ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Learning energy performance of parallel algorithms on an HPC infrastructure

Tesi di laurea magistrale in

Sistemi Operativi M

Relatore

Prof.ssa Anna Ciampolini

Correlatori

Prof.ssa Daniela Loreti

Ing. Marcello Artioli

Dott. Davide De Chiara

Presentata da

Michele Colonna

Sessione Febbraio 2024

Anno Accademico 2022/2023

Abstract

In recent years, the High-Performance Computing (HPC) community has placed a growing emphasis on energy efficiency, recognizing the escalating energy consumption of non-computational components within these systems. This shift has prompted discussions on environmental sustainability in the Information Technology (IT) sector, particularly in HPC, where the term "green" has become a recurring theme. This context sets the stage for the exploration of artificial intelligence (AI) as a key player in predicting and optimizing energy parameters within HPC systems. The coexistence of AI with traditional optimization techniques is highlighted as a promising approach. The thesis, with its central focus on two primary objectives, unfolds within this landscape. Firstly, the thesis aims to compile a detailed dataset encompassing the energy consumption of nodes in an HPC system, specifically tailored to various configurations of jobs executed on the system. Secondly, it seeks to assess the efficacy of an AI model in providing realistic predictions of energy consumption for unknown cases. The specific research focus hones in on parallel algorithms designed for solving linear systems, both in fault-tolerant and non-fault-tolerant versions. This choice is rooted in the widespread applications of linear systems across diverse sectors, adding practical relevance to the undertaken research.

Contents

Introduction **1**

1 Linear systems solver algorithms: IMe and ScaLAPACK **4**

1.1 The Inhibition Method (IMe) 4

1.1.1 The Inhibition Method with reduced memory footage 7

1.2 ScaLAPACK 9

1.3 Fault tolerance management 10

1.3.1 ScaLAPACK Diskless Checkpoining strategy 10

1.3.2 IMe Checksum strategy 11

2 Essential Machine Learning Concepts for Predictive Analysis **16**

2.1 Overview 16

2.2 Regression Models 18

2.2.1 Decision Tree 18

2.2.2 Random Forest 22

2.2.3 Gradient Boosting Decision Tree 23

2.3 Evaluation Metrics for Regression Models 24

2.3.1 Root Mean Squared Error (RMSE) 24

2.3.2 Mean Absolute Error (MAE) 26

2.3.3 Mean Absolute Percentage Error (MAPE) 28

2.4 Tuning the hyperparameters 29

3 Dataset Structure and Implementation 31

3.1 CRESCO6 Cluster Architecture 31

3.2 Dataset Structure 33

3.3 Dataset Implementation 34

3.3.1 Preparation phase for job launch 35

3.3.2 Job Launch Scripts 39

3.3.3 MPI-based Command Line "Tester" 43

3.3.4 Communication with the monitoring node 46

3.3.5 Job results statement 51

3.4 Configuration Parameters 52

4 Predictive Analysis 54

4.1 Data Exploration 54

4.1.1 Data Preprocessing 55

4.1.2 Data Description 56

4.1.3 Boxplots 57

4.1.4 Histograms target 59

4.2 Regressor Evaluation 61

4.3 Prediction error analysis 62

4.3.1 Random Forest 62

Conclusions 65

A Additional analyses for ideally numerically stable dataset 68

A.1 Prediction error analysis 69

A.1.1 Decision Tree 69

A.1.2 Gradient Boosting Decision Tree 70

B Dataset analysis with real numerical stability conditions **72**

B.1 Data Exploration 72

B.1.1 Data Preprocessing 72

B.1.2 Data Description 73

B.1.3 Boxplots 73

B.1.4 Histograms target 74

B.2 Regressors Evaluation 75

B.3 Prediction error analysis 76

B.3.1 Decision Tree 76

B.3.2 Random Forest 77

B.3.3 Gradient Boosting Decision Tree 79

Bibliography **83**

List of Figures **85**

List of Tables **86**

Index **87**

Acknowledgements **87**

Introduction

In recent years, the High-Performance Computing (HPC) community has increasingly emphasized the importance of developing energy-efficient HPC systems. The focus has been on improving both the energy efficiency of computation and the energy efficiencies related to other system components, such as memory or disks. The energy consumption of non-computational components within an HPC system constantly represents an increasing percentage of the overall energy consumption.

Numerous articles have addressed and continue to address the prevailing topic of environmental sustainability, especially in the field of Information Technology (IT) and, more specifically, in areas such as HPC [1], [2]. The term *"green"* has become a recurring theme in discussions on eco-friendly practices in the IT sector.

Top500.org [3] stands out as the website that regularly updates a classification of the most powerful supercomputers globally, providing details on hardware specifications, the organization that owns or uses them, their geographical location, and more. Published twice a year, this ranking, known as the "Top500", lists supercomputers based on their computing capacity, measured in teraflops per second (trillions of floating-point operations per second). Widely used as a benchmark, it assesses the computing capabilities of the world's most advanced machines. Alongside this ranking, a second list has been published since 2008, monitoring the energy consumption of supercomputers. Dubbed the "Green500", it displays results using the GFlops/watts metric for these supercomputers, includ-

ing their peak performance number (for peak power efficiency) and their Linpack performance number (for actual power efficiency).

In nearly all fields, a crucial and increasingly pervasive element is the use of artificial intelligence. Its significant evolution in recent years has opened the door to new possibilities, including the prediction and estimation of energy parameters within systems such as HPC [4]. This approach not only provides advanced projections of energy consumption but also identifies key factors influencing its variations. Such a perspective allows for in-depth analysis, offering concrete insights on how to optimize system performance and achieve energy savings.

Utilizing artificial intelligence in this context is seen as an additional methodology, coexisting with traditional techniques for optimizing computation and improving the energy efficiency of individual system components. This synergy between artificial intelligence and more conventional approaches could represent a significant advancement in the quest for more effective solutions for the overall optimization of high-performance systems.

This thesis aims to achieve several objectives, focusing primarily on two fundamental aspects. Firstly, the goal is to create an extensive dataset containing detailed information on the energy consumption of nodes in a High-Performance Computing system, specifically for each configuration of jobs executed on such a system. Secondly, the intention is to assess the effectiveness of an artificial intelligence model in providing realistic predictions of energy consumption of unkonwn cases.

The specific focus of this thesis revolves around parallel algorithms designed for solving linear systems, presented in both fault-tolerant and non-fault-tolerant versions. This choice stems from the inherent significance of linear systems as a mathematical model with wide-ranging applications across various sectors, including computer protocols, image processing (computer vision), sound manipulation,

building design, railway systems, and other fields.

The emphasis placed on linear systems in addressing real-world problems adds pertinence and practicality to the research undertaken in this thesis. It contributes to the development of predictive models capable of anticipating energy performance in scenarios involving the utilization of HPC systems. This approach aims to proactively intervene to enhance energy efficiency during the operation of such systems.

Thesis outline

The work is structured into several chapters, each dedicated to a specific phase of the research. In Chapter 1, the resolution of linear systems using the IMe (Inhibition Method) and ScaLAPACK algorithms, which will be the two algorithms in the dataset, is presented in detail. In addition to explaining how these algorithms work, strategies for handling potential failures in a high-performance computing (HPC) environment are explored.

Chapter 2 provides the essential fundamentals for predictive data analysis, examining the chosen Machine Learning models, accuracy evaluation metrics, and hyperparameter tuning techniques.

In Chapter 3, the structure of the dataset is described, outlining the steps to prepare the necessary conditions for job execution. Measurement mechanisms, chosen configurations, and the implementation of a second equivalent dataset characterized by more realistic numerical stability, achieved through the use of non-ideal condition number matrices, are detailed.

Finally, Chapter 4 presents the results of predictive analysis based on the IMe and ScaLAPACK algorithms, using the criteria outlined in Chapter 2. Any less relevant graphs from the first dataset are discussed in Appendix A, while Appendix B illustrates the graphs derived from the predictive analysis of the second dataset.

Chapter 1

Linear systems solver algorithms: IMe and ScaLAPACK

Before presenting the main activity of this thesis, it is necessary to set out the concepts behind it. In particular, this section will present the two linear system resolution algorithms used and their parallel implementation to be adopted on HPC systems. In addition, the respective fault tolerance management and recovery methods will be described.

1.1 The Inhibition Method (IMe)

IMe represents a precise approach grounded in the Effect Superposition Theorem, initially formulated in 1963 [5] for the examination of complex linear electric circuits. Subsequently, it found application in analyzing broader scenarios, encompassing physical linear systems [6] and square matrix inversion [7]. Given its original focus on accuracy rather than efficiency, this method involves additional computations compared to more widely adopted algorithms like Gauss-Jordan Elimination (GJE) when tackling linear systems.

The original technique solves a linear system of type $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ is the matrix of $n \times n$ coefficients, $\boldsymbol{b}$ is the vector of constant terms and $\boldsymbol{x}$ is the vector representing the unknowns. Based on this specification, the initial step involves the construction of a matrix $\boldsymbol{T}^{(n)}$, referred to as the *inhibition table*, calculated as follows:

$$T^{(n)} = \left[\begin{array}{ccccc|ccccc} \frac{1}{a_{1,1}} & 0 & \dots & \dots & 0 & 1 & \frac{a_{2,1}}{a_{1,1}} & \dots & \dots & \frac{a_{n,1}}{a_{1,1}} \\ 0 & \frac{1}{a_{2,2}} & \dots & \dots & 0 & \frac{a_{1,2}}{a_{2,2}} & 1 & \dots & \dots & \frac{a_{n,2}}{a_{2,2}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & \dots & \frac{1}{a_{n-1,n-1}} & 0 & \vdots & \vdots & \dots & 1 & \frac{a_{n,n-1}}{a_{n-1,n-1}} \\ 0 & \dots & \dots & 0 & \frac{1}{a_{n,n}} & \frac{a_{1,n}}{a_{n,n}} & \dots & \dots & \frac{a_{n-1,n}}{a_{n,n}} & 1 \end{array}\right]$$

where $a_{i,j}$ are the elements of $\boldsymbol{A}$. The matrix $\boldsymbol{T}^{(n)}$ is a decomposition of the original problem into n sub-problems (one for each row).

IMe employs a *foundamental formula* to progressively decrease the dimensions of $\boldsymbol{T}^{(n)}$ by one row and one column through iterative steps, so that at each step (usually called *level*) l (with $l = n \dots 1$), $\boldsymbol{T}^{(l)}$ represents the original problem decomposed into l sub-problems:

$$t_{i,j}^{(l-1)} = (t_{i,j}^{(l)} - t_{l,j}^{(\iota)} \cdot t_{i,n+l}^{(\iota)}) \cdot h_i^{(l)}, l = n \dots 1,\ i = 1 \dots l-1,\ j = 1 \dots n+l-1$$

where $h_i^{(l)}$ is the i^{th} element of the *auxiliary vector* $\boldsymbol{h}^{(l)}$, which must be recomputed at each level as well:

$$h_i^{(l)} = \frac{1}{1 - t_{l,n+i}^{(l)} \cdot t_{i,n+l}^{(l)}}, \quad l = n \dots 2, i = 1 \dots l-1.$$

Figure 1.1 shows graphically what has been said so far. At each level, the element

$(t_{i,j}^{(l-1)})$ of the inhibition table is recalculated using its previous value $(t_{i,j}^{(l)})$, the elements of the last row $(t_{l,j}^{(l)})$ and column $(t_{i,n+l}^{(l)})$ of the previous inhibition table and the corresponding auxiliary quantity $h_i^{(l)}$. For the calculation of $\boldsymbol{h}^{(l)}$, only the elements in the last row $(t_{l,n+i}^{(l)})$and column $(t_{i,n+l}^{(l)})$ of the inhibition table are required.

To calculate the solution of the system, the vector of constant terms $\boldsymbol{b}$ and the solution vector $\boldsymbol{x}$ are initialised and updated at each level as shown in Table 1.1. When all iterations are completed (i.e. at level $l = 1$), the matrix $\boldsymbol{T}^{(1)}$ has only one row and n columns. The vector $\boldsymbol{x}^{(1)}$ houses the solution of the linear system.

	Initialization	**Update**
$\boldsymbol{b}$	$b_i^{(n)} = \begin{cases} b_n & i = n \\ b_i - t_{n,n+i}^{(n)} b_n & o/w \end{cases}$	$b_i^{(l-1)} = b_i^{(l)} - t_{l-1,n+i}^{(l-1)} b_l^{(l)},\ i = 1 \dots l-2$
$\boldsymbol{x}$	$x_i^{(n)} = \begin{cases} t_{n,i}^{(n)} \cdot b_n & i = n \\ 0 & o/w \end{cases}$	$x_i^{(l-1)} = x_i^{(l)} - t_{l-1,i}^{(l-1)} b_l^{(l)},\ i = l-1 \dots n$

Table 1.1: IMe prescribed steps to compute the system's solution.

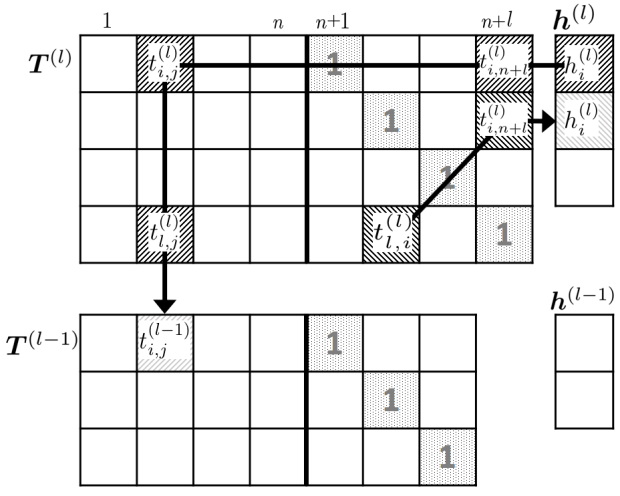


Figure 1.1: Graphical visualization of the fundamental formula and the computation of the auxiliary quantities. (*Source:* [8])

1.1.1 The Inhibition Method with reduced memory footage

As shown in a later study [9], the Inhibition Table $\boldsymbol{T}^{(n)}$ can also be represented in the form of two matrices: $\boldsymbol{E}^{(n)}$ (matrix of *effects*) and $\boldsymbol{K}^{(n)}$ (matrix of *inhibition quantities*), and the vector $\boldsymbol{h}^{(n)}$ was converted to $\boldsymbol{\alpha}^{n}$.

$\boldsymbol{E^n}$

$\frac{1}{a_{1,1}}$	0	$\cdots$	$\cdots$	0
0	$\frac{1}{a_{2,2}}$	$\cdots$	$\cdots$	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
0	$\cdots$	$\cdots$	$\ddots$	0
0	$\cdots$	$\cdots$	0	$\frac{1}{a_{n,n}}$

$\boldsymbol{K^n}$

1	$\frac{a_{2,1}}{a_{1,1}}$	$\cdots$	$\cdots$	$\frac{a_{n,1}}{a_{1,1}}$
$\frac{a_{1,2}}{a_{2,2}}$	1	$\cdots$	$\cdots$	$\frac{a_{n,2}}{a_{2,2}}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	1	$\vdots$
$\frac{a_{1,n}}{a_{n,n}}$	$\cdots$	$\cdots$	$\frac{a_{n-1,n}}{a_{n,n}}$	1

$\boldsymbol{\alpha^n}$

$\frac{1}{1-k_{n,1}k_{1,n}}$
$\vdots$
$\vdots$
1
$1-k_{n,n-1}k_{n-1,n}$

Figure 1.2: Inhibition Table alternative form.(*Source:* [9])

It was also illustrated in [9] that, to reduce the memory space occupied, the matrix can be compressed by taking the $\boldsymbol{K}^{(n)}$ matrix alone and replacing its diagonal with that of the $\boldsymbol{E}^{(n)}$ matrix, generating a single matrix $\boldsymbol{V}^{(n)}$ (Figure 1.3). According to this change, we initialise each $v_{ij}^{(n)}$ element of $\boldsymbol{V}^{(n)}$ as follows:

$$v_{ij}^{(n)} = \begin{cases} \frac{1}{a_{ij}} & \text{if } j = i \\ \frac{a_{ji}}{a_{ii}} & \text{otherwise} \end{cases}$$

with $i, j = 1 \dots n$. The fundamental formula becomes:

$$v_{ij}^{(l-1)} = \alpha_i^{(l)} \cdot \begin{cases} v_{ii}^{(l)}, & \text{if } j = i \\ -v_{il}^{(l)} v_{ll}^{(l)}, & \text{if } j = l \\ v_{ij}^{(l)} - v_{il}^{(l)} v_{lj}^{(l)} & \text{otherwise} \end{cases}$$

with $l = n \dots 2$, $i = 1 \dots l-1$, and $j = 1 \dots n$. To compute the solution we apply the following for all levels $l = n \dots 2$:

$$b_i^{(l-1)} = b_i^{(l)} - v_{l-1,i}^{(l-1)} \cdot b_l^{(l)}, \qquad i = 1 \dots l-2$$

$$x_i^{(l-1)} = x_i^{(l)} - v_{l-1,i}^{(l-1)} \cdot b_l^{(l)}, \qquad i = l-1 \dots n$$

The algorithm used in this thesis to generate configurations with the Inhibition Method is the latter.

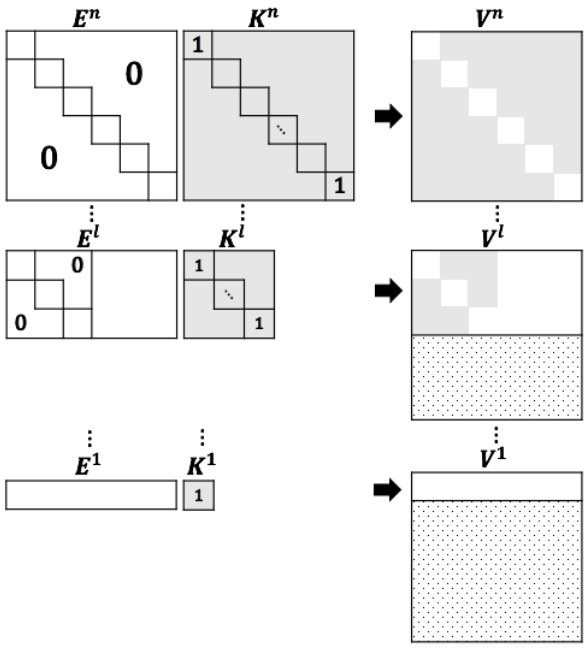


Figure 1.3: Evolution of IMe data structures when compressing $\boldsymbol{E}$ and $\boldsymbol{K}$ into a single matrix $\boldsymbol{V}$. (*Source:* [9])

1.2 ScaLAPACK

ScaLAPACK (Scalable Linear Algebra PACKage) is a high-performance linear algebra library designed for parallel distributed memory machines. It addresses dense and banded linear systems, least squares problems, eigenvalue problems, and singular value problems. ScaLAPACK incorporates key features such as block cyclic data distribution for dense matrices and block data distribution for banded matrices, which can be customized at runtime. It utilizes block-partitioned algorithms to maximize data reuse and includes well-designed low-level modular components, simplifying the parallelization of high-level routines by maintaining consistency with their sequential counterparts.

The objectives of the ScaLAPACK project align with those of LAPACK, focusing on efficiency (aiming for optimal speed), scalability (accommodating growing problem sizes and processor numbers), reliability (including error bounds), portability (across major parallel machines), flexibility (allowing users to create new routines from well-designed components), and ease of use (ensuring a similar interface to LAPACK and ScaLAPACK).

To achieve these goals, the project emphasizes the importance of standards, particularly for low-level communication and computation routines. Notably, ScaLAPACK minimizes machine dependencies by relying on three standard libraries: BLAS (Basic Linear Algebra Subprograms), LAPACK, and BLACS (Basic Linear Algebra Communication Subprograms). LAPACK can run on any machine with available BLAS, and ScaLAPACK is compatible with machines where BLAS, LAPACK, and BLACS are accessible. The library is currently written in Fortran (with the exception of a few symmetric eigenproblem auxiliary routines written in C).

1.3 Fault tolerance management

In the event of executing the parallel implementation of these algorithms on HPC infrastructure for solving a large-scale linear system, it is crucial to consider that the failure of even a single computing processor inevitably disrupts the entire computation. The results obtained from the fault-tolerant versions of these techniques will also be taken into account when creating the datasets in this thesis. Although ScaLAPACK does not have inherent fault-tolerant mechanisms, routines can be modified relatively easily to incorporate the classic fault-tolerant mechanism know as Diskless C/R (Checkpointing/Rollback). Instead IMe one uses a more effective strategy based on a checksum mechanism.

1.3.1 ScaLAPACK Diskless Checkpoining strategy

The Checkpointing is a critical aspect of fault-tolerant computing and forms the basis for rollback recovery. In the event of a failure on the machine running a prolonged computation, without checkpointing, users would need to restart the program from the beginning upon machine recovery, resulting in the loss of all prior computation. However, by periodically saving checkpoints of the program's state to stable storage, users have the option to restart the program from the most recent checkpoint in case of a failure. This process, referred to as rolling back to a stored checkpoint, proves invaluable for long-running computations, allowing users to minimize the amount of lost computation in the event of failures. There are two different ways to make checkpointing: to a stable storage (i.e. disk) or diskless. The second one reduce the overhead removing stable storage and replace it with memory and processor redundancy.

Diskless checkpointing [10] involves two main components:

1) Saving the state of each application processor in memory.

2) Encoding these in-memory checkpoints and storing the encodings in checkpointing processors.

In the event of a failure, system recovery unfolds as follows. Firstly, the non-failed application processors revert to their stored checkpoints in memory. Following this, replacement processors are selected to substitute the failed processors. Ultimately, the replacement processors utilize the checkpointed states of the non-failed application processors, in addition to the encodings stored in the checkpointing processors, to compute the checkpoints of the failed processors. Once these checkpoints are determined, the replacement processors roll back, and the application resumes from the checkpoint.
Plank et al. [11] embedded diskless checkpointing into several matrix operations in the ScaLAPACK distributed linear algebra package, thus making them resilient to single processor failures with low Overhead. Kim et al. [12] extended this work to employ onedimensional parity encoding, which both lowers the overhead and increases the failure coverage.

1.3.2 IMe Checksum strategy

When implementing IMe in parallel mode on an HPC infrastructure to cope with a large linear system, one must consider that a failure affecting even one computing processor inevitably interrupts the entire computation. A simpler and more efficient approach, as outlined in [12], replaces the C/R technique.

For the first formulation of IMe this method involves parallelization by columns and the computation of the sum by rows for all entries of $\boldsymbol{T}^{(l)}$. This results in a checksum vector, denoted as $\boldsymbol{s}^{(l)}$, where each entry is determined through the

following calculation:

$$s_i^{(l)} = \sum_{k=1}^{n+l} t_{i,k}^{(l)} \quad \forall i = 1 \dots l \tag{1}$$

$\boldsymbol{s}^{(l)}$ is stored in the memory of an additional computing processor. In the event of a fault affecting a column of $\boldsymbol{T}^{(l)}$, the entries can be recovered by solving n linear equations. An important characteristic of this technique is that when applying the fundamental formula to $\boldsymbol{s}^{(l)}$, it produces a vector $\boldsymbol{s}^{(l-1)}$, which serves as the checksum vector for $\boldsymbol{T}^{(l-1)}$ once again. This property allows for the calculation of checksums via Eq. (1) only in the initial instance $\boldsymbol{T}^{(n)}$. The processor housing $\boldsymbol{s}^{(l)}$ can then apply the fundamental formula to it, akin to how other processors handle the columns of $\boldsymbol{T}^{(l)}$, persisting as the host for checksums of $\boldsymbol{T}^{(l)}$ for any subsequent level.

This is the case where there is at most a single fault. In the case of multiple faults [8], this technique is extended by appropriately defining m checksum columns. We therefore extend the inhibition table with the checksum matrix $\boldsymbol{S}^{(l)}$ as follows:

$$\Phi^{(l)} = \left[\boldsymbol{S}^{(l)} \mid \boldsymbol{T}^{(l)}\right] = \left[\begin{array}{ccc|ccc} s_{1,1} & \dots & s_{1,m} & t_{1,1} & \cdots & t_{1,n+l} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ s_{n,1} & \cdots & s_{n,m} & t_{n,1} & \cdots & t_{n,n+l} \end{array}\right]$$

According to [13], if the checksum matrix is computed as

$$S^{(l)} = T^{(l)} \cdot \Lambda^{(l)},$$

where $\boldsymbol{\Lambda}^{(l)}$ is a $(n+l) \times m$ matrix suitably conceived for the purpose (such that any square sub-matrix of $\boldsymbol{\Lambda}^{(l)}$ is nonsingular), then in case of a fail-stop involving any m columns of $\boldsymbol{\Phi}^{(l)}$, their values can be recomputed by solving a linear system

for each of its l rows as follows

$$\begin{cases} s_{i,1}^{(l)} = \sum_{k=1}^{n+l} \lambda_{k,1} t_{i,k}^{(l)} & \\ \quad \vdots \qquad\qquad \vdots & \forall i = 1 \dots l \\ s_{i,m}^{(l)} = \sum_{k=1}^{n+l} \lambda_{k,m} t_{i,k}^{(l)} & \end{cases}$$

Every such system has m equations and m unknowns (corresponding to the elements of the failed columns). The important property previously described for the single checksum column remains valid and is extended to the entire checksum matrix, as described in [14]. Finally, in the case of the Inhibition Method with memory optimisation, multiple fault tolerance is adapted as follows [9]: to formulate a versatile algorithm suitable for interconnected nodes, the adoption of a 2D-block cyclic distribution for the initial matrix $\boldsymbol{V}^{(n)}$ was proposed, as shown in Figure 1.4. This distribution effectively partitions $\boldsymbol{V}^{(n)}$ into $N \times N$ blocks, distributing them across a process grid composed of $P \times Q$ processors, with each processor receiving a maximum of $K = \left\lceil \frac{N}{P} \right\rceil \times \left\lceil \frac{N}{Q} \right\rceil$ blocks.

Moreover, in alignment with the original algorithm [14], the row-wise weighted sums computed during initialization must be executed on the matrix $\tilde{\boldsymbol{V}}^{(n)} = \boldsymbol{V}^{(n)} + \boldsymbol{I}$. This adjustment takes into consideration the 1-entries on the diagonal of $\boldsymbol{K}^{(n)}$, which may be overwritten by compression. Subsequently, $\boldsymbol{S}^{(n)}$ undergoes partitioning into distinct blocks:

$$S_{k,pr}^{(n)} = \sum_{q \in Q} \omega_{qr} \tilde{V}_{k,pq}^{(n)}$$

where $S_{k,pr}^{(n)}$ denotes the k^{th} block of the checksum matrix $\boldsymbol{S}^{(n)}$ held by the recovery processor (p, r), where $1 \leq k \leq K$ and $r \in R$. Similarly, $\tilde{V}_{k,pq}^{(n)}$ represents the

k^{th} block of the modified matrix $\tilde{\boldsymbol{V}}^{(n)}$ stored on the computing processor (p, q), excluding the identity matrix. Additionally, w_{qr} stands for the entry at the intersection of row q and column r in a weight matrix $\boldsymbol{W}$. Notably, for the recovery of multiple faults, the checksums must be weighted using a $Q \times R$ matrix, ensuring that any square sub-matrix of W is non-singular [13]. All blocks $S^{(n)}_{k,pr}$ consist of $n_b \times n_b$ elements, and each recovery processor can host a maximum of K blocks, mirroring the capacity of any other computing processor.

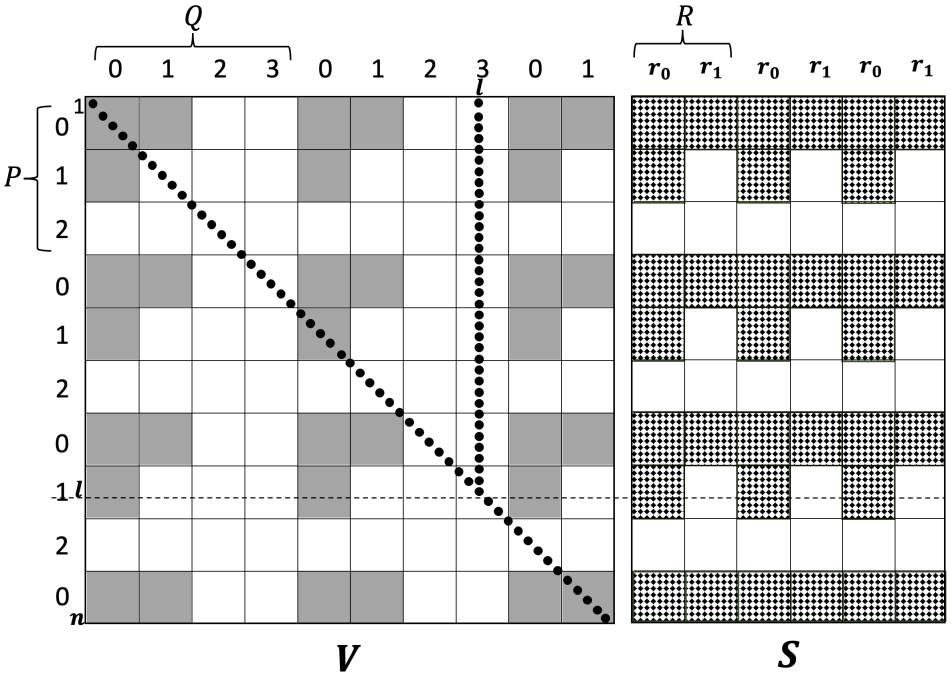


Figure 1.4: 2D block-cyclic distribution of $\boldsymbol{V}$ and $\boldsymbol{S}$ on a 3×4 and 3×2 processor grid, respectively. (*Source:* [9])

Implementing a fault-tolerant system for IMe offers several advantages over the Checkpoint/Restart (C/R) mechanism used in ScaLAPACK. The integration of fault resilience has minimal impact on floating-point operations (flops), memory usage, message count, and volume compared to the non-fault-tolerant implementation.

Traditional C/R mechanisms involve saving the computation state at predetermined intervals. In the event of an error, a rollback to the last checkpoint is necessary. In contrast, IMe employs a continuous checksum at each level, eliminating the need to periodically interrupt the computation for state encoding. Additionally, no extra communication is required among the computational processors.

These considerations are among the reasons why a dataset comparing and analyse the energy performance of these two algorithms was selected.

Chapter 2

Essential Machine Learning Concepts for Predictive Analysis

In this chapter, the focus shifts to key concepts used for predictive analysis after a brief introduction to Machine Learning and its application, with a specific emphasis on Regression. The discussion will center around various regressors and their parameters, providing insights into their predictive capabilities.

2.1 Overview

Machine learning is a prominent field in contemporary computing. Extensive research has been conducted to provide machines with intelligence. Learning, a natural human behavior, has been integrated as an essential aspect of machines as well. Various techniques have been devised for this purpose, with traditional machine learning algorithms finding application in numerous domains. Researchers have invested considerable efforts in enhancing the accuracy of these machine learning algorithms.

Shinde and Shah [15] identified several application domains and subdomains of

machine learning such as computer vision, prediction, semantic analysis, natural language processing and information retrieval.

In this thesis, we explore the application of machine learning in the context of predicting and retrieving information related to the energy analysis of consumption associated with linear system solving algorithms in the High Performance Computing (HPC) environment. A topic of great relevance is energy savings, especially considering the wide use of technology, which involves considerable energy consumption. The aim is to acquire and predict energetic information prior to execution, thus enabling efficient utilisation of such technologies. Regression is coming into play for this purpose.

In its most basic form, **Regression Analysis** allows one to examine the connections between one or more independent variables and a dependent variable.This analysis offers unique information that other techniques often fail to provide.
The main advantages of using regression analysis are as follows:

1) **Determining** whether the independent variables have a **significant correlation** with the dependent variable.

2) **Assess the relative strength of the effects** of the different independent variables on the dependent variable.

3) **Make predictions**.

The exploration of energy analysis will involve the application of sophisticated regression models [16], such as the Decision Tree Regressor, Random Forest Regressor, and Gradient Boosting Decision Tree Regressor. Each of these models will be extensively examined in subsequent sections, highlighting their distinctive features and specific applications in the prediction.

2.2 Regression Models

This section will present the three regression models used to conduct energy performance analyses of the data collected on linear system solvers run on HPC systems: the Decision Tree, the Random Forest, and the Gradient Boosting Decision Tree.

2.2.1 Decision Tree

Decision Trees (DTs) are a form of non-parametric supervised learning method applicable to both classification and regression tasks. Their primary objective is to develop a predictive model for a target variable by deriving uncomplicated decision rules from the dataset's features. In a conceptual sense, envisioning a tree as an approximation of the target variable through segmented, constant pieces enhances understanding.

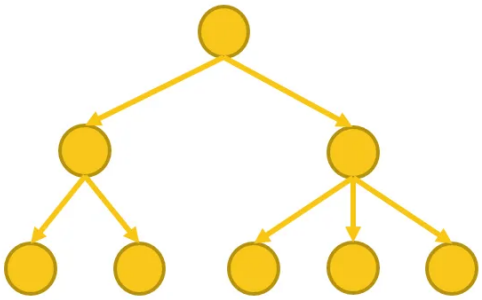


Figure 2.1: Single Decision Tree scheme (*Source:* [17])

This model features a hierarchical tree structure comprising a root node, branches, internal nodes, and leaf nodes. As depicted in Figure 2.1, a decision tree begins with a root node devoid of any incoming branches. The outgoing branches from the root node then flow into internal nodes, also referred to as decision nodes. Both node types, based on available features, conduct evaluations to create homogeneous subsets indicated by leaf nodes or terminal nodes. The leaf nodes encompass all potential outcomes within the dataset.

Advantages of Decision Trees

Decision trees offer several advantages:

1) **Simplicity and Interpretability:** Decision trees are easy to understand and visualize, providing a clear representation of decision-making processes.

2) **Minimal Data Preparation:** Unlike some methods, decision trees require less data preprocessing, handling both numerical and categorical data.

3) **Logarithmic Cost for Predictions:** The cost of predicting data using a decision tree is logarithmic in the number of data points used for training.

4) **Multi-output Capability:** Decision trees can handle problems with multiple outputs.

5) **White Box Model:** Decision trees offer transparency, providing explanations for conditions through easily understandable boolean logic.

6) **Statistical Validation:** Models can be validated using statistical tests, allowing for an assessment of reliability.

7) **Robustness:** Decision trees can perform well even when assumptions about the data's true model are somewhat violated.

Disadvantages of Decision Trees:

However, decision trees have their drawbacks:

1. **Overfitting:** Decision trees can create overly complex models that don't generalize well, known as overfitting. Techniques like pruning and setting tree parameters help mitigate this issue.

2. **Instability:** Small variations in data may lead to entirely different trees.

3. **Non-Smooth Predictions:** Predictions from decision trees are piecewise constant approximations, making them unsuitable for smooth or continuous extrapolation.

4. **Computational Complexity:** Learning an optimal decision tree is NP-complete, leading to heuristic algorithms like the greedy algorithm.

5. **Challenges in Expressing Complex Concepts:** Decision trees struggle to express certain complex concepts, such as XOR, parity, or multiplexer problems.

6. **Bias in Class-Dominated Datasets:** Decision tree learners may create biased trees when certain classes dominate the dataset. It is advisable to balance the dataset before fitting the decision tree to address this issue.

Decision Tree Regression

Shifting the focus to a specific application, consider a 1D regression scenario employing a decision tree. In this context, the decision tree is utilized to fit a sine curve while accommodating noisy observations. Consequently, the tree learns local linear regressions to approximate the sine curve.

The depth of the tree is linked to the complexity of these rules, resulting in a more sophisticated model as the tree deepens. It is noteworthy that when setting the maximum depth of the tree (regulated by the `max_depth` parameter) excessively high, the decision tree captures fine details of the training data, including noise, leading to overfitting. This emphasizes the importance of controlling the model's complexity to avoid an overly detailed representation that may not generalize well beyond the training data.

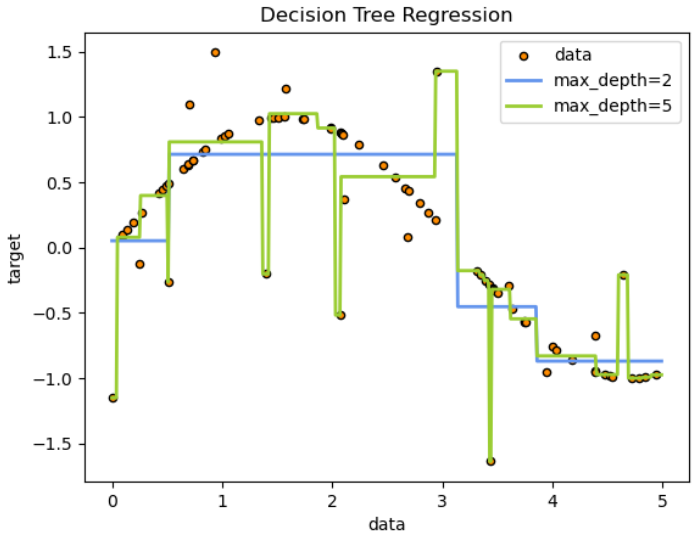


Figure 2.2: Example of a regression tree diagram (*Source:* [18])

2.2.2 Random Forest

Random Forests constitute an ensemble of tree predictors, with each tree relying on values sampled randomly and independently from a common distribution. As the forest size grows, the generalization error consistently converges to a limit. In the context of a forest comprising tree classifiers, the generalization error is influenced by both the individual strength of each tree and the level of correlation among them.

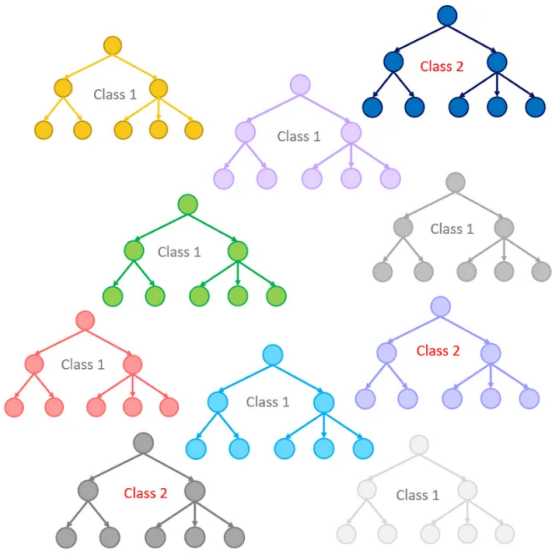


Figure 2.3: Random Forest scheme (*Source:* [17])

2.2.3 Gradient Boosting Decision Tree

Gradient Boosting Decision Trees is a machine learning algorithm that sequentially constructs a series of decision trees, each correcting the residual errors of the previous model. Initially, a "weak" decision tree is built and used for predictions. Residual errors are calculated, and a new tree is constructed to correct these errors. This process is iteratively repeated, with each new tree focusing on the residual errors of the combined model up to that point. Trees are assigned different weights based on their accuracy, and ultimately, predictions from all trees are combined to obtain the final result. This incremental approach helps create a more robust and accurate model.

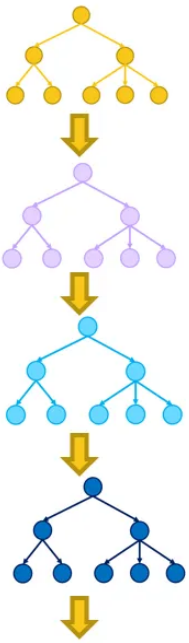


Figure 2.4: Gradient Boosting Decision Tree scheme (*Source:* [17])

In Figure 2.4, the yellow tree is equivalent to the "weak" tree. The residual errors are then calculated and the next purple-colored tree is constructed, which corrects those errors. Again, the blue-colored tree is created, which corrects the errors in the purple tree, and so on.

2.3 Evaluation Metrics for Regression Models

In the context of analyzing applied machine learning models, it is crucial to understand and employ appropriate evaluation metrics to assess the accuracy and performance of regression models. Metrics play a fundamental role in providing a quantitative measure of the models' effectiveness in predicting continuous values. Among the widely used metrics, the Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) stand out . These metrics offer a detailed insight into the discrepancy between the values predicted by the model and the actual values, allowing for a comprehensive understanding of prediction quality. In the subsequent chapters, we will apply these metrics to our analysis, focusing on their implications and contributions to our evaluation. In this section, our emphasis will be on explaining these metrics and their significance in the broader context of regression model assessment.

2.3.1 Root Mean Squared Error (RMSE)

The **Root Mean Squared Error (RMSE)** stands as a widely adopted metric in the realms of machine learning and statistics, serving as a measure of precision for predictive models. This metric calculates the disparities between predicted and actual values by squaring the errors, averaging them, and subsequently taking the square root. RMSE offers a transparent evaluation of a model's performance, where lower values signify enhanced predictive accuracy.

Derived from the square root of Mean Squared Error (MSE), RMSE is alternatively known as the Root Mean Square Deviation. It gauges the average magnitude of errors, focusing on the deviations from the actual values. An RMSE value of zero denotes a flawless fit, reflecting a model that perfectly aligns with the data. The desirability of lower RMSE values is rooted in their indication of superior

model performance and more accurate predictions. Conversely, a higher RMSE suggests substantial deviations from the residuals to the ground truth. RMSE proves versatile across different features, aiding in the assessment of whether a particular feature contributes positively to the model's predictive capabilities.

RMSE formula is:

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

where y are the real values and $\hat{y}$ the predicted values and n is the sample size.

Pros of RMSE

1) **Sensitivity to large errors:** RMSE effectively penalizes significant errors, particularly beneficial when dealing with outliers.

2) **Differentiability:** being differentiable and continuous, RMSE is suitable for optimization algorithms, crucial for gradient-based optimization techniques.

3) **Easy interpretation:** the RMSE value is in the same units as the target variable, facilitating straightforward interpretation of error magnitude in the original data scale.

4) **Widespread adoption:** RMSE's wide usage and recognition make it easy to compare model performance across different studies and applications, contributing to its acceptance in the data science and machine learning communities.

Cons of RMSE

1) **Sensitivity to outliers:** while sensitivity to large errors is advantageous, RMSE's significant weighting of outliers can be a drawback, leading to overly pessimistic evaluations in cases where outliers don't represent the general trend.

2) **Doesn't indicate direction of error:** RMSE lacks information about the direction of errors, which can be crucial in certain applications where understanding error direction is important for informed decision-making.

3) **Assumption of normality:** RMSE assumes errors are normally distributed, potentially impacting metric reliability when this assumption is violated in real-world scenarios.

4) **Not scale-independent:** RMSE is sensitive to the scale of the target variable, posing limitations when comparing models working with different units or when the scale of the target variable varies.

2.3.2 Mean Absolute Error (MAE)

The **Mean Absolute Error (MAE)**, or referred to as L1 loss, distinguishes itself as one of the most straightforward and easily understandable loss functions and evaluation metrics. It is calculated by averaging the absolute discrepancies between predicted and actual values throughout the dataset. In mathematical terms, it denotes the arithmetic mean of absolute errors, concentrating solely on their magnitude, regardless of direction. A diminished MAE value signifies heightened model accuracy.

MAE provides a direct and intuitive measure of the model's average prediction error, proving particularly effective when evaluating accuracy without overly

penalizing large errors. This metric is robust to outliers since it does not square the errors, and its interpretation in original units makes model assessment more accessible. Due to its simplicity as a loss function, MAE is often used in scenarios where differentiability is not crucial, and a more straightforward evaluation is preferable.

MAE formula is:

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

where y are the real values and $\hat{y}$ the predicted values and n is the sample size.

Pros of MAE

1) **Sensitivity to outliers:** MAE is less sensitive to outliers, making it suitable for scenarios where outliers should not unduly impact model evaluation.

2) **Interpretability:** Easily interpretable, representing the average magnitude of errors in the same units as the target variable.

3) **Simplicity:** Straightforward calculation and computational efficiency.

Cons of MAE

1) **Equal treatment of errors:** Treats all errors equally, regardless of their magnitude, which might not align with the actual importance of different errors.

2) **Lack of sensitivity to prediction variability:** Does not differentiate between different levels of variability in predictions.

3) **Non-Differentiability:** Not differentiable at points where the absolute value function is not differentiable, limiting its use in certain optimization scenarios.

4) **Limited information on error direction:** Does not provide information about the direction of errors, which may be crucial in certain applications.

2.3.3 Mean Absolute Percentage Error (MAPE)

The **Mean Absolute Percentage Error (MAPE)**, alternatively called Mean Absolute Percentage Deviation (MAPD), is a metric used to assess the accuracy of a forecasting model, particularly in the context of time series analysis. It calculates the average percentage difference between the predicted values and the actual values in the dataset.

MAPE expresses the prediction errors as a percentage of the actual values, providing a measure of the relative accuracy of the forecasting model. A lower MAPE indicates better accuracy, with 0% indicating a perfect fit.

MAPE formula is:

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{y_i} \cdot 100$$

where y are the real values and $\hat{y}$ the predicted values and n is the sample size.

Pros of the MAPE

1) **Scale-indipendent:** MAPE is scale-independent as its error estimates are in terms of percentage.

2) **Normalization:** All errors are normalized on a common scale (the percentage scale), enhancing ease of understanding.

3) **Avoidance of cancellation issue:** The use of absolute percentage errors in MAPE prevents the problem of positive and negative values canceling each other out.

Cons of the MAPE

1) **Division by Zero Challenge**: MAPE faces a critical challenge when the denominator becomes zero, resulting in a "division by zero" issue.

2) **Bias towards negative errors**: MAPE exhibits bias by penalizing negative errors more than positive errors, potentially favoring methods with lower values.

3) **Sensitivity to actual value alterations:** Due to the division operation, MAPE's sensitivity to changes in actual values leads to varying loss for the same error.

2.4 Tuning the hyperparameters

Parameter tuning is a critical step in refining machine learning models. A widely employed approach is the use of GridSearchCV, which systematically explores a grid of hyperparameter combinations to identify the optimal configuration, thereby improving the model's performance.

GridSearchCV[20], short for *Grid Search Cross-Validation*, is a machine learning technique designed to discover the most effective combination of hyperparameters for a given model. Hyperparameters, which are predetermined parameters not learned during training, include aspects like the maximum depth of a decision tree, regularization strength in linear regression, or the number of neighbors in a k-Nearest Neighbors model.
The GridSearchCV process follows these steps:

1) **Hyperparameter Grid Definition:** Users specify a grid of values for each hyperparameter they wish to optimize. For example, when working with a

decision tree, the grid might include values for the maximum depth, minimum number of samples in a leaf, etc.

2) **Model Creation:** All possible models are generated by combining the specified values from the grid for each hyperparameter.

3) **Cross-Validation:** For each model, a cross-validation procedure is executed. This entails dividing the dataset into different "folds," training the model on some of these folds, and evaluating it on the remaining folds. This process is repeated multiple times (k times, where k is the number of folds), and the average performance is computed.

4) **Performance Evaluation:** The average performance of the model across all folds is calculated using a designated evaluation metric (e.g., accuracy, recall, F1-score, RMSE, MAE, etc.).

5) **Selection of the Best Model:** The model with the highest average performance across the hyperparameter grid is chosen as the optimal model.

GridSearchCV streamlines this process by automating the systematic exploration of the hyperparameter grid, resulting in the selection of the optimal model. This aids in mitigating the risk of overfitting and enhances the model's generalization to new data. However, it's essential to note that GridSearchCV can be computationally demanding, particularly with a large grid or if the model necessitates extensive training. Additionally, more advanced techniques like **RandomizedSearchCV** may provide a more efficient exploration of the hyperparameter space.

Chapter 3

Dataset Structure and Implementation

The most important goal of this thesis is to construct a comprehensive dataset that captures the details and energy performance of algorithmic routines designed to solve linear systems under various configurations. First, a description of the characteristics of the system where these executions were carried out will be presented, followed by the construction details of the dataset, such as the structure and the launch mechanism of the various configurations.

3.1 CRESCO6 Cluster Architecture

The HPC centre of ENEA sited in Portici hosts the main Clusters [21]. The executable files to create the datasets discussed in this thesis are executed on CRESCO6 Cluster. It is a computational system consisting of 434 nodes.
Each node consists of:

- 2 sockets of 24 cores with Intel(R) Xeon(R) Platinum 8160 processor with a clock frequency of 2.10GHz and 192 GB RAM

- An Intel Omni-Path 100 Gb/s interface
- 2 GbE (Gigabit Ethernet) interfaces
- BMC/IPMI 1.8 support and software for remote console management

The architecture of a CRESCO6 node is represented by the following scheme:

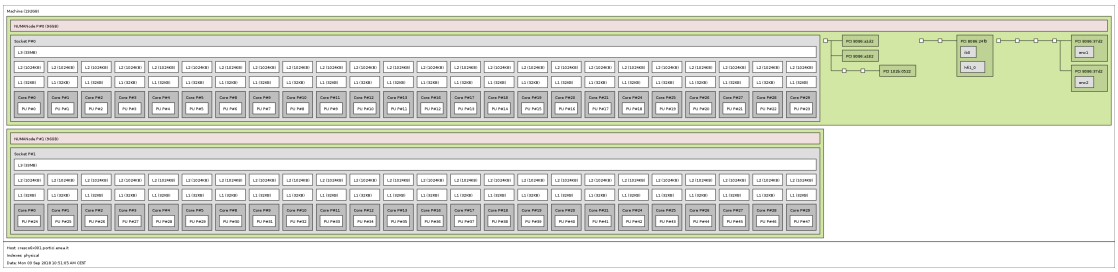


Figure 3.1: Structure diagram of a single node belonging to the CRESCO6 cluster (*Source:* [21])

To use the cluster firstly you have to login to a front-end node. There 2 of them: cresco6x001.portici.enea.it and cresco6x002.portici.enea.it (also for interactive jobs). These nodes are for launching applications via LSF, editing their own launch scripts, or for compilations. Computing nodes are reachable only via LSF. Two file systems are available on this system:

- **AFS**: is the geographically distributed file system common to all ENEAGRID.
- **GPFS**: is the IBM high-performance file system for parallel I/O.

For this thesis, the GPFS file system was used because of its greater amount of memory available to the user.

Launched jobs are placed in a submission queue. The one used for this thesis is `cresco6_48h24`, which is suitable for parallel jobs requiring at least 48 cores and a maximum running time of 24 hours.

3.2 Dataset Structure

The dataset is organized into three distinct blocks of information, each contributing to providing a holistic overview of the executions.

The first block centres on detailing the execution parameters of the routines, meticulously outlining the diverse input configurations of the algorithms slated for execution.

The second block encompasses measurements gathered through sensors positioned on each node of the CRESCO6 cluster. These physical data include critical information such as power and energy consumption, ambient temperature, and the speed of the cooling system's fans of each node.

The final block of information is dedicated to execution statistics, incorporating runtime, average and total memory consumption, as well as any computation errors encountered during the executions.

Through this elaborate and nuanced dataset, the aim is to gain a comprehensive understanding of the algorithmic performance under investigation, thus making a substantial contribution to the optimisation of these computational processes.

To ensure order and easy editing of the scattered data retrieval scripts, the three blocks are divided separately into three groups of files, each of which is stored in a separate directory. The report files, which contain details of the routines' input parameter configurations, are stored in a directory called `reports`. Sensor measurement files are located in the `energy_data` directory, while the output files of each configuration, which include the summary of LSF statistics and the output result of the executed routine, are stored in the `results` directory. Each block of information has associated job ID or job name for each tuple.

For each executed job, the job ID and job name associations and the start and end timestamps of the job provided by LSF are recorded on a further file, so that the data can be merged between the various blocks if necessary, and the

information can be filtered in the execution time range of each job.

The initial step to obtain this dataset involves the careful preparation of the job. For the information obtained from the execution of a job to be valid, it is essential that LSF executes these jobs according to specific requirements both in terms of available resources and execution scheduling. Immediately afterwards, the start-up scripts must be developed, which supplement the job execution command with the parameters of the linear system solver routines to be executed. Next, the code that constitutes the communication mechanism with the sensors located on the cluster nodes must be implemented. Finally, the parameters of the configurations and scripts dedicated to retrieving and processing the essential results taken from the output files of the jobs are defined.

This crucial stage of the process implies a precise orchestration of actions, ensuring that resources are allocated and utilised in such a way that jobs are executed under the conditions necessary to guarantee the validity of the data generated.

3.3 Dataset Implementation

After analysing the dataset structure, this section will show how the preparation processes of the various algorithm configurations will be automated in the form of jobs at execution and how measurements will be carried out. Figure 3.2 shows a diagram of how the submission and execution of a job takes place.

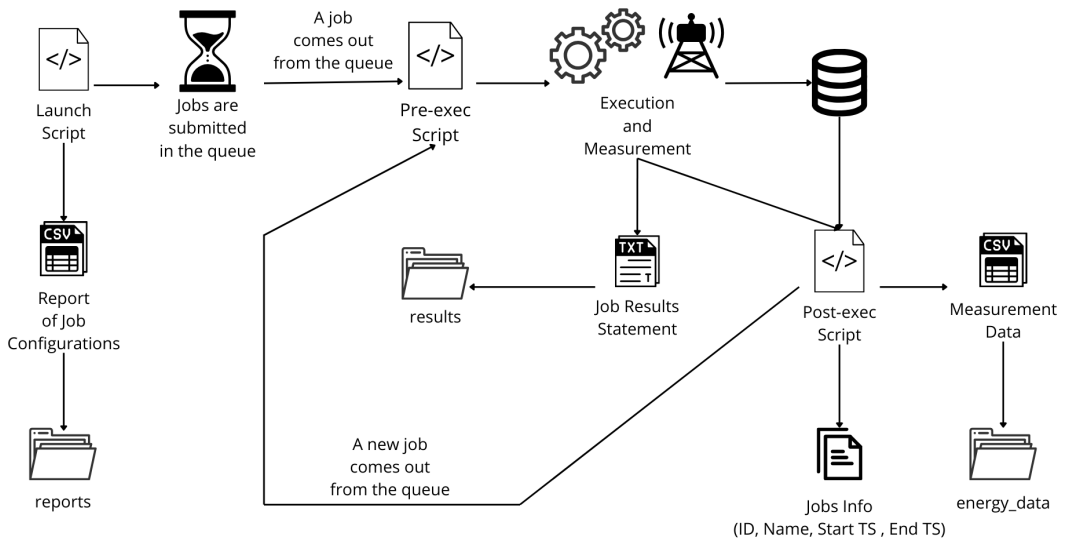


Figure 3.2: Job Execution Mechanism

3.3.1 Preparation phase for job launch

The algorithms, to be executed in parallel on these systems, must be submitted to the LSF scheduler in the form of jobs. Most of the implemented code is written in the form of Bash scripts. After a thorough analysis of the CRESCO6 architecture and a full understanding of the LSF commands, it is necessary to provide specific instructions to LSF before starting routines. This step is essential to successfully co-ordinate the simultaneous execution of numerous subjobs, ensuring efficient scheduling of available resources.
The CRESCO6 cluster is used daily by many users. The first requirements are:

- Nobody user have to use the nodes employed by jobs;
- LSF have to schedule every rank in the minimum number of nodes as possible.

To satisfy these requirements, certain options must be integrated into the job submission command, i.e. `bsub`. In order to request the entire node, it is essential to commit all of its physical processor cores. Each node consists of 48 cores, so to ensure that LSF reserves an entire node, the `-n` option must be specified with the value 48. If more nodes are required, it is imperative to request a number of cores equal to 48 multiplied by the desired number of nodes.

LSF follows a policy of allocating as many job ranks as possible on the cores of the same node. However, if some of a node's cores are already being used by other users, LSF distributes the remaining ranks to an additional node. For example, if 48 processors are required but a node only has 30, the remaining 18 are allocated to another node.

To fulfil the second requirement, the `bsub` command in LSF with the `-R` option is used to specify resource requirements. In this circumstance, the requirement is set to `slots == 48`, indicating that each node must have 48 free slots. In the LSF context of CRESCO6, it is important to note that the number of working slots corresponds to the total number of cores for each node.

In this context, each node is required to have all 48 cores free to execute the job. Even if 48 processors are requested and a node has exactly 48 free cores, this ensures that LSF will assign all processors to the same node. It should be noted that although the `-n` option specifies the maximum number of node cores to use, the job may not use them all. In situations where fewer cores are being used than are specified with `-n`, the unused processors will switch to the IDLE state. The actual number of cores required will be defined in the MPI launch script. This setting ensures efficient allocation of resources, ensuring that jobs are executed on the appropriate nodes and maximising the use of available cores.

To ensure correct measurement of the energy consumption of the jobs through the communication mechanism with the node that acquires the data from the

sensors, a strategy was implemented that requires the sequential execution of the jobs. The key option used to achieve this was the `-w` option of `bsub`, which allows the dependency requirements between jobs to be defined.

In particular, the specified requirement is like `ended(PREVIOUS_JOB_NAME)`, indicating that the job in question can only start after the previously submitted job has finished. The termination condition is valid whether the exit status is `EXIT` (in case of an error) or `DONE` (in case of success) of the predecessor job. It should be noted that even if a job ends with an error state, the next job can still start.

This aspect was considered in the design to maximise overall time efficiency, allowing the workflow to progress even in the presence of individual job errors. This flexibility helps to ensure continuity of operations by reducing possible delays due to individual failures.

Additional options aimed at optimising execution and resource management in the cluster context were also adopted. These options include the control strategy of the job execution duration, the specific cluster submission queue, output and error management, and the monitoring and control of energy consumption.

The `-W` option is used to define the runtime limit of the job. It is important to note that LSF uses this time estimate for scheduling purposes only, facilitating job execution as soon as possible based on prior knowledge of the approximate duration.

In order to direct jobs to a specific queue in the cluster, the `-q` option was used, which allows the submission queue to be specified. In the context of this thesis, the queue selected is `cresco6_48h24`, which is designed for parallel jobs with a required number of cores of 48 or more, with a runtime limit of 24 hours.

The `-o` and `-e` options were implemented to direct the standard output and error output of the job to specific file paths. This practice favours an orderly management of the information generated during job execution.

Given the dynamic nature of node allocation by LSF, a mechanism was adopted to communicate these to the script that collects sensor data. To communicate this information to the monitoring node, the `-E` and `-Ep` options were introduced.

The `-E` option enables the execution of a pre-execution command or script on the first host assigned to the job by LSF, prior to the actual execution of the job. This pre-execution command determines the possibility of starting the job, affecting the scheduling process.

Similarly, the `-Ep` option enables post-execution operations, allowing a specific command or script to be executed at the end of the job.

The main structure of the bsub command is as follows:

```
bsub    -n <NUM_OF_CORES_TO_RESERVE>
        -W <WTIME>
        -q "cresco6_48h24"
        -E "<PRE_EXEC_SCRIPT> <PARAMETERS>"
        -Ep "<POST_EXEC_SCRIPT> <PARAMETERS>"
        -J "<JOB_NAME>"
        -w "<CONDITION>"
        -o "<OUTPUT_FILE>"
        -e "<ERRORS_FILE>"
        -R "slots == 48"
        <MPI_SCRIPT> <MPI_SCRIPT_PARAMETERS>
```

The subsequent section will describe in more detail the communication mechanism between the node submitting the jobs and the node dedicated to monitoring the sensors, which was mentioned above.

3.3.2 Job Launch Scripts

The first job launch script created is the MPI application. CRESCO6 with Open-MPI modules, as the command execution a script with a certain pattern [21]:

```
#!/bin/sh
exe=/afs/enea.it/por/user/raia/Hello_MPI # path of your MPI
   program
HOSTFILE=$LSB_DJOB_HOSTFILE # name of hostfile for mpirun
N_procs=‘cat $LSB_DJOB_HOSTFILE | wc -l‘ # give to mpirun same
   number of slots
mpirun --mca plm_rsh_agent "blaunch.sh" -n $N_procs --hostfile
 $HOSTFILE $exe
```

For the jobs to be submitted for this thesis, it was readjusted as follows:

```
#!/bin/bash
module unload mpi_flavour pgi gcc intel
module load gcc/gcc730 mpi_flavour/openmpi_gcc730-3.1.2

exe=/afs/enea.it/por/user/mcolonna/private/ime/bin/tester
HOSTFILE=$LSB_DJOB_HOSTFILE     # name of hostfile for mpirun
N_procs=$1
NPERSOCKET=$2
shift shift
if [ "${NPERSOCKET}" == "-" ]; then
    mpirun --map-by core --mca plm_rsh_agent "blaunch.sh" -n
       $N_procs --hostfile $HOSTFILE $exe "$@"
else
    mpirun --map-by ppr:${NPERSOCKET}:socket --bind-to core --mca
       plm_rsh_agent "blaunch.sh" -n $N_procs --hostfile
       $HOSTFILE $exe "$@"
fi
sleep 3
```

In this code, the openMPI modules on CRESCO6 were loaded, as the default ones were not appropriate for the requirements of the context. Subsequently, the number of jobs was not bound to the number of available slots, as the nodes had to fulfil specific requirements. In this circumstance, the parameters transmitted included the actual number of cores to be used. Another relevant parameter is the number of cores per socket. This is due to the fact that each job is executed in two distinct modes: with a mapping involving all the nodes' cores, excluding the last one if there are cores in idle mode, or with a partitioning of cores per socket on each node, in order to balance the execution load. This configuration is accompanied by a process-core association to prevent the migration of processes between cores.

Through the input parameter shift operation, input parameters received that are no longer needed are discarded and the others are passed to the tester application during the execution of the mpirun command. After preparing the MPI script, aiming to simplify modifications and improve the scalability and maintainability of the code, the script that is responsible for submitting the various jobs was split up and made more modular.

Three separate *sub-programmes* were created to handle the different instances of the MPI and ScaLAPACK routines, distinguishing between those without fault tolerance, those with Inhibition Method fault-tolerant, and those with ScaLAPACK fault-tolerant. This subdivision was necessary due to the various types of parameters to be passed as input to MPI scripts or processed within them.

Examining these *sub-programmes*, the first script handles the fault-tolerant cases of the IMe and ScaLAPACK routines in a homogeneous manner. Since the MPI application handles both cases similarly, common input parameters include:

- **calculation processes**: number of processes dedicated exclusively to the calculation of the system solution;

- **matrix size**: rank of the input matrix;
- **precision**: numerical representation of real numbers;
- **repetitions**: number of iterations of the internal algorithm (default value used: 1);
- **algorithm**: IMe or ScaLAPACK;
- **socket balancing**: Boolean indicating whether or not to perform socket balancing;
- **wall time**: estimate of the execution time useful to LSF for job scheduling;
- **report file**: path where the job configuration information is to be entered.

This code retrieves the MPI launch, pre-execution and post-execution scripts, and, based on the algorithm and precision parameters, identifies the correct routine to execute. It then calculates the number of nodes needed, the number of cores to be reserved, the number of processes per socket (if balancing is enabled), the scheduling condition, the path to save the output and error files, creates the job name, and finally composes all the parameters collected and calculated to execute the job. The job name is designed in such a way as to allow easy retrieval of that job's configuration through a dedicated script that acts as a parser, thus facilitating the possibility of re-executing the job with the same settings.

As for the second and third scripts, they accept the same parameters as the case without fault tolerance. In addition, they include the following parameters:

- **fault tolerance level**: specifies the level of fault tolerance that can be handled (in this study 1, 2, 4, 8);
- **number of simulated faults**: specifies the number of faults to be simulated. For each fault tolerance scenario, the decision was made to evaluate

both the case with no simulated faults and the case with the maximum number of simulated faults, in order to analyse the behaviour of the system.

The difference between the two scripts lies in the processing of the parameters calculated within them to be included in the job submission command. In the IMe algorithm, before determining the number of nodes required and the other parameters already examined above, it is necessary to compute the spare processes used to repair faults. To ensure optimal statistics, it was decided to simulate failures at one of the lower levels, specifically level 2.

As far as ScaLAPACK is concerned, in addition to the previously mentioned spare processes and parameters, it is necessary to calculate the blocking factor for the load distribution of activities. This parameter varies depending on the number of calculation processes and the rank of the matrix. In addition, the checkpoint used for restoring the system in the event of a failure must be determined. In this context, checkpointing is performed at half the rank of the matrix.

These three scripts are called from a main script dedicated to iterating over the parameter values to be supplied as input. The convenience of this structure lies in the fact that, if one wishes to modify the parameters, it is not necessary to make changes to the scripts that perform the jobs. Instead, if a change to a type of script is necessary, it is only possible to work on the relevant section of code, without affecting the rest.

To facilitate the handling of any inconveniences during the execution of the various jobs, additional utility scripts have been developed, including:

- A script that, starting from the job name, retrieves the parameters transmitted to the execution script.
- A script that displays the jobs that encountered errors during execution.

- A script that, analysing the error files of jobs, re-executes them in the event of non-zero file sizes.
- A script dedicated to extracting information from each job output file.
- A script capable of executing queries passed as a parameter on the database, in order to avoid the inclusion of confidential information in each individual script.

3.3.3 MPI-based Command Line "Tester"

In this paragraph will briefly show the MPI-based command line application used to execute the algorithm routines described in the last chapter the associated parameters to ensure their correct configuration.

Commands

The main commands of the tester are:

- `--help`: prints information about the usage of the command line interface;
- `--list`: print the list of testable routines;
- `--save`: save the generated matrices to files;
- `--run <routine>`: run the test(s) of a routine or a list of routines;

Options

The options used to execute the tests of this thesis are:

- `-nm <integer number>`: specifies the matrix rank;
- `-seed <integer number>`: is the seed of the random generation of the linear system coefficients (in this thesis is always 1);

- `-cnd <integer number>`: is the condition number of the input matrix (in this thesis it assumes values: 1 for generating arrays with ideal conditions of numerical stability and 1000 for those with real conditions of numerical stability);
- `-type <string>`: precision type [s, d, single, double] of the input matrix (routines for different precisions cannot be mixed);
- `-no-cnd-set`: disables matrix pre-conditioning;
- `-no-cnd-readback`: disables condition number checking after generation;
- `-energy-reading`: read Powercap energy counters for each node used at the beginning and end of the run;
- `-r <integer number>`: run multiple times corresponded to the specified number of repetitions (in this case 1);
- `-o <file path>`: saves the output to CSV file
- `-i <file path>`: takes as input matrices base the ones saved at the specified the file path name (.A, .X, .B auto appended)
- `-ft <integer number>`: is the fault-tolerance level (0=none);
- `-fr <integer number>`: simulate faulty MPI ranks;
- `-fl <integer number>`: simulates faulty IMe inhibition level;
- `-npf <integer number>`: is the number of simulated faults (0=none);
- `-nps <integer number>`: number of spare processes for recovery (0=none);
- `-spk-cp <integer number>`: checkpointing interval;
- `-spk-nb <integer number>`: ScaLAPACK blocking factor;

Routines

This software is capable of executing the different routines of several algorithms. In particular, the ones we will use are:

- `IMe-pSGESV-co`
- `IMe-pDGESV-co`
- `IMe-pSGESV-co-ft`
- `IMe-pDGESV-co-ft`
- `SPK-pSGESV`
- `SPK-pDGESV`
- `SPK-pSGESV-ft`
- `SPK-pDGESV-ft`

The first four are IMe routines. The other four are ScaLAPACK routines divided into single and double precision and then into the respective cases with integrated fault tolerance.

3.3.4 Communication with the monitoring node

Each node in the CRESCO6 cluster is equipped with a sensor capable of taking measurements of various parameters, including power consumption, energy consumption and node temperature.

To obtain this data, the command `nodesensors <NODENAME>` can be used. This command is launched on an isolated CRESCO architecture node, which cannot be reached by user access nodes. Each time this command is invoked, the instantaneous values detected by the sensor on the CRESCO6 node passed as input parameter are returned.

Since the node retrieving the sensor data is isolated, communication between it and the node responsible for launching the jobs is necessary in order to specify for each job the nodes to be sampled. To facilitate this communication, a database was chosen. Therefore, in the pre-execution script of the job, which already receives the assignment of all nodes from LSF, this information is extracted and entered into the database, indicating the nodes to be monitored and the time at which to start recording measurement values.

On the other hand, in the post-execution script, instructions are provided to terminate the measurements, perform data retrieval, and calculate any values derived from the measurements. This approach enables the efficient management of sensory data collection and analysis, contributing to the optimised monitoring and management of resources in the context of the CRESCO6 cluster.
The communication process consists of the following phases:

1) The first phase begins with the submission of the job, which is queued in the scheduling queue awaiting execution.

2) Once the necessary resources are available, the LSF (Load Sharing Facility) proceeds to allocate these resources. During this phase, a pre-execution

script is executed on the first allocated node. This script retrieves crucial information, such as the names of the assigned nodes and the job ID, using the `bjobs` command. This information is then entered into a database table, which is accompanied by a lock flag. The latter indicates that the job is about to be executed, and the flag is set to "R". This "unlock" value is essential, as it allows measurement scripts to run continuously.

3) During job execution, a script on the measurement node processes the last row of the table containing information about the running jobs and nodes. If the lock flag is set to "R", the script processes the information from the sensors and saves the values obtained in another database table.

4) Once the job execution is complete, a post-execution script comes into play. In this phase, the value of the lock column in the table indicating the nodes is changed to "D". The measurement script stops the sampling task. The measurement data in the database is then saved to a CSV file in the job's submission node. In parallel, another file stores the start and end data of the job in timestamp format in seconds, thus providing a more precise method for filtering the measurements.

Initially, the procedure for collecting the nodes' energy values was conceived using a single script, which received as input the complete list of nodes employed by the job. For each node, the process would cyclically execute the `nodesensors` command. However, this methodology was discarded due to the relatively low sampling rate: with a list consisting of only one node, the measurement interval was only 1 second (considered the optimal case), while for two 2 nodes it extended to about 3 seconds and even reached 11 seconds to complete a measurement cycle on 16 nodes. Since the job execution time decreases as the number of nodes increases, the best case involved an interval of 9 seconds on 16 nodes, inevitably

generating a significant margin of error in the measurements. These limitations made it necessary to adopt an alternative strategy to ensure acceptable accuracy in energy data collection.

Then, considering that the optimal configuration is achieved by sampling only one node and that the maximum number of nodes used in the relevant jobs is 16, an approach based on 16 parallel instances of the script was adopted. These instances act as daemons and operate cyclically, collecting data from each node every second and subsequently feeding it into the database. The use of different tuples with unique primary keys avoids conflicts during data entry. The script is configured to depend on a single input parameter, indicating at which position in the list it should read the name of the node to be monitored. In the event that no node is present at that position, e.g. if the number of nodes is less than the parameter value representing the position, unnecessary instances will remain inactive, ensuring efficient and effective operation of the monitoring system.

This strategy ensures flexible and optimised management of the sampling process, dynamically adapting to the configuration of nodes present in the context of job execution.

Analysing the CSV file, where the sensor measurements are saved, it contains the following entries:

- **Jobid**: id of LSF job;
- **Nodename**: node name, generally all nodes in the cluster have names with prefix “cresco6x .....”;
- **Tempo**: precise date and time of the reading to the sensors, expressed in the format “Thu 1 Nov 00:02:39 CET 2018”;
- **Timestamp_measure**: timestamp of the measure expressed in unixtime;

- **Sys_power**: Total instantaneous Power Measurement of the computing node in W;
- **Cpu_power**: CPU instantaneous Power Measurement of the computing node in W;
- **Mem_power**: Ram Memory instantaneous Power Measurement of the computing node in W;
- **Sys_util**: percentage of use of the system (single node);
- **Cpu_util**: percentage of use of the CPU's (single node);
- **Mem_util**: percentage of use of the RAM memory (single node);
- **IO_util**: node I/O traffic;
- **Amb_temp**: measured ambient temperature near the computing node.
- **CPU1_Temp**: CPU1 temperature in °C
- **CPU2_Temp**: CPU2 temperature in °C
- **Exh_temp**: Exhaust temperature in °C (air exit of the node);
- **Sysairflow**: air flow of the node measured in CFM (cubic feet to minute) - indicates the flow of air moved;
- **Fan1a**: speed of the cooling fan Fan1a installed in the node expressed in RPM (revs per minute);
- **Fan1b**: speed of the cooling fan Fan1b installed in the node expressed in RPM (revs per minute);

- **Fan2a**: speed of the cooling fan Fan2a installed in the node expressed in RPM (revs per minute);
- **Fan2b**: speed of the cooling fan Fan2b installed in the node expressed in RPM (revs per minute);
- **Fan3a**: speed of the cooling fan Fan3a installed in the node expressed in RPM (revs per minute);
- **Fan3b**: speed of the cooling fan Fan3b installed in the node expressed in RPM (revs per minute);
- **Fan4a**: speed of the cooling fan Fan4a installed in the node expressed in RPM (revs per minute);
- **Fan4b**: speed of the cooling fan Fan4b installed in the node expressed in RPM (revs per minute);
- **Fan5a**: speed of the cooling fan Fan5a installed in the node expressed in RPM (revs per minute);
- **Fan5b**: speed of the cooling fan Fan5b installed in the node expressed in RPM (revs per minute);
- **DCenergy**: energy meter consumed up to the time of reading - useful for making differences between two readings in kWh;
- **Note**: Measurement notes (not always present);
- **Delta_e**: difference between previous and current measurement of energy in KWh for that node;

3.3.5 Job results statement

As mentioned in Section 3.2, the data were organised into three distinct information blocks. After the presentation of the job configuration scripts, which also include the storage of the first block of information, and the mechanisms for communication and data retrieval with the sensors, representing the second block, this section will briefly focus on the data generated by the results of each job, which are stored in the respective output files.
For each job the main information are:

- LSF Resource usage summary:
  - CPU time in seconds
  - Max Memory in MB
  - Average Memory in MB
  - Max Processes
  - Max Threads
  - Run time in seconds
  - Turnaround time in seconds
- Output of MPI application. The most relevant information are:
  - Normwise Relative Error
  - Powercap energy counters

Subsequently, through the previously mentioned utility script (Section 3.3.2), this information was aggregated into a single file.

3.4 Configuration Parameters

The following parameters were chosen to represent this dataset for the different configurations:

- **Algorithms**: Inhibition Method (IMe) with compressed matrix, ScaLAPACK based on Gaussian Elimination
- **Calculating processes**: 64, 100, 144, 256, 400, 484, 576
- **Matrix sizes**: 5280, 10560, 15840, 21120, 26400, 31680, 36960, 42240
- **Accuracy**: single, double
- **Fault tolerance**: 0 (no fault tolerance), 1, 2, 4, 8
- **Number of simulated faults** (only if fault tolerance > 0): 0 , maximum value for fault tolerance
- **Balancing per socket**: yes, no.

With these parameters, a total of 4032 different configurations are obtained. After defining the crucial parameters, the matrices to be used as input for the algorithms in question were generated. Initially, matrices characterised by ideal numerical stability conditions were created. However, to broaden the scope of the analysis, it was subsequently decided to generate matrices with more realistic numerical stability conditions. This choice was guided by the desire to build a second dataset, aimed at investigating possible variations under different numerical stability conditions.

For better organisation and clarity, it was decided to present a summary of each of the two datasets in the form of a table. As shown in Table 3.1, it is mainly

characterised by two fundamental variables: the matrix size and the calculating processes performed.

Each cell in the table corresponds to 72 different configurations generated, considering the variations of the other parameters in relation to the specified number of processes and the rank of the matrix. It is important to note that, due to the long execution times, the configurations for the last three rows of the first two columns on the bottom left (coloured in red) were not actually executed. However, it should be emphasised that despite this limitation, each dataset consists of 3600 configurations instead of 4032, still maintaining a wide spectrum of data for analysis.

mat_size \calc_procs	**64**	**100**	**144**	**256**	**400**	**484**	**576**
5 280	✓	✓	✓	✓	✓	✓	✓
10 560	✓	✓	✓	✓	✓	✓	✓
15 840	✓	✓	✓	✓	✓	✓	✓
21 120	✓	✓	✓	✓	✓	✓	✓
26 400	✓	✓	✓	✓	✓	✓	✓
31 680			✓	✓	✓	✓	✓
36 960			✓	✓	✓	✓	✓
42 240			✓	✓	✓	✓	✓

Table 3.1: Summary of jobs in a dataset

Chapter 4

Predictive Analysis

In this chapter, we will proceed with a little predictive analysis of energy performance, using part of the data obtained through the execution of the previously presented linear system solvers. The initial section will focus on target selection and description of the two datasets, providing details on the distribution of values and other relevant indicators. Subsequently, predictors, evaluation metrics and hyper-parameter optimisation techniques, as outlined in Chapter 2, will be employed in order to identify the optimal regressor. This approach will allow reliable evaluation of predictions and consequently achieve significant energy savings.

In this chapter, the primary graphs of the first dataset, characterized by ideal numerical stability conditions, will be presented. For secondary and additional graphs, as well as those pertaining to the second dataset under realistic numerical stability conditions, refer to the Appendix A and B.

4.1 Data Exploration

In this section, the data characteristics of both datasets will be examined. The chosen targets include the **total energy consumption**, **the maximum and av-**

erage power value of the nodes, and **the running time** of the configurations of the algorithms considered. In addition, a detailed description of the data, once converted into numerical values for the regression, will be provided, accompanied by boxplots of the targets to allow a visualisation of the distribution of the values.

4.1.1 Data Preprocessing

Before starting data analysis, a preprocessing step is necessary to convert categorical data into numerical values for use with a regressor. The fields involved in this transformation are "ALGORITHM", "PRECISION" and "BALANCE". Typically, the "One-Hot Encoder" method is employed, but given that all the data in question have only two distinct values, a more efficient approach involves using a binary variable. The conversion process for each field is as follows:

- **ALGORITHM:**
  - 0 represents 'IMeCO' (compressed matrix IMe)
  - 1 represents 'SPK' (ScaLAPACK)
- **PRECISION:**
  - 0 represents 'single'
  - 1 represents 'double'
- **BALANCE:**
  - 0 represents 'No'
  - 1 represents 'Yes'.

4.1.2 Data Description

Fields \ Index	Mean	Std	Min	25%	50%	75%	Max
ALGORITHM	0.5	0.500069459	0	0	0.5	1	1
PRECISION	0.5	0.500069459	0	0	0.5	1	1
FT	3.333333333	2.789254178	0	1	2	4	8
NF	1.666666667	2.582347582	0	0	0	2	8
CPROCS	314	181.0838168	64	144	256	484	576
SPROCS	29.73333333	45.73302815	0	2	8	40	192
TPROCS	343.7333333	195.4135177	64	148	320	492	768
MATSIZE	22176	11760.75189	5280	10560	21120	31680	42240
NODES	7.668888889	4.02373189	2	4	7	11	16
SPK-CP	4928	6762.638515	0	0	0	10560	21120
BKF	10.38222222	11.63053132	0	0	0	24	25
BALANCE	0.5	0.500069459	0	0	0.5	1	1
NUMxSOCK	10.99333333	11.11289712	0	0	8	23	24
TOTAL_ENERGY	0.076255028	0.140817828	0.00047	0.0059	0.01653	0.0680275	0.93859
MAX_SYS_POWER	334.8583333	32.51745315	150	310	340	360	440
MEAN_SYS_POWER	236.5875569	50.43170759	115	194.1188575	248.362065	280.40282	330.15842
RUNTIME	140.7272222	260.0918769	6	15	36	131	2097

Table 4.1: Description of the dataset data with ideal numerical stability condition

Table 4.1 presents a comprehensive overview of the dataset for each field, such as key statistical indices from ‘pandas.describe‘: mean value, standard deviation, min value, 25^{th}, 50^{th}, and 75^{th} percentiles, and max value. It is important to note that the target variable, representing total energy consumed, is normalized. While the other target variables maintain their current representation for this thesis, it is suggested that future studies consider normalizing these variables as well. Analyzing the impact of normalization on the prediction of these data could be a valuable avenue for further research.

4.1.3 Boxplots

In Figure 4.1, four boxplot plots are shown for the variables 'TOTAL_ENERGY', 'MAX_SYS_POWER', 'MEAN_SYS_POWER' and 'RUNTIME_S'. The boxplots show the median, quartiles and outliers for each variable, providing a compact view of the distribution of values. Outliers are represented by circles, indicating values that deviate significantly from the rest of the sample. These graphs are useful for identifying the presence of any outliers in the data and for assessing the symmetry of the distribution.
The following information can be deduced:

- **TOTAL_ENERGY**: Values are concentrated very close to zero, with a median at or near zero. There are many outliers, indicating that there are some observations with significantly higher total energy values than most of the data.

- **MAX_SYS_POWER**: Values appear to be more distributed than in TOTAL_ENERGY, with a median around 250-300. There are fewer outliers here, indicating a smaller variance than in TOTAL_ENERGY.

- **MEAN_SYS_POWER**: Values appear to be more distributed than in TOTAL_ENERGY, with a median lying around 250. No outliers are shown.

- **RUNTIME_S**: Most values are concentrated near zero, similar to TOTAL_ENERGY, but there are many outliers indicating much longer run times.

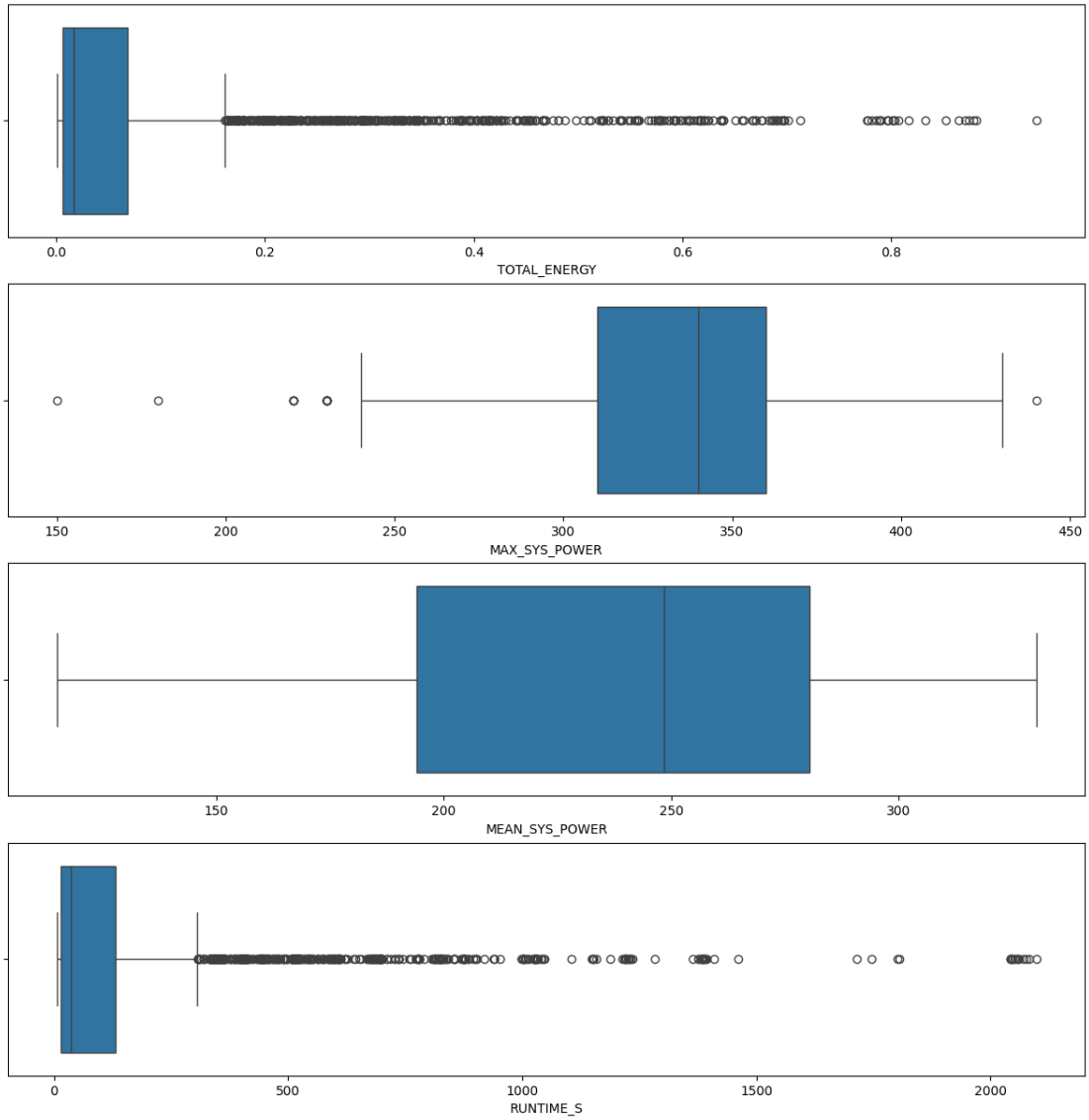


Figure 4.1: Boxplots of total energy consumption, maximum and average power for each node used and runtime

4.1.4 Histograms target

In this subplot, histograms are shown for the target variables, illustrating the frequency of the data in specific intervals (Figure 4.2). Above the histograms, a density curve estimates the probability distribution of the variable. These graphs are useful for visualising the general shape of the data distribution, including the patterns and presence of skewness or bimodal distributions.

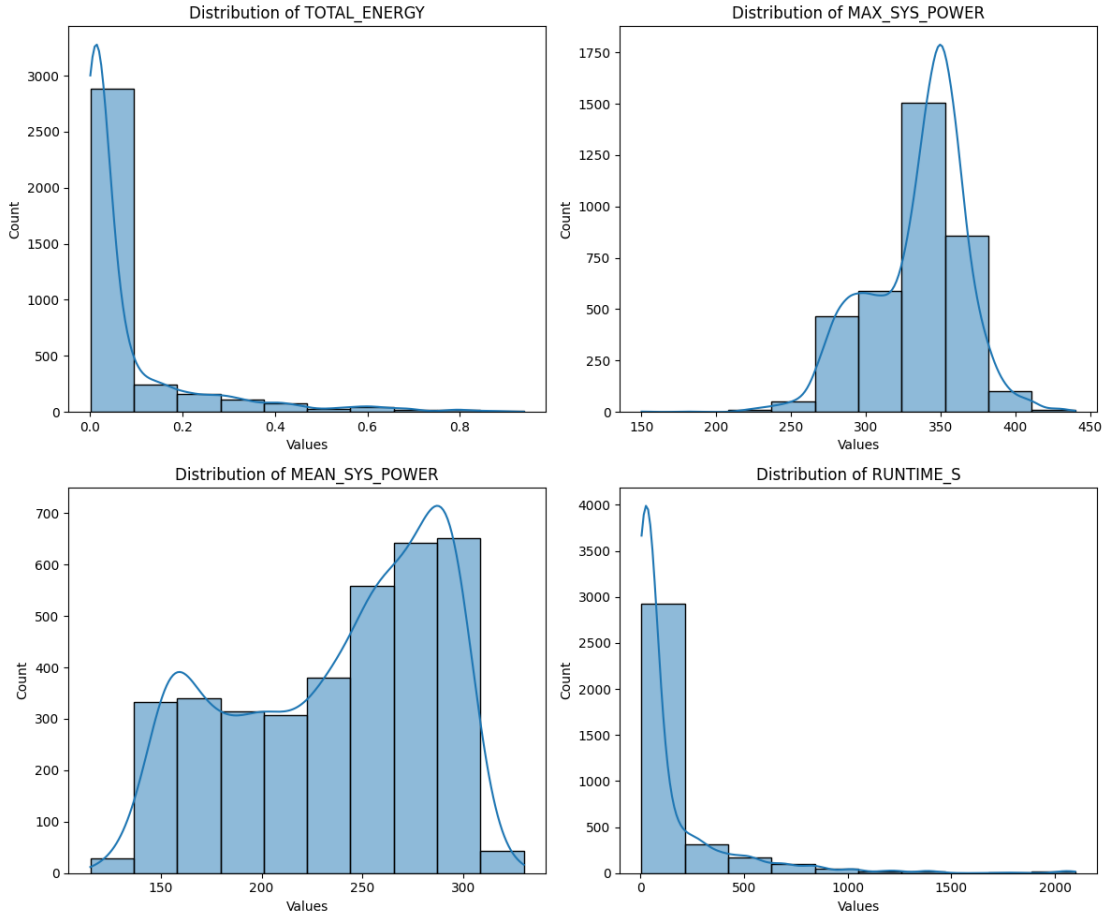


Figure 4.2: Distributions of Energy Performance: Total Consumption, Peak Power, Avg. Power, and Runtime

In this case, it is noted that:

- **TOTAL_ENERGY**: The distribution is strongly skewed to the right, with most values clustered near zero KWh and some extremely high values, as indicated by the long tail on the right.
- **MAX_SYS_POWER**: The distribution of maximum system power values seems more uniform and less skewed than TOTAL_ENERGY, with a slight tendency towards the medium-high values.
- **MEAN_SYS_POWER**: This distribution appears similar to that of MAX-_SYS_POWER, suggesting that the mean value of the system power follows a similar trend to that of the maximum system power.
- **RUNTIME_S**: As with TOTAL_ENERGY, there is a strong skew to the right, with most values concentrated near zero.

4.2 Regressor Evaluation

The table provides a detailed overview of the metrics for the chosen regressors in predicting energy performance. For each regressor, two hyperparameter optimization processes were conducted using gridsearchCV, employing both negative mean squared error and mean squared absolute error as evaluation criteria. Additionally, metrics such as RMSE, MAE, and MAPE were calculated, along with the optimal hyperparameter for depth. To ensure experiment reproducibility, the random state value was fixed at 42. A thorough analysis reveals that the most effective regressor, as expected, is the Random Forest Regressor.

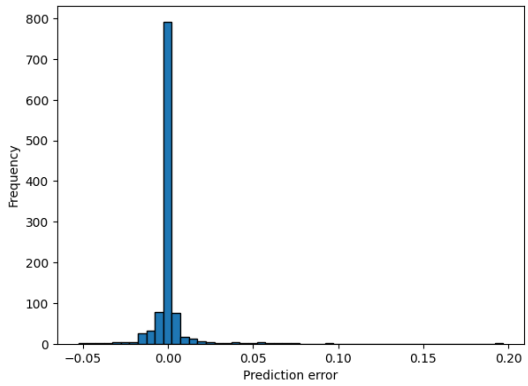
TARGET: TOTAL_ENERGY (Min value : 0.000470 kWh - Max value: 0.938590 kWh)					
REGRESSOR	**TYPE SCORING GRIDSEARCH CV**	**BEST DEPTH**	**RMSE Root Mean Squared Error**	**MAE Mean Absolute Error**	**MAPE Mean Absolute Percentage Error**
Decision Tree	neg_mean_squared_error	7	0.0127546875055387 7	0.00460957337260932	0.12892347597528406
	neg_mean_absolute_error	19	0.013076086258963416	0.00440911111111111	0.07613024544466025
Random Forest	neg_mean_squared_error	8	0.011459619557381045	0.00391048285208939	0.07706775271558028
	neg_mean_absolute_error	13	0.010977759177219614	0.0036212282992650656	0.0621044065764673
Gradient Boosting Decision Trees	neg_mean_squared_error	4	0.010984345149752914	0.004064171446502564	0.19531441428445115
	neg_mean_absolute_error	5	0.010964453587660499	0.003721560408455006	0.11999544031079942
TARGET: MAX_POWER (Min value : 150.000000 W - Max value: 440.000000 W)					
REGRESSOR	**TYPE SCORING GRIDSEARCH CV**	**BEST DEPTH**	**RMSE Root Mean Squared Error**	**MAE Mean Absolute Error**	**MAPE Mean Absolute Percentage Error**
Decision Tree	neg_mean_squared_error	8	11.721032536216727	7.840206936800857	0.025459493945647525
	neg_mean_absolute_error	10	12.67629088504843	7.966831722253927	0.02418583101927385
Random Forest	neg_mean_squared_error	10	10.071289391584031	6.899977766413612	0.02081705971122439
	neg_mean_absolute_error	11	10.241775561215347	6.9472041660891115	0.020960671799750048
Gradient Boosting Decision Trees	neg_mean_squared_error	5	10.538027940147591	7.216216035020971	0.02165217702090875
	neg_mean_absolute_error	6	10.339164947223738	7.089667493848047	0.021379468805083495
TARGET: MEAN_POWER (Min value : 115.000000 W - Max value: 330.158420 W)					
REGRESSOR	**TYPE SCORING GRIDSEARCH CV**	**BEST DEPTH**	**RMSE Root Mean Squared Error**	**MAE Mean Absolute Error**	**MAPE Mean Absolute Percentage Error**
Decision Tree	neg_mean_squared_error	7	7.8011153440732155	5.72857807241245	0.026113611768348765
	neg_mean_absolute_error	9	7.297785836367988	5.188574471984356	0.024404831082133947
Random Forest	neg_mean_squared_error	15	6.574396178443816	4.659535641571242	0.02215968926861537
	neg_mean_absolute_error	11	6.530530828193816	4.632852686227598	0.022011408105103197
Gradient Boosting Decision Trees	neg_mean_squared_error	5	5.9429056481767875	4.319605873630149	0.020523226809441252
	neg_mean_absolute_error	5	5.9429056481767875	4.319605873630149	0.020523226809441252
TARGET: RUNTIME (Min value : 6.000000 s - Max value: 2097.000000 s)					
REGRESSOR	**TYPE SCORING GRIDSEARCH CV**	**BEST DEPTH**	**RMSE Root Mean Squared Error**	**MAE Mean Absolute Error**	**MAPE Mean Absolute Percentage Error**
Decision Tree	neg_mean_squared_error	10	22.238984402183785	7.29594903278829	0.06542877264761252
	neg_mean_absolute_error	10	22.238984402183785	7.29594903278829	0.06542877264761252
Random Forest	neg_mean_squared_error	18	19.41031338156751	7.041209949695367	0.06054693612839134
	neg_mean_absolute_error	12	19.65149666938452	7.03334351312457	0.06011087396678247
Gradient Boosting Decision Trees	neg_mean_squared_error	6	14.637845002519697	5.560338101119648	0.06530000056868551
	neg_mean_absolute_error	6	14.637845002519697	5.560338101119648	0.06530000056868551

Table 4.2: Summary table of regressor prediction evaluation on the first dataset

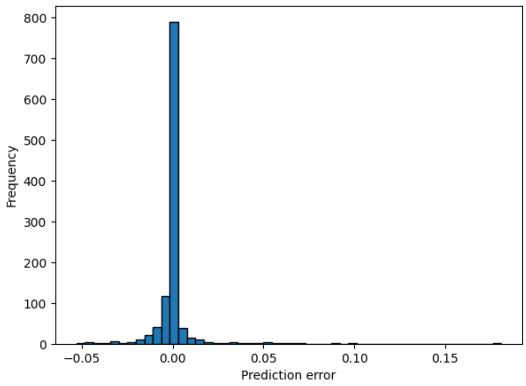
4.3 Prediction error analysis

In this section, the final analysis compares the error distributions of the Random Forest regressor using tuning scores with *neg_mean_squared_error* and *neg_mean_absolute_error* for hyperparameters. Initially, the frequency of error values is presented for all configurations and each target (Figure 4.3). Subsequently, the prediction errors are analyzed by dividing the data based on prediction precision (single or double). The results of similar analyses for other regressors can be found in Appendix A.

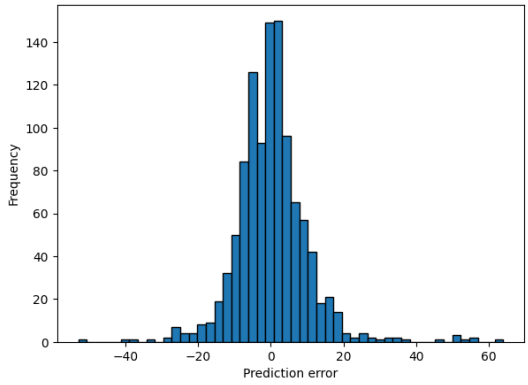
4.3.1 Random Forest



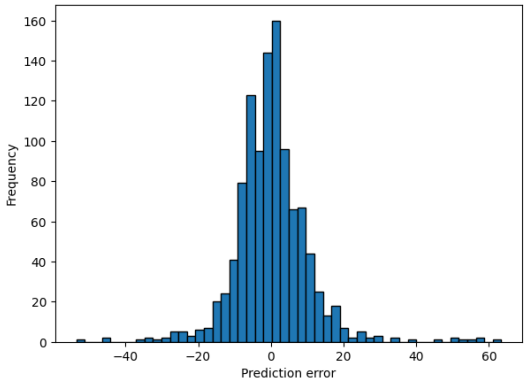
(a) Total energy - Score Tuning: Neg MSE



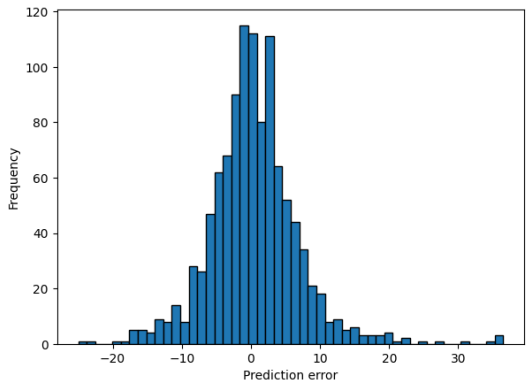
(b) Total Energy - Score Tuning: Neg MAE



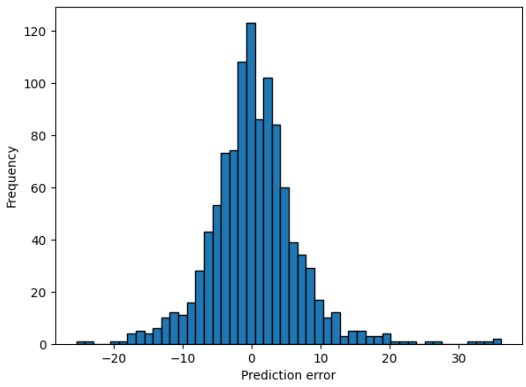
(c) Max Power - Score Tuning: Neg MSE



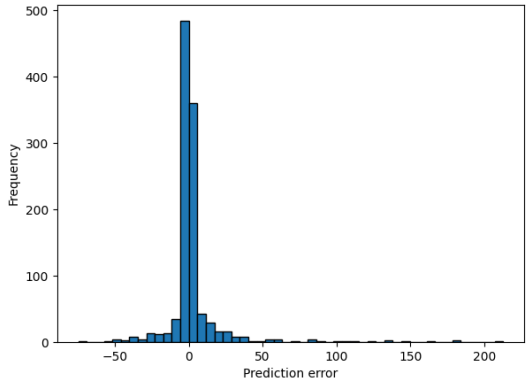
(d) Max Power - Score Tuning: Neg MAE



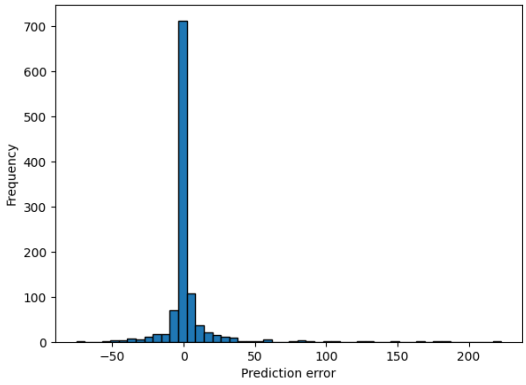
(e) Mean Power - Score Tuning: Neg MSE



(f) Mean Power - Score Tuning: Neg MAE



(g) Runtime - Score Tuning: Neg MSE



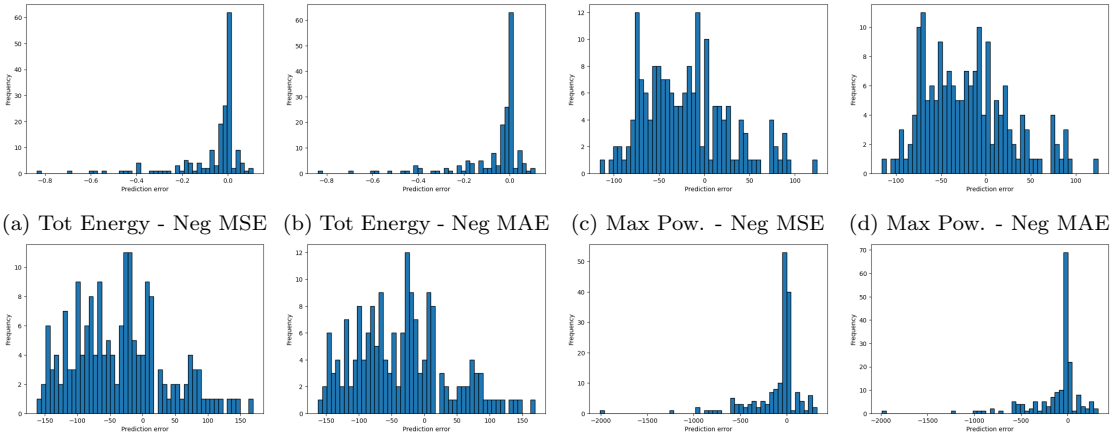
(h) Runtime - Score Tuning: Neg MAE

Figure 4.3: Random Forest error predictions distribution

It can be noted that the majority of cases exhibit values around zero, with higher precision observed for the target variables *Total Energy* and *Runtime*. Specifically, as illustrated in figures 4.3b,4.3d,4.3f,4.3h there is a slight improvement in scoring using *neg_mean_absolute_error* (Neg MAE).

The next page will briefly show the prediction errors divided into single (Figure 4.4) and double (Figure 4.5) precision values.

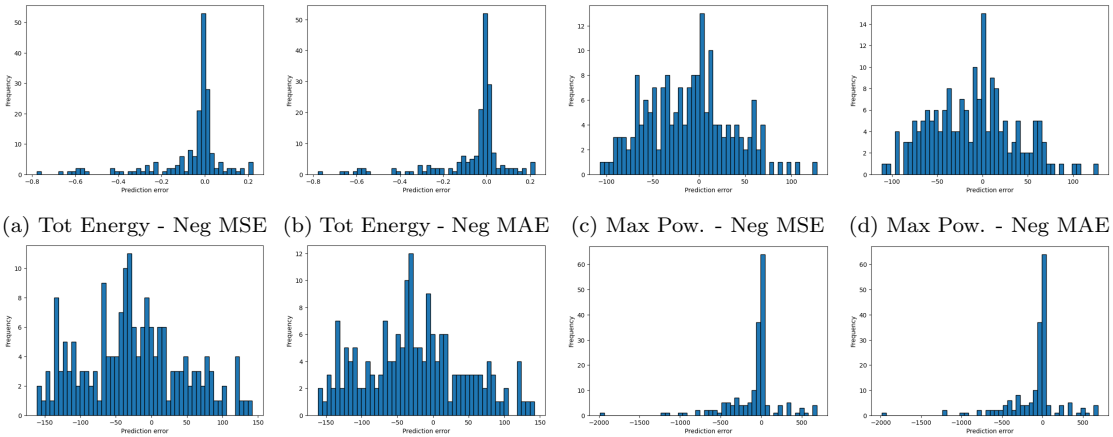
Noteworthy is that, for total energy consumed and runtime, the prediction error shows minimal difference between the single and double precision cases when the model is trained on both precision types. However, a higher accuracy in predicting maximum and average power values per node emerges in the double precision cases.



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure 4.4: Random Forest error predictions distribution in single precision cases



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure 4.5: Random Forest error predictions distribution in double precision cases

For further investigations, it might be interesting to explore the possibility of training two separate models, each for only one precision type. Next, evaluate how the prediction error varies with respect to the results presented in the graphs, considering the two models separately.

Conclusions

The main objective of this project was to analyze the energy performance of two linear solvers, the Inhibition Method and ScaLAPACK, run on a high-performance computing system, considering different configurations. In the course of this analysis, an attempt was made to exploit the opportunity of integrating machine learning techniques to gain benefits by exploiting the predictive capabilities of these approaches.

During the execution of algorithms to solve linear systems, the collection of a large amount of energy data was completed. Communication mechanisms were developed to transfer data from the HPC scheduler to the programs responsible for detection from physical sensors. Subsequently, codes were created to aggregate and organize this data.

The two datasets generated offer comprehensive details on the energy parameters of each job, including individual nodes, memory usage, CPU, cooling system and other relevant aspects. In addition, these datasets make it possible to extract information about groupings of parameters or physical components of the HPC system, such as nodes used for execution and the breakdown of processes by sockets. Energy data were also acquired from software tools such as powercap, with the goal of future verification of the consistency of the information against the precise measurements provided by sensors directly installed on the nodes.

As for the data analysis, although in its early stages, it has already provided

significant answers that further support the adoption of machine learning techniques for energy prediction. The predicted results show much higher accuracy than expected, and regression techniques show a significant correlation between initial job configuration and energy performance. In particular, the Random Forest Regressor proved optimal for the first dataset, with error metrics confirming the overall high accuracy.

Future Developments

Looking to the future, the prospects for developing this work open the way for a number of exciting possibilities. Among the challenges and goals outlined in the following pathway are:

1) **Depth Investigations and More Accurate Analysis:** this could involve specific insights into individual nodes, detailed assessments of memory, CPU, cooling system and node temperature utilization.

2) **Refinement of Machine Learning Techniques:** exploration of machine learning techniques could go further, with the aim of further improving the accuracy of energy predictions. The most crucial features that influence energy performance would be interesting to identify, thus helping to develop more sophisticated predictive models.

3) **Optimization of Parameter Tuning Techniques:** Deepen the study of parameter tuning techniques, with a focus on finding more efficient GridSearch CV approaches. This could lead to more robust models that are adaptable to various job configurations.

4) **Introduction of Neural Networks:** In addition to regressors, introducing neural networks could be a significant next step. Neural networks can cap-

ture complex relationships in the data and could offer superior performance, especially considering the complexity of interactions in HPC systems.

5) **Improving the Scheduling Process:** Use the data collected and estimates provided by artificial intelligence to optimize the scheduling process in HPC systems. This could lead to more efficient management of resources and execution time.

6) **Dataset Expansion and Future Perspectives:** Expanding the dataset is a logical step to improve the representativeness and generalization of the models. Doing so could involve collecting data on different HPC architectures and execution environments.

7) **Wider Applications and Subsequent Projects:** Use the results of this work as the basis for larger projects involving vast amounts of energy data. This could include collaborating with other research projects or applying the knowledge gained in broader contexts, such as system-level energy optimization or designing more efficient algorithms.

Ultimately, the future of this work offers a broad spectrum of opportunities to explore, innovate, and contribute significantly to the field of energy performance in HPC systems.

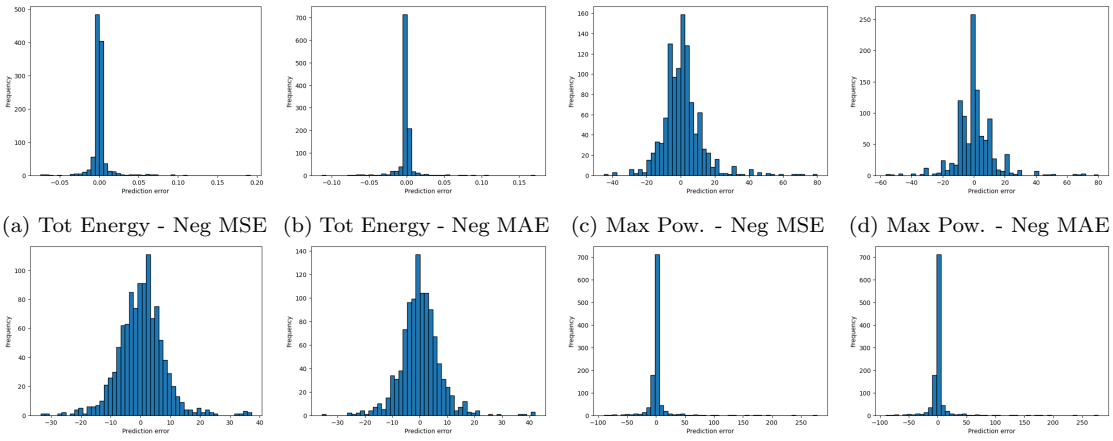
Appendix A

Additional analyses for ideally numerically stable dataset

In this appendix, graphs are presented on the error prediction of the two regressors analyzed in Chapter 4: first, of the classical Decision Tree (subsection A.1.1) and then of the Gradient Boosting Decision Trees model (subsection A.1.2). The graphs illustrate the discrepancy between the actual data and the predictions, divided into three subsets: one for all configurations (resp. Figures A.1 and A.4) without distinction of precision and two more to distinguish single (resp. Figures A.2 and A.5) and double (resp. Figures A.3 and A.6) precisions. It is emphasized that both regressors were trained using a dataset that included both types of precision.

A.1 Prediction error analysis

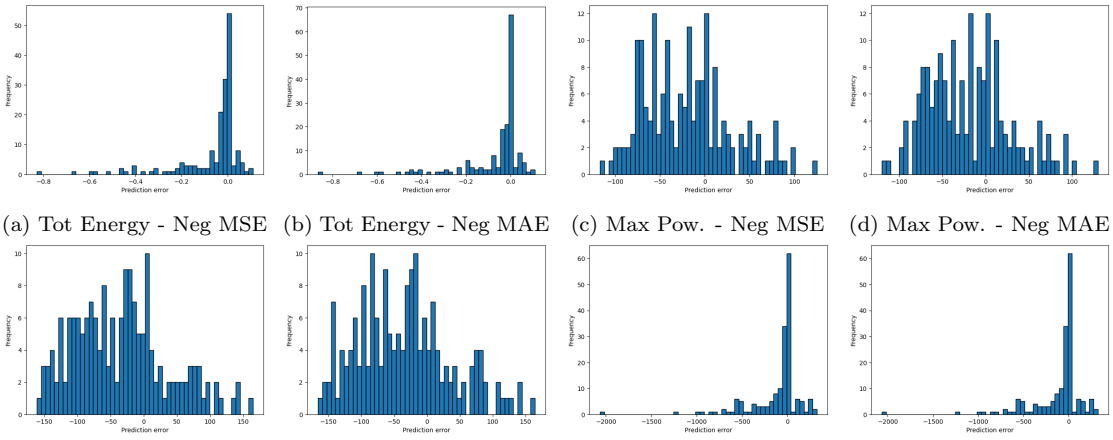
A.1.1 Decision Tree



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

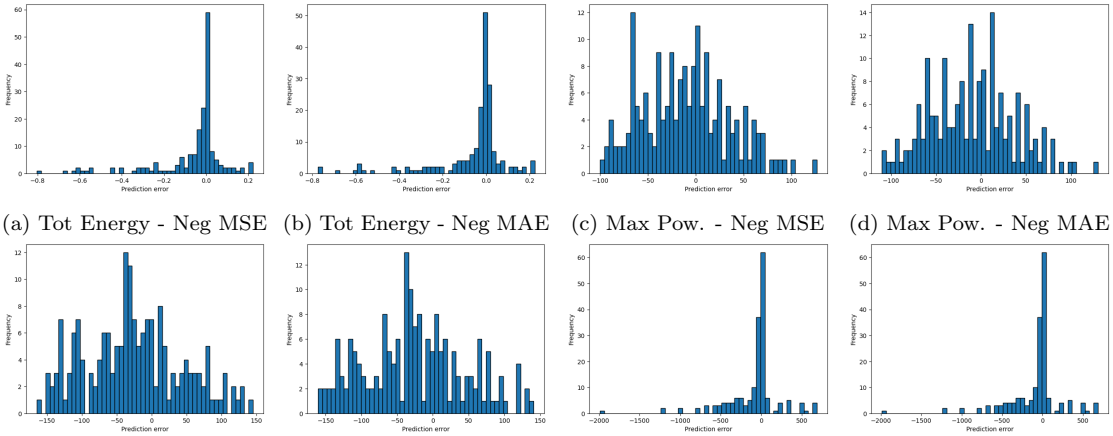
Figure A.1: Decision Tree error predictions distribution



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure A.2: Decision Tree error predictions distribution in single precision cases

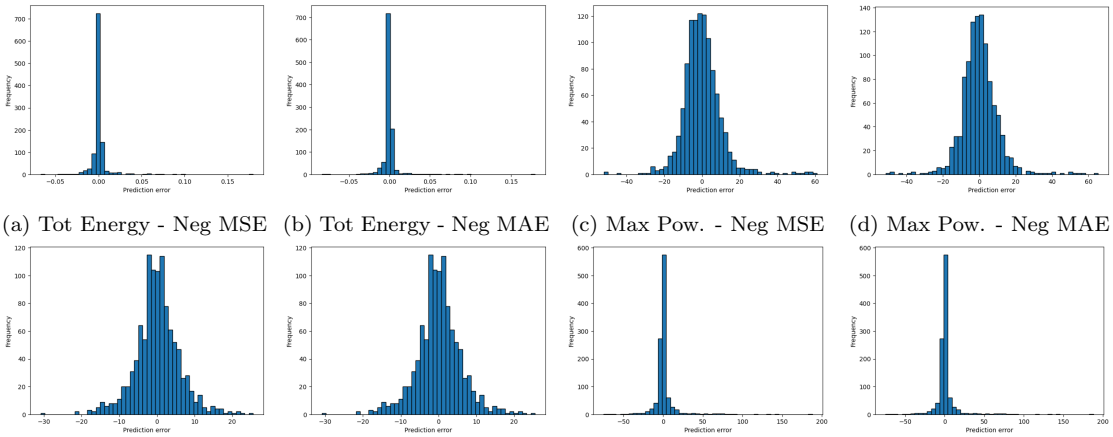


(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure A.3: Decision Tree error predictions distribution in double precision cases

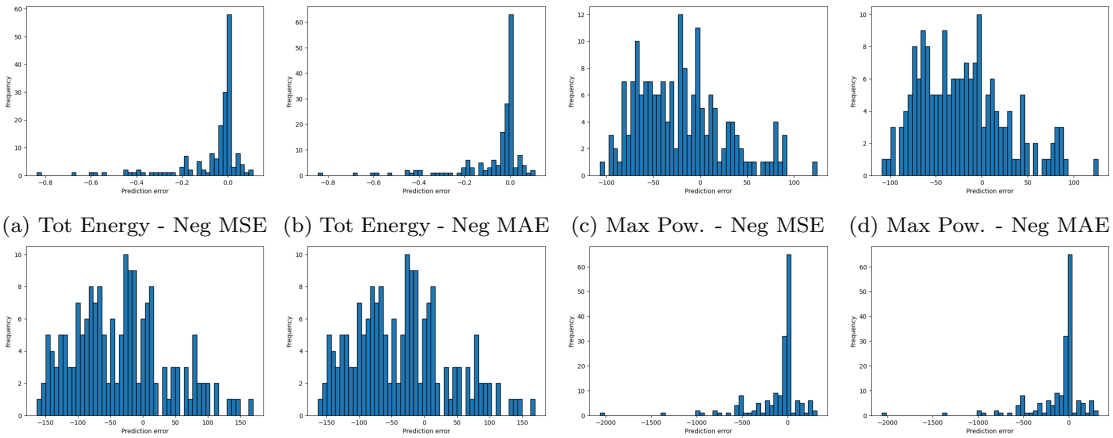
A.1.2 Gradient Boosting Decision Tree



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

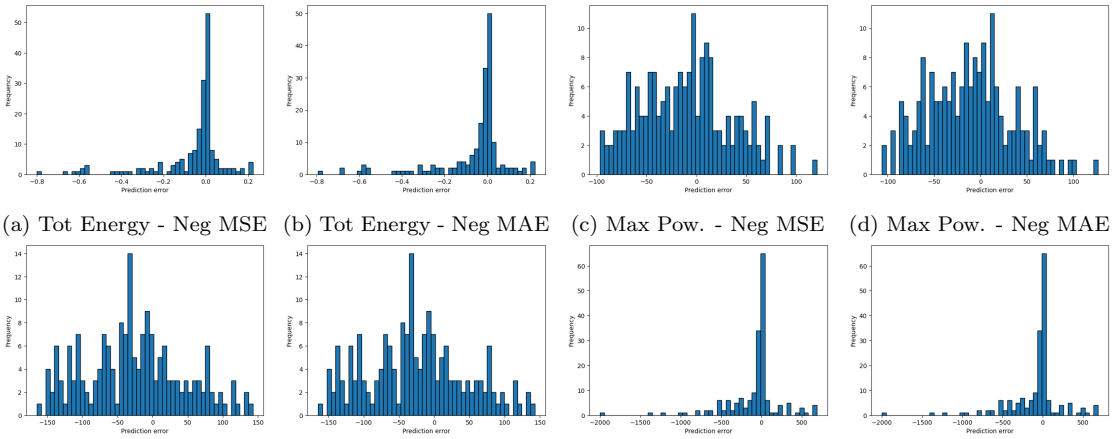
Figure A.4: Gradient Boosting Decision Tree error predictions distribution



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure A.5: Gradient Boosting Decision Tree error predictions distribution in single precision cases



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure A.6: Gradient Boosting Decision Tree error predictions distribution in double precision cases

Appendix B

Dataset analysis with real numerical stability conditions

This Appendix provides data from the same analyses carried out in Chapter 4 and Appendix A for the second dataset created, namely, the one featuring matrices with realistic numerical stability conditions.

The availability of this data is bound to be very useful for future comparative analyses, not only with the first dataset but also to explore other critical aspects in performance evaluation, particularly in the context of more realistic and complex scenario simulations.

B.1 Data Exploration

B.1.1 Data Preprocessing

As both datasets are structured in the same way, the same data preprocessing described in Paragraph 4.1.1 has been applied.

B.1.2 Data Description

Fields \ Index	Mean	Std	Min	25%	50%	75%	Max
ALGORITHM	0.5	0.5000694589153869	0.0	0.0	0.5	1.0	1.0
PRECISION	0.49972222222222223	0.5000693817441682	0.0	0.0	0.0	1.0	1.0
FT	3.335277777777778	2.790066914550512	0.0	1.0	2.0	4.0	8.0
NF	1.6666666666666667	2.5823475817683295	0.0	0.0	0.0	2.0	8.0
CPROCS	314.0	181.08381680308827	64.0	144.0	256.0	484.0	576.0
SPROCS	29.735277777777778	45.731954945898835	0.0	2.0	8.0	40.0	192.0
TPROCS	343.73527777777775	195.41113654720175	64.0	148.0	320.0	492.0	768.0
MATSIZE	22176.0	11760.751885507818	5280.0	10560.0	21120.0	31680.0	42240.0
NODES	7.668888888888889	4.023731890431845	2.0	4.0	7.0	11.0	16.0
SPK-CP	4928.0	6762.638515313564	0.0	0.0	0.0	10560.0	21120.0
BKF	10.382222222222222	11.630531318246515	0.0	0.0	0.0	24.0	25.0
BALANCE	0.5002777777777778	0.5000693817441682	0.0	0.0	1.0	1.0	1.0
NUMxSOCK	10.998333333333333	11.111998851572782	0.0	0.0	16.0	23.0	24.0
TOTAL_ENERGY	0.07588855555555556	0.14034929283605288	0.00043	0.0057875	0.016425000000000002	0.0674275	0.99293
MAX_SYS_POWER	332.69444444444446	37.124573034626316	190.0	310.0	340.0	360.0	810.0
MEAN_SYS_POWER	237.13445114722222	50.17862726176367	111.25	194.42708	248.238925	281.280795	323.29204
RUNTIME_S	140.45333333333335	260.5358063824153	6.0	15.0	36.0	128.25	2367.0

Table B.1: Description of the dataset data with real numerical stability condition

B.1.3 Boxplots

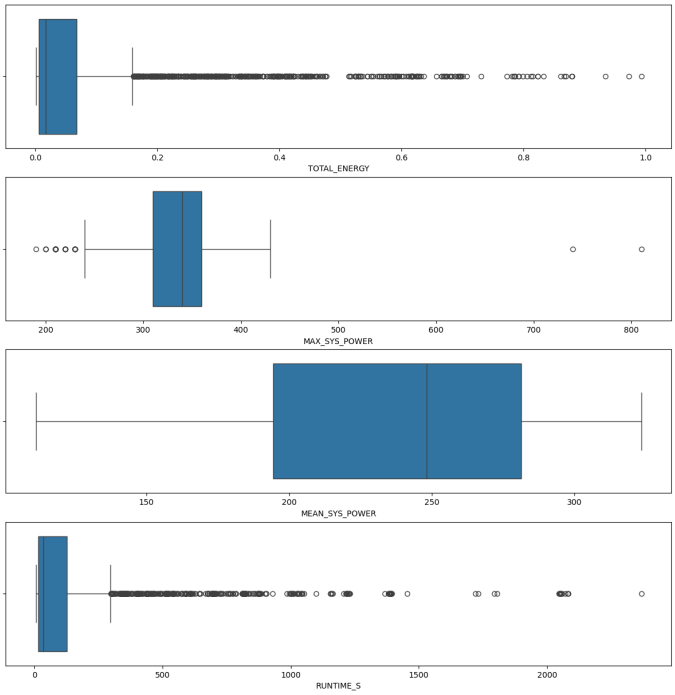


Figure B.1: Boxplots of total energy consumption, maximum and average power for each node used and runtime

B.1.4 Histograms target

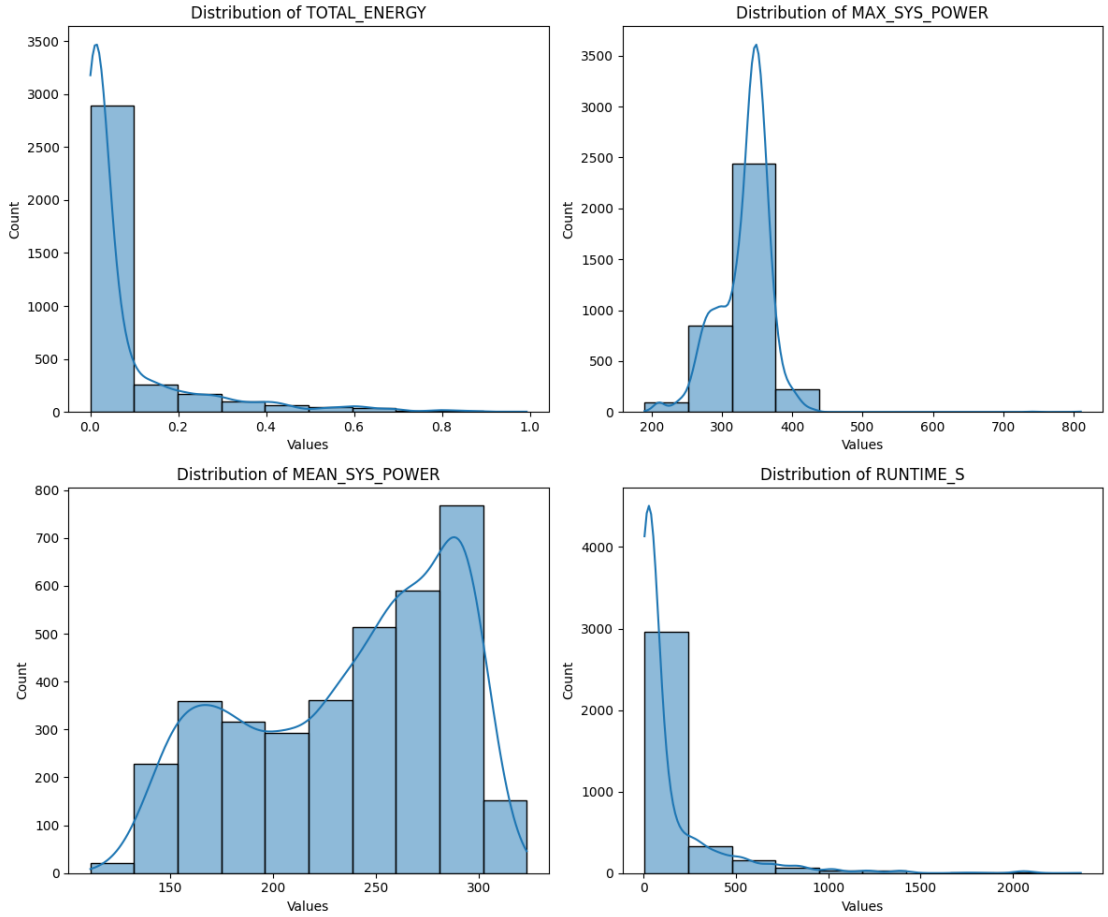


Figure B.2: Distributions of Energy Performance: Total Consumption, Peak Power, Avg. Power, and Runtime

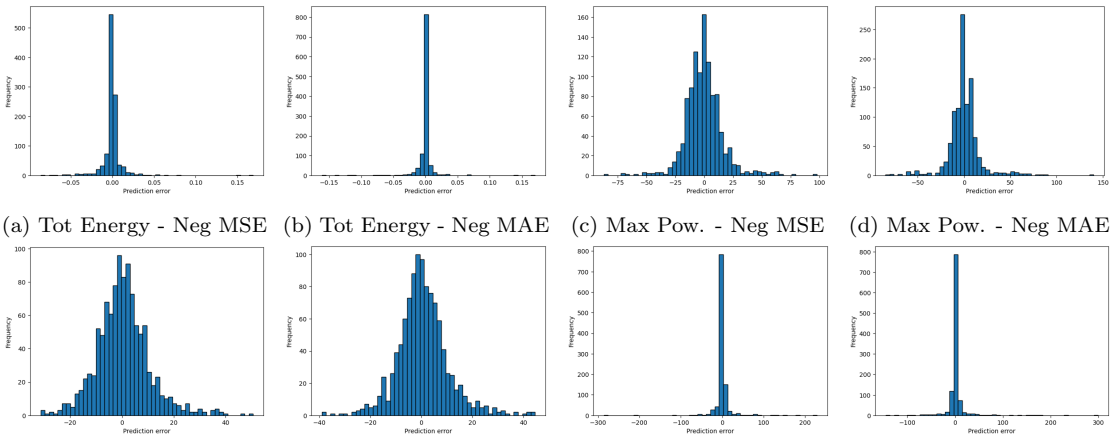
B.2 Regressors Evaluation

TARGET: TOTAL_ENERGY (Min value : 0.000430 kWh - Max value: 0.933830 kWh)					
REGRESSOR	**TYPE SCORING GRID SEARCH CV**	**BEST DEPTH**	**RMSE Root Mean Squared Error**	**MAE Mean Absolute Error**	**MAPE Mean Absolute Percentage Error**
Decision Tree	neg_mean_squared_error	6	0.012677891758296266	0.005380234815344958	0.21626462919047512
	neg_mean_absolute_error	9	0.013818456161842631	0.004229301260162477	0.08283302086508638
Random Forest	neg_mean_squared_error	7	0.010510454294855502	0.003923716804769786	0.11082853754088962
	neg_mean_absolute_error	14	0.01029298495650228	0.0034602339076107913	0.06465906203724821
Gradient Boosting Decision Trees	neg_mean_squared_error	4	0.009471948138803653	0.00348564658335038	0.18117993501409718
	neg_mean_absolute_error	5	0.009841154065001169	0.0032789628119311138	0.10917922830372573
TARGET: MAX_POWER (Min value : 190.000000 W - Max value: 430.000000 W)					
REGRESSOR	**TYPE SCORING GRID SEARCH CV**	**BEST DEPTH**	**RMSE Root Mean Squared Error**	**MAE Mean Absolute Error**	**MAPE Mean Absolute Percentage Error**
Decision Tree	neg_mean_squared_error	6	15.950352526764423	11.012386004972942	0.0341947511029825
	neg_mean_absolute_error	8	17.422814022241557	10.542704500572725	0.03259291985974746
Random Forest	neg_mean_squared_error	7	15.56142791778739	10.252386523263118	0.03164025906615516
	neg_mean_absolute_error	8	16.079502853698937	10.15436820219703	0.031303663043796066
Gradient Boosting Decision Trees	neg_mean_squared_error	4	16.34298004700934	10.904020415733665	0.03353489836385893
	neg_mean_absolute_error	5	16.80221003205115	10.710370092813783	0.03292080902937197
TARGET: MEAN_POWER (Min value : 111.250000 W - Max value: 321.366160 W)					
REGRESSOR	**TYPE SCORING GRID SEARCH CV**	**BEST DEPTH**	**RMSE Root Mean Squared Error**	**MAE Mean Absolute Error**	**MAPE Mean Absolute Percentage Error**
Decision Tree	neg_mean_squared_error	7	10.284141520130754	7.523980357051382	0.03480608007643323
	neg_mean_absolute_error	8	10.027103004959717	7.252885337234795	0.03407978056750638
Random Forest	neg_mean_squared_error	9	8.48625552480795	6.1457455966687125	0.0293638953141297
	neg_mean_absolute_error	10	8.556555371124416	6.076015524637254	0.02914376353848104
Gradient Boosting Decision Trees	neg_mean_squared_error	5	7.712039392436465	5.690627771000274	0.027524888248922054
	neg_mean_absolute_error	5	7.712039392436465	5.690627771000274	0.027524888248922054
TARGET: RUNTIME (Min value : 6.000000 s - Max value: 1015.000000 s)					
REGRESSOR	**TYPE SCORING GRID SEARCH CV**	**BEST DEPTH**	**RMSE Root Mean Squared Error**	**MAE Mean Absolute Error**	**MAPE Mean Absolute Percentage Error**
Decision Tree	neg_mean_squared_error	9	21.410858174061943	6.767014365546298	0.06487049973710668
	neg_mean_absolute_error	17	22.28386127337515	6.92530864197531	0.07214101695964685
Random Forest	neg_mean_squared_error	14	20.235359549239764	6.857134592887136	0.06371312433570141
	neg_mean_absolute_error	14	20.235359549239764	6.857134592887136	0.06371312433570141
Gradient Boosting Decision Trees	neg_mean_squared_error	6	16.182932593757208	6.086177888879056	0.07922660739897572
	neg_mean_absolute_error	6	16.182932593757208	6.086177888879056	0.07922660739897572

Table B.2: Summary table of regressor prediction evaluation on the second dataset

B.3 Prediction error analysis

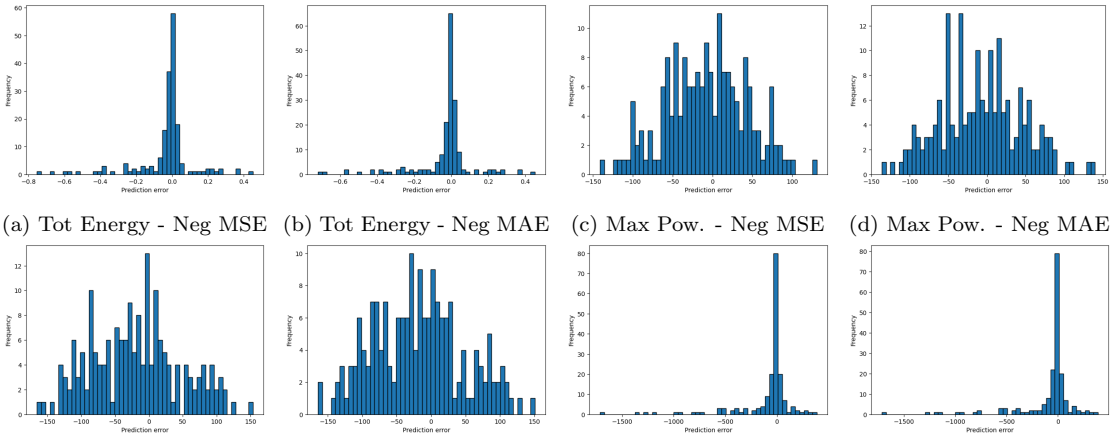
B.3.1 Decision Tree



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

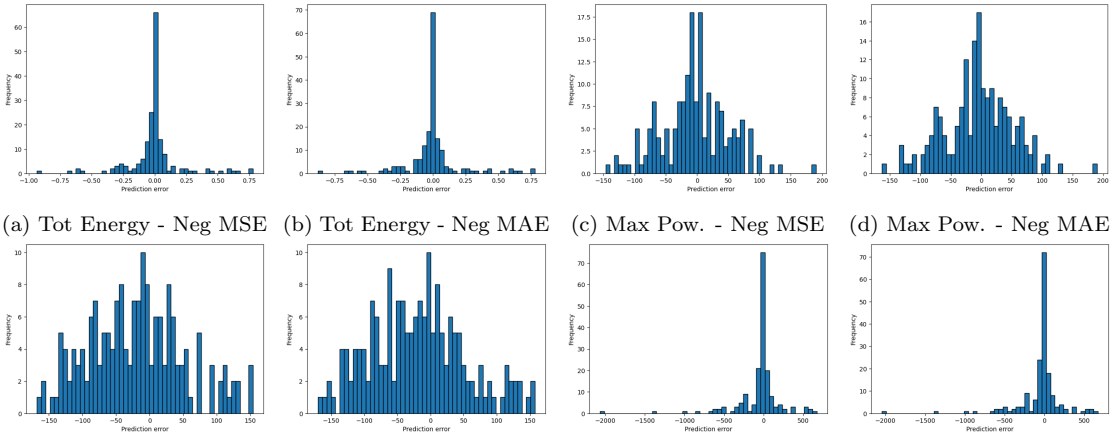
Figure B.3: Decision Tree error predictions distribution



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure B.4: Decision Tree error predictions distribution in single precision cases

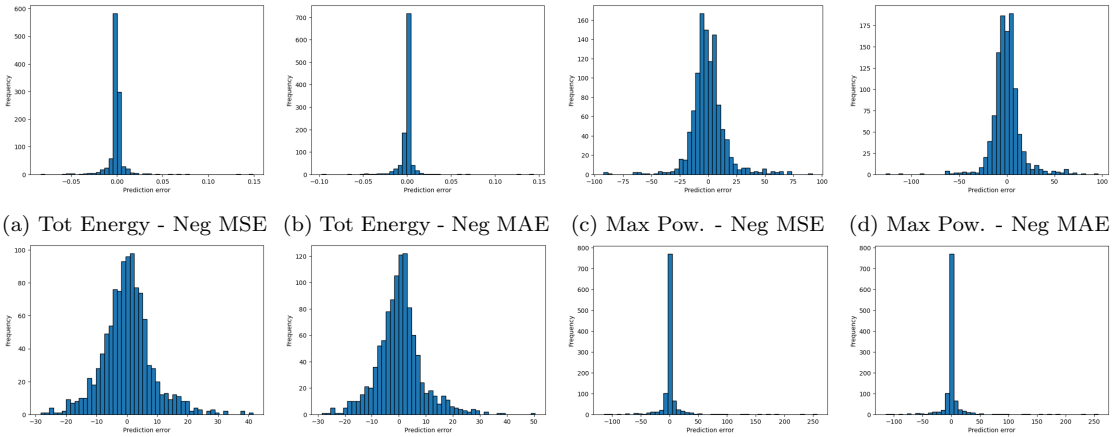


(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure B.5: Decision Tree error predictions distribution in double precision cases

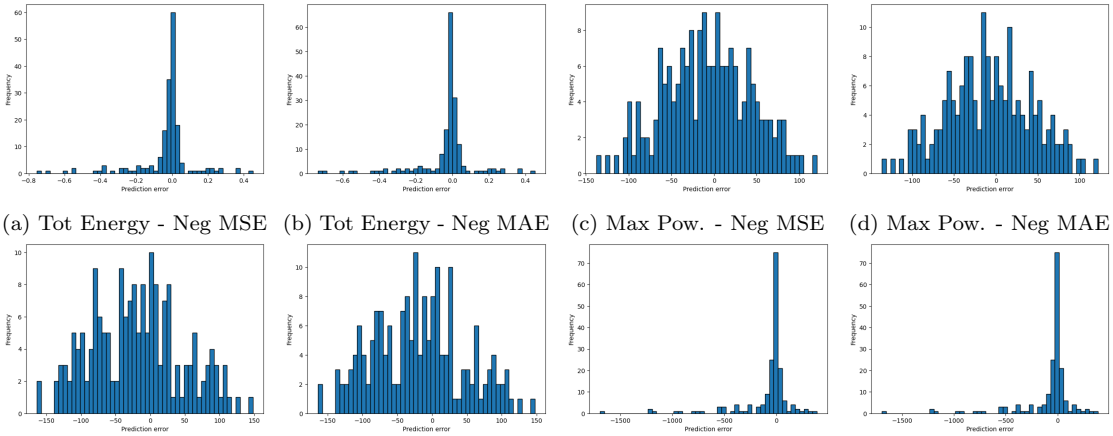
B.3.2 Random Forest



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

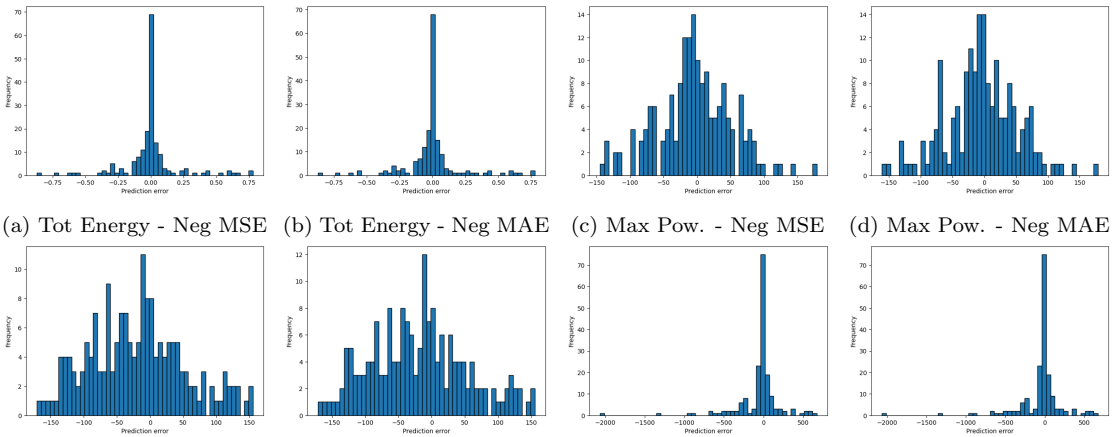
Figure B.6: Random Forest error predictions distribution



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure B.7: Random forest error predictions distribution in single precision cases

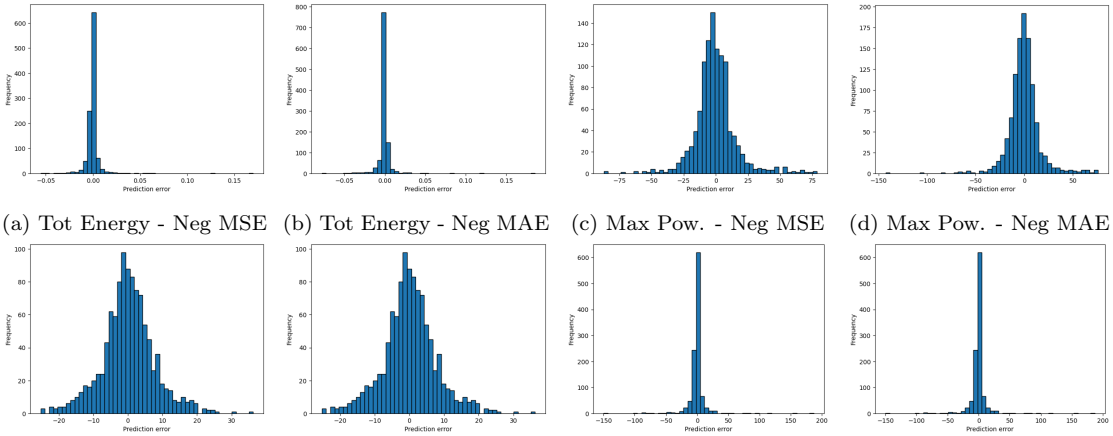


(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure B.8: Random Forest error predictions distribution in double precision cases

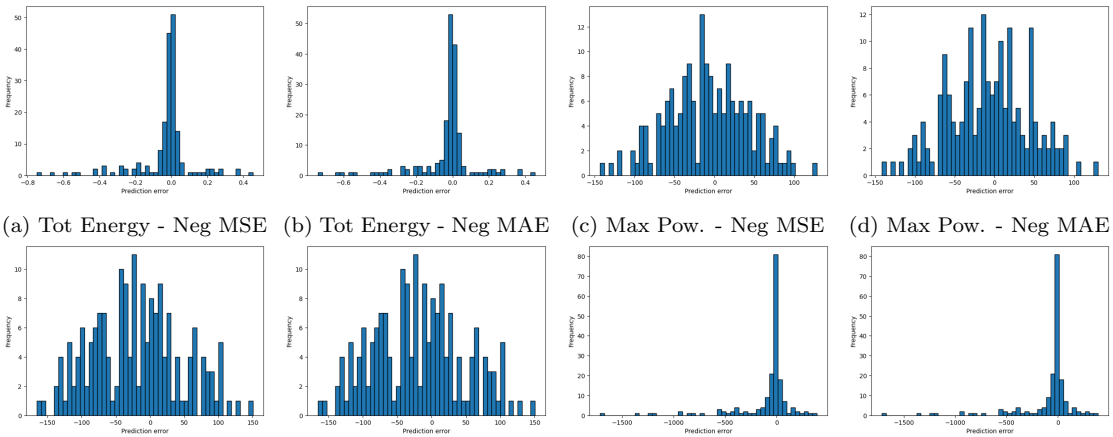
B.3.3 Gradient Boosting Decision Tree



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

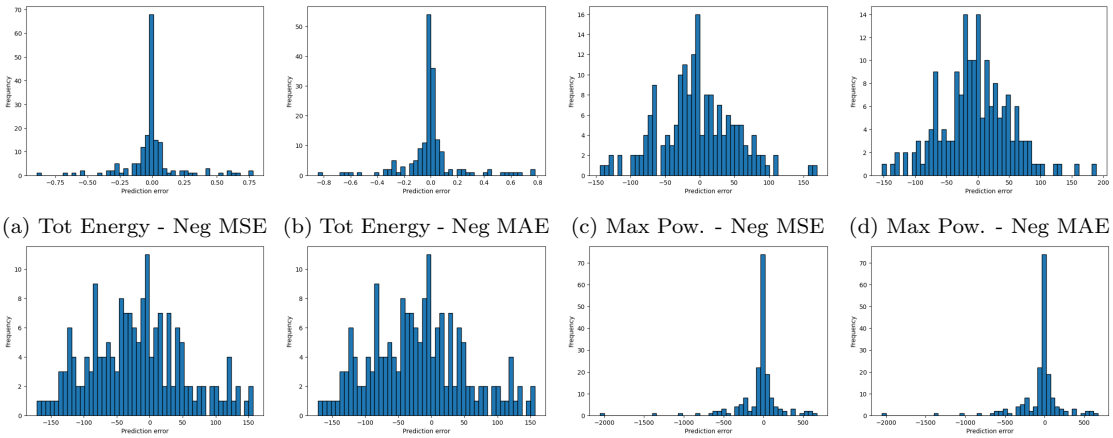
Figure B.9: Gradient Boosting Decision Tree error predictions distribution



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure B.10: Gradient Boosting Decision Tree error predictions distribution in single precision cases



(a) Tot Energy - Neg MSE (b) Tot Energy - Neg MAE (c) Max Pow. - Neg MSE (d) Max Pow. - Neg MAE

(e) Mean Pow. - Neg MSE (f) Mean Pow. - Neg MAE (g) Runtime - Neg MSE (h) Runtime - Neg MAE

Figure B.11: Gradient Boosting Decision Tree error predictions distribution in double precision cases

Bibliography

[1] Kenneth O'brien et al. "A Survey of Power and Energy Predictive Models in HPC Systems and Applications". *ACM Comput. Surv.* 50.3 (June 2017). ISSN: 0360-0300. DOI: 10.1145/3078811.

[2] Pierre-François Dutot et al. "Towards Energy Budget Control in HPC". *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2017, pp. 381–390. DOI: 10.1109/CCGRID.2017.16.

[3] *TOP500 - The List*. Online. Accessed: 2024-01-21. URL: https://www.top500.org/.

[4] Dan Zhao et al. "A Green(er) World for A.I." *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2022, pp. 742–750. DOI: 10.1109/IPDPSW55747.2022.00126.

[5] F. Ciampolini. "Un metodo di soluzione dei circuiti lineari". L.10 (1963).

[6] F. Filippetti and M. Artioli. "IMe: 4-term formula method for the symbolic analysis of linear circuits". *IEEE Transactions on Circuits and Systems* 51-I.3 (2004), pp. 526–538. DOI: 10.1109/TCSI.2003.822374.

[7] M. Artioli. "Symbolic techniques addressed to electric circuit analysis and diagnosis". Ph.D. dissertation. University of Bologna, Mar. 2001.

[8] Daniela Loreti Loreti, Marcello Artioli, and Anna Ciampolini. "Solving Linear Systems on High Performance Hardware with Resilience to Multiple Hard Faults". *2020 International Symposium on Reliable Distributed Systems (SRDS)*. Shanghai, China, 2020, pp. 266–275. DOI: 10.1109/SRDS51746.2020.00034.

[9] Daniela Loreti, Marcello Artioli, and Anna Ciampolini. "Rollback-free recovery for a high performance dense linear solver with reduced memory footprint". *Submitted to IEEE Transaction on Parallel and Distributed Systems*. 2024.

[10] J.S. Plank, Kai Li, and M.A. Puening. "Diskless checkpointing". *IEEE Transactions on Parallel and Distributed Systems* 9.10 (Oct. 1998), pp. 972–986. DOI: 10.1109/71.730527.

[11] James S. Plank, Youngbae Kim, and Jack J. Dongarra. "Fault-Tolerant Matrix Operations for Networks of Workstations Using Diskless Checkpointing". *Journal of Parallel and Distributed Computing* 43 (Sept. 1997), pp. 125–138.

[12] Y. Kim, J. S. Plank, and J. J. Dongarra. "Fault Tolerant Matrix Operations for Networks of Workstations Using Multiple Checkpointing". *High Performance Computing on the Information Superhighway, HPC Asia '97*. Seoul, Korea, Apr. 1997, pp. 460–465.

[13] Z. Chen. "Scalable Techniques for Fault Tolerant High Performance Computing". Ph.D. dissertation. Knoxville, TN, USA: University of Tennessee, 2006.

[14] Marcello Artioli, Daniela Loreti, and Anna Ciampolini. "Fault Tolerant High Performance Solver for Linear Equation Systems". *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. Lyon, France, 2019, pp. 113–11309. DOI: 10.1109/SRDS47363.2019.00022.

[15] Pramila P. Shinde and Seema Shah. "A Review of Machine Learning and Deep Learning Applications". *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. 2018, pp. 1–6. DOI: 10.1109/ICCUBEA.2018.8697857.

[16] Leo Breiman et al. *Classification and regression trees*. Wadsworth, 1984.

[17] Rosaria Silipo. "Ensemble models: Bagging & Boosting - Analytics Vidhya - Medium". en. *Medium* (Dec. 2021). URL: https://medium.com/analytics-vidhya/ensemble-models-bagging-boosting-c33706db0b0b.

[18] *Scikit-learn: Machine Learning in Python*. Online. Accessed: 2024-01-17. 2023. URL: https://scikit-learn.org/stable/user_guide.html.

[19] Padhma M. *A comprehensive introduction to evaluating Regression Models*. Online. Nov. 2023. URL: https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/.

[20] *Scikit-learn: Tuning the hyperparameters of an estimator*. Online. Accessed: 2024-01-18. 2023. URL: https://scikit-learn.org/stable/modules/grid_search.html.

[21] *CRESCO - Computational RESearch Centre on COmplex Systems*. Online. Accessed: 2024-01-15. URL: https://www.afs.enea.it/project/eneagrid/Resources_en/CRESCO_documents/index.html.

List of Figures

1.1 Graphical visualization of the fundamental formula and the computation of the auxiliary quantities. (*Source:* [8]) 6
1.2 Inhibition Table alternative form.(*Source:* [9]) 7
1.3 Evolution of IMe data structures when compressing $\boldsymbol{E}$ and $\boldsymbol{K}$ into a single matrix $\boldsymbol{V}$. (*Source:* [9]) 8
1.4 2D block-cyclic distribution of $\boldsymbol{V}$ and $\boldsymbol{S}$ on a 3×4 and 3×2 processor grid, respectively. (*Source:* [9]) 14

2.1 Single Decision Tree scheme (*Source:* [17]) 18
2.2 Example of a regression tree diagram (*Source:* [18]) 21
2.3 Random Forest scheme (*Source:* [17]) 22
2.4 Gradient Boosting Decision Tree scheme (*Source:* [17]) 23

3.1 Structure diagram of a single node belonging to the CRESCO6 cluster (*Source:* [21]) 32
3.2 Job Execution Mechanism 35

4.1 Boxplots of total energy consumption, maximum and average power for each node used and runtime 58
4.2 Distributions of Energy Performance: Total Consumption, Peak Power, Avg. Power, and Runtime 59

4.3 Random Forest error predictions distribution 63
4.4 Random Forest error predictions distribution in single precision cases 64
4.5 Random Forest error predictions distribution in double precision cases 64

A.1 Decision Tree error predictions distribution 69
A.2 Decision Tree error predictions distribution in single precision cases 69
A.3 Decision Tree error predictions distribution in double precision cases 70
A.4 Gradient Boosting Decision Tree error predictions distribution . . . 70
A.5 Gradient Boosting Decision Tree error predictions distribution in single precision cases 71
A.6 Gradient Boosting Decision Tree error predictions distribution in double precision cases 71

B.1 Boxplots of total energy consumption, maximum and average power for each node used and runtime 73
B.2 Distributions of Energy Performance: Total Consumption, Peak Power, Avg. Power, and Runtime 74
B.3 Decision Tree error predictions distribution 76
B.4 Decision Tree error predictions distribution in single precision cases 76
B.5 Decision Tree error predictions distribution in double precision cases 77
B.6 Random Forest error predictions distribution 77
B.7 Random forest error predictions distribution in single precision cases 78
B.8 Random Forest error predictions distribution in double precision cases 78
B.9 Gradient Boosting Decision Tree error predictions distribution . . . 79
B.10 Gradient Boosting Decision Tree error predictions distribution in single precision cases 79
B.11 Gradient Boosting Decision Tree error predictions distribution in double precision cases 80

List of Tables

1.1 IMe prescribed steps to compute the system's solution. 6

3.1 Summary of jobs in a dataset 53

4.1 Description of the dataset data with ideal numerical stability condition 56

4.2 Summary table of regressor prediction evaluation on the first dataset 61

B.1 Description of the dataset data with real numerical stability condition 73

B.2 Summary table of regressor prediction evaluation on the second dataset 75

Acknowledgements

First and foremost, I would like to express my gratitude to Professor Anna Ciampolini and Professor Daniela Loreti for their essential support and assistance in the completion of this thesis, marking the conclusion of my master's degree.

I also extend my thanks to Eng. Marcello Artioli and Dr. Davide De Chiara from ENEA, who were consistently available during my internship leading up to this thesis and throughout its writing, providing valuable guidance and knowledge.

Special mention goes to Professor Andrea Borghesi for his assistance, especially in the data analysis within the context of AI, and to Dr. Marta Chinnici for her administrative and bureaucratic support.

A simple thank you is not enough, but I want to express my gratitude to my parents, my brother, my grandparents, my aunts, uncles, and cousins, who have supported me morally and economically.

To my friends Giordano and Michele, Flavio, and all my other friends from Molise, thank you for your support and for sharing beautiful moments together whenever we meet, despite being scattered across Italy.

In Bologna, I want to acknowledge the many people who have enriched my life:

To Alessandra, one of the first dear people I met in Bologna.

To Francesca, Alin, and Ada, who have become some of my closest friends.

To Generoso, for enduring and supporting me through countless moments.

To Marco ("Disi"), for the lengthy conversations and advice.

To Giuseppe ("Peppone"), whose charisma makes it impossible not to become friends.

To Aldo, Alessandro, and Enrico, with whom I shared the emotions and disappointments of our great passion, Formula 1, every Sunday, besides our studies.

To Fabiana, who is always there for me, has taught me so much and is without a doubt one of the most important people I know.

To the "O' sole mio" group, as I owe a lot to you for getting me here.

Thanks to my other close group of friends: Antonio, Federica, Filippo, Gabriele, Giacomo, Giorgia, Guglielmo, Stefano, and Lorenzo.

I also extend my thanks to all the other computer engineers I've met in these 5 and half years.

Now, it's time to acknowledge my second family: UniOne study room. I want to mention a few names (in no particular order): Riccardo, Francesca, Anixia, Serena, Emanuela, Gloria, Ilaria, Saverio, Sara, Carlotta, Matilde, Rachele, Beniamino, Andrea, Federica and many others. Even if not mentioned, you are all in my heart.

Moreover, I would like to thank the staff at UniOne, Rita, Roberto, Nicoletta, and Gerardo, for making me feel at the study room literally like home.

As I approach the conclusion, I want to express gratitude to those who shared the AlmaEnglish journey with me, particularly Arianna and Roberta.

Over the years I've met people from other engineering courses and from various faculties in general. Others I met during a week's vacation in Barcelona. There are so many of you that I cannot name them all. Perhaps I forgot to mention someone, and with this sentence I take the opportunity to thank you.

As one chapter closes, I hope you'll be present in the ones that follow.

This marks the end of the thesis, and I thank you all.