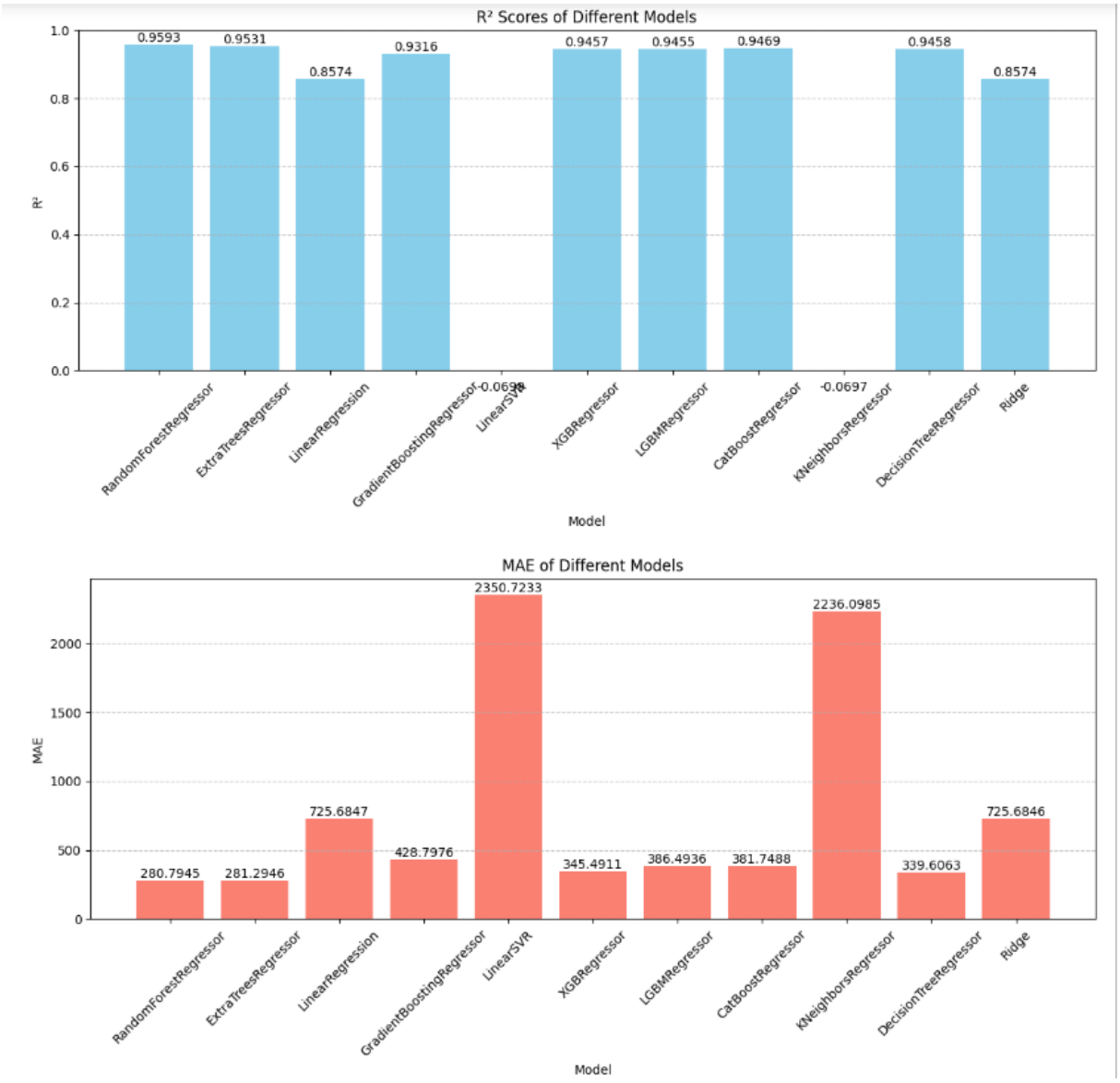


Figure 37: Initial power prediction supervised learning regression models using the power consumption per sample feature.

- **Redundant Information:** When two variables are highly correlated, they carry similar information. This redundancy means that the model does not need to learn complex relationships to capture the underlying patterns in the data.

- **Reduced Variance and Overfitting:** Simple models are less prone to overfitting because they do not have the capacity to learn overly complex patterns. When the data has highly correlated features, the model benefits from the reduced noise and variance, leading to more stable and generalizable predictions.

- **Linear Relationships:** Highly correlated variables often suggest a linear or near-linear relation-

ship. Simple models like linear regression excel at capturing linear relationships without the need for advanced non-linear transformations or complex architectures.

- **Easier Interpretation:** Simple models with highly correlated variables are easier to interpret. For instance, in linear regression, the coefficients will directly reflect the strength and direction of the relationship between the variables.
- **Less Need for Feature Engineering:** With highly correlated variables, the need for advanced feature engineering is reduced because the relationship is already apparent in the original features.

However, there are some caveats to this scenario:

- **Multicollinearity:** If two or more predictors are highly correlated in a regression model, it can cause multicollinearity, which makes it difficult to determine the individual effect of each variable. In such cases, while the model might perform well, the interpretability and reliability of the coefficients may suffer.
- **Non-Linear Relationships:** If the relationship between the variables is non-linear or involves complex interactions, simple models may not capture it well, even if the variables are highly correlated.
- **Correlation Does Not Imply Causation:** A high correlation between two variables does not necessarily imply a causal relationship. Simple models may perform well in such cases, but they cannot infer causality.
- **Data Distribution and Noise:** If the data has a lot of noise or is not well-distributed, even highly correlated variables may not lead to good model performance.

In Figure 37, the two metrics used to evaluate the model's performance are: the R-squared score (R^2) and the Mean Absolute Error (MAE). The R^2 score, also known as the coefficient of determination, indicates how well the model explains the variance in the actual data. An R^2 of 0.96 means that the model explains 96% of the variance in the power consumption data. This is a very high R^2 value, indicating that the model fits the data well and has strong predictive power, as most of the variance in the actual power values is captured by the model. The MAE represents the average absolute difference between the predicted and actual power values. An MAE of 281.8 means that, on average, the model's predictions are off by around 281.8 watts (further analysis in Section 5.3.8).

The performance of the `RandomForestRegressor` was evaluated with different numbers of estimators. In Figure 38 the number of trees are 50, 100, 200, 300 and 500. Notably, the model achieved similar R^2 scores regardless of complexity for all models, with only a difference of 0.0005 for all models; all configurations achieved an R^2 of approximately 0.96. In the MAE metric there a difference of <2 between all models represented in Figure 38 (from 50 trees to 500 trees).

This indicates that increasing the number of estimators beyond a certain point did not yield significant improvements, suggesting that 100 estimators were sufficient for near optimal performance (as far as model complexity). In Figure 38, it is possible to observe the different models performance and with it understand the closeness of the values showing that complexity does not provide the model with significant improvement in performance as it was expected and already discussed.

5.3.6 Feature Importance

Given the scenario of power consumption prediction, it is highly probable that metrics related to hardware utilization, such as the number of GPUs requested, have significant importance for regression models. This is supported by the high correlation factor observed in Figure 24 and the findings of Antici et al. [2023], which highlight that GPU-bound jobs tend to consume more power compared to CPU-bound jobs. As the number of GPUs requested is directly related to hardware usage, it is expected that higher utilization of hardware resources will result in increased power consumption.

In this context, it is noteworthy that the model could still perform reasonably well even with a reduced number of metrics. The key metrics, particularly those related to GPU usage, provide substantial predictive power. This suggests that in scenarios where obtaining a wide range of metrics is challenging, focusing on critical metrics like the number of GPUs requested can still enable the development of effective power consumption predictors in real HPC system environments.

5.3.7 Further Analyzing the model

The model does not benefit from complexity so to further improve the model it was necessary to take other approaches. Initially, the data had to be scaled (Figure 40), as discussed previously in Section 4.1.1, to improve generalization to new data and reduce sensitivity to the wide range of values. However, this did not necessarily improve the model. Due to the skewness of the data, the model faced another issue, which was occasional occurrences. In Table 9 it is observable that the 75th percentile is 1096.01 which means that 75% of data is comprised under the value of 1096.01. In fact 87% of data is under the mean value (2289W). Figure 41 shows the skewness of the data, on the left of the red vertical line is comprised

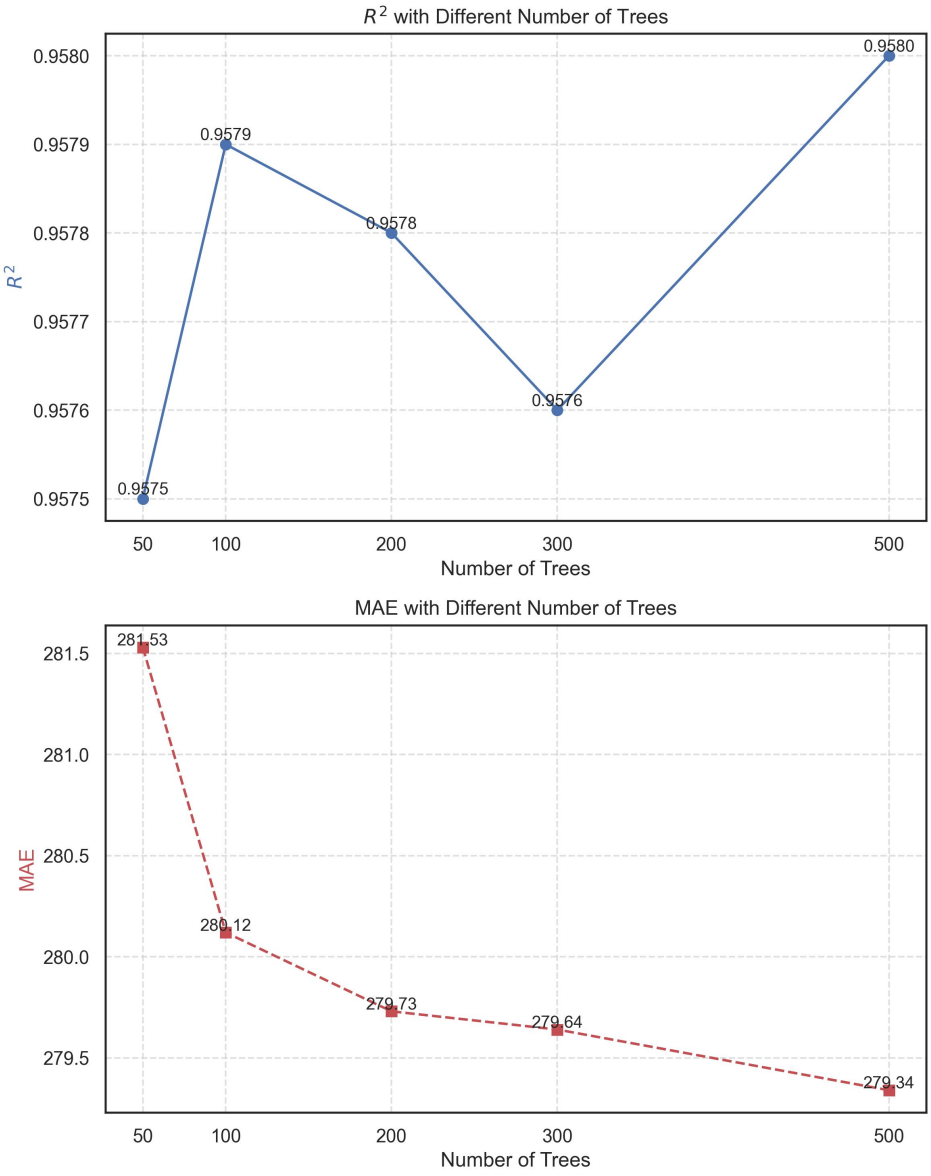


Figure 38: `RandomForestRegressor` model performance with increasing complexity.

90% of data values, on the left the green vertical line are comprised 99% of Power consumption values. Although the values on the right of the green line are unusually high, it is possible that a job ran for days and only had its power consumption sampled at the end, or that the measurement tool introduced an error. However, there is no way to determine this with certainty. Therefore, simply removing these jobs is not justified, instead, it is important to carefully interpret the results.

5.3.8 Contextual Analysis of `MAE` and R^2

The Mean Absolute Error (`MAE`) values across different intervals of power consumption are summarized in Table 10. The `MAE` for the entire dataset is 280.1 watts, indicating the average magnitude of prediction

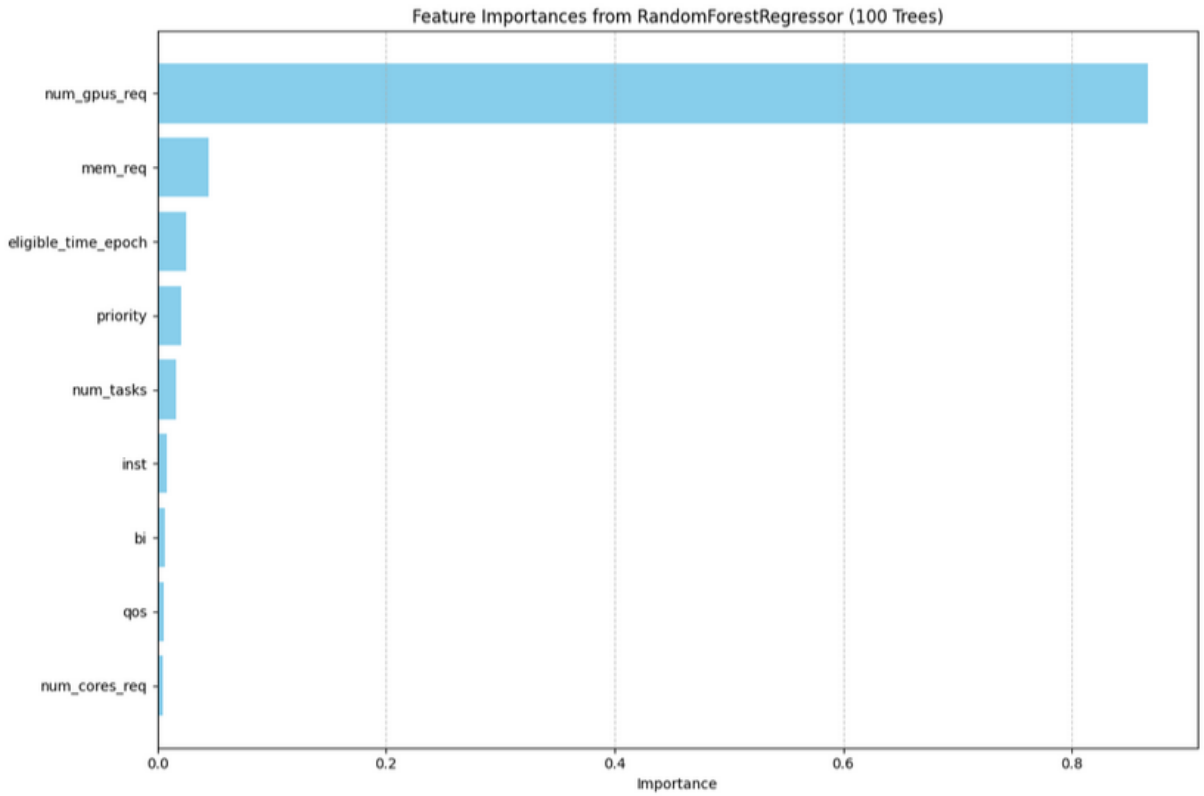


Figure 39: Feature Importance in power consumption per sample prediction with the `RandomForestRegressor` model.

errors without considering their direction. When broken down by specific percentiles, the `MAE` shows variability based on the power consumption levels. For the 50th percentile (median), the `MAE` is significantly lower at 76.63 watts, reflecting that predictions are more accurate around the median power consumption. As higher percentiles are considered, such as the 75th percentile with an `MAE` of 86.09 watts, the 90th percentile with an `MAE` of 116.05 watts and 99th percentile with 190.24 watts, the error growth is stable and shows good results achieving the final result of 280.1 watts of error in the wide range of data values.

In fact the `MAE` for values above the 99th percentile is notably higher at 9164.41 watts. This happens because those values are extremely high (ranging from 24090.38 watts to 313297.5 watts) and very unusual occurrences. In graph 42 is possible to observe the way the error naturally grows as the Power consumption values grow demonstrating that the model struggles significantly with the highest power consumption levels. These results suggest that while the model performs reasonably well for typical power consumption levels, its accuracy diminishes for extreme values, highlighting potential areas for model improvement in handling outliers and high-end predictions.

Overall the `MAE` of the model with 100 trees, is 280W which is significantly lower than the mean (2289.80) of the actual values Table 9. This indicates that, on average, the model's predictions are

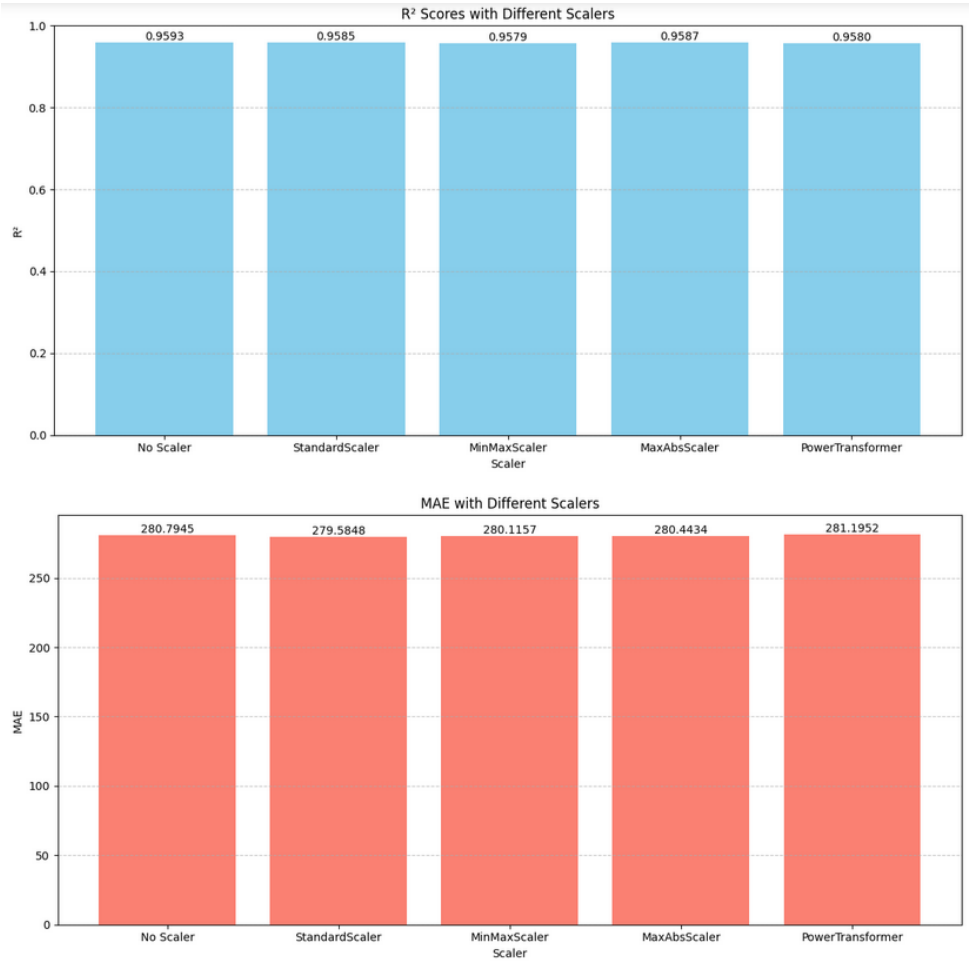


Figure 40: Different `scalers` Performance in the power consumption prediction `RandomForestRegressor` model.

Interval	**Value**	**MAE**
25th Percentile	570.00	75.61
50th Percentile (Median)	780.00	76.63
75th Percentile	1096.01	86.09
90th Percentile	3563.19	116.05
99th Percentile	24090.38	190.24
100th Percentile	313297.50	280.1

Table 10: Table of power consumption prediction model results analysis.

reasonably close to the true values, especially in relation to the average scale of the data.

Relative to the standard deviation of the actual values, which is quite high (8556.27), reflecting substantial variability in the data, a `MAE` of 280W is suggesting that the model performs well compared to the

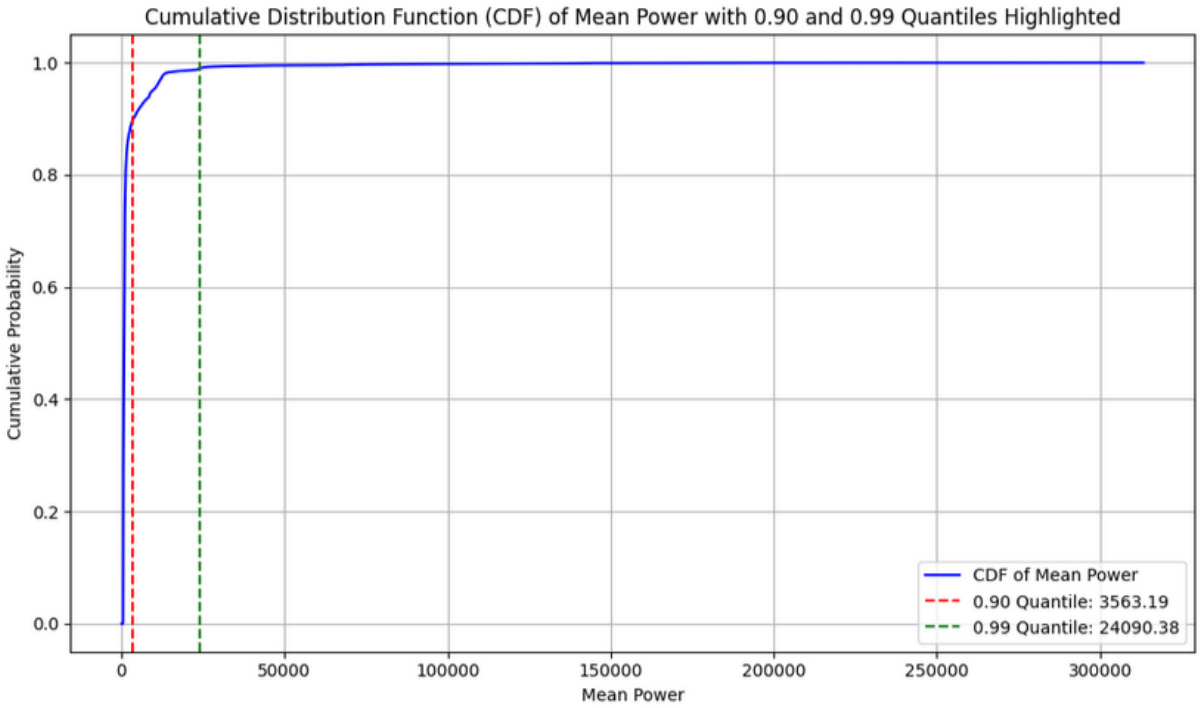


Figure 41: Cumulative Distribution Function of the power consumption per sample metric.

overall spread of the data.

A R^2 value of 0.96 for the model with 100 trees, indicates that approximately 96% of the variance in power consumption is explained by the model. This is significantly higher than the average scale of the data, suggesting that the model's predictions align well with the actual values, providing a high level of explanatory power.

In the context of the substantial standard deviation of the actual values (12006.26), the high R^2 value of 0.96 reflects that the model accounts for most of the variability in the data. This implies that despite the large spread in the data, the model effectively captures the underlying patterns, thus offering strong predictive performance relative to the data's variability.

Achieving an R^2 of 0.96 reflects that the model performs exceptionally well in explaining the variance in the data. While no model is perfect, this high R^2 value indicates that the model's predictions are highly accurate, with minimal unexplained variance, thus providing robust performance in the context of the dataset's wide range and variability.

5.3.9 Power Consumption Predictions Result Discussion

For a visual analysis, Figure 43 presents two different types of plots comparing the predicted data to the actual target data. These plots show a significant overlap between the predicted and actual values,

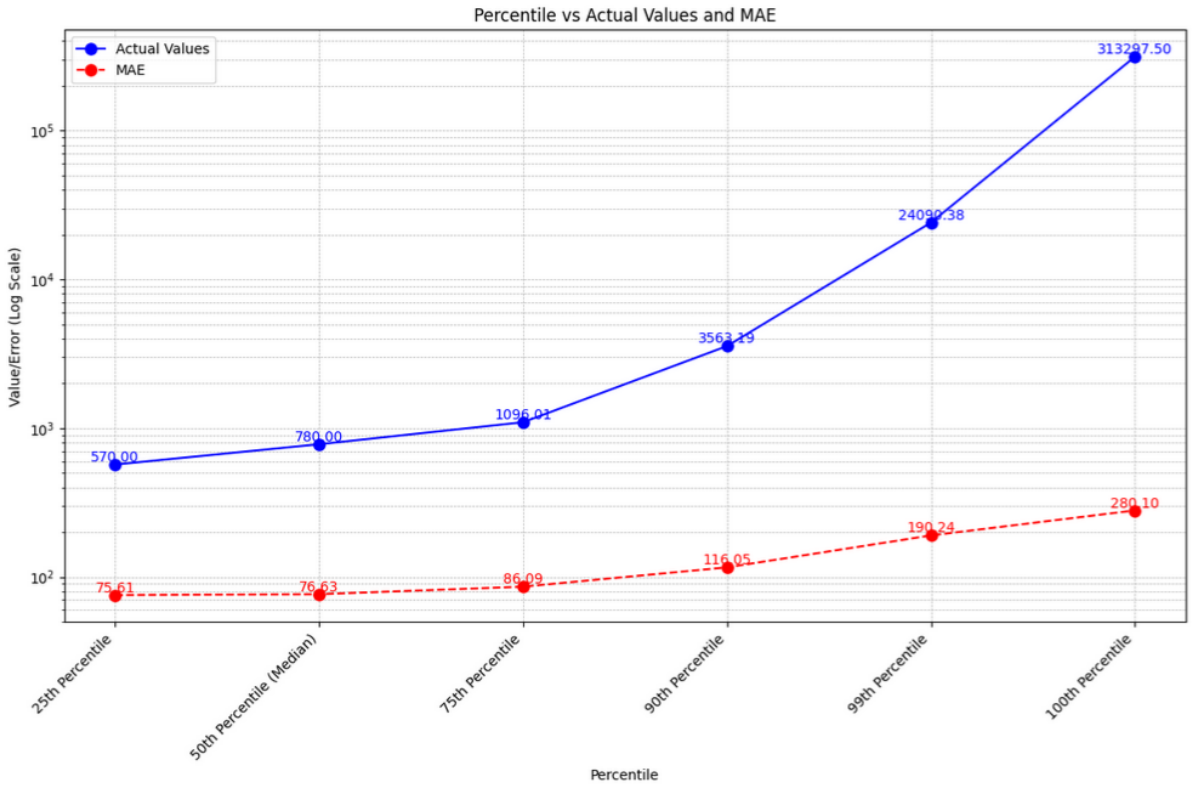


Figure 42: Power consumption distribution across different percentile and their respective associated prediction error.

further demonstrating the accuracy of the model (`RandomTreesRegressor` with 100 trees). The first plot is a Line Plot and its purpose is to visually compare how well the model's predictions align with the actual values over different samples. The closer the green line (predicted values) is to the blue line (actual values), the better the model's predictions. The second plot is a Scatter Plot and its purpose is to assess the accuracy of the predictions in a more granular way. Points close to the red dashed line indicate accurate predictions, while points further away indicate less accurate predictions.

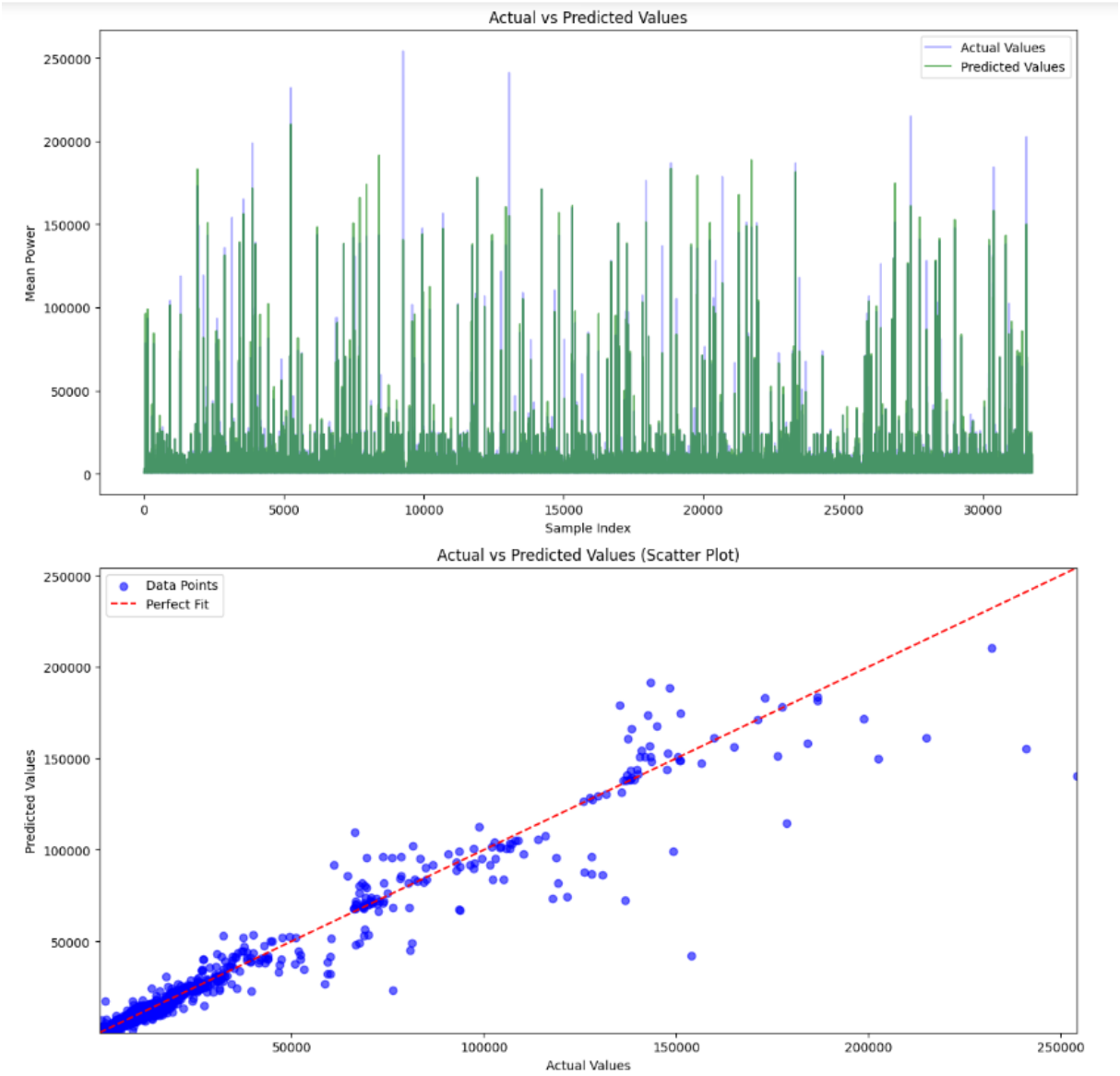


Figure 43: Actual values vs. Predicted values of power consumption, bar and scatter plots.

The scatter plot in Figure 44 and the residual plot in Figure 45 provide insights into the model's performance across the Training, Test, and Validation datasets, highlighting the distribution of predicted power consumption values and their corresponding residual errors. The overall Mean Absolute Error (MAE) for the combined dataset is 133 watts.

Both figures demonstrate a substantial concentration of points around the reference lines, indicating accurate predictions. However, a few outliers are noticeable, positioned significantly away from the reference lines. This is an anticipated outcome due to the inherent uncertainties in predicting power consumption within an HPC environment, which can be influenced by various factors such as resource contention, workload variability, and system load fluctuations. Such variability can lead to deviations from expected runtime, resulting in the observed outliers in the predictions.

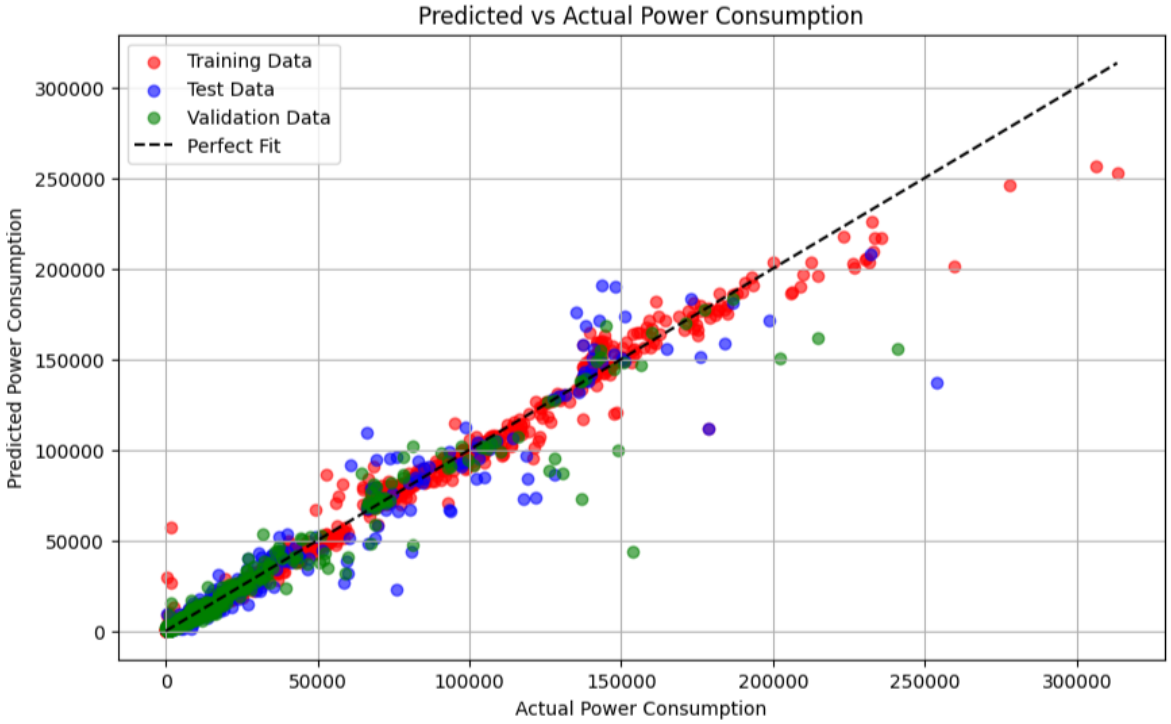


Figure 44: Scatter plot of the power consumption regressor final model across all data.

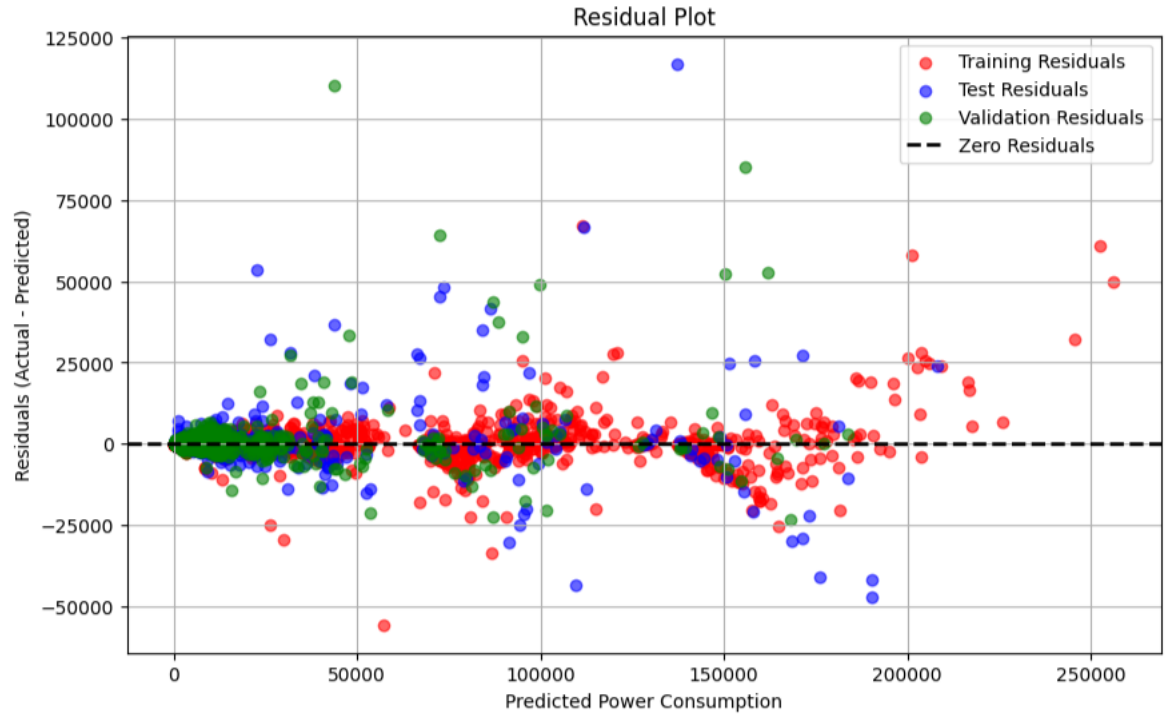


Figure 45: Residual error plot of the power consumption regressor final model across all data.

Chapter 6

Conclusions and Future Work

With the increasing prevalence of High-Performance Computing (HPC) systems, this thesis investigated job behavior within these environments and evaluated models and techniques for understanding the relationships between job features using various Machine Learning models. It presents a synthetic dataset and a Supervised Learning-based approach for predicting job characteristics in HPC systems. The proposed model effectively distinguishes between regular and irregular jobs and, for irregular jobs, estimates both runtime and power consumption. By improving the understanding of job behavior in heterogeneous HPC environments, this predictive capability supports better resource planning and allocation, ultimately contributing to enhanced energy efficiency and performance.

To address the limitations of available data on HPC metrics, a method for synthesizing a dataset was developed. This synthetic dataset was created by combining job metrics from the datasets provided in Antici et al. [2023] and Netti [2021]. The combined data contained metrics obtainable before a job is executed by a system. Additionally, the data was augmented by resampling (with error margin) the regular jobs, as their main characteristic is being deterministic/repetitive (see Section 3.3.3). This data synthesis approach was instrumental in training the Machine Learning models, ensuring sufficient and relevant data for modeling purposes, making the predictive models both reliable and applicable in real-world scenarios. The generated dataset comprised 278,982 jobs with various metrics, including the number of instructions, number of branch instructions, amount of memory requested, number of CPU cores requested, number of GPUs requested, quality of service (QoS) requested, priority level, runtime (in seconds), eligible time, and total power consumption (see Section 3.1.2). Furthermore, the data was carefully analyzed and balanced to create an almost even distribution between regular (43%) and irregular (57%) jobs, as detailed in Section 3.3.4.

Using this dataset, a classification predictor was developed employing a supervised learning model—`ExtraTreesClassifier`. This model outperformed others supervised learning models and deep supervised learning models. The final configuration used 50 estimators (trees), was scaled by `MaxAbsScaler`.

Recursive Feature Elimination (RFE) selected 8 features out of the 9 available: the number of instructions, number of branch instructions, amount of memory requested, number of GPUs requested, number of cores requested, eligible time, priority, and quality of service, only eliminating the number of tasks (parallelization) from the group (see Figure 16). Adjustments were also made to class weights and prediction thresholds, among other parameters, as discussed in Section 4.1. The classifier achieved an accuracy of 96.7%. Although the accuracy could have been further improved, a decision was made to prioritize a model that produced more false negatives than false positives, a trade-off beneficial in scheduling scenarios (see Section 4.1.5).

Following the classification process, the dataset was filtered to include only the jobs labeled as irregular, resulting in a reduced dataset of 158,800 jobs (see Section 5). With this refined dataset, a regression predictor for runtime was developed using a pipeline of supervised learning models, classifying the data into different quartiles (and smaller intervals when necessary) using a stacking classifier composed by a combination of ExtraTreesClassifier, XGBoost and SGD Classifier and afterwards applying a regressor model for each interval (ExtraTrees, RandomForest or Huber depending on the interval). This approach allowed for the narrowing of analysis ranges for each regressor, considering that the runtime data spanned from 0 seconds to nearly a full day (see Section 5.2.3). The final model achieved an overall Mean Absolute Error (MAE) of 495 seconds. However, this error was distributed differently across various intervals in the data, ranging from 1 second in the lower 25th percentile (with a limit of 16 seconds) to 3554 seconds in intervals approaching the maximum runtime of 86,395 seconds (see Section 5.2.7).

After developing the time prediction model, the research shifted focus to predicting power consumption. The power consumption predicted in this investigation refers to the power consumption per sample (see Section 5.3.2). The RandomForestRegressor model was selected for this task, as it outperformed others in its category (see Figure 37). This model was scaled with a MinMaxScaler, and all features were retained, as they contributed to improving the model's performance. The number of GPUs requested was the most important feature in the regression process, significantly outweighing other features in terms of importance (see Figure 39). The final model consisted of 100 trees and achieved an R-squared value of 0.96, with a Mean Absolute Error (MAE) of 280 watts (see Section 5.3.8).

Figure 46 shows the overall performance and structure of the entire model from classification of jobs to both target variables (run time and power consumption predictions), this image shows the flow of processes inside the model, the models used in each step, the results obtained by these models along with an estimation of prediction time in each step. The Pipeline model indicated in Figure 46 runtime model is more detailed in Figure 30.

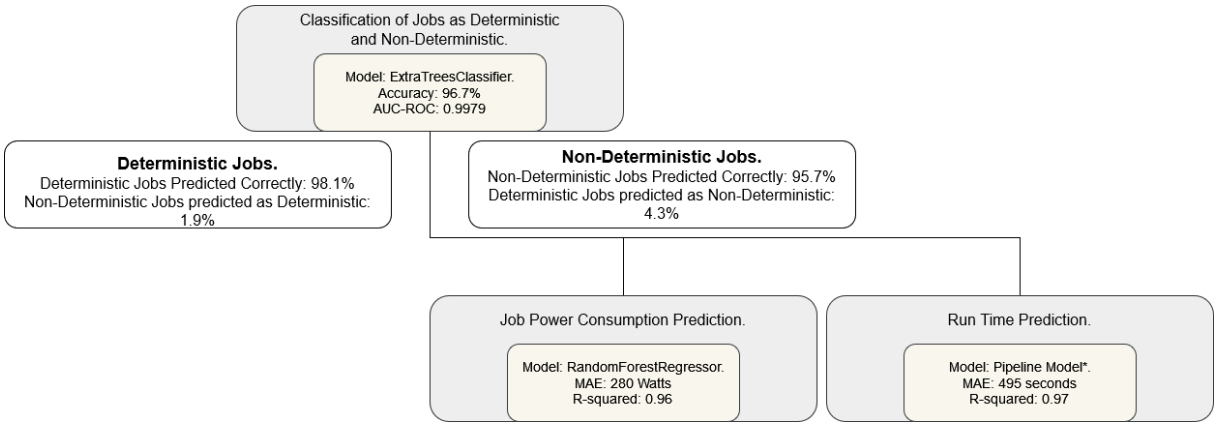


Figure 46: Overall developed model performance across all steps.

`Ensemble` models such as `RandomForest` and `ExtraTrees`, which were used in most of the classification and regression tasks, exhibited superior performance due to their ability to handle high-dimensional data, balance bias-variance tradeoff, and resist noise. The structure of `HPC` jobs, which includes diverse metrics such as resource allocation and job size, is well-suited for these models. Unlike algorithms such as `SVM` and `KNN`, which are sensitive to high-dimensional data or computationally expensive, `ensemble` models are both efficient and robust, yielding accurate predictions while minimizing overfitting risks. The complete model, along with the dataset and a Python notebook containing usage instructions and examples, is available in Carvalho [2024].

6.1 Prospects for Future Work

Future research should prioritize the creation of comprehensive datasets that encompass a wide range of `HPC` job characteristics and system metrics. These datasets should include processed data containing diverse `HPC` job metrics along with general system characteristics, which are essential for advancing both Machine Learning applications and `HPC` system optimization.

In the domains of burst time and power consumption prediction, future efforts should extend beyond the development of models that merely predict `HPC`-related variables such as power consumption and job completion time. A critical focus should be placed on analyzing real `HPC` data to identify which variables most significantly influence these predictions. Understanding the impact of different features on prediction accuracy is vital for enhancing model performance. Future research should aim to determine the most informative features and how they can be effectively utilized to refine predictive models, thereby identifying the key variables that contribute to more accurate and reliable predictions.

Another focal point for future work in the area must be the integration of advanced predictive and

classification models into HPC cluster management systems. Implementing these models within a cluster environment and integrating them with scheduling mechanisms could lead to substantial improvements in scheduling efficiency. By leveraging predictive models to forecast job performance, resource requirements, and system behavior, it is possible to develop schedulers that surpass current solutions. Such integration could result in more intelligent and adaptive scheduling strategies, optimizing resource utilization and minimizing job completion times.

Bibliography

A. B. Abdallah, D. R. Hill, A. Tolk, M. K. Traoré, and B. P. Zeigler. List of authors. In M. K. Traoré, editor, *Computational Frameworks*, page 111. Elsevier, 2017. ISBN 978-1-78548-256-4. doi: https://doi.org/10.1016/B978-1-78548-256-4.50008-9. URL https://www.sciencedirect.com/science/article/pii/B9781785482564500089.

A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Pearson, 2006.

F. Antici, M. S. Ardebili, A. Bartolini, and Z. Kiziltan. Pm100: A job power consumption dataset of a large-scale production hpc system. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023)*, pages 1–8, Denver, CO, USA, November 12–17 2023. ACM. doi: 10.1145/3624062.3624263. URL https://doi.org/10.1145/3624062.3624263. This work is licensed under a Creative Commons Attribution International 4.0 License.

L. Baolin, B. Rohan, D. Wang, S. Samsi, V. Gadepally, and D. Tiwari. Toward sustainable hpc: Carbon footprint estimation and environmental implications of hpc systems. *arXiv preprint arXiv:2306.13177*, 2023. URL https://arxiv.org/abs/2306.13177.

J. Berral, I. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavaldà, and J. Torres. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking (e-Energy 2010)*, pages 215–224, 2010. doi: 10.1145/1791314.1791349.

A. Bhuiyan, F. Reghenzani, W. Fornaciari, and Z. Guo. Optimizing energy in non-preemptive mixed-criticality scheduling by exploiting probabilistic information. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3906–3917, 2020. doi: 10.1109/TCAD.2020.3012231.

J. Bielecki and M. Śmiałek. Estimation of execution time for computing tasks. *Cluster Computing*, 26: 3943–3956, 2023. doi: 10.1007/s10586-022-03774-1.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. ISBN: 978-0387310732.

A. Borghesi et al. M100 dataset 1: From 20-03 to 20-12. Zenodo, 2023. Available at: https://zenodo.org/records/7588815.

S. Borkar. Thousand core chips: A technology perspective. In *Proceedings of the 44th Annual Design Automation Conference*, pages 746–749, 2007. doi: 10.1145/1278480.1278667.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, 2000. doi: 10.1145/339647.339657.

G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2005.

N. F. Carvalho. Performance and carbon footprint modelling of irregular tasks, 2024. URL https://github.com/NFilCarv/Performance-and-Carbon-Footprint-Modelling-of-Irregular-Tasks. GitHub repository.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

C. Cheng, G. Li, and J. Fan. Deep q learning cloud task scheduling algorithm based on improved exploration strategy. *Journal of Cloud Computing: Advances, Systems and Applications*, 24(4-5), 2023.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., 1991. ISBN 0471062596.

M. Dayarathna, Y. Wen, and R. Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794, 2016. doi: 10.1109/COMST.2015.2481183.

J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI'04: Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 137–150, 2004.

J. Dongarra, P. Luszczek, and A. Petitet. The linpack benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2005. doi: 10.1002/cpe.994.

J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

Y. Gao and P. Zhang. A survey of homogeneous and heterogeneous system architectures in high performance computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 170–175, 2016. doi: 10.1109/SmartCloud.2016.36.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

I. J. Goodfellow et al. Generative adversarial nets. Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2014. URL https://arxiv.org/abs/1406.2661.

T. Helmy, S. Al-Azani, and O. Bin-Obaidellah. A machine learning-based approach to estimate the cpu-burst time for processes in the computational grids. 12 2015. doi: 10.1109/AIMS.2015.11.

J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2017.

T. Herault and Y. Robert. *Parallel Algorithms*. CRC Press, 2015.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

C.-H. Hsu and W.-C. Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC '05)*, pages 1–10. IEEE, 2005.

G. James et al. *Introduction to Statistical Learning: With Applications in R*. Springer, New York, 2013. ISBN: 978-1461471370.

G. Karypis and V. Kumar. A fast and high-quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3149–3157, 2017.

D. Kingma and M. Welling. Auto-encoding variational bayes. Proceedings of the International Conference on Learning Representations (ICLR), 2013. URL https://arxiv.org/abs/1312.6114.

D. B. Kirk and W. mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2016.

H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2011.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin-Cummings Publishing, 1994.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539.

M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the 8th Heterogeneous Computing Workshop*, pages 30–44, 1999.

M. Melnik and D. Nasonov. Workflow scheduling using neural networks and reinforcement learning. *Procedia Computer Science*, 156:29–36, 2019. doi: https://doi.org/10.1016/j.procs.2019.08.126. URL https://www.sciencedirect.com/science/article/pii/S1877050919310440. 8th International Young Scientists Conference on Computational Science, YSC2019, 24-28 June 2019, Heraklion, Greece.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

A. Netti. Hpc-oda dataset collection. Zenodo, 2021. Available at: https://zenodo.org/records/4671477.

T. Patel et al. Dataset from paper: What does power consumption behavior of hpc jobs reveal? Zenodo (CERN European Organization for Nuclear Research), February 2020. URL https://doi.org/10.5281/zenodo.3666632. Available at: https://doi.org/10.5281/zenodo.3666632.

A. K. Pawel Czarnul, Jerzy Proficz. Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments. *Scientific Programming*, 2019, 2019. doi: 10.1155/2019/8348791.

K. Pearson. Mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London A*, 187:253–318, 1896. doi: 10.1098/rsta.1896.0007.

M. L. Pinedo. *Scheduling*. Springer New York, NY, 2012. doi: https://doi.org/10.1007/978-1-4614-2361-4.

L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: Unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6639–6649, 2018.

A. Putera and U. Siahaan. Comparison analysis of cpu scheduling: Fcfs, sjf and round robin. *International Journal of Engineering Development and Research, vol. 4*, 2016.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

N. Roblex, T. Claude, D. Petr, and M. Youssef. Energy concerns with hpc systems and applications, 2023.

D. Rodriguez, D. Gomez, D. Alvarez, and S. Rivera. A review of parallel heterogeneous computing algorithms in power systems. *Algorithms*, 14(10), 2021. ISSN 1999-4893. doi: 10.3390/a14100275. URL https://www.mdpi.com/1999-4893/14/10/275.

B. Rountree, M. Schulz, and B. R. de Supinski. Practical performance-power profiling in hpc. In *Euro-Par 2012: Parallel Processing Workshops*. Springer, 2012.

P. Samal, S. Jha, and R. K. Goyal. Cpu burst-time estimation using machine learning. In *EEE Delhi Section Conference (DELCON)*, 2022. doi: 10.1109/DELCON54057.2022.9753639.

P. G. Shambharkar, S. Bhambri, and V. Kumar. Multiple resource management and burst time prediction using deep reinforcement learning. In *Proceedings of the 2019 International Conference*, 2019. URL https://api.semanticscholar.org/CorpusID:208002889.

S. Shankar and A. Reuther. Trends in energy estimates for computing in ai/machine learning accelerators, supercomputers, and compute-intensive applications. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, September 2022. doi: 10.1109/HPEC55821.2022.9926296. URL http://dx.doi.org/10.1109/HPEC55821.2022.9926296.

C. Shyalika, T. Silva, and A. Karunananda. Reinforcement learning in dynamic task scheduling: A review. *SN Computer Science*, 1:306, 2020. doi: 10.1007/s42979-020-00326-5.

A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Springer New York, NY, 10th edition, 2018. ISBN 978-1-119-32091-3.

C. Silva, R. Vilaça, A. Pereira, and R. Bessa. A review on the decarbonization of high-performance computing centers. *Renewable and Sustainable Energy Reviews*, 2024. doi: 10.1016/j.rser.2023.136403. URL https://www.sciencedirect.com/science/article/pii/S1364032123008778.

B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986. ISBN: 0412041812.

C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. doi: 10.2307/1412159.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, 2018. ISBN: 978-0262039246.

A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 3rd edition, 2007. URL https://www.pearson.com/store/p/modern-operating-systems/P100000039820.

Tarockey et al. Synthetic data generation. Microsoft.com, March 2024. URL https://learn.microsoft.com/en-us/ai/playbook/capabilities/experimentation/synthetic-data-generation. Available at: https://learn.microsoft.com/en-us/ai/playbook/capabilities/experimentation/synthetic-data-generation.

H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002. doi: 10.1109/71.993206.

M. P. Uddin, M. Sohrawordi, and M. I. Afjal. Developing operating system simulation software for windows based system by c# .net framework and an android application by java and xml. 2017.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

V. Weaver and J. Dongarra. Can hardware performance counters produce expected, deterministic results? In *3rd Workshop on Functionality of Hardware Performance Monitoring*, Atlanta, GA, December 4 2010.

S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Z. Xiao, S. Ma, and S. Zhang. Learning task allocation for multiple flows in multi-agent systems. In *2009 International Conference on Communication Software and Networks*, pages 153–157, 2009. doi: 10.1109/ICCSN.2009.22. URL https://api.semanticscholar.org/CorpusID:18633290.

Part III
Appendices

Appendix A
Details of results

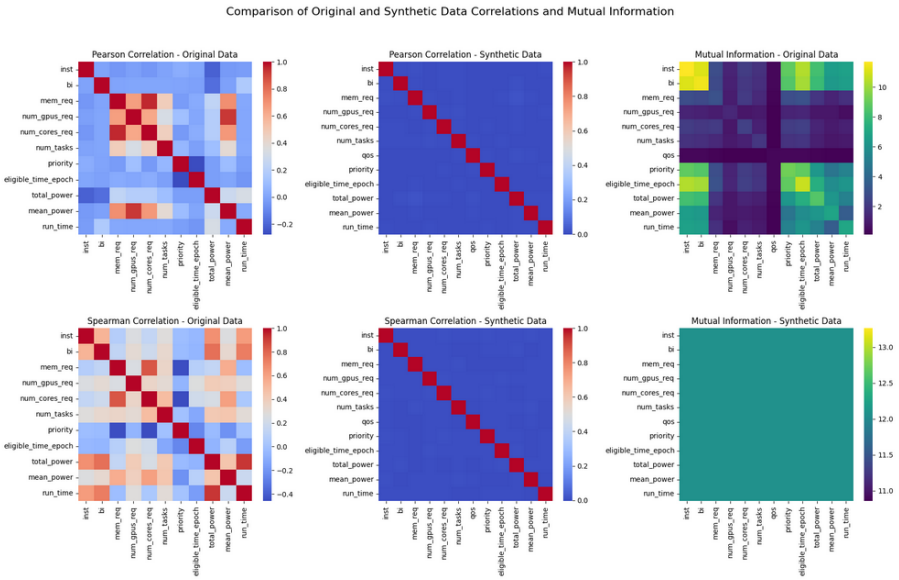


Figure 47: Data Augmented with KDE.

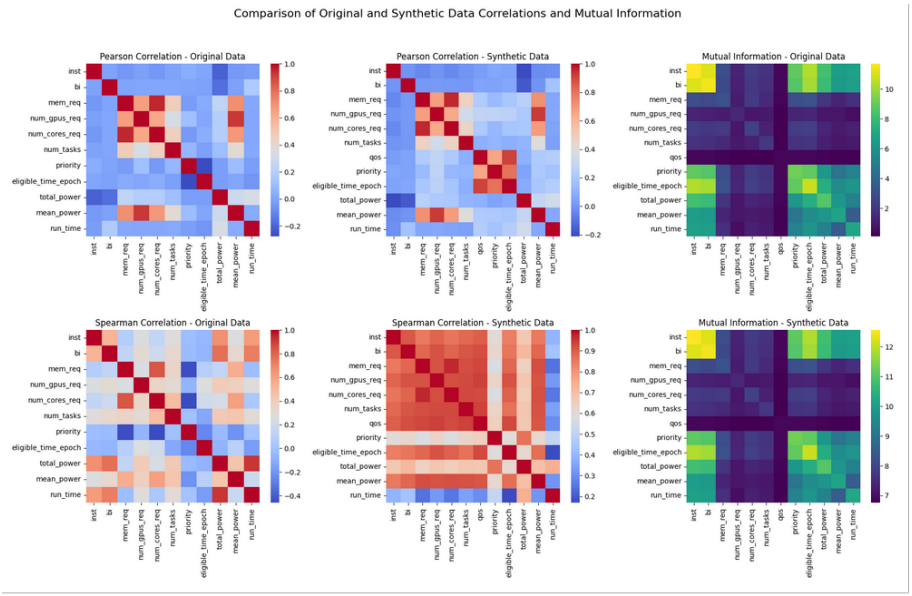


Figure 48: Data Augmented with GAN.

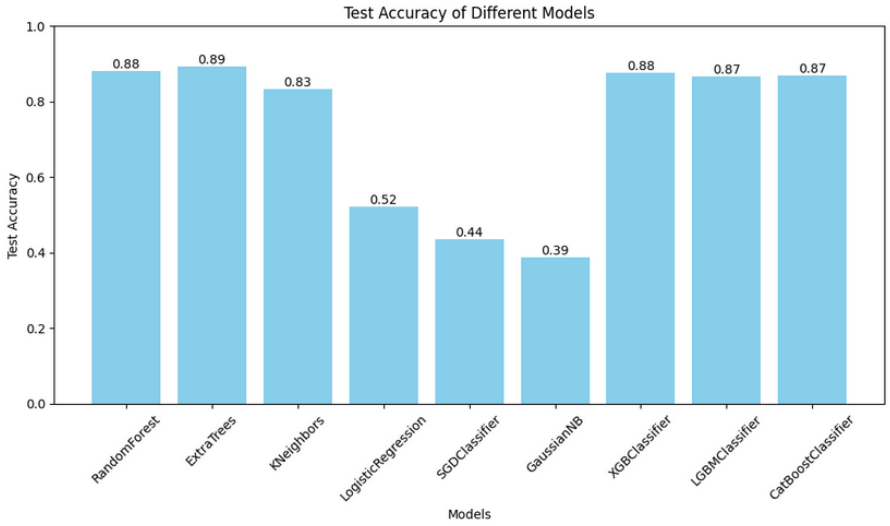


Figure 49: Classifier Models Performance in Multi-Classification of the run time data.

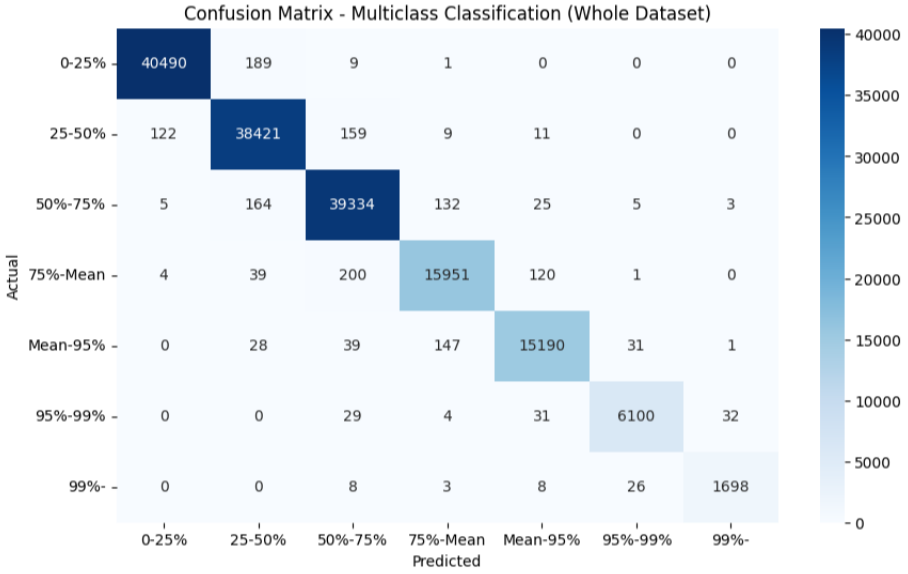
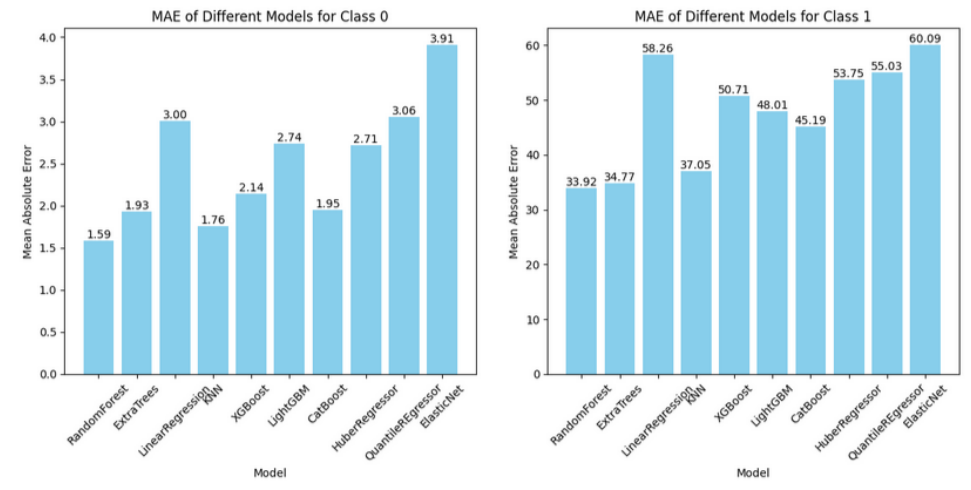


Figure 50: Stacking Multi-Classifier of run time data Confusion Matrix for the whole dataset.



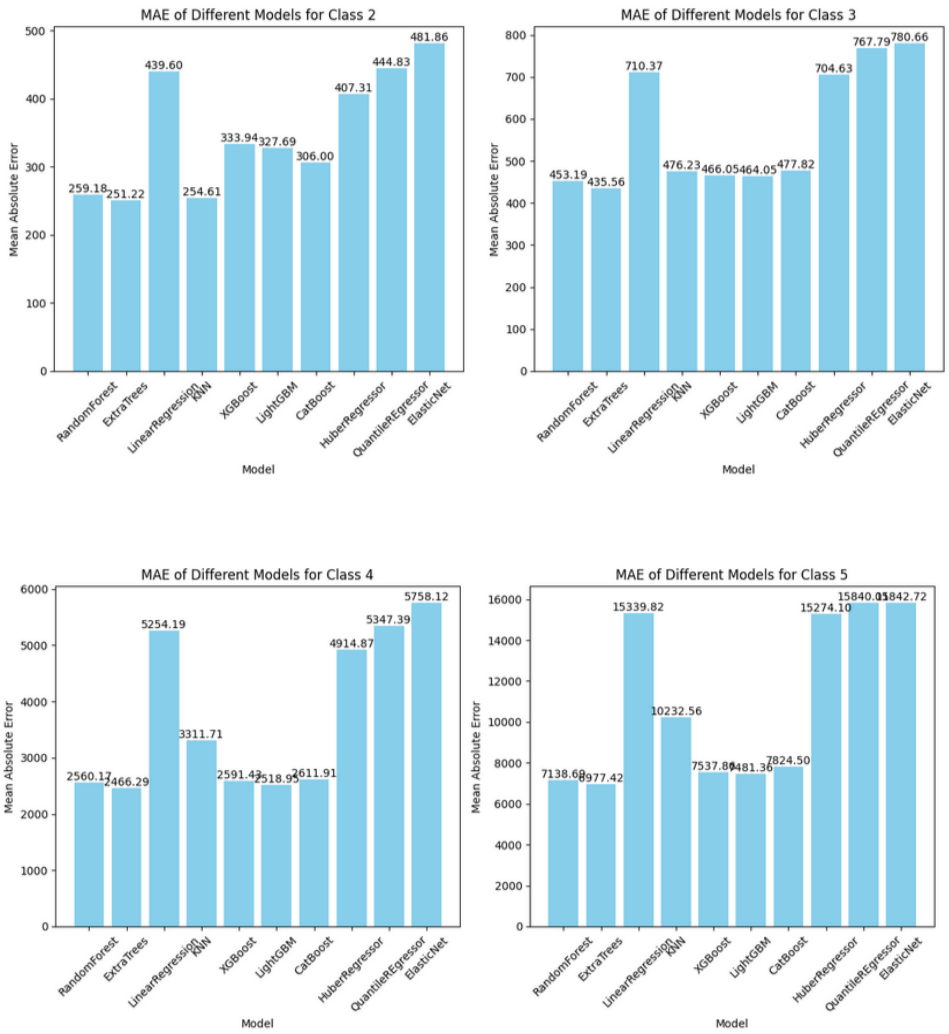


Figure 51: Regressor performance within each class of run time prediction.

Appendix B
Listings

Table 11: Summary of `ExtraTreesClassifier` Parameters used in 4

Parameter	**Value**
n_estimators	50
criterion	'gini')
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	False
oob_score	False
n_jobs	-1
random_state	42
verbose	0
warm_start	False
class_weight	{0: 1, 1: 100}
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

Table 12: Summary of `RandomForestRegressor` Parameters used for power prediction in 5.3

Parameter	**Value**
n_estimators	100
criterion	'squared_error'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'auto'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	-1
random_state	42
verbose	0
warm_start	False
ccp_alpha	0.0
max_samples	None

Table 13: Summary of `StackingClassifier` and Base Learners Parameters used in run time prediction 5.2.3.

Parameter	Value
Stacking Classifier	
final_estimator	LogisticRegression(C=1.0, max_iter=3000, solver='lbfgs', multi_class='multinomial')
Base Learner 1	ExtraTreesClassifier
n_estimators	500
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
bootstrap	False
random_state	42
Base Learner 2	SGDClassifier
max_iter	3000
tol	1e-4
alpha	1e-5
early_stopping	True
learning_rate	'adaptive'
eta0	0.01
loss	'modified_huber'
class_weight	'balanced'
n_iter_no_change	10
Base Learner 3	XGBClassifier
n_estimators	500
learning_rate	0.05
subsample	0.8
random_state	42

Table 14: Summary of `RandomForestRegressor` Parameters for Classes 0, 1 used in run time prediction 5.2.3

Parameter	Value
n_estimators	50
criterion	'squared_error'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'auto'
max_leaf_nodes	None
bootstrap	True
oob_score	False
n_jobs	-1
random_state	42
verbose	0

Table 15: `ExtraTreesRegressor` Parameters for Classes 2,3,4 and 5 used in run time prediction 5.2.3

Parameter	Value
n_estimators (Classes 3, 4)	200
n_estimators (Classes 2, 5)	500
criterion	'squared_error'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'auto'
bootstrap	False
random_state	42
n_jobs	-1

Table 16: Summary of `HuberRegressor` Parameters for Class 6 used in run time prediction 5.2.3.

Parameter	Value
epsilon	1.35
max_iter	100
alpha	0.0001
warm_start	False
fit_intercept	True
tol	1e-05

Appendix C
Tooling

Table 17: Versions of Software and Libraries

Library/Software	**Version**
Python	3.10.7
Jupyter Notebook	8.5.0
pandas	1.5.0
numpy	1.26.2
scikit-learn	1.4.2
sklearn.model_selection	1.4.2
sklearn.preprocessing	1.4.2
sklearn.metrics	1.4.2
sklearn.neighbors	1.4.2
sklearn.neural_network	1.4.2
sklearn.feature_selection	1.4.2
sklearn.ensemble	1.4.2
matplotlib	3.6.1
seaborn	0.12.0
joblib	1.2.0
xgboost	2.0.3
lightgbm	4.5.0
tensorflow	2.16.1
tensorflow.keras.models	2.16.1
tensorflow.keras.layers	2.16.1
tensorflow.keras.regularizers	2.16.1
tensorflow.keras.optimizers	2.16.1
tensorflow.keras.activations	2.16.1
scikeras	0.13.0
keras.callbacks	2.16.1
random	Built-in Python library
uuid	Built-in Python library
statistics	Built-in Python library
itertools	Built-in Python library
math	Built-in Python library
Overleaf	TeX 2024 compiler (XeLaTeX)
Draw.io	Web-based diagramming tool