Accelerating Level-1 Cache Design Space Exploration using Machine Learning Approach for Miss Rate and Power Prediction

A Thesis submitted to Gujarat Technological University

for the Award of

Doctor of Philosophy

in

Electronics and Communication Engineering

by

Dave Hetal Vinubhai

[Enrollment No. 209999915011]

under the supervision of

Dr. Nirali A. Kotak

GUJARAT TECHNOLOGICAL UNIVERSITY
AHMEDABAD, GUJARAT, INDIA

[August – 2026]

Accelerating Level-1 Cache Design Space Exploration using Machine Learning Approach for Miss Rate and Power Prediction

A Thesis submitted to Gujarat Technological University

for the Award of

Doctor of Philosophy

in

Electronics and Communication Engineering

by

Dave Hetal Vinubhai

[Enrollment No. 209999915011]

under the supervision of

Dr. Nirali A. Kotak

GUJARAT TECHNOLOGICAL UNIVERSITY
AHMEDABAD, GUJARAT, INDIA

[August – 2026]

Declaration

I declare that the thesis entitled **"Accelerating Level-1 Cache Design Space Exploration using Machine Learning Approach for Miss Rate and Power Prediction"** submitted by me for the degree of Doctor of Philosophy is the record of research work carried out by me during the period from **February 2021 to August 2026** under the supervision of **Dr. Nirali A. Kotak**, Assistant Professor, L.D. College of Engineering, Ahmedabad (India), and this has not formed the basis for the award of any degree, diploma, associateship, fellowship, or titles in this or any other University or other institution of higher learning.

I further declare that the material obtained from other sources has been duly acknowledged in the thesis. I shall be solely responsible for any plagiarism or other irregularities, if noticed in the thesis.

Signature of the Research Scholar:  Date: **07/08/2026**

Name of Research Scholar: **Dave Hetal Vinubhai**

Place: Ahmedabad

Certificate

I certify that the work incorporated in the thesis **"Accelerating Level-1 Cache Design Space Exploration using Machine Learning Approach for Miss Rate and Power Prediction"** submitted by **Dave Hetal Vinubhai** was carried out by the candidate under my supervision/guidance. To the best of my knowledge: (i) the candidate has not submitted the same research work to any other institution for any degree/diploma, Associateship, Fellowship or other similar titles (ii) the thesis submitted is a record of original research work done by the Research Scholar during the period of study under my supervision, and (iii) the thesis represents independent research work on the part of the Research Scholar.

Signature of the Research Supervisor: Date: **07/08/2026**

Name of Supervisor: **Dr. Nirali A. Kotak**

Place: Ahmedabad

Coursework Completion Certificate

This is to certify that **Dave Hetal Vinubhai,** Enrollment No. **209999915011,** is enrolled in the PhD program in the faculty of **Electronics and Communication Engineering** of Gujarat Technological University, Ahmedabad.

(Please tick the relevant option(s))

☐ He/She has been exempted from the coursework (successfully completed during the M.Phil Course)

☐ He/She has been exempted from the Research Methodology Course only (successfully completed during the M.Phil Course)

☑ He/She has successfully completed the PhD coursework for the partial requirement for the award of a PhD Degree. His/ Her performance in the coursework is as follows-

Grade Obtained in Research Methodology [PHD20-01]	Grade Obtained in Research and Publication Ethics [PHD20-02]	Grade Obtained in Self-Study Course/ Contact Program [PHD20-03]
BC	AB	AA

Supervisor's Sign

(Dr. Nirali A. Kotak)

Date: **07/08/2026**

Originality Report Certificate

It is certified that the PhD Thesis titled **"Accelerating Level-1 Cache Design Space Exploration using Machine Learning Approach for Miss Rate and Power Prediction"** by **Dave Hetal Vinubhai** has been examined by us. We undertake the following:

a. Thesis has significant new work/knowledge as compared to already published or under consideration to be published elsewhere. No sentence, equation, diagram, table, paragraph, or section has been copied verbatim from previous work unless it is placed under quotation marks and duly referenced.

b. The work presented is original and the work of the author (i.e., there is no plagiarism). No ideas, processes, results, or words of others have been presented as the Author's own work.

c. There is no fabrication of data or results that have been compiled/analyzed.

d. There is no falsification by manipulating research materials, equipment, or processes, or changing or omitting data or results such that the research is not accurately represented in the research record.

e. The thesis has been checked using **Turnitin** Software (copy of originality report attached) and found within limits as per GTU Plagiarism Policy and instructions issued from time to time (i.e., permitted similarity index <=10%).

Signature of the Research Scholar:  Date: **07/08/2026**

Name of the Research Scholar: **Dave Hetal Vinubhai**

Signature of the Research Supervisor:  Date: **07/08/2026**

Name of the Research Supervisor: **Dr. Nirali A. Kotak**

Place: Ahmedabad

Hetal Dave

Ph.D. Thesis

Quick Submit

Quick Submit

Gujarat Technological University

Document Details

Submission ID
trn:oid:::1:3540660422

Submission Date
Apr 17, 2026, 2:04 PM GMT+5:30

Download Date
Apr 17, 2026, 2:06 PM GMT+5:30

File Name
Thesis_Hetal_Dave_2099999150111.pdf

File Size
6.3 MB

121 Pages

29,156 Words

160,814 Characters

7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text
- Small Matches (less than 9 words)

Match Groups

- 167 Not Cited or Quoted 7%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

5% Internet sources
4% Publications
1% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Ph.D. Thesis Non-Exclusive License to

GUJARAT TECHNOLOGICAL

UNIVERSITY

In consideration of being a Research Scholar at Gujarat Technological University, and in the interests of the facilitation of research at the University and elsewhere, I, **Dave Hetal Vinubhai**, having **209999915011**, hereby grant a non-exclusive, royalty-free, and perpetual license to the University on the following terms:

a. The University is permitted to archive, reproduce, and distribute my thesis, in whole or in part, and/or my abstract, in whole or in part (referred to collectively as the "Work") anywhere in the world, for non-commercial purposes, in all forms of media;

b. The University is permitted to authorize, sub-lease, sub-contract, or procure any of the acts mentioned in paragraph (a);

c. The University is authorized to submit the work at any National / International Library, under the authority of their "Thesis Non-Exclusive License";

d. The Universal Copyright Notice (©) shall appear on all copies made under the authority of this license;

e. I undertake to submit my thesis, through my University, to any Library and Archives. Any abstract submitted with the thesis will be considered to form part of the thesis.

f. I represent that my thesis is my original work, does not infringe any rights of others, including privacy rights, and that I have the right to make the grant conferred by this non-exclusive license.

g. If third-party copyrighted material was included in my thesis for which, under the terms of the Copyright Act, written permission from the copyright owners is required, I have obtained such permission from the copyright owners to do the acts mentioned in paragraph (a) above for the full term of copyright protection.

h. I understand that the responsibility for the matter as mentioned in paragraph (g) rests with the authors / me solely. In no case shall GTU be liable for any acts/omissions/errors/copyright infringement from the publication of the said thesis or otherwise.

i. I retain copyright ownership and moral rights in my thesis, and may deal with the copyright in my thesis, in any way consistent with rights granted by me to my University in this non-exclusive license.

j. GTU logo shall not be used /printed in the book (in any manner whatsoever), being published, or any promotional or marketing materials, or any such similar documents.

k. The following statement shall be included appropriately and displayed prominently in the book or any material being published anywhere: "The content of the published work is part of the thesis submitted in partial fulfillment of the requirements for the award of the degree of PhD in **Electronics and Communication Engineering** of the Gujarat Technological University."

l. I further promise to inform any person to whom I may hereafter assign or license my copyright in my thesis of the rights granted by me to my University in this nonexclusive license. I shall keep GTU indemnified from any and all claims from the Publisher(s) or any third parties at all times resulting or arising from the publishing, use, or intended use of the book / such similar document or its contents.

m. I am aware of and agree to accept the conditions and regulations of Ph.D., including all policy matters related to authorship and plagiarism.

Date: **07/08/2026**

Place: Ahmedabad

Signature of the Research Scholar

Recommendation of the Research Supervisor: ---------- Recommended

Signature of the Research Supervisor

Thesis Approval Form

The viva-voce of the PhD Thesis submitted by **Dave Hetal Vinubhai** (Enrollment No. **209999915011**), entitled **"Accelerating Level-1 Cache Design Space Exploration using Machine Learning Approach for Miss Rate and Power Prediction"**, was conducted on 07/08/2026 (Friday) at Gujarat Technological University.

(Please tick any one of the following options)

☑ The performance of the candidate was satisfactory. We recommend that they be awarded the PhD degree.

☐ Any further modifications in research work recommended by the panel after 3 months from the date of the first viva-voce, upon request of the Supervisor or request of the Independent Research Scholar, after which the viva-voce can be re-conducted by the same panel again.

(briefly specify the modifications suggested by the panel)

☐ The performance of the candidate was unsatisfactory. We recommend that they should not be awarded the PhD degree.

(The panel must give justifications for rejecting the research work)

(Dr. Nirali A. Kotak)
Name and Signature of Supervisor with Seal

(Dr. Bijayananda Patnaik)
External Examiner 1

(Dr. Ameer P. M.)
External Examiner 2

Abstract

Modern devices must be developed quickly while delivering high performance and rich features. It creates a strong need for high-performance, energy-efficient processors. In such a processor, the level-1 cache is an integral part, as it directly affects its performance, power consumption, and overall cost. The cache size, line size, and associativity influence the performance and power consumption, which are essential for the cache design of an application. An optimal cache design that considers its parameters, such as size, line size, associativity, and replacement policy, spans an ample design space. Thus, optimal cache configurations must be identified and analyzed within this large design space. It is very time-consuming to analyze the given application with different cache configurations using architectural simulators. It encourages the use of predictive modeling, enabling machine learning to rapidly estimate cache performance and power consumption, thereby accelerating design space exploration. This work presents a rapid method for predicting the performance (miss rates) and power consumption of level-1 cache memory for an application, leveraging machine learning to accelerate design space exploration during the early design phase. This thesis proposes two predictive models that utilize cache configurations and the application characteristics. 28 cache configurations are considered, defined by size, line size, and associativity, along with 20 different applications. The proposed models achieve high accuracy for L1-I (correlation coefficients of 0.987 and 0.992 for miss rate and power, respectively) and L1-D (correlation coefficients of 0.989 and 0.994 for miss rate and power, respectively), with lower error measures (MAE and RMSE) for both miss rate and power consumption prediction. They also demonstrate good generalization, quickly predicting miss rates and power consumption for new (test) applications with 28-cache configurations simultaneously, providing 12x to 148x speedups over traditional architectural simulation. This approach significantly reduces the designer's effort in exploring the cache design space by minimizing time-consuming simulations. Furthermore, by applying Pareto-optimal selection, level-1 cache configurations are identified that achieve a balanced trade-off between miss rate and power consumption for an application from the entire design space, further reducing the design space. These all demonstrate the method’s ability to accelerate cache design and serve as a practical tool for early-stage architectural decisions.

Acknowledgement

At first, I bowed my head in sincere thanks to **Almighty God** for all the blessings, guidance in life, and the right direction that have led me toward completing my PhD.

I am sincerely grateful to my guide, **Dr. Nirali A. Kotak**, Assistant Professor at L. D. College of Engineering, Ahmedabad, for her continuous motivation, invaluable advice, and guidance. I have known her since 2005 during the bachelor's degree program, and after several years, I had the opportunity to work under her guidance again. Her wisdom, knowledge, and commitment to the highest standards greatly inspired and motivated me. I am grateful to her for her unwavering support throughout my research journey, which she generously provided through academic and technical guidance. I am thankful to the respected members of the Doctoral Progress Committee - **Dr. Vishal S. Vora**, Professor, Atmiya University, Rajkot, and **Dr. Narendrasinh B. Gohil**, Associate Professor, GEC, Rajkot- for their valuable suggestions, comments, and support, which were essential in improving the quality of my research work.

I extend my heartfelt thanks to **Gujarat Technological University** for providing the academic platform and support for my research, and to the entire staff of the **PhD Section** for their support throughout the various phases of my doctoral work.

I am incredibly thankful to **my beloved parents**, who have given me the gift of education, and for their lifelong sacrifices, blessings, and unwavering support. I sincerely appreciate my husband, **Dr. Jay Teraiya, an Associate Professor at NFSU, Gandhinagar, who has been a pillar of strength in my life and has stood beside me with love and support throughout** this long journey. A very special mention goes to my daughter, **Rajvee**, and my son, **Yugveer,** who have been silent contributors to this thesis. Their innocent presence, patience, and precious time helped me to stay strong and focused on my work. I am grateful to **my sibling** for their well-wishes, support, and motivation. I am thankful to **parents-in-law** for their blessings.

I am also grateful to **Dr. Rajendra D. Patel,** Associate Professor at Marwadi University, Rajkot, for his insightful discussion and guidance on the research, which significantly guided the research work in the right direction. I wish to thank my peer research scholar, **Ms Mital Solanki**, Lecturer at AVPTI, Rajkot, for her support during research work. Furthermore, I would like to express my deepest gratitude to all who have contributed to this research, and I humbly acknowledge their vital support.

Finally, I offer my heartfelt thanks to all those who, knowingly or unknowingly, supported, encouraged, and blessed me in this journey.

Dave Hetal Vinubhai

Table of Contents

List of Abbreviations

Sr. no.	Abbreviation	Description
1	ALU	Arithmetic & Logic Unit
2	ANN	Artificial Neural Network
3	ARM	Advanced RISC Machine
4	ARRF	Additive Regression with Random Forest
5	CACTI	Cache Access and Cycle Time Information
6	CC	Correlation Coefficient
7	CMOS	Complementary Metal-Oxide-Semiconductor
8	CPU	Central Processing Unit
9	DRAM	Dynamic Random Access Memory
10	DSE	Design Space Exploration
11	FIFO	First In First Out
12	FPGA	Field Programmable Gate Array
13	IPC	Instruction Per Cycle
14	ISA	Instruction Set Architecture
15	KB	Kilo Byte
16	L1-D	Level-1 Data Cache
17	L1-I	Level-1 Instruction Cache
18	LRU	Least Recently Used
19	MAE	Mean Absolute Error
20	MB	Mega Byte
21	ML	Machine Learning
22	MLP	Multilayer Perceptron
23	MPSoC	Multiprocessor System On Chip
24	RF	Random Forest
25	RISC	Reduced Instruction Set Computer
26	RMSE	Root Mean Square Error
27	SE	System Emulation
28	SRAM	Static Random Access Memory
29	SVM	Support Vector Machine

List of Figures

List of Tables

List of Appendices

CHAPTER – 1

Introduction

Modern processors deliver excellent performance and are used across diverse domains. A cache memory affects the performance, power consumption, and cost of a complex processor. This chapter first introduces cache memory, then discusses design space exploration, the motivation behind the work, the problem statement, and the research contributions.

1.1. Cache Memory

There has been a steady increase in the use of mobile devices over the last decade. The technological marketplace and consumers demand high performance, low power consumption, more features, and cost-effective solutions from modern devices [1]. Handy devices, such as smartphones, drones, digital cameras, and GPS navigators, persistently incorporate novel functionalities [2]. A shorter time-to-market, combined with a relatively low price for higher performance and rich features, is crucial for both consumer and industrial modern devices.

Today's complex processor contains dedicated processing units, such as application-specific instruction set processors, a general-purpose computing unit, a memory system, and communication means, including buses or a network. Modern processors deliver excellent performance, and memory systems require a similar level of performance[3][4]. The speed gap arises because the processor and memory system, such as DRAM, do not operate at the same speed [5]. This speed gap will widen as CMOS technology approaches its fundamental limits [6]. As mentioned in work [7], there have been no significant advances in memory speed over the past two decades. That means the processor has to wait most of the time for

data from the main memory. Consequently, the limitation of the memory speed significantly reduces the processor's performance.

The cache is static random-access memory (SRAM) on-chip and off-chip to the processor, and is essential for bridging this gap by storing recently referenced data or instructions [8]. Intel's 80486 processor marked the first time cache was designed, with an 8KB size, 16B line size, and 4-way set associativity. The cache supports multiple levels in the memory hierarchy to balance size, access time, and cost, which are categorized as level 1 (L1), level 2 (L2), level 3 (L3), and occasionally level 4 (L4). A specific processor with its memory hierarchy is shown in Figure 1.1 [9].

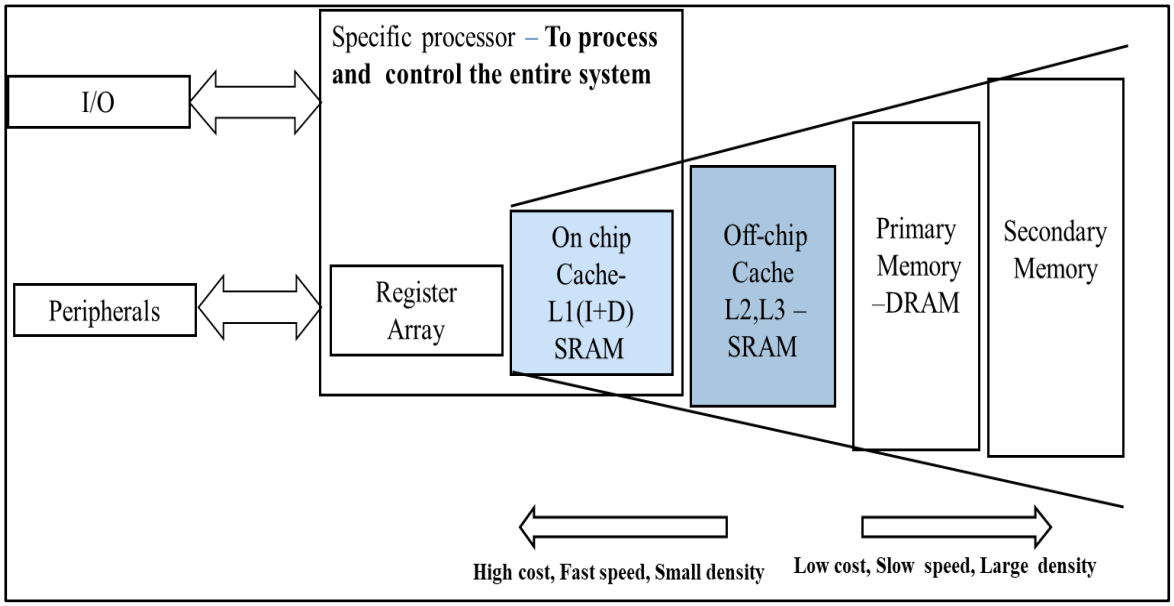


FIGURE 1.1 Specific processor with its memory hierarchy [9]

The L1 is a faster, smaller, and more expensive on-chip cache, designed to fit within the processor's area. Thus, the cache enables the processor to reuse and access recently referenced data [10][11]. The L1 cache is particularly critical for processor performance. The L1 cache is unified or separate, i.e., instruction (L1-I) or data (L1-D), and is critical to processor performance. The i586 processor, also known as the Pentium, began using a split L1 cache, with 8KB allocated to the L1-I cache and 8KB to the L1-D cache [12]. Today, Apple Silicon M1 and M2 chips have separate L1-I and L1-D caches. The M2 high-performance cores feature 192KB of L1-I cache and 128KB of L1-D cache, whereas the energy-efficient cores have 128KB of L1-I cache and 64KB of L1-D cache [13]. L1-I stores

instructions only, and L1-D stores data only, while the unified cache stores both instructions and data. The working principle of a cache is based on the locality of reference, encompassing both spatial and temporal locality. Spatial locality refers to a location that is referenced, and its data is brought into the cache; data from nearby locations is also likely to be available in the future. Temporal locality refers to the situation in which the same data is requested again shortly after it was previously requested.

The cache is the fastest memory in the memory hierarchy. Having an on-chip L1 cache, the processor tends to have fewer accesses to the external bus. Hence, overall system performance will improve. It drastically enhances memory access time from 60 ns in dynamic random access memory (DRAM) to 10 ns in L1 cache. However, the on-chip cache occupies a significant portion of the processor dies [14]. As a result, the cache consumes a considerable amount of processor power, i.e., 20% to 40% [2][15][16], which should be kept as low as possible, providing an opportunity to optimize the cache. Traditionally, power consumption has been a primary concern for portable and mobile devices; nowadays, the desktop and server domains are also concerned with it.

Cache parameters significantly influence processor performance. The designer's primary goal in designing today's processor is to deliver high performance and low power consumption concurrently [17]. Additionally, an application's performance depends on the cache [2][18][19]. Hence, the cache is an excellent candidate for consideration [16] as its design strongly influences performance and energy efficiency. The authors [10][20] have provided a detailed explanation of the cache.

The L1 cache design depends on its design parameters and performance parameters. The design parameters include cache size, line size, associativity, replacement policy, write strategy, and level, while the performance parameters are miss rate, power consumption, bandwidth, latency, and access time. These parameters are vital for cache design. Each cache design parameter affects its performance parameter to varying degrees. A larger cache results in additional silicon area and increased power consumption, while a small cache leads to degraded performance [11][21]. Moreover, different applications require different cache design parameters [11][16][22], necessitating careful selection of the cache. Some applications require high performance from L1 cache, such as weather forecasting, real-time gaming, and high-performance computing workloads. While some applications demand

lower power consumption, in which energy efficiency and battery life are more important, such as embedded systems, IoT sensors, portable health devices, industrial control systems, and low-power mobile devices.

The optimization of cache design involves determining the parameters that best suit the application's needs rather than relying on fixed, generic hardware configurations. Hence, it is essential to balance cache design parameters, the performance improvement they offer, and the system cost. The design space of a single L1 cache comprises the design parameters, i.e., cache size, line size, and associativity, which will make the cache configuration. A system designer must study various cache configurations in terms of performance before designing a cache and analyze a vast number of them. Therefore, finding a suitable cache configuration that meets the system application's requirements is difficult [23]. To find a suitable cache configuration for each system application, efficient design space exploration (DSE) is essential [19][24]. The designer applies a DSE during the very early stages of the cache's design cycle [24][25], which is an important and tedious task [22][26]. During the DSE stage, researchers and engineers evaluate new design ideas or optimize existing designs before committing to the costly, time-consuming process of hardware fabrication.

1.2. Design Space Exploration

A single cache comprises the following tunable design parameters: cache size, line size, associativity, replacement policy, write strategy, and cache level. The possible combinations of these parameters are called design points, and all design points together form the design space. Each design point in such a design space corresponds to a unique cache configuration. A typical design space can have hundreds of configurations [27], each offering exceptional access and performance for a given memory access sequence. Selecting an appropriate cache configuration for the application is crucial to optimizing cache performance. A system designer faces a vast array of cache configurations that need to be suitably selected for the application, and must decide on design options to explore various design decisions based on the system's application, thereby achieving a shorter time-to-market with reasonably reduced design cost and effort [22]. The suitable cache configuration satisfies the given criteria at the time of design. The design, analysis, and research to identify an optimal solution in the minimum time are called DSE. It enables the understanding and study of the most suitable cache configuration, thereby reducing the time required to search for design

metrics. The optimal solution satisfies the design constraints, and the suitable cache configuration can significantly affect overall performance and power consumption for a particular application. Therefore, DSE is beneficial for understanding the impact of cache configuration on system performance, identifying potential limitations, and enhancing overall system efficiency.

This section explains the necessity of DSE in modern computer architecture. Let us start by assuming a design space with limited parameters to clarify design configuration choices. In the example below, design points were calculated by considering the cache design choices of the standard x86 ISA. Cache configuration choices are Cache levels, L1 cache size, L2 cache size, Associativity, Replacement policy, and Prefetching algorithms. Table 1.1 presents the calculation of the design points, which are unique combinations of these values; the total number of design points is $3 \times 3 \times 3 \times 3 \times 2 \times 2 = 324$. By taking only 6 primary parameters, there are 324 design points, which might seem manageable, but simulating each would take many hours. The above example is provided for understanding purposes only. In practice, the number of design points can increase exponentially as design complexity rises. The work [16] further supports the complexity of the cache design space by showing a considerable increase in the number of cache design points, especially when configurable parameters are dependent.

TABLE 1.1 Calculation of design points

Configuration parameters	Design choices	Calculations
Cache level	L1,L2,L3	3
L1 size	8KB,32KB,64KB	3
L2 size	128KB,256KB,1 MB	3
Associativity	2way, 4way, 16way	3
Replacement algorithms	LRU, FIFO	2
Prefetching algorithms	2	2

Thus, the cache design space is quite ample; as a result, fast and competent DSE is necessary. Optimal design choices considerably affect the success or failure of the final design and can help avoid wasting time and effort. In addition, different applications require different optimal cache configurations [19]. Moreover, analyzing each design choice using the

existing tool or hardware prototyping is cumbersome and often impractical, requiring more time and resources; if done manually, it is even more time-consuming. Thus, cache DSE involves the physical execution, evaluation, and comparison of multiple cache configurations to find an optimal or near-optimal configuration that meets the application's requirements, making it challenging due to the ample design space of a complex system.

The cache performance analysis should be carried out in the early design phase, as changes later in the design process are more costly in terms of design costs and time-to-market. A suitable method is necessary to serve as the primary vehicle for analysis. DSE has recently received significant research attention [27]. The designers conduct an exhaustive search and use their intuition to find the optimal cache design. This method is not helpful when the design space is enormous, as it is the slowest and thus very time-consuming. The authors [24] detailed various methods for DSE of the embedded systems. In earlier work, simulation-based, analytical, and evolutionary methods explored cache design space. A computer learns without explicit programming from its experience, utilizing a rapid method known as machine learning (ML). Researchers have begun solving system architectural and optimization problems using an ML method that is beneficial for exploring the design space of the memory hierarchy [28].

1.3. Motivation for This Work

The cache size, line size, and associativity influence the miss rate and power consumption, which are essential for the cache design of an application [18]; thus, an optimal cache configuration must be analyzed and identified using an architectural simulator during cache DSE. These simulators are accurate at generating results. Therefore, it eliminates the need for actual hardware. However, simulating the given application with different cache configurations is very time-consuming [29][30][31], as it requires simulating each configuration individually, which slows down analysis of large cache configurations and can take hours or days. i.e., during the experiments of this work, a bfs application took 9 hours to simulate 28 cache configurations. The authors [32] surveyed, reviewed, analyzed, and classified various simulation methods and tools for computer architecture, including gem5, SimpleScalar, SimOS, MARSSx8, Simics, and many others. They also addressed the challenges and proposed solutions to make their use case understood by system and architecture researchers.

Moreover, different applications require different cache design parameters, so for each application, the designer must repeat the detailed cache design analysis. To enable such designs, a tool is needed that can rapidly evaluate large design spaces to identify interesting cache configurations that can then be explored with detailed simulations [33].

To address this limitation, a faster, more intelligent DSE approach is needed as design complexity grows rapidly. In today's world, ML is ubiquitous across every aspect [25][34][35]. Over the years, engineers and researchers have begun adopting ML methods to solve DSE and computer-architecture-related problems [30][31][36][37]. It offers a fast, precise, and effective solution for design optimization compared to time-intensive simulators [36]. Moreover, ML can explore vast configuration spaces, as it can effectively handle large training datasets and learn non-linear relations among configurations. The authors [31] broadly surveyed the adoption of ML in computer architecture through a new understanding of the ML-systems relationship.

An ML algorithm can capture patterns within a dataset, such as microarchitectural cache design parameters and application profiling, and identify relationships between these patterns and performance parameters, such as the miss rate and power consumption of a particular application. The authors in [24] noted that one could train an ML model using a training dataset derived from microarchitecture parameters and application-profiling information to forecast the processor's performance. This work is conducted based on this opinion, and the training dataset is prepared using microarchitecture parameters, such as cache size, line size, and associativity. Moreover, application-specific characteristics are included in the training dataset to predict cache performance and power consumption.

Furthermore, the level-1 cache cannot be too large, as it is designed to occupy the same area as the processor. Thus, the processor can get data or instructions quickly because it does not need to search a large area. Limited storage space can lead to a higher miss rate. Therefore, it is essential to consider the level-1 cache in the analysis. Additionally, the importance of considering the L1-I and L1-D caches on processor performance is discussed in detail in [38].

The proposed work aims to accelerate level-1 cache DSE by predicting key performance metrics—miss rate and power consumption—using an ML method, as it is a promising,

rapid, and accurate solution for accelerating DSE in computer architecture today. Thus, by maintaining high accuracy while designing quickly. The proposed models forecast L1-I and L1-D cache miss rates and power consumption for applications, considering 28 distinct cache configurations that vary in cache size, associativity, and line size, along with the characteristics of 20 diverse domain applications from 5 standard benchmark suites. Moreover, the trained models can quickly and accurately predict the L1-I and L1-D miss rates and power consumption of a new (test) application using cache configurations and the application's characteristics, without requiring additional simulation.

Some configurations yield the optimal miss rate in this cache design space, while others produce the optimal power. On the other hand, a miss-rate-power-optimal cache confirmation is often required. Pareto-optimal analysis is applied to determine the cache configuration (a specific combination of cache design parameters [size, line size, associativity]) that achieves a balanced trade-off between miss rate and power consumption for an application, based on predicted values, assisting in rapid, informed cache design decisions.

1.4. Definition of the Problem, Objectives, and Research Contribution

1.4.1 Definition of the Problem

Conventional methods are analytical and simulation-based [24]. An analytical method provides good DSE speedup but requires a complex mathematical formula and is less accurate. The simulation-based method yields precise results, but it is very time-consuming [29]. Due to the large number of possible configurations to be explored, exhaustive simulation is challenging when targeting application-specific architectures. Consequently, traditional methods show their limitation in early cache DSE. Furthermore, selecting miss-rate-power-optimal cache configurations from a vast design space is challenging.

Although prior studies have used ML-based methods to predict processor- and cache-related metrics and demonstrated their effectiveness in reducing design exploration time, some research challenges remain, as discussed in Chapter 2 (Section 2.6 - Summary and Research Gap).

To address these challenges, this work aims to accelerate the prediction of L1-I and L1-D cache miss rates and power consumption across different cache configurations and applications using ML methods, thereby reducing the need for additional simulations. Furthermore, the predicted miss rate and power values are used in a Pareto-based multi-objective analysis to identify cache configurations that offer a balanced trade-off between miss rate and power consumption.

1.4.2 Objectives and Scope of Work

To achieve the goal, the researcher's aims are :

1. To develop ML-based regression models for forecasting L1-I and L1-D cache miss rate and power consumption values using different cache configurations and benchmark applications' characteristics

2. To investigate and evaluate different regression models for selecting suitable predictive models for L1-I and L1-D cache miss rate and power consumption prediction.

3. To examine the generalization ability of the proposed ML models by predicting miss rates and power consumption for previously unseen applications using different cache configurations and application characteristics without requiring additional architectural simulations.

4. To identify miss-rate-power-optimal cache configurations using Pareto-based multi-objective analysis.

5. To develop an ML-based cache DSE framework that considerably reduces exploration time compared to architectural simulation while maintaining prediction accuracy.

Scope of Work

- The work is carried out for a level-1 instruction and data cache, with cache size, line size, and associativity as its design configuration.

- The evaluation is based on 20 applications from 5 widely used benchmark suites (GraphBIG, PolyBenchC, MiBench, CortexSuite, and Rodinia) across diverse domains, including graph processing, linear algebra, embedded computing, and ML.

- Applications are simulated in the gem5 simulator to obtain L1-I and L1-D miss rates, while power consumption is measured using gem5 and CACTI.

- Applications are profiled using the Intel Pintool to collect their characteristics, such as the number of data transfers, control flow, ALU instructions, basic blocks, and dynamic instructions.

- Two predictive regression models are developed using the Weka tool: one to predict miss rate (Additive regression with a random forest as the base learner) and one to predict power consumption (Multilayer perceptron).

- A comparative evaluation with other ML models is conducted using popular, suitable metrics such as the correlation coefficient, MAE, and RMSE.

- Miss-rate-power-optimal cache configurations are selected via a Pareto-optimal approach and validated in the architectural simulator.

1.4.3 Research Contribution

The following are the proposed research contributions.

1. A unified ML-based cache DSE framework is proposed for level-1 cache performance prediction in a common optimization workflow.

2. ML-based regression models are trained for L1-I and L1-D cache miss rate and power consumption using the application's characteristics and different cache configurations.

3. A comparative evaluation is performed across multiple trained regression models to select the best-fitting model based on its predictive accuracy.

4. The proposed predictive models quickly estimate cache miss rate and power consumption of new (test) applications without requiring additional architectural simulations, yielding estimates closer to simulation results.

5. Pareto-optimal analysis (multi-objective optimization) is employed to identify the application's L1-I and L1-D cache configurations that provide a balanced trade-off between miss rate and power consumption, based on the proposed model's predicted values.

6. The proposed framework enables considerable acceleration of cache DSE over simulation without compromising prediction accuracy.

The research publications made during this PhD work are listed in the Publications section.

1.5. Overall Organization of Thesis

The thesis is organized into nine chapters, each beginning with a brief introduction and concluding with a summary. As discussed earlier, the first chapter focused on cache memory and the fundamentals of DSE. It also explained the work's motivation, the problem's definition, objectives, scope, and research contributions. The remainder of the thesis is divided into the given chapters.

Chapter 2 – This chapter reviews the literature on various approaches to cache performance evaluation and DSE, including analytical, simulation-based, evolutionary, and ML methods. Moreover, it also describes multiple simulation and modeling tools used for performance evaluation and DSE.

Chapter 3 – This chapter details the proposed research methodology, including the main steps: Training dataset generation, ML model training, predictions from the proposed models on a new (test) application, and selection of Pareto-optimal cache configurations.

Chapter 4 – This chapter provides details on the benchmark applications, cache parameters, and open-source tools used in this research.

Chapter 5 – This chapter presents the implementation of the proposed methodology, providing details on generating the training dataset, training the ML predictive models, and Pareto-optimal analysis.

Chapter 6 – This chapter covers the results and observations of the work. It compares the performance of trained regression models, evaluates the generalization ability of the proposed models across all applications, identifies miss-rate-power-optimal cache configurations via Pareto analysis, reports the achieved speedup, and compares the proposed model with prior ML-based work. Lastly, this section also discusses the key insights from the work.

Chapter 7 – This chapter concludes the presented work and provides a roadmap to future work.

List of References – This section lists the references used to carry out this research work.

List of Publications – This section lists the publications that emerged during this research work.

Appendix – This section provides supporting material for the main text, including the simulation command-line interface setup, the feature-and-target relevance table, the Pareto-optimal program and algorithm, and the Pareto front graphs.

CHAPTER – 2

Literature Review

Designers employ multiple evaluation approaches to navigate the design space and identify an appropriate cache configuration, thereby reducing chip fabrication costs and achieving shorter time-to-market. Based on the level of detail required, they rely on analytical approaches, architectural simulators, and even hardware prototypes. Past work focused on uniprocessor or multiprocessor systems with one or more levels of caches in the memory hierarchy—most prior work employed analytical, simulation-based, and evolutionary methods. A limited number of prior works have also used the ML approach. The various performance evaluation techniques and the DSEs presented in prior work are explained in Sections 2.1-2.4. Various simulation and modeling tools identified in the literature for performance evaluation and DSE are also described in Section 2.5.

2.1. Analytical Approaches

An analytical approach is used to evaluate the performance of different cache configurations and to carry out the cache DSE by developing a mathematical model. This method is known for its rapid evaluation of a large number of design points; however, it is less accurate compared to simulation [23].

Liang and Mitra [15] proposed an analytical approach based on a control flow graph (CFG) algorithm for L1-I cache DSE. They experimented with SimpleScalar and CACTI using the MiBench and SPEC2000 benchmark suites. Focused parameters are L1-I size, line size, and associativity. They provided the foundational work for cache DSE, achieving high accuracy and faster estimation than simulation-based methods. However, their work is only for L1-I and requires a complex analytical model. Pan and Jonsson [39] developed an analytical model that utilizes the reuse distance distribution as input to estimate cache miss rates for

various configurations, including cache size, associativity, and replacement policy. Steen et al.[33] proposed an analytical model based on application characteristics, including instruction mix, branch characteristics, and memory-related characteristics. They provided the characteristics of these applications, along with the x86 processor's architectural parameters, to their proposed analytical model and estimated the performance and power consumption. They demonstrated that by using application characteristics for estimation, their analytical model achieved a 25× speedup. The methodology for generating features for the training dataset in this work aligns with theirs, as the features are generated from application characteristics and cache architecture parameters (i.e. cache configurations). Sabarimuthu and Venkatesh [40] proposed an analytical model to determine the L2 cache miss rate based on the L1 cache reuse distance profile. They experimented with the multicore simulator Sniper and 8 cache configurations using the PARSEC and SPLASH benchmarks. They achieved a 10.168 speedup in their work.

To facilitate a suitable decision at the early system design stage, Cheng et al. [29] developed an analytical model to determine suitable cache design parameters. Their method is 150-250 times faster than the traditional simulation approach. They used the reuse distance metric to analyze cache performance, focusing on optimal multilevel cache design. They estimated the miss rate of a multilevel cache with full associativity and an LRU replacement policy. They collected metrics using Pintool and compared the results with those from the SimpleScalar simulator. Chijindu et al**.** [41] estimated the cache hit rate using their proposed analytical model, which incorporates reuse distance and binomial probability theory, for the MiBench benchmark. They varied the L1-D cache size and associativity with an LRU replacement policy and extracted reuse distances using the Intel Pintool. They validated their proposed model against sim-cheetah simulations from the SimpleScalar simulator. However, they successfully estimated cache performance in terms of hit rate but did not focus on power-aware cache design or cache DSE.

2.2. Simulation-Based Approaches

In the simulation-based approach, performance evaluation of different cache configurations under various benchmarks is conducted using architectural simulators to identify optimal configurations. Simulators can provide detailed and accurate analysis, but they are very slow

[23][25]. Nawinne and Parameswaran [26] broadly surveyed various simulation approaches for cache in their work.

The cache DSE, using a simulation-based approach, has made considerable progress over the years. Hill and Smith [42] presented one of the earliest prior works evaluating cache performance by considering cache associativity. The authors investigated the impact of varying associativity on cache miss rates across different workload characteristics, employing trace-based evaluation methods. They revealed that cache performance depends heavily on application behavior and access patterns, and that it is not uniformly improved by increasing associativity. They laid the groundwork for subsequent research in cache performance evaluation and DSE by providing a basic understanding of cache design tradeoffs.

Another earlier and significant effort in application-specific L1 cache optimization is presented in [43]. The authors proposed a simulation approach to analyze cache-miss behavior across different cache configurations, accounting for cache line size, the number of cache lines, and associativity. Their methodology reads a memory execution trace to collect cache performance statistics for all possible cache configurations, and then selects the configuration with the optimal miss rate. They used the inclusion property to simulate multiple cache configurations simultaneously. They revealed that cache performance is application-dependent. Tojo et al. [44] improved Janapsatya's [43] approach by introducing additional correlation properties among cache configurations.

The selection of L1 cache configuration using a simulation approach for embedded applications, which utilizes PLRU or FIFO as a cache replacement policy, was proposed by Tawada et al. [21] and is both fast and accurate. They used the CRCB (Configuration Reduction approach by the Cache Behavior) approach and the MediaBench application programs. Their algorithms determined an optimal L1 cache configuration that minimizes the total memory access time by varying cache parameters. The authors [45] explored the cache design space to select the performance-per-power-optimal cache configuration for an embedded system. The experiment was carried out on the L1 and L2 caches for applications from the MiBench and PacketBench benchmark suites. They evaluated the performance using the Multi-2-Sim simulator and obtained power parameters from the CACTI tool. To reflect realistic timing effects, they back-annotated power details (from CACTI) to the

simulator. They identified the cache size that provides either the highest performance or the lowest power. Their work offers an essential understanding of the cache size selection. In [46], the cache design space is further reduced by using the cache power models from [45] to achieve optimal performance per unit power. All the above-mentioned work achieved highly accurate results. However, the major limitation of their simulation-based work is reliance on a memory-access trace of the cache configuration, which results in significantly higher runtime. Saeed and Arshad [47] employed a simulation-based method to evaluate multilevel cache energy using the MARSS and CACTI tools on the BARNES, FMM, WATER-N, and WATER-S benchmarks. They considered the L1 and L2 caches.

Some past work carried out cache DSE using hardware-based trace-driven simulation [11][19]. Nawinne et al. [19] explored the L1 and L2 cache design space using hardware-based trace-driven simulation with hSim and CACTI 6.5, examining parameters such as cache size, associativity, and line size and evaluating performance on SPEC2006 and MiBench benchmarks. Schneider et al. [11] also proposed a hardware-based, fast method to explore the cache design space. They measured cache hit rates across different configurations by collecting memory traces using a cache simulator. These traces worked as an input to the FPGA. They reported that their hardware-based FPGA cache simulator is 53x faster than the fastest software-based cache simulator.

The authors in [48] proposed a hybrid (analytical-simulation) approach for L1-I cache DSE that combines static code analysis with the LLVM tool to prune the cache design space and cycle-accurate simulation with SimpleScalar to evaluate and optimize the reduced design space. They carried out a basic-block analysis of an application and calculated the line size and the number of lines for the MiBench embedded benchmark. The achieved average speedup is 4-10 times compared to detailed simulations, while maintaining near-optimal performance. Their research demonstrated the usefulness of analysis-guided cache DSE. However, the approach remains dependent on simulation to obtain the performance value of the identified cache configuration for each application, unlike the ML-based approach.

2.3. Evolutionary Approaches

Evolutionary approaches are commonly employed in both software and hardware design. It is a population-based method that iterates over multiple solutions and contributes to them.

Then, a novel population of solutions grows in each iteration. It is beneficial for solving multi-objective optimization problems. Application-specific multiprocessors require optimizing their cache levels to achieve optimal performance, reduced energy consumption, and a smaller area. The application-specific system runs applications periodically, which require cache configuration with a cache hierarchy, line size, and associativity, necessitating multiple iterations to determine the optimal cache levels. Genetic algorithms [1][49] and heuristic techniques [22][50][51] are two primary methods of the evolutionary approach.

A genetic algorithm is a natural evolution for optimizing and searching problems [24]. The well-known genetic algorithm is NSGA-II (Non-Dominated Sorting Genetic Algorithm), a multi-objective evolutionary algorithm [52]. Yessin et al. [49] proposed CERE, a genetic algorithm–based cache DSE method that uses gem5 and CACTI, and targets an ARM processor. The study explored the L1 and L2 cache hierarchies for web-browsing applications, with L1-D size, L2 size, line size, and associativity as design parameters. The time reduction for the exploration cost is approximately 1% of the exhaustive simulation time when identifying near-optimal cache designs, demonstrating that the approach is practical for large cache design spaces. The authors [1] proposed a grammatical evolution method using the Trimaran, Dinero IV, SimpleScalar, and CACTI tools to find optimal L1-I and L1-D cache configurations for a given application. By achieving significant performance improvements in L1 cache configurations; their proposed approach confirms the appropriateness of the evolutionary method for optimizing cache performance. Although their approach is multi-objective cache optimization, they rely on simulation-based evaluation. This limitation underscores the need for data-driven, ML-based models.

The authors [50] proposed a heuristic method to explore the design space of L1-D and victim cache using SimpleScalar and CACTI tools. Benchmarks are MiBench and Powerstone. Considered parameters are cache size, associativity, line size, and victim cache size. Their approach is simulation-driven and heuristic-dependent, which requires repeated evaluations for each application. Gianelli et al. [51] employed a heuristic approach to application-specific cache tuning of general-purpose GPUs (GPGPUs) to optimize the performance and power of L1-D and L2 cache configurations using the Rodinia 3.1 benchmark. They concluded that dynamic cache tuning could significantly reduce energy–delay product and improve IPC.

Balasubadra et al. [22] pruned the L1 cache design space using a Bayesian Belief Network (BNN), identified optimal L1 cache configurations (size, line size, and associativity) for ALPbench and Mediabench applications in fewer iterations, and proposed a hybrid DSE framework. The proposed method has successfully reduced the number of simulation iterations and identified the optimal cache configuration. The authors [53] combined the artificial bee colony (ABC) optimization with design of experiments (DoE) to propose a metaheuristic method to reduce cache DSE time. They obtained L1-D and L2 cache configurations optimized for performance and energy consumption. They employed CACTI 6.5 for cache power modeling, the Infinity simulator, and the Splash2 & MiBench benchmarks. Their results showed that a heuristic-based DSE could be used to identify performance-optimized cache configurations.

All the above-mentioned prior work has successfully applied either genetic algorithms or heuristic techniques to cache DSE and optimization problems. However, this method remains dependent on either extensive simulation or heuristics, underscoring the importance of an ML approach for fast, scalable cache DSE and optimization.

An evolutionary-based method is flexible and practical for cache DSE. Still, its main limitations include high simulation overhead [16] and limited scalability as the number of cache design parameters grows, making the search impractical. Additionally, it struggles to generalize across diverse applications. [54][55].

2.4. Machine Learning-Based Approaches

The machine learning (ML) based algorithm can learn the complex relationship pattern between cache design and performance parameters from training data. Moreover, ML models can evaluate the entire cache design space, whereas the architectural simulator can evaluate each cache configuration in each run. Among various ML models, the supervised learning approach has gained broad adoption in cache DSE, where models are trained on a labelled dataset generated by simulation or profiling. The supervised learning models are categorized into classification problems, which select an optimal cache configuration from a discrete set, and regression problems, which predict continuous values such as cache miss rates or power consumption. As ML-based predictive models provide acceptable accuracy

and reduce exploration time, they offer an alternative solution to repeated simulations for evaluating cache performance and DSE.

Over the past decade, ML-based methods have gained attention across many research fields. ML is now widely recognized as a design and computer-architecture-related problem-solving tool for designers and researchers. Prior ML-based studies are presented in the following section.

2.4.1 Classification-Based ML Approaches

The authors [56] determined the optimal number of cache levels and their sizes for commercial applications using a data mining-based approach. They prepared the training dataset using the SimICS x86 simulator and then trained the decision tree using the See5 tool. Using the greedy algorithm, they determined the most suitable cache size for each level. However, its applicability to a broader range of applications is unclear, as experiments have been conducted only on two representative commercial data mining applications, C4.5 and Apriori. The authors [57] proposed an ML-based model for selecting the optimal cache line size that employs a multilayer perceptron (MLP) neural network. They collected memory access traces using Intel's Pintool and obtained cache performance metrics using the SMPCache trace-driven simulator. The proposed model was trained using the Weka 3.8 toolkit and evaluated on data mining applications from the NUmineBench benchmark. The reported prediction accuracy for cache line size selection is 95%. Although their approach is accurate, it only considers one cache parameter, such as cache line size, based on the miss rate, without considering the power consumption value of the cache, and relies on memory traces, which require high storage.

Navarro et al. [58] proposed a classification ML approach for L1-D cache DSE using an RF model. They treated cache optimization as a configuration-selection problem and identified the optimal cache configuration within a predefined cache design space. The authors relied on CACTI for cache power modeling and generated a training dataset using gem5. The proposed RF model is trained and validated using Weka 3.8 ML toolkit. They experimented on an x86-target processor using MiBench benchmark by varying L1-D cache parameters such as cache size, line size, and associativity. Their classification-based approach can

effectively identify a suitable cache configuration that meets application requirements in a shorter timeframe than a simulation-based approach.

Thesneema and Bhaskaran [59] proposed a random subspace model to determine the optimal cache level for PARSEC benchmark applications running on a multicore architecture. The authors considered cache size, associativity, line size, and cache type as design parameters, and performance metrics such as execution time (generated using gem5) and power consumption (generated using CACTI). All these collected data are features to determine the most suitable cache level. They used 10-fold cross-validation to train the model and reported precision, recall, and F-measure as evaluation metrics. Their experimental results show that a classification-based method achieves high accuracy in determining the cache level. However, the approach treats cache optimization as a discrete classification problem, aiming to predict the cache level rather than continuous values for cache performance or power consumption, thereby limiting its applicability to fine-grained cache DSE.

2.4.2 Regression-Based ML Approaches

Many prior works have proposed statistical regression-based predictive approaches to characterize the relationship between processor design parameters and performance [60][61]. A linear regression predictive model was proposed by Joseph et al. [60] to describe the relationship between processor design parameters and performance, based on experiments spanning a design space with 26 diverse design parameters. Linear models have limitations in identifying nonlinear relationships between processor design parameters and their responses. The observation in [30] reveals that the relationship between architectural parameters and performance is nonlinear.

The linear regression model is a primary approach in modeling processor performance. In contrast, modern ML models can address the limitations of these predictive models. Examples of powerful ML models include random forests (RF), neural networks (NN), decision trees (DT), and support vector machines (SVM). These models can learn complex nonlinear patterns from large training datasets and provide higher accuracy in predicting the target value (performance) across different workloads. Shahid et al. proposed [62] the regression-based approach using an NN for fast multiprocessor systems-on-chip MPSoC DSE. The authors experimented with the SESC simulator and the Multi-Cube Explorer tools

on the SPLASH-2 benchmark suite. Their proposed NN-based model learns the relationship between system design and performance parameters, enabling rapid prediction of target values for early-stage MPSoC DSE, outperforming a simulation-based method. However, it still shows limited applicability due to the model's reliance on a simulation-driven training dataset and its inability to model the cache separately.

The artificial neural network (ANN) model was trained in [63] using 15 applications to predict the CPU IPC (Instruction Per Cycle) performance metric. This work is among the earliest to employ an ML-based approach to effectively explore a large architectural and memory system design space through time-intensive simulation. The authors modeled the simulator as a nonlinear function of architectural parameters, rather than simulating each design point. They trained their model on a subset of selected design points and used it to predict the IPC for the remaining design points in the design space. To achieve robust training and accurate predictions, they used cross-validation. Essentially, the authors demonstrate that ML modeling can reduce prediction time compared to simulation techniques, such as SimPoint, without compromising paramount accuracy. Vazquez et al. [16] proposed an artificial neural network (ANN)-based ML method for cache tuning. They demonstrated that the ANN-based ML model could effectively predict cache energy consumption and provide the optimal (lowest-energy) cache configuration, without relying on a time-consuming simulator. The authors trained their ANN model on execution characteristics of applications gathered using the SimpleScalar simulator to predict the energy of L1 cache configurations. They experimented with 15 applications from the EEMBC and MiBench benchmarks and 18 L1 cache configurations, varying parameters such as cache size, line size, and associativity. This ML-based work explores the potential of ML-refined cache DSE using data-driven predictive models, bypassing simulation-driven DSE. However, they accurately predicted cache energy quickly; the scope of their work is limited to a single objective, namely cache energy, without considering cache performance, i.e., the miss rate.

The work, NAPEL [64], proposed by Singh et al. for quick instruction per cycle (IPC) and power consumption prediction in near-memory computing (NMC) architectures, using an ML-based framework. They trained their RF model on hardware-independent characteristics of 12 applications, gathered through LLVM-based analysis, along with microarchitecture parameters. To reduce simulation time, they used representative sampling from a large

sample of the application input space using the design of experiments (DoE) method. To ensure robust evaluations and generalization across applications, the authors used a leave-one-out application approach: they trained the model on all applications, left one out as a new application for testing, and then iteratively considered all applications. The trained NAPEL can predict 220 times faster than the NMC simulator without additional simulation and accurately predict the IPC and power consumption of new applications, demonstrating the capacity of ensemble learning, such as RF, to capture the nonlinear relationships in the training data. These represent an important step towards the ML-based architectural performance evaluation and DSE. In this thesis, the work also adopts the same principle for its leave-one-out validation approach, as in [64].

Sen and Imam [28] proposed ML-based prediction for a hybrid main-memory system, which is combined by DRAM and non-volatile memory parameters and trained RF, SVM, multilayer perceptron (MLP), and gradient-boosting (GB) to predict latency, power consumption, bandwidth, and memory read/write, by preparing a training dataset using NVmain and gem5, and considered 2 applications. The authors employed gem5 to gather memory traces and NVmain to simulate the hybrid memory. They trained their models using learning-curve hyperparameter tuning to avoid overfitting. In their framework, the best-fitting models for prediction are an MLP for bandwidth prediction, an RF for latency prediction, GB and RF for memory read/write prediction, and an SVM for power consumption prediction. The authors reported their experimental results, showing that the proposed ML models can predict memory responses with reasonable accuracy and at a faster rate than simulations. Although this work aims for hybrid memory rather than cache memory, it sets an example by employing an ML-based method as a substitute for time-consuming simulation in architectural performance evaluation and DSE.

Similarly, Hasan et al. [55] proposed a regression-based ML framework for DSE in DRAM and non-volatile memory systems, aiming to achieve faster evaluation than simulation. The authors trained SVM, RF, and GB models using training data generated by gem5, integrated with NVMain. The trained models characterized the relationships between memory design parameters and performance metrics, including bandwidth; average and total latency (predicted using SVM); memory reads and writes (predicted using RF); and power consumption (predicted using GB). The authors reported their experimental results, showing that the proposed regression-based ML models can predict memory responses with realistic

accuracy and at a faster rate than simulations. However, the approach targeted mainly DRAM and used only 2 applications. Conversely, the approach in this work targeted the cache and used 20 diverse domain applications from 5 standard benchmark suites. The authors [28][55] adopted multiple ML models to predict different target values, suggesting that training the model according to target values is practical and acceptable for prediction. The work in [65] predicted the cache miss distribution using application execution traces and proposed an LSTM-based deep learning model. The authors collected traces of application behavior from the Sysbench benchmark. They used DynamorIO to collect application traces and PyTorch to train their model to predict the cache-miss distributions of the level 1 and last-level caches. They focused exclusively on the cache performance metric, analyzing cache miss distributions with a trace-based LSTM model, without considering power consumption or multi-objective cache DSE.

The author [66] proposed an ANN model to predict LRU conflict misses in L1-D and formulated the cache performance prediction problem as a regression problem. The features and labels for the training dataset were prepared using a stack distance histogram of 15 applications generated via simulations with gem5 and 9 cache configurations. While they successfully employed a regression ML model that accurately forecasts an L1-D miss rate closer to simulation and is effective for cache performance modeling, their work primarily targets miss rate prediction. They do not consider power consumption or cache DSE. The quantitative comparison, based on RMSE values, between the work [66] and the proposed model is shown in Figure 6.8 (Section 6.5, "Comparison with Prior ML-based Cache DSE Work ").

2.5. Simulation and Modeling Tools for Performance Evaluation and DSE

As in the previous section, prior research on cache performance evaluation and DSE was surveyed. These approaches differ in their optimization techniques; however, almost all of these works, in one way or another, rely on simulation and modeling tools. The cache hierarchy plays a vital role in nearly every processor. Some simulators can simulate a complete processor, including the cache hierarchy, cores, and on-chip interconnect, while others may only simulate the cache levels. These methods support both highly accurate (but slower) and fast (but less accurate) techniques [24]. Simulation and modeling tools are

essential for advancing research in computer architecture. This section discusses simulation and modeling tools, specifically full-system, trace-driven, functional, and energy/power modeling tools, commonly used in the cache DSE literature.

2.5.1 Full-System Simulators

Full-system simulator offers the highest accuracy for cache evaluation by simulating the entire system and supporting a rich set of ISAs. Still, the simulation speed is slow at the cost of accuracy [32]; as is usually the case, it can simulate only a single cache configuration per simulation. Thus, exploring a vast design space is computationally expensive.

The majority of full-system simulators for cache evaluation include SimOS [32], Simics [32], gem5 [68], and MARSSx86 [32]. Among these, gem5 is widely adopted by researchers and academics for studying processor and cache performance. The gem5 can be operated in both functional and timing modes. SimOS is a full-system functional simulator, while Simics can execute programs in either forward or backward directions, making it useful for system-level debugging. MARSSx86 is a fast full-system x86 simulator. It is the most sensitive to changes in cache size. Full-system simulators are known for their accuracy, but their use is limited in architectural DSE due to their high computational cost. As a result, many researchers used trace–driven or functional simulators or adopted ML models to improve scalability.

2.5.2 Trace-Driven Simulators

During the execution of an application or workload, a dynamic sequence of memory references is recorded; this log is called the application's trace. Trace-driven simulators take application memory access traces as input and produce performance metrics, such as the cache miss rate. Trace-driven simulators are broadly categorized as software-based and FPGA-based. In the software-based simulation method, an architectural simulator collects the application's memory traces and then reruns them to evaluate different cache configurations. Dinero IV [32], CASPER [69], and Cheetah [70] are trace-driven simulators. Among them, Cheetah is a single-pass simulator that efficiently simulates different cache configurations.

In the FPGA-based simulation method, parts of the simulation are implemented in a trace-driven hardware environment, thereby accelerating cache evaluation. HAsim [32] and FireSim [32] are FPGA-based simulators. FPGA-based simulation supports higher simulation speeds. Still, significantly high development costs, longer setup time, and less flexibility are reasons they are less widely adopted in cache DSEs. Trace-driven simulators use a single trace to evaluate different cache configurations, making them faster than full-system simulations; however, they generate a considerable number of traces, requiring gigabytes of disk space to store them [24].

2.5.3 Timing and Functional Simulators

An instruction set simulator (ISS) simulates processors at the instruction level, along with their associated caches. ISS is a functional, timing, or cycle-accurate simulator, depending upon its fundamental model. It simulates a single cache configuration per simulation run. It offers good performance, faster than a full-system simulator, while maintaining moderate accuracy. SimpleScalar [71] and GPGPU-Sim [32] are usually employed ISSs. SimpleScalar is a comprehensive toolset that supports various simulation models, including sim-safe, a functional simulation model, and sim-fast, a speed-optimized version of sim-safe.

Although ISS supports fair speed for cache evaluation and early-stage DSE, its moderate accuracy in system-level analysis limits its application to detailed cache evaluation and analysis. Accordingly, ISS can be used in conjunction with analytical models or replaced by ML-based predictive models to further reduce the cost of cache DSE.

2.5.4 Energy and Power Modeling Tools

The importance of power and energy modeling tools for architectural and cache DSE has increased as power- and energy-efficient processor and computer system design has become more prevalent. CACTI [72], Wattch [32], and McPAT [32] are well-known power and energy modeling tools.

CACTI is widely used by researchers for power, energy, access latency, and area modeling in cache systems, making it a de facto standard for cache-level power estimation. Wattch is a power modeling tool used to model the energy consumption of microarchitectural

components of the processor and to analyze and optimize power consumption. Wattch can also be integrated with other architectural simulators, such as SimWattch, which integrates Simics and Wattch to enable analysis of system-level power consumption. Finally, McPAT can simulate the timing, area, and power of a multicore processor based on its architectural parameters.

2.6. Summary and Research Gap

Table 2.1 lists an overview of representative approaches mentioned in the literature review for evaluating the processor/memory performance and DSE, facilitating quick comparison. Additional studies are detailed in the corresponding sections of this Chapter.

In summary, earlier studies have proposed analytical, simulation-based, and evolutionary methods for analyzing cache configurations and cache DSEs, and have demonstrated their effectiveness. The existing literature either emphasizes selective cache parameters, relies on complex mathematical equations and slow architectural simulations, or restricts itself to specific design points, thereby revealing limitations in scalability across diverse applications and large cache configurations, as well as high exploration costs.

Many prior studies have successfully applied ML-based methods to reduce the time required to explore the cache design space, underscoring their effectiveness in predicting cache performance and supporting cache DSE. Existing ML-based studies have considered cache miss rate and power consumption separately rather than integrating them into a unified ML-based predictive cache DSE framework. Many ML-based methods in the literature have been evaluated on a limited number of applications, whereas the proposed work evaluated 20 applications across 5 benchmark suites. These applications encompass graph processing, linear algebra, embedded computing, and ML.

To address the limitations of conventional methods for evaluating cache performance and DSE, the proposed work leverages an ML model's predictive power to estimate the level-1 cache miss rate and power consumption across various cache configurations and application-specific characteristics. Moreover, because of the regression-based method, the design space could be explored in more detail, facilitating Pareto-optimal analysis. From the predicted

values, the optimal cache configuration was determined that balances miss rate and power consumption for a given application.

Due to distinct benchmark applications, classification-based model evaluation metrics, different feature sets, and higher-level (CPU/GPU) performance metrics, the direct quantitative comparison of results across most ML studies is limited. These studies have been discussed qualitatively with respect to methodology and problem framing. In one way or another, prior ML-based studies gathered data for their training datasets from gem5, CACTI, and Pintool, which encouraged the use of the same toolchain in this work. The subsequent chapter presents the research methodology of the proposed work.

TABLE 2.1 Representative approaches for performance evaluation and DSE

Sr. no.	Approach type	Reference	Target system	Methodology/ Algorithm	Simulator/Tools	Benchmark suit	No. of applications used for the training dataset	Target metrics
1	Analytical	Liang & Mitra (2013) [15]	L1-I	CFG	SimpleScalar, CACTI	MiBench, SPEC2000	NA	Hit rate
2	Analytical	Cheng & Ren (2021) [29]	L1 & L2	Reuse distance metric	SimpleScalar, Pintool	SPEC CPU2006	NA	Miss rate
3	Analytical	Chijindu et al. (2021) [41]	L1-D	Reuse distance and binomial probability theory	SimpleScalar, Pintool	MiBench	NA	Hit rate
4	Simulation	Andhi et. al. (2006) [43]	L1-I & L1-D	Reading execution trace and using inclusion property	Dinero IV	Mediabench	NA	Miss rate
5	Simulation-based	Mehdi et. al. [45] (2011)	L1 & L2	Cycle-accurate simulation of all possible design points, Back-annotating power details (from CACTI) to Simulator	Multi-2-Sim, CACTI	MiBench, PacketBench	NA	Execution time & energy
6	Hardware-based FPGA simulator	Schneider et al (2014) [11]	L1-I	Memory-trace-driven FPGA	SimpleScalar ,Tensilica Xtensa	Mediabench, SPEC CPU2000	NA	cache hit
7	Analytical+ simulation	Patel & Rajwat (2015) [48]	L1-I	Basic-block analysis	LLVM, SimpleScalar	MiBench	NA	IPC & miss rate
8	Genetic algorithm-evolutionary	Alvarez et al. (2016) [1]	L1-I & L1-D	GE	Trimaran, Dinero IV, SimpleScalar, CACTI	Mediabench	NA	Execution time & energy
9	Heuristic-evolutionary	Navarro et al. (2015) [50]	L1-D & victim cache	Sequential tuning and interleaved heuristic	SimpleScalar, CACTI	MiBench, Powerstone	NA	Execution time & energy
10	Heuristic-evolutionary	Balasubadra (2017) [22]	L1-I & L1-D	BNN	Xtensa architecture simulator	ALPbench, Mediabench	NA	Execution time & energy
11	Data mining	Elakkumanan et al. (2005) [56]	Cache hierarchy	Cache access statistics, DT	SimICS_x86, see5	C4.5 and Apriori	2	Number of cache levels & sizes

12	ML classification	Khakhaeng & Chantraporn chai (2016) [57]	L1	Trace collection, MLP	Pintool, SMPCache, Weka	NUMineBench	3	Line size
13	ML classification	Navarro et al. (2020) [58]	L1-D	Frequency of dynamic instructions, RF	gem5, CACTI, Weka	MiBench and other	507	Optimal L1-D
14	ML classification	Thasneema & Bhaskaran (2021) [59]	Cache hierarchy	Execution time and power, Random subspace	gem5, CACTI, Weka	PARSEC	7	Optimum cache levels
15	ML predictive modeling	Ipek et al. (2006) [63]	CPU	Processor design parameters, ANN	SESC, CACTI	SPEC CPU 2000,SPEC OMP,NAS	15	IPC
16	ML Regression	Vazquez et al. (2019) [16]	L1	Execution characteristics collection of applications, ANN	SimpleScalar, Matlab	EEMBC MiBench	15	Energy
17	ML regression	Singh et al. (2019) [64]	Near Memory Computing	DoE, RF	LLVM, Ramulator, Pintool, Scikitlearn	PolyBench and Rodinia	12	IPC & power
18	ML multi-model regression	Hasan et al. (2021) [55]	DRAM and NVM	SVM, RF, and GB	NVMain, gem5, Scikitlearn	CosmoGAN and LeNet	2	Power, latency, bandwidth, & memory read/write
19	ML regression	Ling et al. (2017) [66]	L1-D	Stack distance theory, ANN	gem5, Matlab	MiBench 1.0, Mediabench 2, Mobybench 2.0	`15	LRU-cache conflict misses
20	ML regression	Jelacic et al (2025) [65]	L1 & LLC	Application traces, LTSM	DynamorIO, PyTorch	Sysbench	4	Miss distributions
		Note: NA shows approaches that do not need ML training						

CHAPTER – 3

Proposed Research Methodology

This chapter presents a unified framework for L1-D and L1-I miss rates, with power-consumption prediction for L1-D and L1-I, and Pareto-optimal analysis as a multi-objective optimization.

3.1. Overview of the Proposed Methodology

The key goal of the proposed methodology is to quickly and accurately predict miss rate and power, which are used to identify an application's miss-rate-power-optimal level-1 cache configurations for cache DSE. It requires a feature set (cache design parameters and application characteristics), label values (miss rate and power consumption), and an ML model to identify patterns among them and predict the target values across the entire design space (28 cache configurations) [67]. This section describes the research methodology as shown in Figure 3.1. The methodology involves several steps, including four main ones.

1. Training dataset generation,
2. ML model training,
3. Proposed models' prediction on a new (test) application, and
4. Pareto-optimal cache configurations selection.

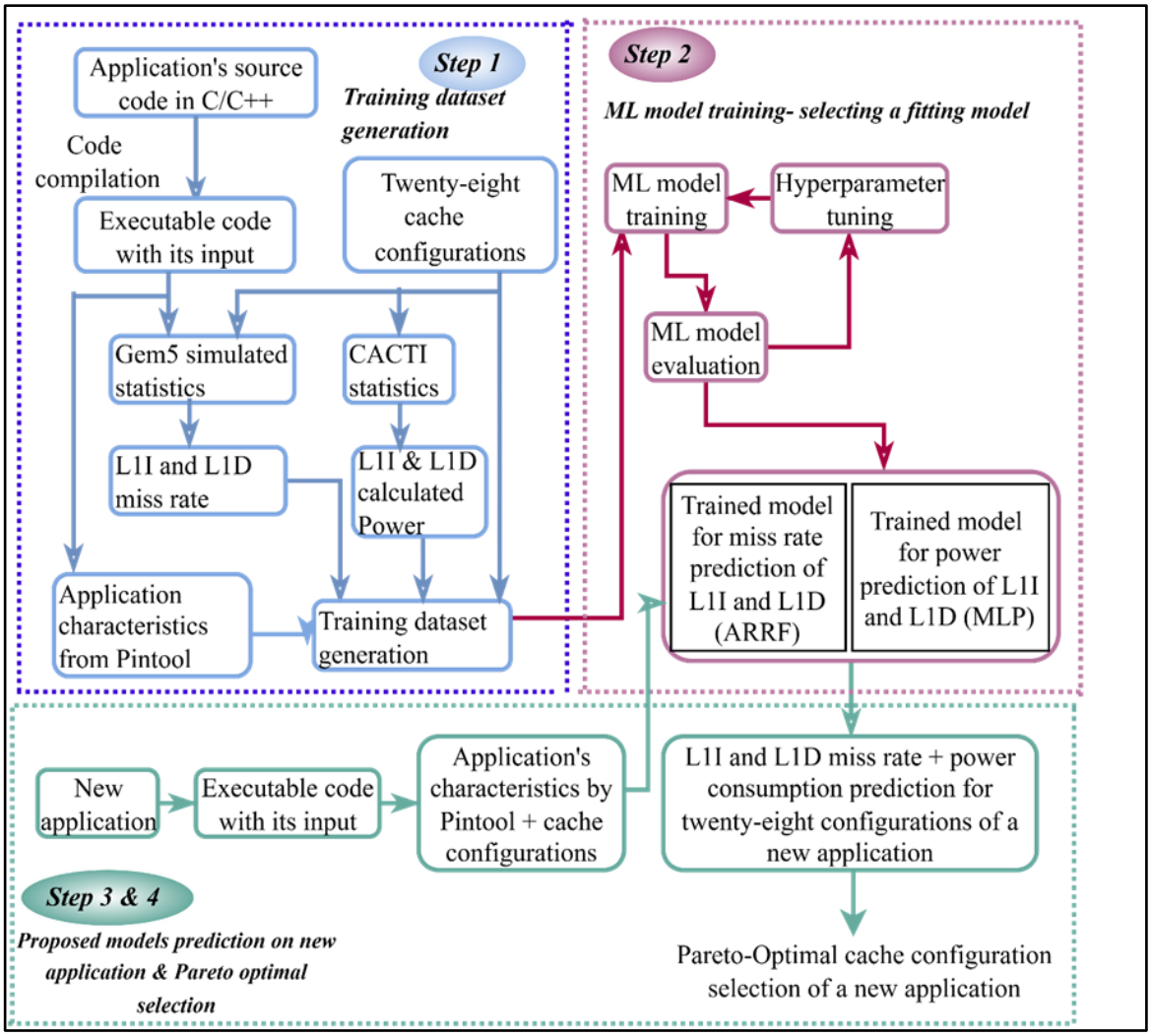


FIGURE 3.1 The proposed methodology for cache miss rates and power prediction

3.2. Main Stages of the Proposed Methodology

In the first step, training datasets were generated that include labels such as miss rate and power consumption, as well as features such as cache configurations and application characteristics. To carry out the experiments, the C/C++ source code for different applications was compiled into their executables. The executable and its input data were simulated using gem5, considering L1-I and L1-D cache configurations. 20 applications (considered applications are listed in Chapter 4, Table 4.1) were simulated with 28 cache configurations (considered cache configurations are listed in Chapter 5, Table 5.2 “Cache configurations design space”). The desired statistics were collected from the gem5 output, generated at the end of each simulation.

For power prediction, the CACTI tool was run to collect power-related statistics for L1-I & L1-D, which were then combined with gem5 statistics to calculate power for all applications. The collected statistics and power consumption calculation are given in Chapter 5 (Table 5.3 "Collected statistics from gem5 and CACTI ").

Thus, using gem5 and CACTI, cache performance metrics values were collected as the labels for the training dataset. After that, Pintool was used to profile the application's executable and its input, and the application's characteristics were gathered. The gathered application characteristics and 28 cache configurations were used as features, with miss rate and power consumption as labels (targets) for the training dataset. The detailed explanation of the dataset generation and feature construction process is presented in Section 5.1, "Training Dataset Generation Process," and Subsection 5.2.1, "Feature Construction and Selection," respectively.

In the second phase of the work, the ML models were trained. To train and evaluate ML models, the prepared training dataset was provided to Weka. Various ML models were experimented with for target prediction. Among them, the model that provided the best results for the generated training dataset was selected. The best-fitting model for miss rate is ARRF, and for power, MLP. The hyperparameters of both ML models were tuned to lower their prediction errors.

Once the model is trained to the desired accuracy and low error, the third step evaluates the model's generalization ability by predicting target values for new applications across all 28 cache configurations simultaneously (a detailed explanation of the ML training process is presented in Chapter 5). Two different ML models are proposed to predict two target values: one for the miss rate and the other for power consumption, since the miss rate is more sensitive to application behavior. At the same time, cache configurations influence the power consumption. The authors [28][55] successfully applied multiple ML models to predict different target values, suggesting that training the model on target values is practical and acceptable for prediction.

In cache design, if one performance parameter (objective) improves, the others worsen. Thus, an optimal cache configuration must be found that balances the trade-offs between performance objectives. In the real world, a multi-objective design problem is generally

addressed, in which two or more contradictory performance objectives are optimized simultaneously to design a cache. It is impossible to find a single cache configuration that simultaneously optimizes both miss rate and power consumption. The combined analysis of miss rate and power consumption supports efficient cache DSE. Thus, in the last phase, after predicting the target values for an application, the cache configurations that satisfy constraints on predicted miss rate and power consumption were determined using Pareto-optimal analysis. The Pareto-optimal set comprises cache configurations that represent different balances among the objectives. In Pareto optimization, two cache configurations are compared based on whether one dominates the other in terms of performance. It is used to significantly narrow the cache design search space and identify balanced cache configurations. Using the predicted miss-rate and power-consumption values from the proposed models across various cache configurations, a Pareto-optimal analysis was conducted to identify the optimal configurations for the L1-I and L1-D cache DSE. The same analysis was also performed on gem5-simulated values for comparison.

This chapter outlined the key steps taken to conduct this research. Benchmark applications, cache configuration, and simulators & tools were used to conduct the research experiment. In the next chapter, all of these are introduced. An implementation for each step of the research methodology is detailed in Chapter 5.

CHAPTER – 4

Experimental Environment: Benchmark, Cache Parameters, and Tools

Cache parameters are crucial in processor design. The cache design parameters considered in this work are cache size, line size, and associativity. This chapter provides a comprehensive overview of benchmark applications, the cache's design and performance parameters, and an introduction to the simulators and tools used to implement the proposed methodology.

4.1. Benchmark Applications

The benchmark suite is a collection of representative applications. This application program can be run on the simulator and mimic the characteristics of a specific real application system. Designers utilize it to compare and evaluate designs, specifically studying the processor's response in detail during the design and architecture phases of the processor development. Thus, they must examine how their new designs affect system performance early in the design process. Additionally, in computer architecture research, a scientific investigation can be conducted using standard benchmarking techniques. Generally, a benchmark suite contains application programs and their corresponding input files.

To carry out the experiment and evaluation, 20 applications were used from 5 diverse benchmark suites: GraphBIG [73], PolyBenchC [74], MiBench [75], CortexSuite [76], and Rodinia [77]. These benchmarks span domains such as graph processing, linear algebra, embedded computing, and ML, and are freely accessible. Table 4.1 presents the benchmark applications used in this work in detail.

TABLE 4.1 Benchmark applications with their description [67]

Benchmark suits	Application	Domain	Description
GraphBIG	Connected Component (cc)	Graph processing	Compute a set of connected sub-graphs from the specific graph
	Degree Centrality (dc)		A specific case of generic graph synthesis on social media observation
	Depth-First Search (dfs)		Deals with Graph traversal
	Graph Construction (gc)		Analyzed the graph for image processing
	Graph Update (gu)		Dynamic graphs computing
	Page Rank (pr)		Measuring the graph node's property by assigning a rank to each node.
	sssp		The single-source shortest path for smart navigation
	Triangle Count (tc)		Finding the number of triangles in the graph provided as input
	ufind		Random graph construction and finding
	ut		Graph traversal operations
PolyBenchC	atax	Linear algebra	Matrix transpose and vector multiplication
	bicg		BiCG Sub Kernel of BiCGStab Linear Solver
	gemver		Vector multiplication and matrix addition
	gesummv		Scalar, vector, and matrix multiplication
	mvt		Matrix-vector product and Transpose
MiBench	dijkshatra	Network	Computes the shortest routes among sets of nodes in a graph
	patricia		Instead of complete trees with thinly dispersed leaf nodes, a data structure is used.
	qsort	Automotive	Sorts an extensive array of strings in ascending order
CortexSuite	Principal Component Analysis (pca)	Machine learning	Extracts features from multivariate datasets
Rodinia	Breadth-First Search (bfs)	Graph processing	Large graphs involving millions of vertices

The GraphBIG benchmark suite is a comprehensive benchmark for graph processing that accelerates research in the field and was developed by researchers at the Georgia Institute of Technology in the USA. It is particularly focused on real-world use cases of IBM System G. It includes benchmarks for both CPU and GPU. Applications considered in the GraphBIG CPU benchmark for this work are cc, dc, dfs, gc, gu, tc, pr, sssp, ufind, and ut.

The PolyBench benchmark spans various application domains, including image processing, linear algebra, physics simulation, dynamic programming, and statistics, and was developed at the Ohio State University in the United States. It contains applications in C and Fortran. This work used atax, bicg, gemver, gesummv, and mvt from the linear algebra domain from PolyBench version C-4.2.1. The MiBench benchmark suite follows the EEMBC benchmark. It was developed by the University of Michigan in the United States. It represents the embedded system domain and is divided into six groups. From these, dijkshatra & patricia were considered from the network domain, and qsort from the automotive domain.

The CortexSuite benchmark is an extended version of the SD-VBS vision benchmark suite, developed at the University of California, San Diego. The domains of application for the CortexSuite benchmark include ML, natural language processing, and computer vision, and it supports small, medium, and large datasets. A pca application was taken from the ML domain with a large input size from CortexSuite. The Rodinia benchmark suite is a collection of parallel programs for heterogeneous computing with both CPUs and GPUs. It supports CUDA, OpenCL, and OpenMP, and was initially developed at the University of Virginia in the US. A bfs was used, which is a widely practiced graph processing application, and the OpenMP of the Rodinia version 3.1 from the CPU benchmark, in this work.

A statistical analysis of the characteristics of all 20 applications was carried out. Table 4.2 presents a quantitative analysis of application characteristics for benchmark selection. An analysis shows significant variation in the minimum and maximum values, with a relatively high coefficient of variation. It reveals that the benchmark suites include the diverse nature of application characteristics. (The theoretical relevance between application characteristics and the cache performance metrics is presented in Table B.1, Appendix B).

TABLE 4.2 Quantitative analysis of application characteristics for benchmark selection

Application characteristic	Minimum (Min)	Maximum (Max)	Arithmetic mean	Standard deviation	Coefficient of variation (CV%)
Input size	3.05×10^4	2.62×10^7	5.31×10^6	9.17×10^6	172.62
Data transfer instructions	7.48×10^4	2.79×10^8	9.32×10^7	7.69×10^7	82.51
Control flow instructions	4.17×10^4	1.37×10^8	3.00×10^7	2.85×10^7	94.96
ALU instruction	7.60×10^4	2.19×10^8	6.79×10^7	4.69×10^7	69.06
Dynamic instruction count	2.25×10^5	7.28×10^8	2.39×10^8	1.68×10^8	70.15
Basic block count	4.65×10^4	1.58×10^8	3.60×10^7	3.32×10^7	92.44

4.2. Cache Parameters

Cache is a small, fast memory in the memory hierarchy that stores recently referenced instructions and data [6]. The best-suited cache design can considerably improve performance by reducing the miss rate by storing frequently accessed instructions/data in the cache, so the processor can access them without accessing main memory. By reducing the number of main memory accesses, the cache can also help reduce the processor's overall power consumption.

Cache parameters significantly influence the processor's performance. The cache design parameters are cache size, line size, associativity, replacement policy, write strategy, and cache level. The possible combinations of these design parameters are called cache configurations. All these cache design configurations together are collectively referred to as the cache design space. Various cache design parameters, such as size, line size, and associativity, can significantly affect cache performance, which in turn affects the processor. An analysis of cache performance in terms of miss rate and power consumption, with varying cache design parameters, is crucial for finalizing the cache design [9][78].

Different applications have varying requirements for cache design parameters [22]. The optimal cache parameters will vary depending on the application type. Hence, the system

designer must carefully select a cache configuration and make reliable architectural choices. The cache design and performance parameters are as follows:

1. Cache size: The total storage capacity of a cache, measured in bytes and expressed in powers of 2, ranging from KB to MB. It is calculated as Associativity × Number of sets × Line size—the more significant the cache, the more data it stores closer to the processor. Hence, different processors have different cache sizes. L1 is designed to be small, i.e., between 2KB and 64KB [79]. Due to L1's small size, the processor needs to search a smaller area, resulting in faster data access.

2. Line size: Also known as block size, it represents the amount of data read or written with each cache access from main memory during a miss [22]. Generally, line size is kept from 16 to 256 bytes. A larger line size holds more words in the cache. The application influences how well the system designer selects a particular line size.

3. Associativity: It concerns different mapping methods that determine how main memory and cache are mapped, enabling data transfer from main memory to cache. The types of associativity include direct-mapped, N-way set-associative, and fully associative. The direct-mapped cache, also known as a one-way cache, maps a single main memory location to a single cache location. The cache is designed as an N-set in an N-way set-associative method. In the N-way set-associative design, the main and cache memories are organized into N sets. Both have the same number of blocks. Each address in main memory is free to map to any location within a particular set in cache memory. Generally, the associativity is 2, 4, 8, or 16, and the set size ranges from 1 to 1,024. A set-associative cache lies between direct-mapped and fully associative caches in terms of complexity and flexibility. Any address in main memory is mapped to any location in the cache, resulting in a fully associative mapping. It always has one set size.

   The typical Intel x86 core is designed with a 32KB L1 cache size, a 64B line size, and 8-way associativity. A different design point is that every single combination of these design parameters. For instance, a 32KB cache with a 64B line size and 4-way set associativity is one particular design point. By altering these cache

design parameters, the design space can be created that covers all possible design points.

4. Replacement and Write policy: The replacement policy determines which line to remove from the cache when the cache is not empty. It includes least recently used (LRU), first-in-first-out (FIFO), and random. When the processor has new data to write to the cache, the main memory should also be updated with this more recent data, as determined by the write policy. There are two types of write policies: write-back and write-through. In write-back, the processor updates only the cache, and main memory is updated when a cache entry is evicted. In the write-through, the processor writes to both the cache and main memory simultaneously.

5. Cache level: Modern processors support a hierarchy of cache levels, including L1, L2, and L3. Each level can have different sizes, associativity, and replacement policies.

6. Miss rate: If the processor requires data or instructions, it first checks the cache. If it is found in the cache, the processor processes the information; this is called a hit. If it is not found, a miss occurs, and the processor fetches the line from main memory, which takes longer. The processor fetches this line and also stores it in the cache for future use [80]. There are three types of misses: compulsory, capacity, and conflict [10]. When the processor accesses memory for the first time, a compulsory miss takes place. The capacity miss is due to the small space, and the conflict miss is caused by N-way set associativity, which maps two lines to the same location. The miss rate is the number of times the cache misses the memory location requested by the processor.

7. Access time: The time required by the processor to fetch instructions or data from the cache. The L1 cache supports fast access times, allowing the processor to bring instructions or data with minimal or no delay.

8. Power consumption: Another critical performance parameter of a cache is power consumption, particularly for power-limited platforms, such as embedded and mobile devices. Selecting the correct cache configuration with a power-aware

> design can reduce power consumption without significantly degrading system performance. The primary sources of power consumption in a circuit are dynamic and leakage (static) power. The dynamic power of the cache equals dynamic energy per cache access divided by time per access. The dynamic energy is due to the charging and discharging of the load capacitance in the circuit, as well as to logic-switching current. The leakage power resulting from logic switching current [18].

The cache design parameters significantly affect its performance parameters. Table 4.3 compares different cache designs and performance parameters [9]. With a larger cache, the processor fetches data that is more likely to be in the cache, resulting in a lower miss rate; however, this also increases power consumption, hardware complexity, and cost, as well as conflict misses. It slows down the access time. Hence, the cache size should be chosen for the design only up to the limit at which the desired performance is achieved. A larger line size supports spatial locality: when a location is referenced, its data are brought into the cache, and nearby location data are also obtainable in the future [78]. Thus, the miss rate will be decreased. It also reduces compulsory misses, supporting fast access time, but increases conflict misses. Direct mapping increases the miss rate but supports speedier access times. The miss rate is affected by the N-way set-associative cache. Higher associativity means more lines per set, which offers more opportunities for data to be in cache. For a given cache size, associativity can reduce the miss rate [8]. For example, if the cache size is 8MB and the associativity is changed from one-way to two-way, then this change reduces cache misses by almost 44% [10]. However, increasing associativity means more ways are looked up simultaneously per access, thereby increasing energy per access. Thus, large caches with higher set associativity support fewer miss rates but longer access times and more power consumption. Due to full associativity, power consumption and internal design complexity increased, but it supports the lowest miss rate and conflict misses. Hence, it is necessary to design a cache with an appropriate associativity.

This section provides a basic understanding of how cache design parameters affect performance, underscoring the need to select a suitable cache configuration via DSE to optimize performance for a given application within a reasonable timeframe.

TABLE 4.3 Comparison of different cache designs and performance parameters [9]

Sr no.	Cache deign parameter	Sub type	Cache performance parameter						Commonly used size
			MR	HR	AT	PC	HC	Other	
1	Larger cache size	-	L	H	S	H	1	1. Supports spatial locality 2. Conflict misses increase 3. More cost	A*No of Set*Line size
2	Larger line	-	L	H	F	-	0	1. Supports spatial locality 2. Decreases compulsory misses 3. Increases conflict misses	4B to 256 B
3	Associativity	DM	H	LT	FT	L	0	Cost is less	A = 1
		FA	LT	HT	ST	HT	2	1. Good if the cache size is small 2. Decrease conflict misses	Set size =1
		SA	L	Better	FT	H	1	-	Set size= 1 to 1024 A = 2,4,8
4	Replacement algorithm	FIFO	L	-	-	-	0	-	3
		LRU	L	-	-	H	2	1. Most effective algorithm 2. Not suitable for high associative 3. Most expensive hardware design	
		Radom	H	L	-	-	0	1. Suitable for high associative 2. Cheapest hardware design	
5	Multi- level	-	L	H	S	-	2	1. Cache Coherency maintained, 2. Miss Penalty can be reduced.	L1,L2,L3

MR= Miss Rate, HR = Hit Rate, AT = Access Time, PC = Power Consumption, HC = Hardware Complexity, Associativity =A, DM = Direct Mapped, FA= Full Associative, SA= Set Associative, FIFO = First In First Out, LRU = Least Recently Used, Low = L, High = H, Slow = S, Fast= F, Lowest = LT, Highest = HT, Slowest = ST, Fastest = FT, Hardware Complexity highest =2

4.3. Introduction of Simulator and Tools

This work was carried out using a simulator and tools. The proposed work employed gem5, CACTI, the Intel Pintool, and Weka, all open-source development tools. Figure 4.1 presents the complete experimental environment and its interaction. The gem5 simulator and CACTI were used to generate label values for the training dataset, and the Pintool was employed to collect application characteristics as features for the training dataset. The ML models were trained and evaluated using the Weka tool. The proposed research methodology is explained in Chapter 3 (Figure 3.1). This section introduces each tool used in this work and justifies the selection of the simulation and ML tools.

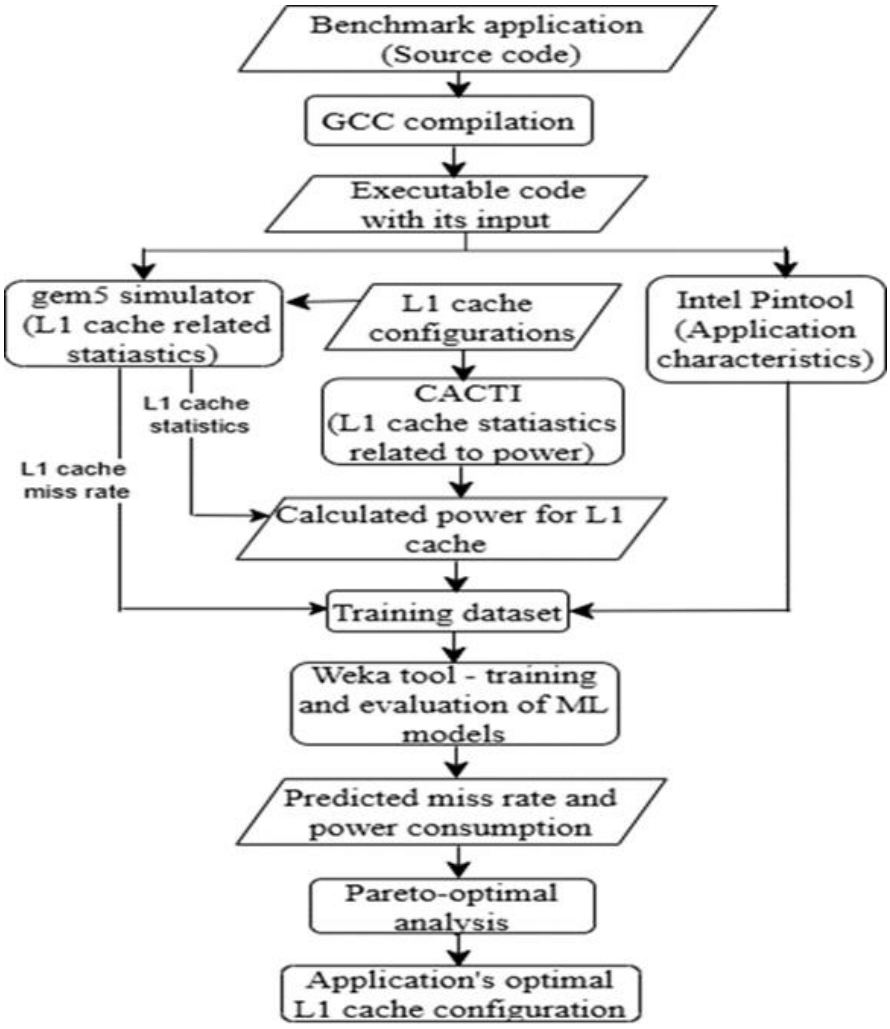


FIGURE 4.1 Complete experimental environment

4.3.1 The gem5 Simulator and CACTI

In computer architecture research, the gem5 [68][81] simulator is well known. Developers have actively maintained it, and it is widely accepted in academia, industry, and research institutions [82]. It originated from the combination of GEMS (General Execution Driven Multiprocessor Simulator) and the M5 simulator [32]. It can simulate both interactive and complex benchmark applications and is developed in C++ and Python. It supports various CPU types, including Atomic-Simple, Timing-Simple, Out-of-Order (O3), In-Order, and KVM (Kernel-based Virtual Machine). It also supports many instruction set architectures (ISAs), comprising x86, ARM, POWER, RISC-V, Alpha, MIPS, and SPARC. Gem5 supports two simulation modes: full system (FS) and syscall emulation (SE). It simulates the operating system as a whole in FS mode, while executing user-mode binaries without invoking kernel-mode system calls, as in a real operating system, in SE mode. The gem5 simulator provides two cache models: the classic cache and the Ruby cache. It also offers excellent flexibility in configuring various system parameters. It generates three output files — stats.txt, config.ini, and config.json — in the m5out folder for each run after the simulation completes, providing detailed information about numerous parameters. The config.ini file provides information about what is simulated, along with its parameter values, and the stats.txt file provides the output statistics in text format.

To generate label values for ML model training, applications were simulated in gem5 using the x86 ISA, the Timing-Simple CPU model, the SE simulation mode, and the classic cache memory model in this work. To simulate the x86 ISA, the essential architectural parameters for the gem5 simulation, as shown in Table 4.4, were used. Since this work aims to use gem5 to generate label values for the training dataset, rather than a detailed study such as pipeline hazard, this setup was employed because it provides a trade-off between accuracy and speed in computing L1-cache statistics. The L1-I and L1-D-related parameters were modified at the command line during simulation execution. The simulation results were analyzed using a stats.txt output file, which provides valuable statistics.

TABLE 4.4 Architectural parameters for the simulated system in gem5

Sr. no.	Parameter	Value
1	CPU type	TimingSimpleCPU
2	CPU clock frequency	2GHz
3	L1-D size/line size/associativity	Variable
4	L1-I size/line size/associativity	Variable
5	L1-D &L1-I MSHR	4
6	L1-D &L1-I write buffer	8
7	L1-D & L1-I cache hit latency	4 cycles
8	Replacement policy	LRU
9	Memory bus width	16 Bytes (128 bits)
10	Access latency	10 cycles
11	DRAM type	DDR3-1600

CACTI was developed in Hewlett-Packard labs. It can estimate cache area, access times, and power dissipation for a given set of parameters [72]. There are mainly two types of power consumption: dynamic power (resulting from switching activity) and leakage (static) power (due to transistor leakage). The CACTI tool had been used to extract dynamic read & write energy and leakage power for L1 cache configurations. Table 4.5 lists the most essential parameters configured in CACTI for the experiment that generates the statistics listed in Table 5.3 in the next Chapter. The L1-I and L1-D-related parameters were modified on the command line when running CACTI.

TABLE 4.5 Important configured parameters for running CACTI

Sr. no.	Parameters name	L1-I	L1-D
1	Size, Line size, Associativity	Variable	Variable
2	Technology node (nm)	22nm	22nm
3	Number of banks	1	1
4	read-write port	0	1
5	Exclusive read port	1	0
6	Exclusive write port	0	0
7	Cell/transistor type in data array	itrs-hp	itrs-hp
8	Cell/transistor type in tag array	itrs-hp	itrs-hp
9	Cache type	SRAM_CACHE	SRAM_CACHE
10	Temperature (K)	360	360

The detailed description and the equations used to calculate the L1 cache power consumption from gem5 and CACTI-derived L1 cache-related statistics are presented in Subsection 5.1.1 and equations (5.1) and (5.2), respectively.

4.3.2 Pintool

In particular, application profiling has become essential in system design owing to the growing number of data–demanding, memory-intensive applications. It is used to extract the application's characteristics using specific metrics to decide which architecture offers balanced performance and energy efficiency [83]. Application profiling tools can examine the functional operations of application program instructions and generate their characteristics, such as instruction mix and instruction dependencies. The Pin [84] is a powerful dynamic binary instrumentation tool from Intel Corporation that can be used to profile an application's characteristics. As input, Pintool takes the application's executable and its input file to produce a bitstream. This bitstream is instrumented with function calls to link it to a Library tool, which performs various analyses of the application.

4.3.3 WEKA

Weka ("Waikato Environment for Knowledge Analysis") is a popular data mining package developed at the University of Waikato, New Zealand, implemented in Java that offers a comprehensive collection of ML algorithms, along with data preprocessing, evaluation tools, and visualization aids. It can be used with graphical user interfaces (GUIs) for easy access. Weka is used in academia, research, and industrial applications. The Weka toolkit supports many input file formats, including ARFF (Attribute-Relation File Format) and CSV (Comma-Separated Values- for storing tabular data).

Weka supports cross-validation to train models, evaluate their performance, and assess generalization. In cross-validation, the data are split into multiple folds, and models are trained and tested iteratively on different combinations of these folds. The training model process using cross-validation supports robust model performance measurement by reducing the data-split effect. The ML community considers cross-validation as a best practice.

The ML approach involves two phases: training and testing. In the training phase, the dataset is used to train the ML model. The model trained on the data can make predictions with fair accuracy. Then, in the second phase, called the testing phase, the trained model is evaluated on data it has not seen before to ensure it predicts correctly and generalizes well. Weka provides all the facilities for ML model training and testing through its GUI. If the trained

model is not achieving the desired accuracy, it can be retrained by adjusting its tunable parameters (hyperparameters), thereby enabling it to learn more effectively. Different ML algorithms have specific hyperparameters (variable parameters) that can be set to train a well-fitting model on the desired training dataset [25]. Different models support different hyperparameters, such as the learning rate, momentum, number of neurons, number of trees, and tree depth, which decide the model's complexity. Well-tuned hyperparameters play an essential role in determining the model's performance [85] and improving generalization. It also helps to reduce overfitting (It occurs when the model fits the training data well, but does not fit the test data due to model complexity) or underfitting (It occurs when the model does poorly in the training data, but well in the test data due to model simplicity) [86]. Weka supports hyperparameter tuning of the ML algorithm before training.

Once the ML model is trained by properly tuning its hyperparameters on the training dataset, its performance can be measured using various metrics. Weka supports standard metrics for evaluating the performance of regression-trained models, such as Correlation Coefficient (CC), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE). The CC indicates how the model fits on the training data by measuring the linear relationship between actual and predicted values. Its higher value, closer to 1, is desirable for a better-trained model. MAE measures the average absolute difference between predicted and actual values of the trained model [87]. For a better-trained model, a lower MAE is desirable. The RMSE is calculated by taking the square root of the average squared difference between the predicted and actual values [87]. It measures how close the predictions are to the actual value. As with MAE, a lower RMSE indicates a better-trained model. The CC, MAE, and RMSE were used to evaluate the model's performance in target predictions during ML model training in this work.

4.3.4 Justification of Selected Simulation and ML Tools

Table 4.6 compares selected simulation and ML tools used in this research with representative alternative tools. Gem5, CACTI, Pin, and Weka were selected because they were suitable for the objectives of the research work rather than for claims of superiority over other tools. Moreover, they are used in the academic research literature, which enhances the credibility of the experimental methodology.

TABLE 4.6 Comparison of selected tools with alternative tools

Category	Selected tool	Alternative tool
Simulator	gem5 [81] : It can simulate both interactive and complex benchmark applications. It supports various CPU types and ISAs, configurable cache hierarchies, and generates detailed output statistics. The community actively supports it.	SimpleScalar [71] : An older simulator with limited capacity to simulate both interactive and complex benchmark applications. The support of an active community is limited.
Power engine	CACTI [72] : It is widely used in the literature to estimate cache area, access times, and power.	McPAT [32] : It is primarily used to estimate the entire processor's power.
Application profiling	Intel Pintool [84] : Using Pintool, an application can be profiled without modifying its source code.	DynamoRIO [65] : Comparable functionality with the Pin tool, but Pin has been widely used by computer architecture research to extract application characteristics.
ML framework	Weka [92] : It supports a comprehensive collection of ML algorithms, data preprocessing, and evaluation tools, and can also be used with graphical user interfaces (GUIs) for easy access. It is an open-source platform.	Scikit-learn [55] : It also supports a comprehensive collection of ML algorithms, but programming is required to train ML models.

4.4. Summary

This chapter covers benchmarks, cache parameters, and the simulator and tools used in this work. Understanding the effect of different cache parameters is vital. The cache design space is enormous, but informed design choices and careful exploration are essential for achieving optimal performance in a processor. By using suitable DSE techniques, designers can select cache configurations that meet an application's requirements more quickly. The success of the final product depends on an effective cache design.

Table 4.7 summarizes the overall development environment. The GNU Compiler Collection (GCC) was used to compile an application's source code into executable code. The next

chapter presents the implementation of the methodology, which uses all that was explained in this chapter.

TABLE 4.7 Development environment

Sr. no.	Development tools/Platform	
1	Host machine	Ubuntu 22.04 (64-bit) with an Intel Core i5-4300M CPU and 8 GiB RAM.
2	gem5	Version 22.1
3	CACTI	Version 7.0
4	Pin	Version 3.31
5	Weka	Version 3.8.6
6	GNU GCC/G++	Version 11.4.0

CHAPTER – 5

Implementation of the Methodology

The experimental setup for the proposed methodology in this research work was implemented using open-source tools: gem5, CACTI, Pin, and Weka. As mentioned in the previous chapter, gem5 facilitates the simulation of the application for the target processor architecture. CACTI estimates cache area, access times, and power dissipation for a given set of parameters. A Pin is an instrumentation tool used to profile an application's characteristics. Weka is a wide-ranging collection of ML algorithms, along with data preprocessing and evaluation tools. Benchmark applications from diverse domains, spanning multiple benchmark suites, were used to implement and validate the proposed methodology.

This chapter details the implementation of the methodology by presenting the training dataset generation process in Section 5.1 and the ML models training process in Section 5.2. Section 5.3 describes Pareto-optimal analysis.

5.1. Training Dataset Generation Process

A supervised ML model requires both labels and features for the training dataset. As this work aims to predict miss rate and power consumption, these values must be generated for the training dataset labels. This section explains the detailed steps of the experiment to create labels (miss rate and power consumption) in subsection 5.1.1 and features (application characteristics) in subsection 5.1.2.

5.1.1 Labels Generation for Training Dataset

Firstly, the applications were downloaded with their provided input file and Makefile. Then, applications were studied, and 20 suitable applications were selected from 5 widely known benchmark suites (listed in Table 4.1) spanning diverse domains. The C/C++ source code

for all 20 applications was compiled separately into executable binaries using a Makefile provided with each application, targeting the x86 ISA with GCC and G++.

Installing gem5 requires several prerequisites, which must be installed first. The gem5 version 22.1 was cloned and built for the x86 ISA on the Ubuntu host machine. The target application's compiled executable was simulated in gem5 to generate cache-related statistics. Executable code for the application must be passed to gem5 via command-line options for the simulation to proceed. The prepared command-line configuration (an example of the command-line configuration is provided in Appendix A) includes a gem5-built option for x86 ISA, SE (syscall emulation) simulation mode, CPU type (i.e., Timing-Simple), the benchmark application executable and its input file, and cache-related parameters such as L1-I and L1-D cache sizes, line sizes, and associativity. With varying L1-I and L1-D cache sizes, line sizes, and associativity, each benchmark application was simulated using the gem5 simulator. All 20 applications were simulated using gem5 across 28 L1-I and L1-D cache configurations. A fixed LRU replacement policy was chosen. Table 5.1 lists the L1 cache design parameter values (all powers of 2) considered in the experiment.

TABLE 5.1 Cache design parameter values

Cache design parameters	Value
Cache size	1 KB to 1024KB
Line size	8B to 64B
Associativity	1-way to 16-way

28 cache configurations were simulated. Here, it is explained how it was prepared. The associativity from 1-way to 16-way was varied with a 32KB cache and a 64B line size; these configurations were named config 1-5. The considered line sizes were 8B to 32B for the cache configurations 6 to 8, with a 32KB cache size and 1-way associativity. Then the cache size was varied from 1 KB to 1024KB with 1-way associativity, considering line sizes of 16B and 64B in configs 9-18 and 19-28, respectively. Table 5.2 presents the cache design space, which comprises all 28 L1 cache configurations. Because the simulator takes a long time to produce output, this work deliberately considered 28 useful cache configurations to obtain a practical label value for the training dataset.

TABLE 5.2 Cache configurations design space [67]

Cache configurations	Size-line size-associativity	Cache configurations	Size-line size-associativity
Config 1	32KB-64B-1Way	Config 15	128KB-16B-1Way
Config 2	32KB-64B-2Way	Config 16	256KB-16B-1Way
Config 3	32KB-64B-4Way	Config 17	512KB-16B-1Way
Config 4	32KB-64B-8Way	Config 18	1024KB-16B-1Way
Config 5	32KB-64B-16Way	Config 19	1 KB-64B-1Way
Config 6	32KB-8B-1Way	Config 20	2KB-64B-1Way
Config 7	32KB-16B-1Way	Config 21	4KB-64B-1Way
Config 8	32KB-32B-1Way	Config 22	8KB-64B-1Way
Config 9	1 KB-16B-1Way	Config 23	16KB-64B-1Way
Config 10	2KB-16B-1Way	Config 24	64KB-64B-1Way
Config 11	4KB-16B-1Way	Config 25	128KB-64B-1Way
Config 12	8KB-16B-1Way	Config 26	256KB-64B-1Way
Config 13	16KB-16B-1Way	Config 27	512KB-64B-1Way
Config 14	64KB-16B-1Way	Config 28	1024KB-64B-1Way

After completing the simulation, the gem5-simulated statistics were collected from the stats.txt output file. The generated stats.txt file contains many statistics, including general execution, CPU cache, memory controller statistics, and more. A sample stats.txt file of the qsort application is shown in Figure 5.1. The extracted statistics for the experiment are highlighted in Figure 5.1. The L1-I and L1-D miss-rate values can be directly extracted from the gem5 output file for the training dataset.

To collect the power consumption label, statistics from both gem5 and CACTI were used. From the gem5 output file, the simulation time (sim-second: the time required for the application to execute, as computed by gem5 for a particular cache configuration), the number of read accesses, and the number of write accesses were collected. Table 5.3 lists all the extracted statistics used for the experiment. CACTI was employed to generate dynamic read & write energy, as well as leakage power, for all 28 L1-I and L1-D cache configurations, separately for each. CACTI-generated statistics (listed in Table 5.3) were combined with the gem5 output statistics to calculate the power consumption of all 28 L1-I and L1-D cache configurations and 20 applications for the target values in the training dataset.

```
---------- Begin Simulation Statistics ----------
simSeconds                                   3.868638                       # Number of seconds simulated (Second)
simTicks                                     3868638288000                  # Number of ticks simulated (Tick)
finalTick                                    3868638288000                  # Number of ticks from beginning of simulation
simFreq                                      1000000000000                  # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  1415.05                        # Real time elapsed on the host (Second)
hostTickRate                                 2733917208                     # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                   8529904                        # Number of bytes of host memory used (Byte)
simInsts                                     283005829                      # Number of instructions simulated (Count)
simOps                                       511177797                      # Number of ops (including micro ops) simulated (Count)
hostInstRate                                 199997                         # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                   361243                         # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                      1000                           # Clock period in ticks (Tick)
system.cpu.numCycles                         7737276576                     # Number of cpu cycles simulated (Cycle)
system.cpu.dcache.overallHits::cpu.data      80252885                       # number of overall hits (Count)
system.cpu.dcache.overallMisses::cpu.data    15456272                       # number of overall misses (Count)
system.cpu.dcache.overallMissLatency::cpu.data 924915018000                 # number of overall miss ticks (Tick)
system.cpu.dcache.demandAccesses::cpu.data   95652673                       # number of demand (read+write) accesses (Count)
system.cpu.dcache.overallAccesses::cpu.data  95709157                       # number of overall (read+write) accesses (Count)
system.cpu.dcache.overallMissRate::cpu.data  0.161492                       # miss rate for overall accesses (Ratio)
system.cpu.dcache.demandAvgMissLatency::cpu.data 60060.157755               # average overall miss latency in ticks ((Tick/Count))
system.cpu.dcache.ReadReq.accesses::cpu.data 57867793                       # number of ReadReq accesses(hits+misses) (Count)
system.cpu.dcache.WriteReq.accesses::cpu.data 37784880                      # number of WriteReq accesses(hits+misses) (Count)
system.cpu.icache.overallHits::cpu.inst      339240724                      # number of overall hits (Count)
system.cpu.icache.demandMisses::cpu.inst     46493463                       # number of demand (read+write) misses (Count)
system.cpu.icache.overallMisses::cpu.inst    46493463                       # number of overall misses (Count)
system.cpu.icache.demandMissRate::total      0.120532                       # miss rate for demand accesses (Ratio)
system.cpu.icache.overallMissRate::cpu.inst  0.120532                       # miss rate for overall accesses (Ratio)
system.cpu.icache.ReadReq.accesses::cpu.inst 385734187                      # number of ReadReq accesses(hits+misses) (Count)
system.cpu.icache.ReadReq.missRate::cpu.inst 0.120532                       # miss rate for ReadReq accesses (Ratio)
system.cpu.icache.ReadReq.missRate::total    0.120532                       # miss rate for ReadReq accesses (Ratio)
    .
    .
    .
    .
-.--------- End Simulation Statistics   ----------
```

FIGURE 5.1 Sample stats.txt file of the qsort application

Dynamic energy is the product of the number of cache accesses and the energy per cache access (N×E) [83]. The dynamic power (N×E)/T has units J/S, or Watts. The dynamic and leakage (static) power of the L1 cache were considered. The power calculation was derived from Equations (5.1) and (5.2).

TABLE 5.3 Collected statistics from gem5 and CACTI

Sr. no.	Statistics name	Description
1	*T*	sim-seconds (s)
2	*NL1Dr*	number of L1 data reads (read access)
3	*NL1Dw*	number of L1 data writes (write access)
4	*NL1Ir*	number of L1 instruction reads (read access)
5	*EL1Dr*	dynamic energy per data read (joules/read)
6	*EL1Dw*	dynamic energy per data write (joules/write)
7	*EL1Ir*	dynamic energy per instruction read (joules/read)
8	*PL1D_leakage*	data cache leakage power (watts)
9	*PL1I_leakage*	Instruction cache leakage power (watts)

For L1-I cache:

$$PL1I_{total} = PL1I_{dynamic} + PL1I_{leakage} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (5.1)$$

Where,

$$PL1I\_dynamic = \frac{(\mathrm{N}L1I\mathrm{r} \times \mathrm{E}L1I\mathrm{r})}{T}$$

Since instruction caches are designed for read operations only, this work used read-related statistics to achieve this goal.

For L1-D cache:

$$PL1D_{total} = PL1D_{dynamic} + PL1D_{leakage} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (5.2)$$

Where,

$$PL1D\_dynamic = \frac{(NL1Dr \times EL1Dr) + (NL1Dw \times EL1Dw)}{T}$$

The dynamic cache power was calculated as shown in the above equations by following the architecture-level approach of Wang et al. [88] and the 'accesses × energy-per-access' formulation of Michael et al [50][89].

5.1.2 Applications Characteristics Generation for Training Dataset

In this stage of implementation, all application characteristics were generated using the Intel Pintool by running the application's executable binary with its input file. Then, the application's characteristics were collected as raw data, such as dynamic instruction count, basic block count, category mix, memory read/write, memory write, memory read, and no-memory instruction count, from MypinTool, Catmixtool, and ldstmixtool in Intel Pintool [90] (Finalized characteristics are listed in the next section, Table 5.4). From the Catmixtool, dynamic counts were considered, which provide actual runtime execution counts while the application is running, helping assess real performance and behavior. A sample output file

of the Catmixtool of the cc application is shown in Figure 5.2. The extracted statistics for the experiment are highlighted.

Thus, the training dataset consists of independent variables (cache configurations and the application's characteristics) and dependent variables (miss rate and calculated power consumption). A training dataset was prepared for L1-I and L1-D. For L1-I, two training datasets were used: one for the miss-rate target and another for the power-consumption target. Same for L1-D. Thus, four training datasets were finally prepared for the training process.

```
#
# CC_dynamic counts
#
#num category                count-unpredicated  count-predicate
#
  3 AES                      0                   6702153
  6 AVX                      521993              0
  7 AVX2                     1030943             6729864
 13 AVX512_VBMI              0                   2885447
 14 AVX512_VP2INTERSECT      0                   92
 16 BINARY                   20507291            0
 17 BITBYTE                  3458                150190
 19 BMI1                     455165              256938
 20 BMI2                     114871              0
 21 BROADCAST                407106              0
 22 CALL                     2885452             0
 23 CET                      0                   2538873
 25 CLFLUSHOPT               0                   441756
 28 CMOV                     0                   849174
 30 COND_BR                  21636004            12
 32 CONVERT                  33655               0
 33 DATAXFER                 42666370            0
 34 DECIMAL                  0                   3775996
 44 HRESET                   0                   5283985
 46 INTERRUPT                0                   2
 48 IOSTRINGOP               0                   226
 54 LOGICAL                  13447330            0
 55 LOGICAL_FP               21402               0
 57 MISC                     7180842             0
 62 NOP                      153149              0
# eof
```

FIGURE 5.2 Sample output file of the Catmixtool of the cc application

5.2. ML Models Training Process

After collecting all labels and feature values as described in Section 5.1, the appropriate ML model must be trained on the prepared dataset. This section presents the ML model training process in detail, explaining feature selection in 5.2.1 and model training in 5.2.2.

5.2.1 Feature Construction and Selection

Choosing both the relevant features based on the target values to be predicted and the total number of features is a thoughtful approach to training an ML model with high prediction accuracy and good generalization. The features should be relevant to the target values; i.e., one can choose cache configurations and application behavior as features to predict miss rates and power consumption. The number of features is also essential, as too many can cause an ML model to overfit, a model-complexity problem, and increase training time. Conversely, a smaller number of features may not be sufficient to differentiate between training data sets, preventing the ML model from learning functional patterns and leading to underfitting. Thus, training an ML model with relevant features reduces overfitting (underfitting), lowers computational cost, improves model performance, and enhances generalization.

Firstly, in the experiment, several raw data points were collected by profiling the applications using the Pintool, which could serve as the training dataset for the ML model. Then, both feature construction and feature selection were applied to ensure that all application characteristics were covered and to remove redundant features, thereby helping to train the proposed model for accurate prediction. Manual feature construction was employed, combining multiple sub-features into a single meaningful feature. The constructed features are data transfer count, control flow count, and ALU instruction count. The data_transfer feature was created by adding the frequency of data transfer and the clflushopt (Cache Line Flush Optimized - x86 instruction) - load-and-store-related instructions. The control_flow was created by adding the counts of conditional-branch, unconditional-branch, and call-related instructions. The ALU_instruction feature accumulated the counts of logical, logical-floating-point, binary, and decimal instructions.

These constructed features provide a high-level abstraction of subfeatures, thereby simplifying the feature space while maintaining significant application relevance. After constructing the features, dynamic_instruction_count, basic-block_count, data_transfer_count, control_flow_count, ALU_instruction_count, AVX_instruction_count, memory_read/write, memory_write, memory_read, and no_memory_instruction_count were retained as application characteristics, along with the cache configuration, including its size, line size, and associativity.

Then the training of the first ML model began, but the expected results were not achieved. After checking various ML models for training, it was decided to examine the feature set to achieve the desired result. Then, feature selection was applied to remove redundant and irrelevant features from the collected set. A feature selection filter, specifically the ClassifierAttributeEval attribute evaluator with the Ranker search method in Weka, was used. The models were retrained only for the suitable features.

The finalized feature set includes cache configuration and application characteristics, comprising 9 features: cache_size, line_size, associativity, data_transfer, control_flow, ALU_instruction, application's input_file_size, dynamic_instruction, and basic_block [67]. Table 5.4 lists the finalized feature set, which was identified as valuable and relevant for training the proposed ML model. (The theoretical relevance between selected features and the target is presented in Table B.1, Appendix B)

TABLE 5.4 List of final feature sets for the training process

Sr. no.	Feature	Feature description
1	cache_size	L1 cache size in KB
2	line_size	L1 cache line size
3	associativity	L1 cache number of ways
4	data_transfer	number of load and store instructions
5	control_flow	number of control flow (branch and call) instructions
6	ALU_instruction	number of ALU (logical, logical-floating-point, binary, bit-byte, and decimal) instructions
7	input_file_size	application's input file size in MB
8	dynamic_instruction	total number of instructions count
9	basic_block	number of basic blocks in the program

5.2.2 Training the ML Models

After generating labels and selecting all relevant features, the final training dataset was prepared. The training dataset contained 20 applications × 28 cache configurations = resulting in 560 samples. It consisted of a feature set as independent variables (cache configuration and the application's characteristics, as listed in Table 5.4) and labels as dependent variables (miss rate and calculated power consumption). Training datasets for L1-I and L1-D were prepared. For L1-I, two training datasets were prepared: one for the miss-rate label and another for the power-consumption label. Same for L1-D. Thus, four training datasets were ready. To maintain methodological consistency across experiments, this work used a unified set of ML model features for miss rate and power consumption, as well as for L1-D and L1-I.

Now, a suitable ML model must be selected to train on the dataset and accurately predict the target values. The generated training dataset was provided as input to the Weka tool [91][92] for model training and evaluation. The leave-one-out [64][67] method was applied, with 19 applications in the training set and one reserved for testing. Likewise, iteratively, each application was tested. Then, multiple regression ML models (Additive Regression, Random Forest, RandomSubspace, KNN, SMOReg, and Multilayer Perceptron) were trained on the training dataset in Weka, and the best-fitting model was selected using 10-fold cross-validation. Then, after calculating the average, evaluation metrics were reported for all trained models, including the correlation coefficient, MAE, and RMSE. The comparison among various trained models is provided in Chapter 6. The additive regression with a random forest (ARRF) and the Multilayer Perceptron (MLP) best predicted the desired miss rate and power consumption, respectively, on the training dataset, as they had the highest correlation coefficient and the lowest MAE and RMSE.

The hyperparameters of ARRF and MLP were tuned to train the best-fitting model (with a high correlation coefficient and lower error). For the additive regression model, the learning rate (shrinkage) was set to 0.9 [93][94], and a random forest was used as the base learner of the additive regression with 200 iterations [67]. For MLP, the learning rate was set to 0.01, the number of hidden layers to 4, the number of training epochs to 400, and normalizedAttributes was set to the “true” value (as it can improve the network's

performance). The remaining hyperparameters were kept at their default value for both models. Table 5.5 lists the proposed models' hyperparameters.

TABLE 5.5 The proposed models' hyperparameters

Hyperparameters	Values
AR model parameter	
Batch size	100
Classifier	RF
Number of iterations	10
Shrinkage (learning rate)	0.9
RF classifier parameter	
Bag size percentage	100
Batch size	100
Max depth	0 (unlimited)
Number of iterations(number of trees)	200
Seed	1
numFeatures	0, int(log_2(#predictors) + 1)
MLP parameter	
Learning rate	0.01
Hidden layers	4
Number of training epochs	400
normalizedAttributes	True
Momentum	0.2

Based on the empirical analysis, the hyperparameter values shown in Table 5.5 were selected for both ML models (ARRF and MLP). Initially, the important hyperparameters of all candidate ML models were systematically varied and evaluated. Their prediction performance was evaluated using the correlation coefficient (CC), MAE, and RMSE. Hyperparameters were tuned that are expected to have a major effect on the learning behavior of the respective ML models. The experimental analysis revealed that hyperparameter tuning for RF, RandomSubspace, Knn, and SMOReg did not yield performance improvements over their default settings in Weka. Therefore, their default hyperparameter values were used in the experiments.

In contrast, the additive regression model, with a tuned hyperparameter (i.e., learning rate) and a random forest as the base learner, improved the prediction of the miss rate. Similarly, the MLP, with tuned hyperparameters such as learning rate, number of hidden layers, and the number of training epochs, and normalizedAttributes, improved the power prediction.

Tuned hyperparameters can directly control the model's learning process, i.e., by providing contributions in successive iterations for additive regression and by improving convergence and representational ability in MLPs. Further parameter tuning did not improve predictive performance. Thus, the remaining hyperparameter values were set to their default values provided in Weka.

Furthermore, an additional statistical analysis was performed to compare and select ML models using a corrected paired t-test in the Weka experimenter [91], with the complete training dataset and 10-fold cross-validation. A significance threshold was set at 0.05 in a corrected paired t-test. ARRF (miss rate prediction) and MLP (power prediction) were set as baselines in the test-base in a corrected paired t-test. As RMSE is an important regression error metric that provides a direct measure of prediction accuracy, it was reported as a significant metric for the analysis. A lower RMSE value is desirable. An additional statistical analysis provides quantitative support for comparing ML models. Further, it supports selecting the ARRF model for miss-rate prediction and the MLP model for power prediction. The detailed additional statistical analysis for ML model selection is presented in Table 5.6 for miss-rate prediction and in Table 5.7 for power prediction.

TABLE 5.6 Additional statistical analysis for ML model selection for miss rate prediction

Cache	ML models	RMSE	Corrected paired t-test
L1-D	ARRF	0.010	Baseline
	RF	0.015	Considerably inferior
	RandomSubspace	0.04	Considerably inferior
	Knn	0.03	Considerably inferior
	MLP	0.06	Considerably inferior
L1-I	ARRF	0.004	Baseline
	RF	0.01	Considerably inferior
	RandomSubspace	0.02	Considerably inferior
	Knn	0.01	Considerably inferior
	MLP	0.02	Considerably inferior

TABLE 5.7 Additional statistical analysis for ML model selection for power prediction

Cache	ML models	RMSE	Corrected paired t-test
L1-D	MLP	0.004	Baseline
	ARRF	0.01	Considerably inferior
	SMOReg	0.01	Considerably inferior
	RandomSubspace	0.05	Considerably inferior
	Knn	0.02	Considerably inferior
L1-I	MLP	0.004	Baseline
	ARRF	0.006	Considerably inferior
	SMOReg	0.009	Considerably inferior
	RandomSubspace	0.049	Considerably inferior
	Knn	0.028	Considerably inferior

Once the model's training is complete, the time-consuming simulation step is no longer required to obtain feature values for new applications, as the feature set can be prepared using cache configurations and application characteristics (from Pintool) for the proposed model.

The proposed model predicted the target values of the new application. The MAE and RMSE were calculated for the new application using the formulae in [28], Equations (5.3) and (5.4).

$$MAE = \frac{\sum_{i=1}^{n} | Pi - Si |}{n} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5.3)$$

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(Pi - Si)^2}{n}} \ldots\ldots\ldots\ldots\ldots\ldots (5.4)$$

Where n denotes the total number of data points, Pi equals the proposed model's predicted target value, and Si equals the simulated (actual) target value.

5.3. Pareto-Optimal Analysis

One of the important focus points of the proposed work is to determine which cache configuration choice yields the best results in terms of miss rate and power consumption for

each application. In this regard, after predicting the target values for an application, the cache configuration that satisfies constraints on the miss rate and power consumption could be determined. A configuration is called Pareto-optimal if it cannot be improved in any objective without compromising another objective (in this case, miss rate and power consumption). It can be obtained using Pareto-optimal analysis [95][96][97].

To understand the miss rate-power trade-off, Figure 5.3 presents an analysis of miss rate and power across 28 simulated cache configurations for the L1-D of the cc application.

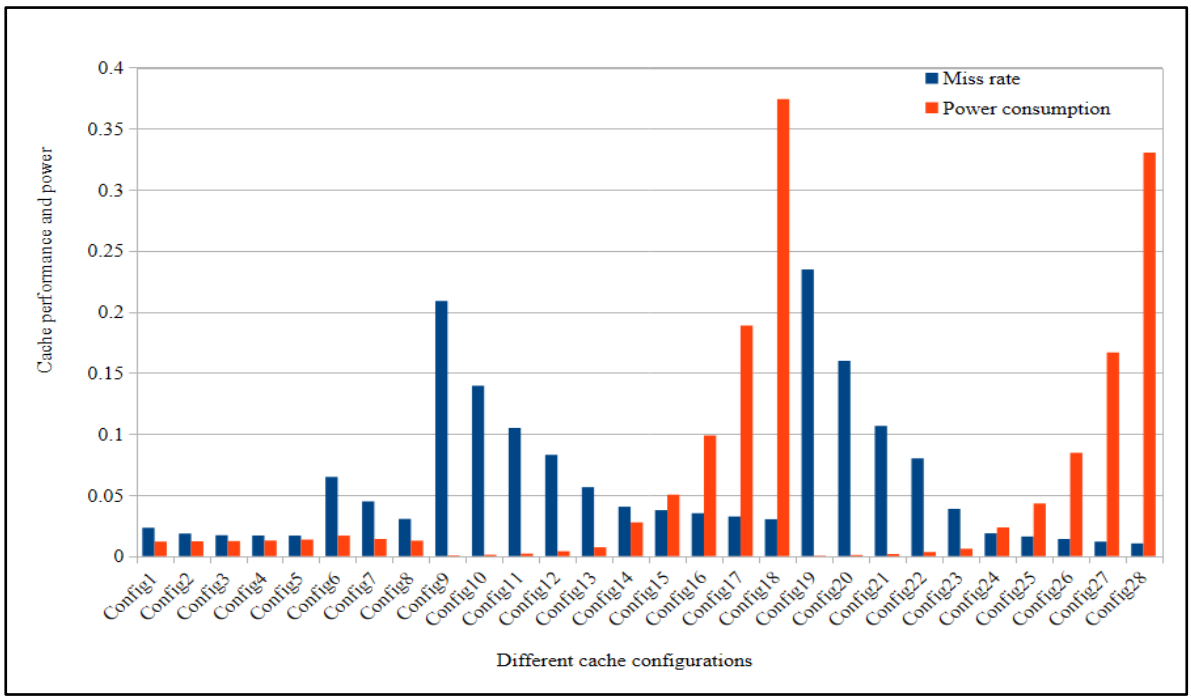


FIGURE 5.3 Miss rate and power analysis across different cache configurations

It shows that various cache configurations exhibit higher power consumption with lower miss rates, and vice versa. In contrast, some balanced configurations support moderate miss rates and power consumption, suggesting Pareto-optimality. Some are suboptimal configurations that consume more power without corresponding performance benefits. This analysis underscores the need for a multi-objective cache DSE and Pareto analysis.

Figure 5.4 illustrates the conceptual Pareto-optimal curve in cache design for the miss rate-power tradeoff [95] and is used only for basic understanding of the Pareto-optimal selection methodology employed in this work. The dashed line in Figure 5.4 is the Pareto-optimal

curve (Pareto front) formed by non-dominated cache configurations, which are broadly clustered into low-miss-rate (high-power-consuming) configurations, low-power (high-miss-rate) configurations, and balanced (miss-rate-power tradeoff) configurations. In the cache DSE process, from this Pareto-optimal set, one can choose, based on the cache design requirements, either the low-miss-rate configuration (for high-performing applications), the low-power configuration (for prioritized battery life), or the balanced configuration. It depends on the constraints of the target application. The balanced configurations support the most performance benefits without exponentially increasing power penalties.

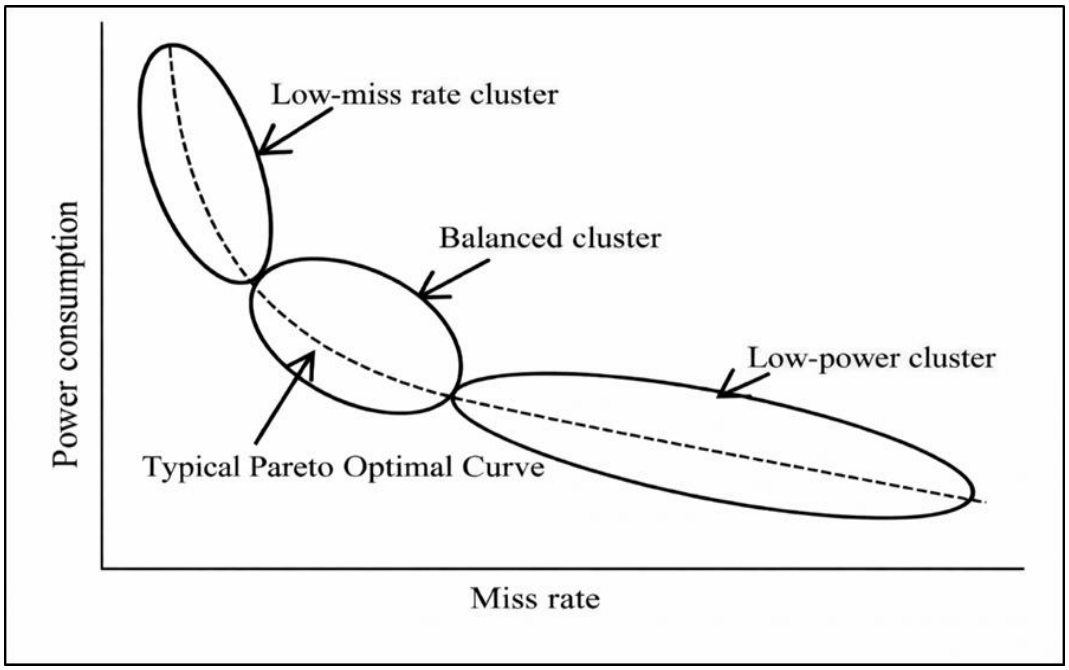


FIGURE 5.4 Conceptual Pareto-optimal curve in cache design for miss rate-power tradeoff [95]

To perform a Pareto analysis, a Python program was used that implements the Brute-Force Non-Dominated Sorting algorithm and the weighted-sum method with min-max normalization. The input parameters for the Python program include 28 cache configurations, miss rates, and power consumption values (the Python program snippet and the Pareto dominance algorithm are provided in Appendix C). Table 5.8 presents the input parameters for the Python program: 28 cache configurations, along with their miss rates and power consumption values (predicted and simulated) for the L1-D cc application, for Pareto analysis.

TABLE 5.8 28-cache configurations with their miss rate and power consumption values for the cc application for Pareto analysis

Cache config.	Predicted		Simulated		Cache config.	Predicted		Simulated	
	Miss rate	Power	Miss rate	Power		Miss rate	Power	Miss rate	Power
Config 1	0.033	0.010	0.023	0.012	Config 15	0.046	0.046	0.038	0.050
Config 2	0.027	0.010	0.018	0.012	Config 16	0.043	0.091	0.035	0.099
Config 3	0.026	0.011	0.017	0.012	Config 17	0.039	0.188	0.032	0.189
Config 4	0.026	0.012	0.017	0.013	Config 18	0.035	0.358	0.030	0.374
Config 5	0.026	0.014	0.017	0.013	Config 19	0.244	0.001	0.235	0.000
Config 6	0.075	0.017	0.065	0.017	Config 20	0.171	0.002	0.160	0.001
Config 7	0.055	0.016	0.045	0.014	Config 21	0.119	0.002	0.107	0.002
Config 8	0.040	0.013	0.030	0.013	Config 22	0.091	0.003	0.080	0.003
Config 9	0.226	0.007	0.209	0.001	Config 23	0.043	0.006	0.039	0.006
Config 10	0.160	0.007	0.140	0.001	Config 24	0.027	0.019	0.019	0.023
Config 11	0.122	0.007	0.105	0.002	Config 25	0.023	0.038	0.016	0.043
Config 12	0.099	0.009	0.083	0.004	Config 26	0.020	0.080	0.014	0.085
Config 13	0.062	0.011	0.056	0.007	Config 27	0.017	0.174	0.012	0.167
Config 14	0.050	0.025	0.040	0.028	Config 28	0.015	0.347	0.010	0.330

This work used Brute-Force Non-Dominated Sorting, a multi-objective optimization method [96], to identify Pareto dominance by comparing each cache configuration to all others. This method finds configurations that are not dominated by any other configuration, selects those that are not simultaneously inferior in both miss rate and power, and forms the Pareto-optimal set (a Pareto-dominance example is given in Appendix C.2).

This Pareto–optimal set forms the Pareto front [97], a curve defined by the non-dominated cache configurations. The cache configurations closest to the bottom-left corner (as shown in Figure 5.4) [95] can provide a balanced trade-off between objectives.

After finding the Pareto optimal set, the Python program used the weighted sum method (WSM) [98] with min–max normalization (a Multi-Criteria Decision Making technique) - a score-based ranking method - to select the final configuration that provides a balanced trade-off between miss rate and power consumption. It calculates a score based on the weighted-average principle. It assigns weights to each objective and then sums them, converting multiple objectives into a single aggregated scalar objective function. The Pareto front and its graph-related values for the cc application are presented in the next Chapter (Figure 6.5 and Table 6.7, Section 6.3, "Pareto-Optimal Cache Configuration Selection").

Using the predicted target values from the proposed models across 28 cache configurations, a Pareto-optimal analysis was conducted to determine the optimal cache configuration for the L1-I and L1-D cache DSEs in terms of miss rate and power consumption for all 20 applications. The same analysis was also performed on gem5-simulated values for comparison. In the next Chapter (Section 6.3, Pareto-optimal Cache Configuration Selection), the case study compares the identified cache configuration, which provides a balanced trade-off between miss rate and power consumption for the application, based on predicted values, with the simulated data.

5.4. Summary

In this chapter, the experiment, carried out mainly using the training dataset generation and the training of the ML models with the tools mentioned in the previous chapter, along with the Pareto analysis method, has been presented. This data set serves as the basis for the results and observations in the next chapter.

CHAPTER – 6

Results and Observations

This chapter presents the study's key results and observations from the information generated in the previous chapter. It compares the performance of trained regression models (Section 6.1) and evaluates the generalization capacity of the proposed models across all applications (Section 6.2). This chapter also identifies optimal cache configurations via Pareto analysis (Section 6.3) and analyzes the runtime and computational complexity of the proposed framework (Section 6.4). Furthermore, this chapter compares the proposed model with prior ML-based work (Section 6.5) and discusses the key insights from this work (Section 6.6).

6.1. Comparison of Regression Models

Table 6.1 (for L1-I) and Table 6.2 (for L1-D) compare various trained regression models for target value prediction, using 10-fold cross-validation on the training dataset to identify the best-fit model. Among them, ARRF and MLP outperformed in predicting the miss rate and power consumption, respectively, with high correlation coefficients and low errors, as highlighted in Tables 6.1 and 6.2. A higher correlation coefficient is better; lower errors (MAE and RMSE) are better.

Additive regression (AR) is a gradient boosting technique that fits a model to the residuals left by the base classifier from the previous iteration and makes predictions by combining them, thereby improving its performance with each iteration [91][93][99]. The RF [100][101] is known to be used for regression and classification-based problems. It builds many decision trees from subsets of the data and combines their outputs, thereby reducing overfitting and improving accuracy. It is highly significant and widely used for producing reliable results. It uses bagging, making it an ensemble learning method.

TABLE 6.1 Comparison of regression models for target value prediction of L1-I

Target	ML models	Correlation coefficient	MAE	RMSE
Miss rate	ARRF	0.987	0.002	0.005
	RF	0.977	0.003	0.007
	RandomSubspace	0.896	0.010	0.017
	Knn	0.923	0.007	0.014
	MLP	0.742	0.017	0.023
Power	MLP	0.992	0.005	0.006
	ARRF	0.989	0.007	0.010
	SMOReg	0.980	0.008	0.009
	RandomSubspace	0.968	0.030	0.046
	Knn	0.966	0.009	0.029

An ensemble learning technique can enhance the model's performance after combining different models [100]. The additive regression with random forest base learner (ARRF) could learn nonlinear patterns of the training dataset and provided higher accuracy in predicting the target value (miss rate) across different applications. Thus, the combination of additive regression and RF (an ensemble learning technique) in the proposed model yields better performance than other ML models.

TABLE 6.2 Comparison of regression models for target value prediction of L1-D

Target	ML models	Correlation coefficient	MAE	RMSE
Miss rate [67]	ARRF	0.989	0.006	0.011
	RF	0.976	0.011	0.024
	RandomSubspace	0.913	0.027	0.046
	Knn	0.881	0.027	0.055
	MLP	0.854	0.037	0.056
Power	MLP	0.994	0.004	0.006
	ARRF	0.986	0.005	0.008
	SMOReg	0.986	0.004	0.008
	RandomSubspace	0.966	0.023	0.037
	Knn	0.964	0.007	0.026

An MLP is a type of ANN [31] that uses backpropagation for training and supports a feedforward NN with one or more hidden layers [25]. ANNs are recognized as effective models for predicting power consumption [16][102]. Compared to linear regression, an ANN can model nonlinearity [25]. Compared to a decision tree, an ANN can be trained to output

a continuous range of values, whereas a decision tree is typically trained to output a few discrete values [16].

6.2. Generalization Ability of Proposed Models

As mentioned in the previous chapter, using the leave-one-out concept [64], the proposed model's generalization ability was evaluated by predicting the target value and calculating its MAE and RMSE using the formulas as provided in the previous chapter (Equations 5.3 and 5.4). Figure 6.1 compares the proposed ARRF model's MAE with two other ML models (ARRF vs. an RF and a random subspace) across 20 applications for miss-rate prediction on L1-I (a) and L1-D (b).

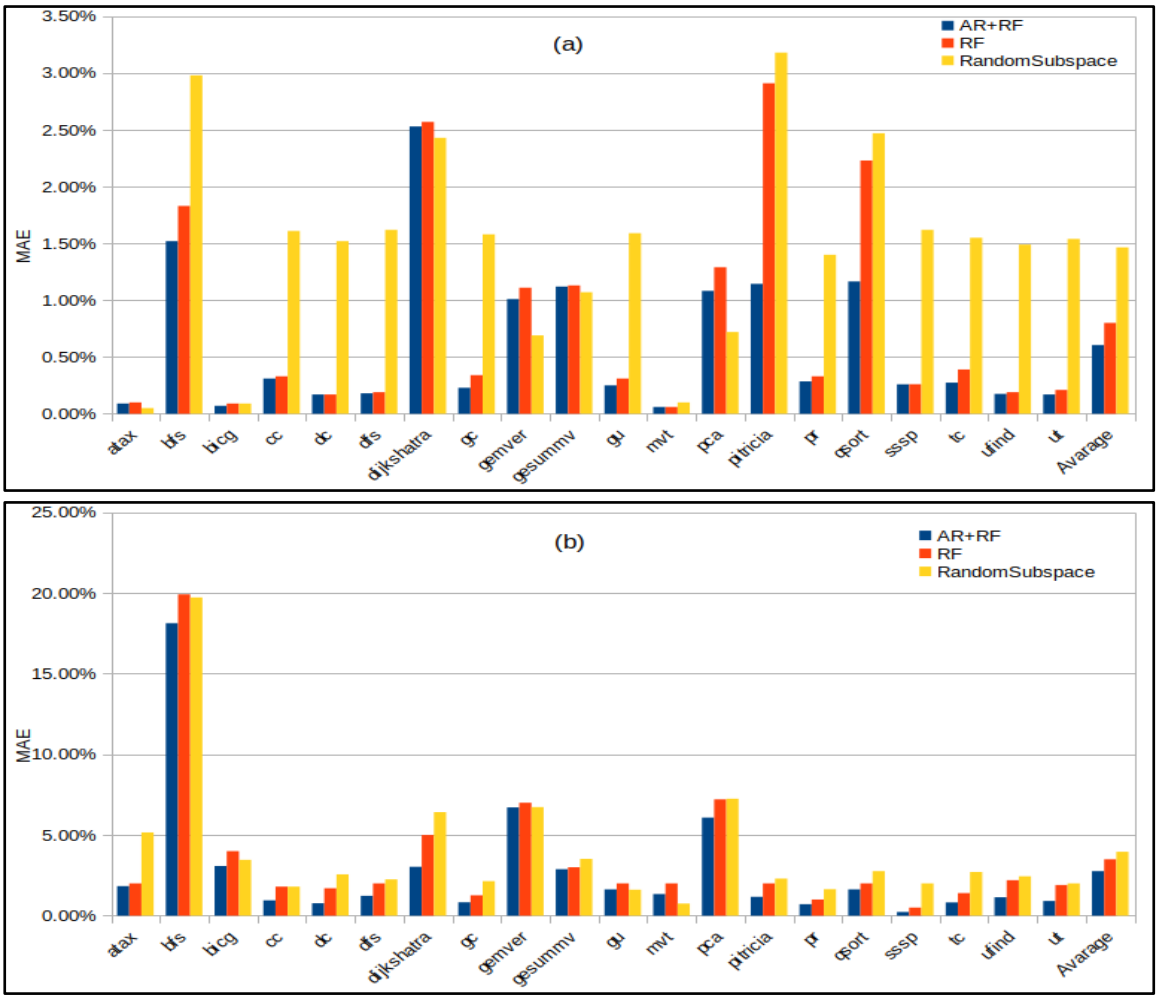


FIGURE 6.1 MAE for miss rate prediction for L1-I (a) and L1-D (b)

For power prediction, Figure 6.2 compares the proposed MLP model's MAE with those of two other ML models (MLP vs. an SMOreg and an ARRF) across 20 applications, specifically L1-I (a) and L1-D (b). The same comparison is presented in Figures 6.3 and 6.4 for RMSE.

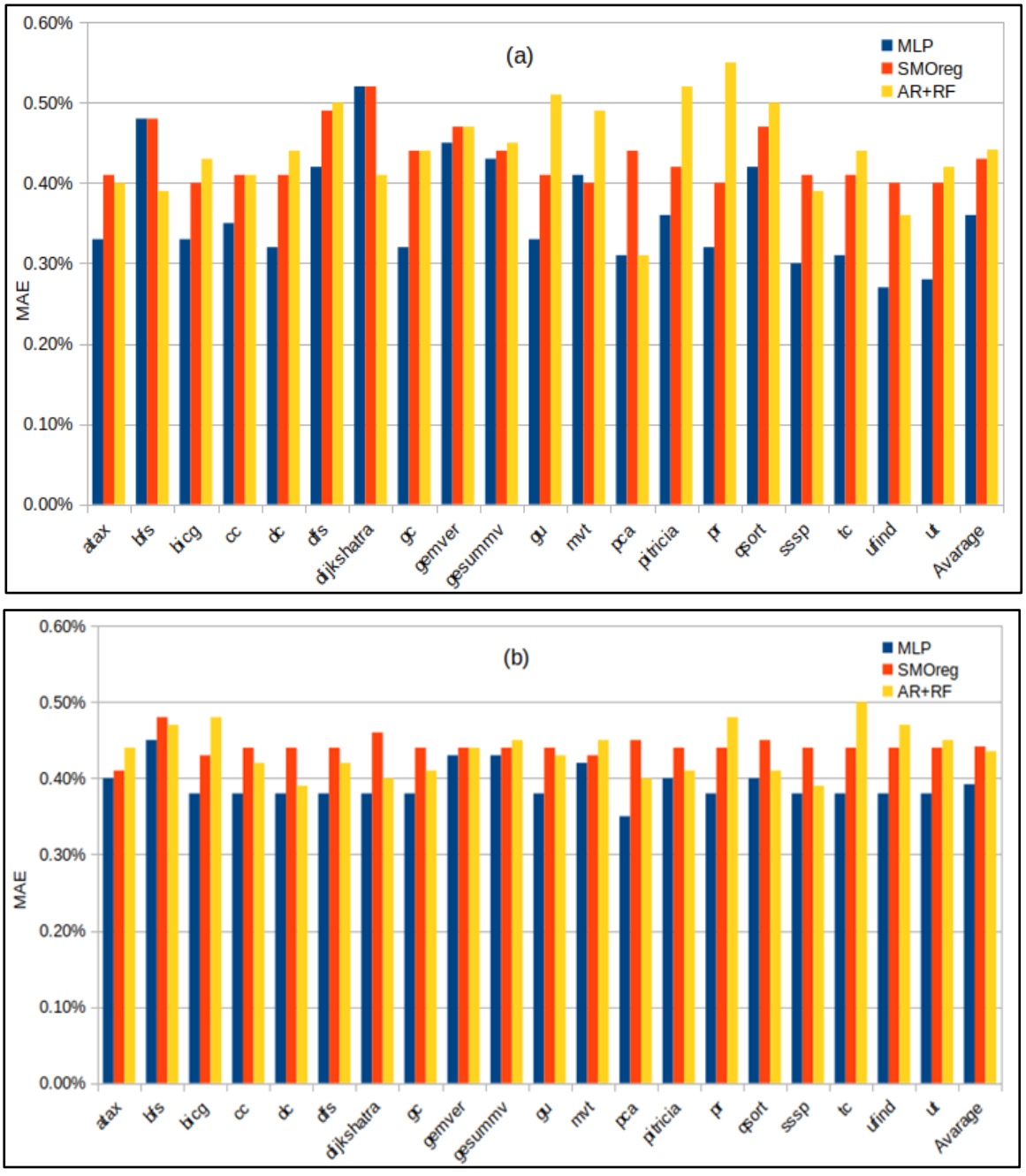


FIGURE 6.2 MAE for power prediction for L1-I (a) and L1-D (b)

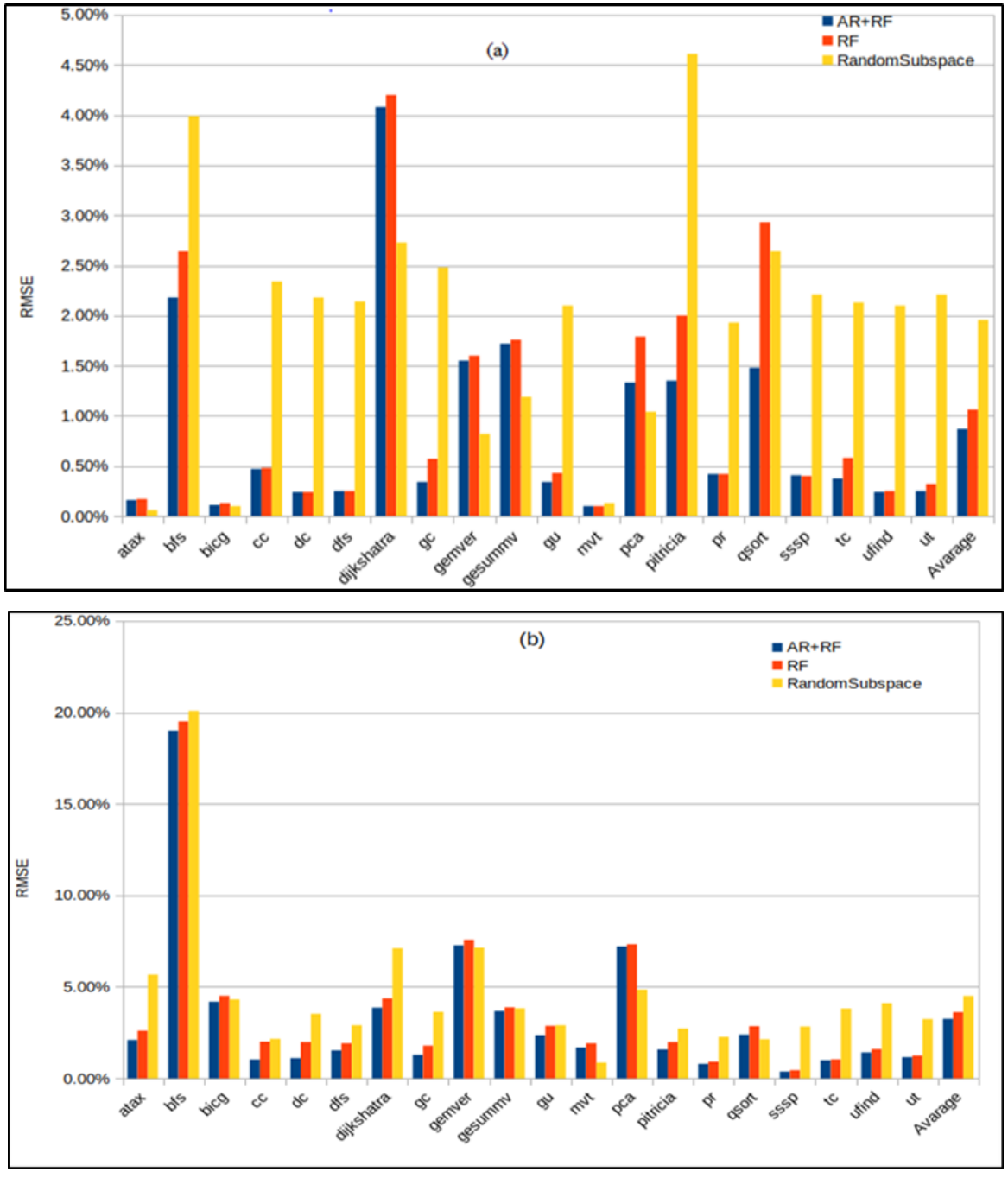


FIGURE 6.3 RMSE for miss rate prediction for L1-I (a) and L1-D (b)

FIGURE 6.4 RMSE for power prediction for L1-I (a) and L1-D (b)

These comparisons demonstrate that, in most cases, the average MAE and RMSE, and their mean values, are lower than those of other evaluated ML models, indicating that the proposed model can nearly predict the target value across most considered applications.

Although Figures 6.1-6.4 provide a graphical comparison of the evaluated ML models based on application-wise average prediction error, Tables 6.3 and 6.4 present an additional statistical comparison using 95% confidence intervals (CIs) for the average prediction errors. The CI provides a statistical measure of variability in average model performance [103],

along with an additional analysis to enable reliable comparisons among evaluated ML models. 95% CIs were calculated by following the work described by the author in [103].

Table 6.3 presents the average MAE and RMSE, along with their corresponding 95% CIs, for the L1-I and L1-D miss rate predictions. Among the three evaluated models, the ARRF's average MAE and RMSE are the lowest, and its CIs are the smallest for L1-I, indicating good prediction accuracy and consistent performance across applications. The ARRF's average MAE and RMSE are the lowest, and its CIs for the compared models are similar for L1-D miss-rate prediction, indicating good predictive accuracy while maintaining comparable stability. Hence, the CIs analysis further supports the selection of the proposed ARRF model for miss-rate prediction.

TABLE 6.3 Statistical comparison of miss rate prediction models

ML models	MAE				RMSE			
	L1-I		L1-D		L1-I		L1-D	
	Mean	95% CI	Mean	95% CI	Mean	95% CI	Mean	95% CI
ARRF	0.60%	±0.31	2.76%	±0.019	0.87%	±0.005	3.25%	±0.020
RF	0.80%	±0.42	3.49%	±0.020	1.06%	±0.005	3.62%	±0.019
Random Subspace	1.47%	±0.41	3.96%	±0.019	1.96%	±0.006	4.51%	±0.019

Table 6.4 presents the average MAE and RMSE, along with their corresponding 95% CIs, for the L1-I and L1-D power predictions. Among the three evaluated models, the MLP has the lowest average MAE and RMSE across both tasks, indicating good predictive accuracy. The CIs for all evaluated models are very small, indicating that the prediction errors remain consistent across the applications. Hence, the CIs analysis further supports the selection of the proposed MLP model for power prediction.

TABLE 6.4 Statistical comparison of power prediction models

ML models	MAE				RMSE			
	L1-I		L1-D		L1-I		L1-D	
	Mean	95% CI	Mean	95% CI	Mean	95% CI	Mean	95% CI
MLP	0.36%	±0.0003	0.39%	±0.0001	0.48%	±0.0007	0.58%	±0.0001
ARRF	0.43%	±0.0002	0.44%	±0.0001	0.88%	±0.0002	0.82%	±0.0002
SMOreg	0.44%	±0.0003	0.44%	±0.0002	0.68%	±0.0004	0.62%	±0.0002

Table 6.5 presents the proposed model's new application accuracy, quantified by MAE and RMSE, for predicting target values for L1-I and L1-D. Lower MAE and RMSE values indicate better predictive accuracy. A small prediction error across the majority of applications demonstrates the proposed methodology's ability to capture the non-linear relationships among cache configurations, application characteristics, and the corresponding target variable. It also shows that the proposed methodology generalizes well across applications selected from different benchmark suites (GraphBIG, PolyBenchC, MiBench, CortexSuite, and Rodinia) and diverse domains.

TABLE 6.5 Obtained MAE and RMSE values on the new applications

Model	**ARRF (Miss rate)**				**MLP (Power)**			
Metrics	**MAE (%)**		**RMSE (%)**		**MAE (%)**		**RMSE (%)**	
Applications	**L1-I**	**L1-D**	**L1-I**	**L1-D**	**L1-I**	**L1-D**	**L1-I**	**L1-D**
atax	0.09	1.83	0.16	2.10	0.33	0.40	0.40	0.58
bfs	1.52	18.13	2.18	19.00	0.48	0.45	0.87	0.64
bicg	0.07	3.08	0.11	4.19	0.33	0.38	0.40	0.54
cc	0.31	0.95	0.47	1.03	0.35	0.38	0.42	0.57
dc	0.17	0.77	0.24	1.11	0.32	0.38	0.40	0.57
dfs	0.18	1.23	0.25	1.53	0.42	0.38	0.50	0.57
dijkshatra	2.53	3.03	4.08	3.86	0.52	0.38	0.86	0.58
gc	0.23	0.83	0.34	1.29	0.32	0.38	0.40	0.57
gemver	1.01	6.71	1.55	7.27	0.45	0.43	0.59	0.62
gesummv	1.12	2.88	1.72	3.68	0.43	0.43	0.51	0.59
gu	0.25	1.63	0.34	2.36	0.33	0.38	0.41	0.57
mvt	0.06	1.34	0.10	1.68	0.41	0.42	0.49	0.59
pca	1.08	6.00	1.33	7.21	0.31	0.35	0.38	0.52
patricia	1.14	1.17	1.35	1.58	0.36	0.40	0.47	0.59
pr	0.29	0.71	0.42	0.79	0.32	0.38	0.42	0.56
qsort	1.17	1.63	1.48	2.39	0.42	0.40	0.52	0.59
sssp	0.26	0.23	0.41	0.37	0.30	0.38	0.40	0.57
tc	0.27	0.82	0.38	0.99	0.31	0.38	0.40	0.57
ufind	0.18	1.14	0.24	1.42	0.27	0.38	0.37	0.56
ut	0.17	0.91	0.25	1.16	0.28	0.38	0.37	0.56
average	0.61	2.76	0.87	3.25	0.36	0.39	0.48	0.58

Moreover, it is observed that smaller errors occur in many applications, whereas relatively larger prediction errors occur in a few applications. These differences show that the prediction accuracy depends on the application's memory access patterns and its characteristics. Also, the prediction errors are sufficiently small for cache DSE, as the aim

of the proposed methodology is to rapidly predict cache performance with minimal loss in accuracy, thereby helping identify suitable cache configurations for applications before detailed architectural evaluation. The discussion about the applications showing relatively higher prediction error is given in Section 6.6.

Table 6.6 illustrates the proposed model's generalization ability by reporting the average MAE and RMSE for target prediction on new data for L1-I and L1-D. Overall, the proposed model's average MAE across all applications for L1-I and L1-D indicates adequate generalization to new applications. The discussion about the average MAE and RMSE for target values prediction is given in Section 6.6

TABLE 6.6 Average MAE and RMSE values on the new applications

Target	MAE (%)	RMSE (%)
For L1-I		
Miss rate	0.61	0.87
Power	0.36	0.48
For L1-D		
Miss rate	2.76	3.25
Power	0.39	0.58

6.3. Pareto-Optimal Cache Configuration Selection

Using Pareto–optimal analysis in the last stage of the research methodology, as shown in Chapter 3, it is possible to identify optimal cache configurations that achieve optimal miss rate and power. The Pareto-optimal analysis for cache DSE was performed using the proposed model-predicted and gem5-simulated values for comparison. This analysis identifies the set of miss-rate-power-optimal cache configurations for an application. The predicted target values for each application were analyzed across all 28 cache configurations using Pareto-optimal analysis (Table 5.8 presents all 28 cache configurations along with their miss rates and power values for the cc application).

Figure 6.5 shows the Pareto front graph for L1-D miss rate (x-axis) and power consumption (y-axis) in the cc application, comparing predicted (a) and simulated values (b). The graph shows dominated and non-dominated cache configurations. Non-dominated cache configurations represent the Pareto front (red line). Each point on the graph corresponds to

the miss rate (lower is desirable) and power consumption (lower is desirable). The graph shows all three types of cache configurations: low-miss-rate (high-power-consuming), low-power (high-miss-rate), and balanced (miss-rate-power tradeoff), which constitute a small subset of the whole design space of 28 considered cache configurations (considered 28 cache configurations are shown in Chapter 5, Table 5.2, “Cache configurations design space”).

The Pareto front representation reveals that both miss rate and power consumption are conflicting objectives. Figure 6.5 shows the overall similarities in the positions and shapes of the predicted and simulated Pareto front graphs, suggesting that the ML-predicted Pareto front for the cc application is nearly accurate with the simulated result. The cache design space can be explored using the proposed model in the same way as the simulation. To demonstrate the Pareto front of Figure 6.5, Table 6.7 presents the Pareto-optimal cache configurations (generated by the Python program Brute-Force applied to the data from Table 5.8) for L1-D in the cc application, along with their miss rates and power consumption (predicted and simulated). Table 6.7 lists the 10 Pareto-optimal cache configurations (as predicted and simulated) out of 28 in total. The last column shows their min-max scores.

One can select the cache configuration from the Pareto-optimal configurations obtained based on the application's requirements; i.e., Configurations 28, 27, 26, and 25 are low-miss-rate configurations with higher power consumption due to their higher cache sizes. Configurations 23, 22, 21, and 19 are lower-power configurations with higher miss rates due to their lower cache sizes. Configuration 3 is balanced (miss-rate-power tradeoff). Based on its minimum score (as highlighted), obtained with balanced target values. Both the predicted and simulated results match; configuration 3 (32KB-64B-4Way) is the balanced cache configuration for the cc application.

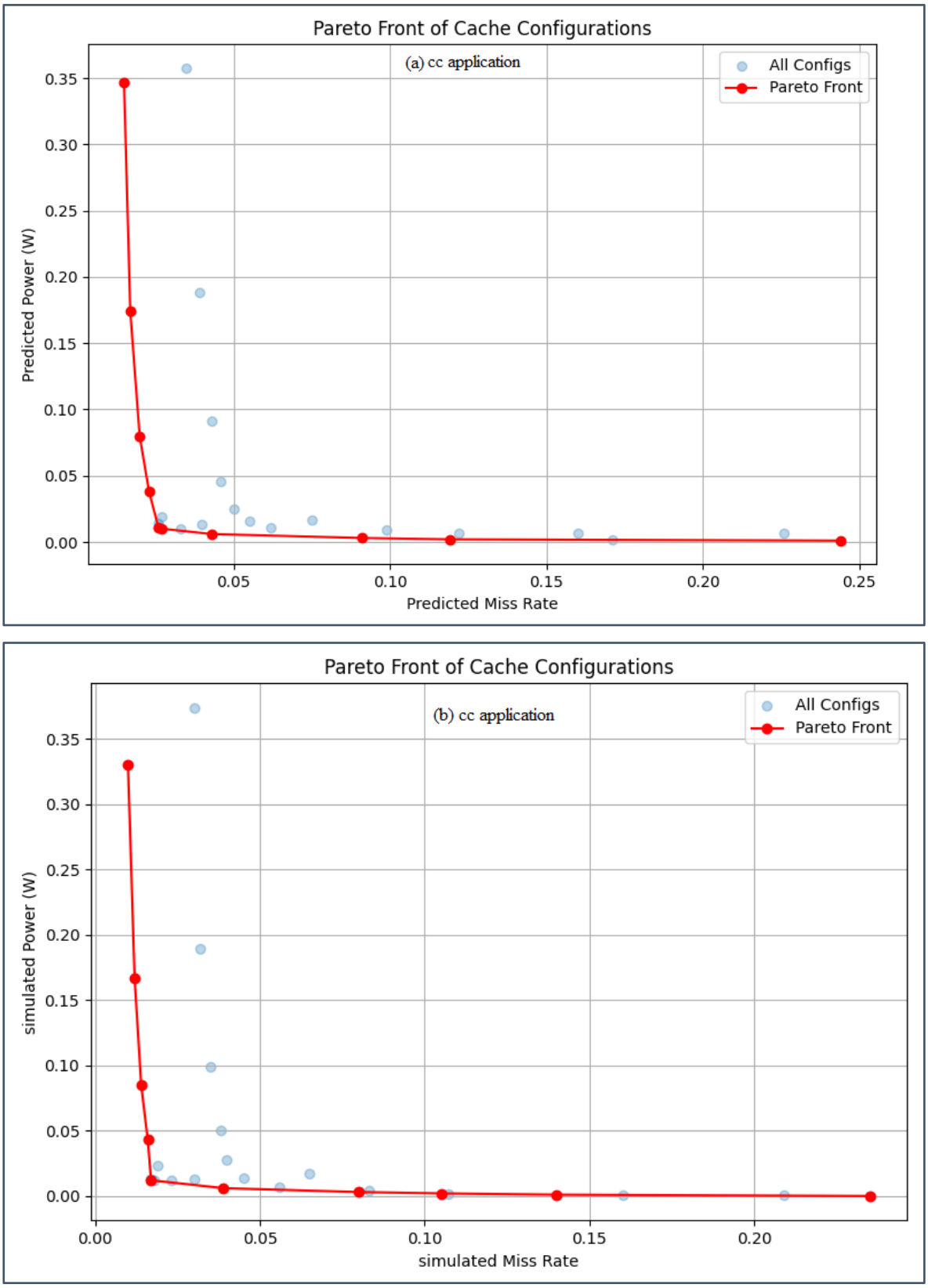


FIGURE 6.5 Comparison of predicted (a) and simulated (b) Pareto front for the cc application

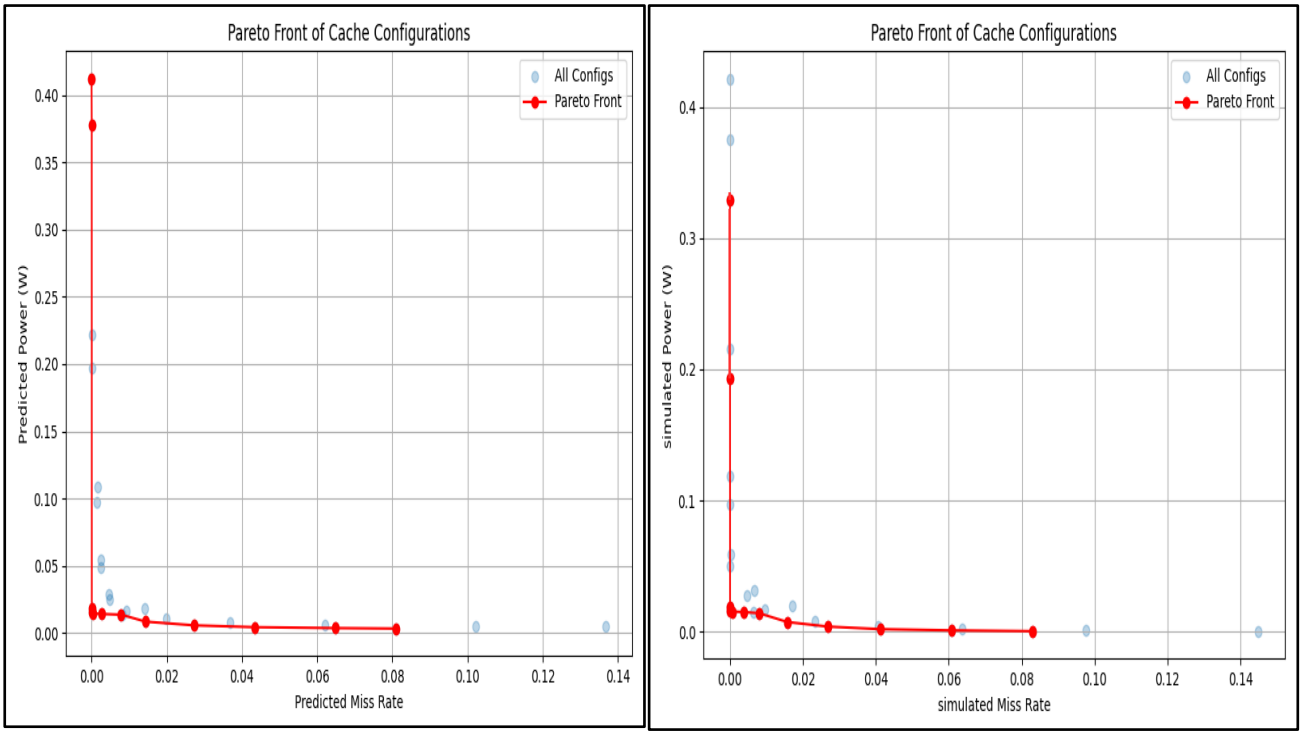
TABLE 6.7 Pareto-optimal cache configurations for the cc application

Cache config	Miss rate	Power	Min-Max score	Cache config	Miss rate	Power	Min-Max score
Predicted				**Simulated**			
Config 3	0.026	0.011	0.0385	Config 3	0.017	0.012	0.0337
Config 2	0.027	0.010	0.0392	Config 23	0.039	0.006	0.0735
Config 23	0.043	0.006	0.0684	Config 25	0.016	0.043	0.0785
Config 25	0.023	0.038	0.0709	Config 26	0.014	0.085	0.1377
Config 26	0.020	0.080	0.1251	Config 22	0.080	0.003	0.1601
Config 22	0.091	0.003	0.1688	Config 11	0.105	0.002	0.2141
Config 21	0.119	0.002	0.2285	Config 27	0.012	0.167	0.2575
Config 27	0.017	0.174	0.2544	Config 10	0.140	0.001	0.2904
Config 28	0.015	0.347	0.5000	Config 28	0.010	0.33	0.5000
Config 19	0.244	0.001	0.5000	Config 19	0.235	0	0.5000

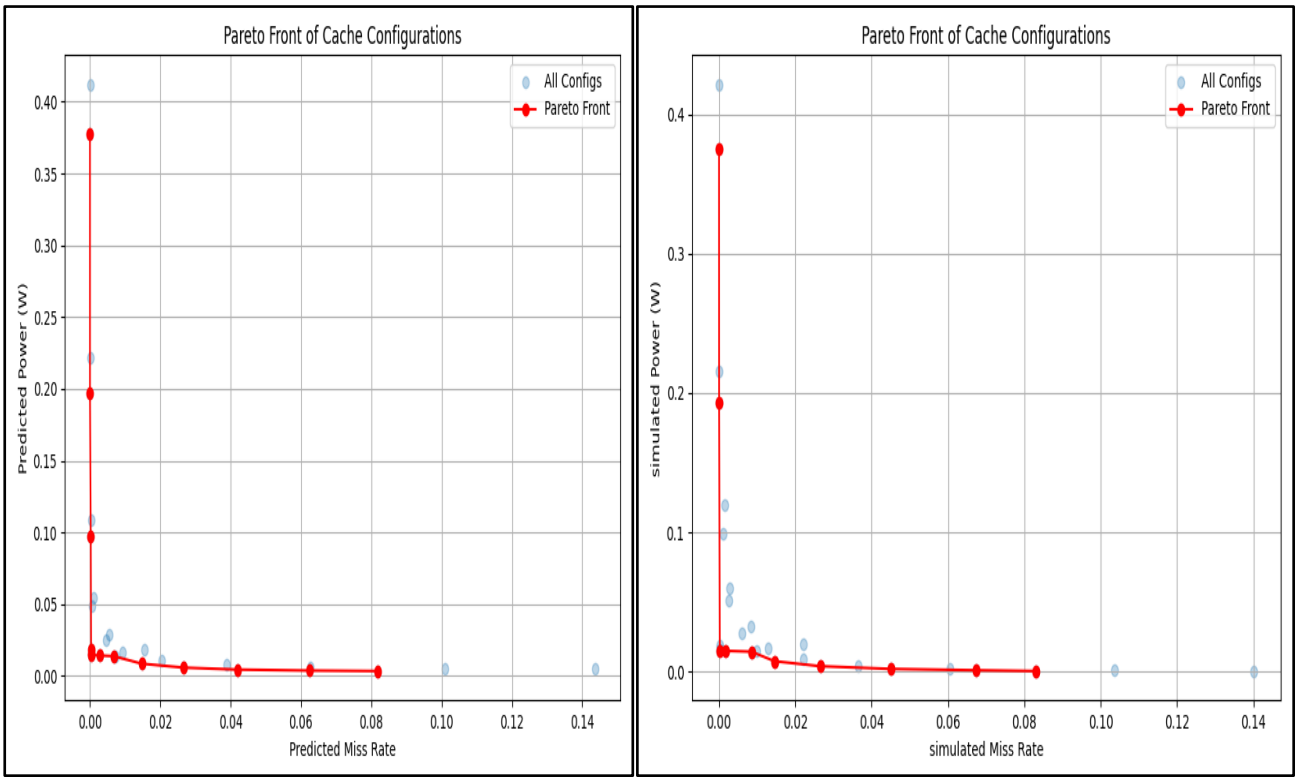
For the case study, the predicted and simulated miss-rate-power-balanced L1-I cache configurations for four applications (ut, dc, dfs, and bicg), along with their miss rates and power consumption, are shown in Table 6.8. An ut, dc, and dfs show superior alignment between predicted and simulated optimal configurations. The miss rate and power values obtained from the proposed method's predictions were very close to those from simulations. It shows that ML-based prediction and selection can be effectively applied to cache DSE in these applications. For bicg, the simulated Pareto-optimal selection was 32KB-64B-4-way, while the proposed model predicted 32KB-64B-16-way. Both configurations showed a balanced trade-off between miss rate and power, but differed slightly in associativity. The predicted and simulated Pareto front graphs for these applications (ut, dc, dfs, and bicg) are shown in Figure 6.6.

TABLE 6.8 Miss-rate-power-balanced L1-I cache configurations for representative applications

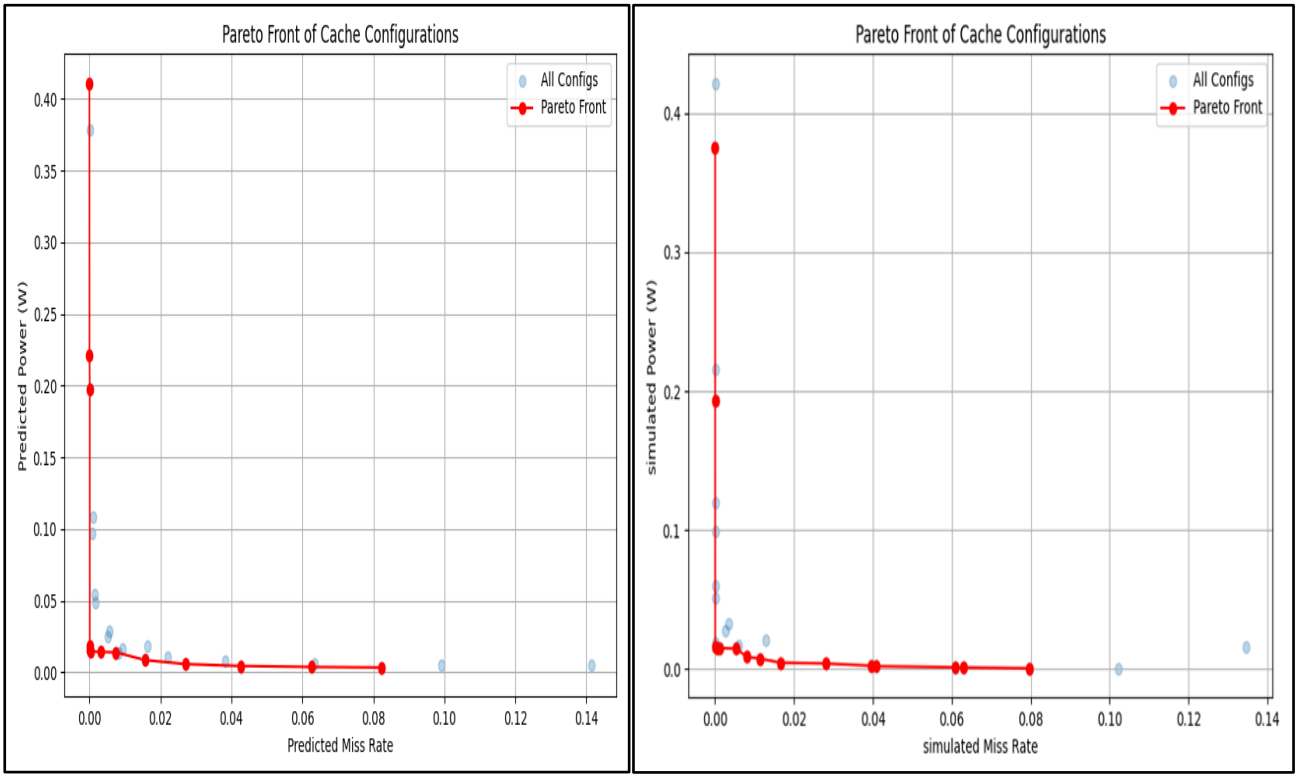
Applications	Method	Balanced cache configuration (Size-Line-Assoc.)	Miss rate	Power(W)
ut	Predicted	32KB-64B-8-way	0.0001	0.0156
	Simulated	32KB-64B-8-way	0.0001	0.0159
dc	Predicted	32KB-64B-4-way	0.0004	0.0150
	Simulated	32KB-64B-4-way	0.0002	0.0152
dfs	Predicted	32KB-64B-8-way	0.0001	0.0156
	Simulated	32KB-64B-8-way	0.0002	0.0159
bicg	Predicted	32KB-64B-16-way	0.0002	0.0193
	Simulated	32KB-64B-4-way	0.0005	0.0154



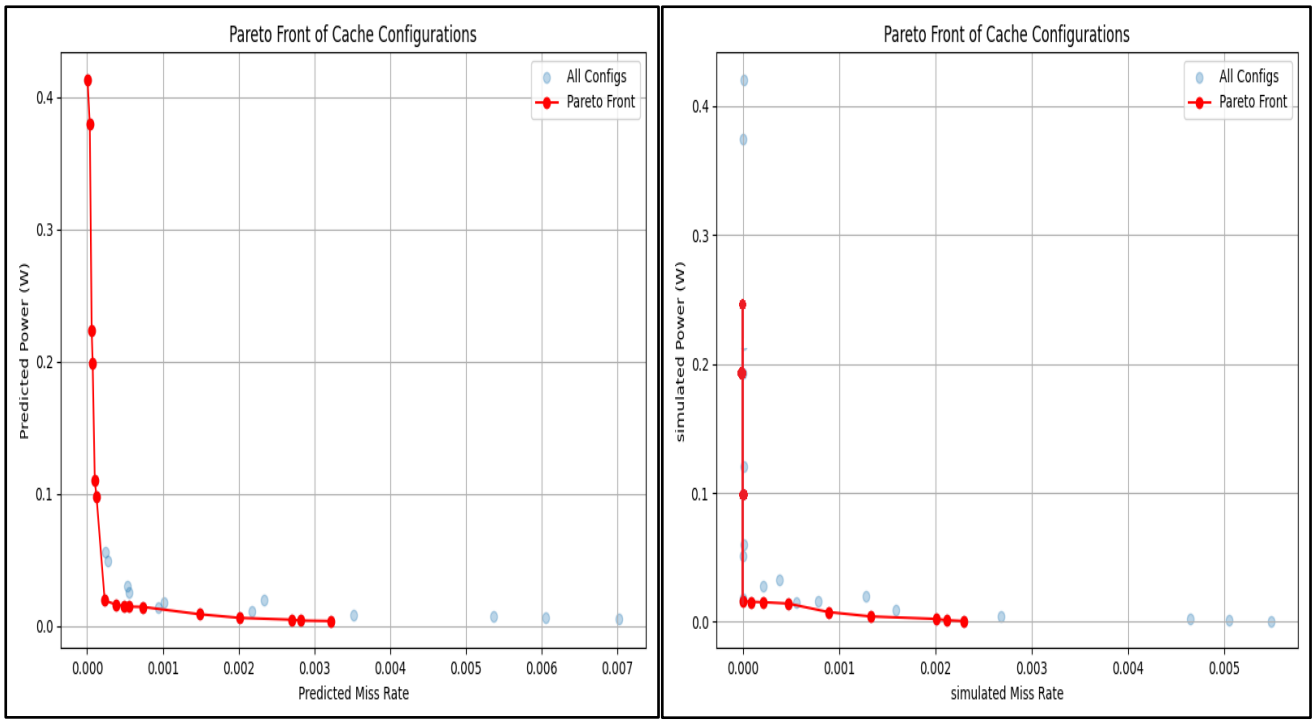
(a) ut


Pareto Front of Cache Configurations
All Configs
Pareto Front
Predicted Power (W)
Predicted Miss Rate
Pareto Front of Cache Configurations
All Configs
Pareto Front
simulated Power (W)
simulated Miss Rate

(b) dc


Pareto Front of Cache Configurations
All Configs
Pareto Front
Predicted Power (W)
Predicted Miss Rate
Pareto Front of Cache Configurations
All Configs
Pareto Front
simulated Power (W)
simulated Miss Rate

(c) dfs



(d) bicg

FIGURE 6.6 Predicted and simulated Pareto front graphs of applications for L1-I (a) ut, (b) dc, (c) dfs, and (d) bicg

Table 6.9 presents the predicted and simulated balanced Pareto-optimal L1-I cache configurations for the remaining applications. In some cases, such as bfs, dijkstra, gemver, and qsort, the proposed models suggested a larger, more associative cache as optimal, whereas simulations favored a smaller, direct-mapped cache. i.e., for the bfs application, according to the proposed model, the balanced cache configuration is 32KB-64B-16-way, whereas, according to the simulation, the optimal one is 16KB-64B-1-way. The proposed model identified alternative yet valid configurations, demonstrating its potential to expand the set of efficient design points.

Table 6.10 shows the predicted and simulated miss-rate-power-balanced L1-D cache configurations, along with their miss rates and power values for dc, gu, sssp, ut, pr, and qsort applications. In most cases, the predicted and simulated Pareto-optimal results closely matched, demonstrating the reliability of the ML-driven DSE. The predicted and simulated Pareto front graphs for these applications (dc, gu, sssp, ut, pr, and qsort) are shown in Figure 6.7.

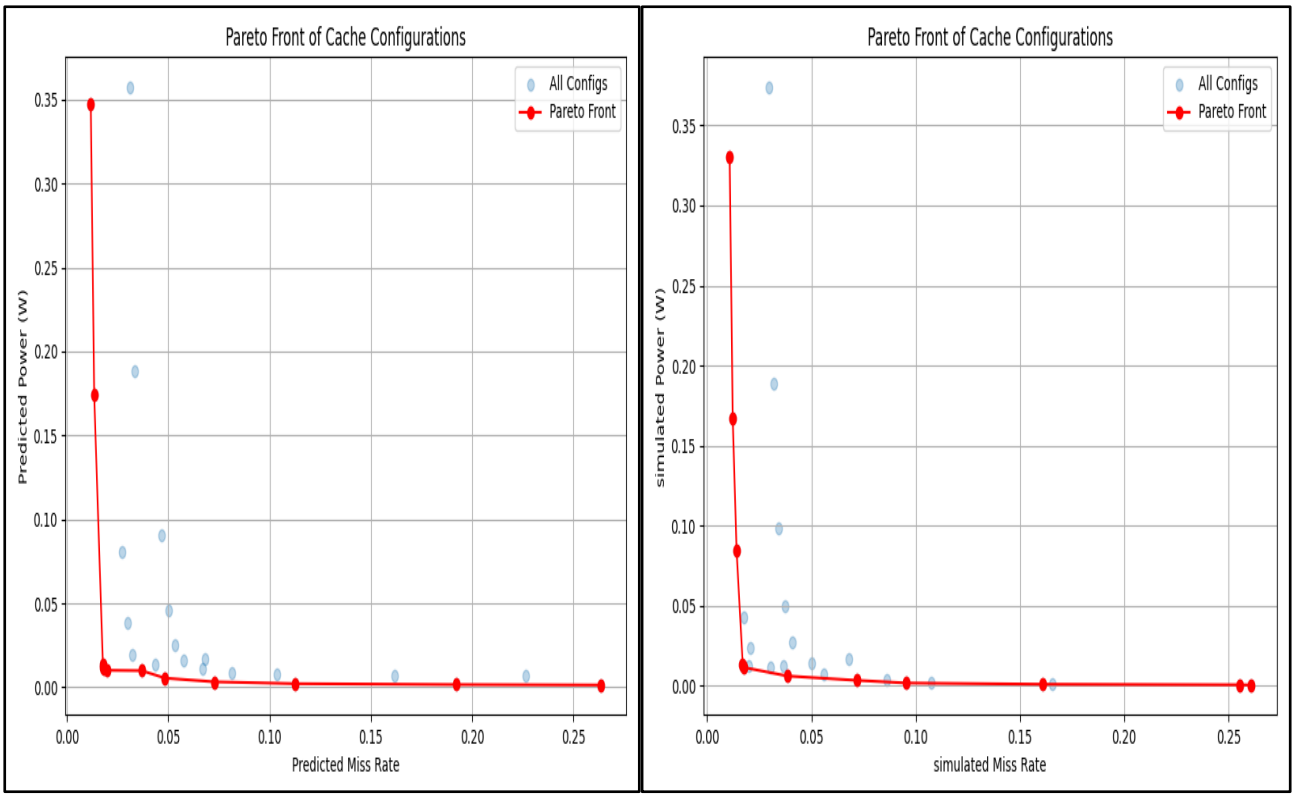
TABLE 6.9 Predicted & simulated miss-rate-power-balanced L1-I cache configurations for remaining applications

Application	Predicted balanced cache config. [Size (KB)-Line (B)-Assoc. (way)]	Simulated balanced cache config. [Size (KB)-Line (B)-Assoc. (way)]	Application	Predicted balanced cache config. [Size (KB)-Line (B)-Assoc. (way)]	Simulated balanced cache config. [Size (KB)-Line (B)-Assoc. (way)]
atax	32-64-4	8-64-1	mvt	32-64-4	16-64-1
bfs	32-64-16	16-64-1	pca	8-64-1	32-64-1
cc	32-64-8	32-64-8	patricia	8-64-1	32-64-16
dijkshatra	32-64-16	8-64-1	pr	32-64-8	32-64-8
gc	16-64-1	16-64-1	qsort	32-64-16	8-64-1
gemver	32-64-16	8-64-1	sssp	16-64-1	32-64-8
gesummv	8-64-1	8-64-1	tc	32-64-4	32-64-4
gu	32-64-8	32-64-8	ufind	32-64-4	32-64-4

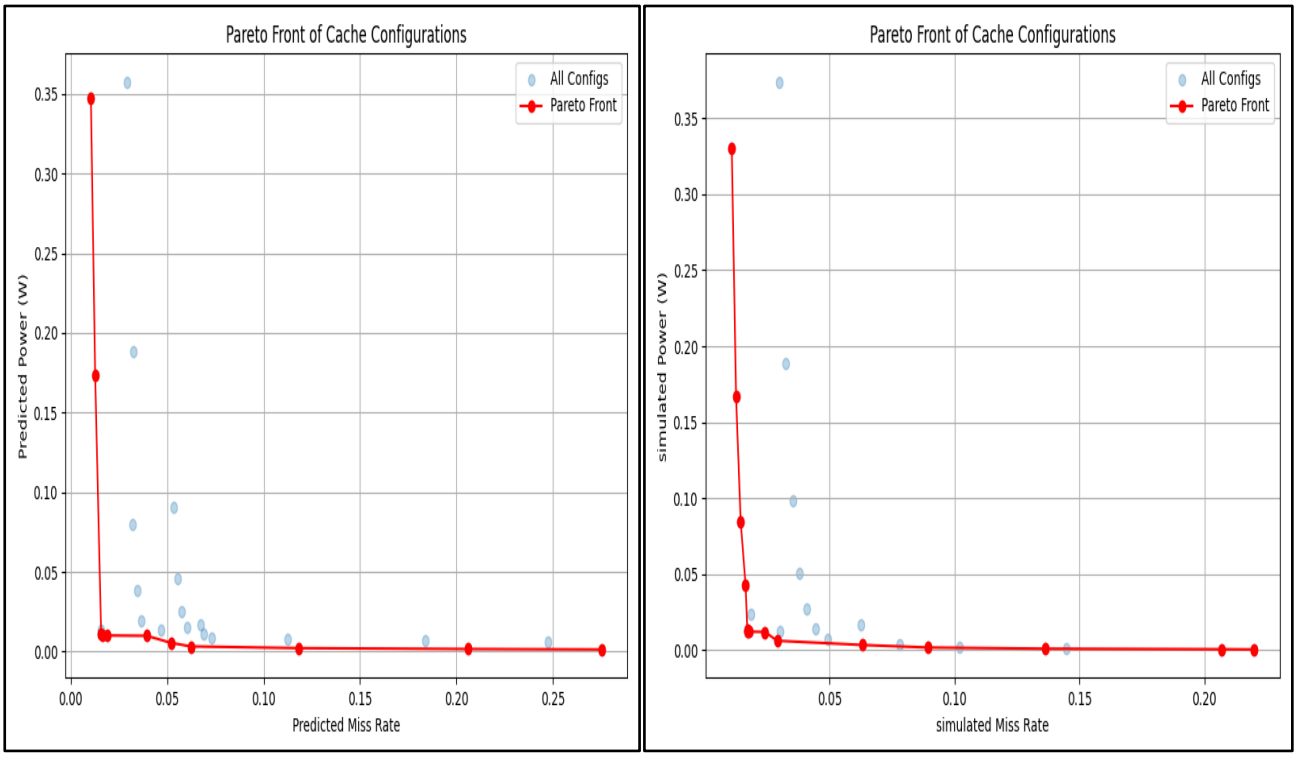
Table 6.11 represents the predicted and simulated balanced Pareto-optimal L1-D cache configurations for the remaining applications. The Pareto front graphs for the remaining applications for L1-D and L1-I are provided in Appendix D.

TABLE 6.10 Miss-rate-power-balanced L1-D cache configurations for representative applications

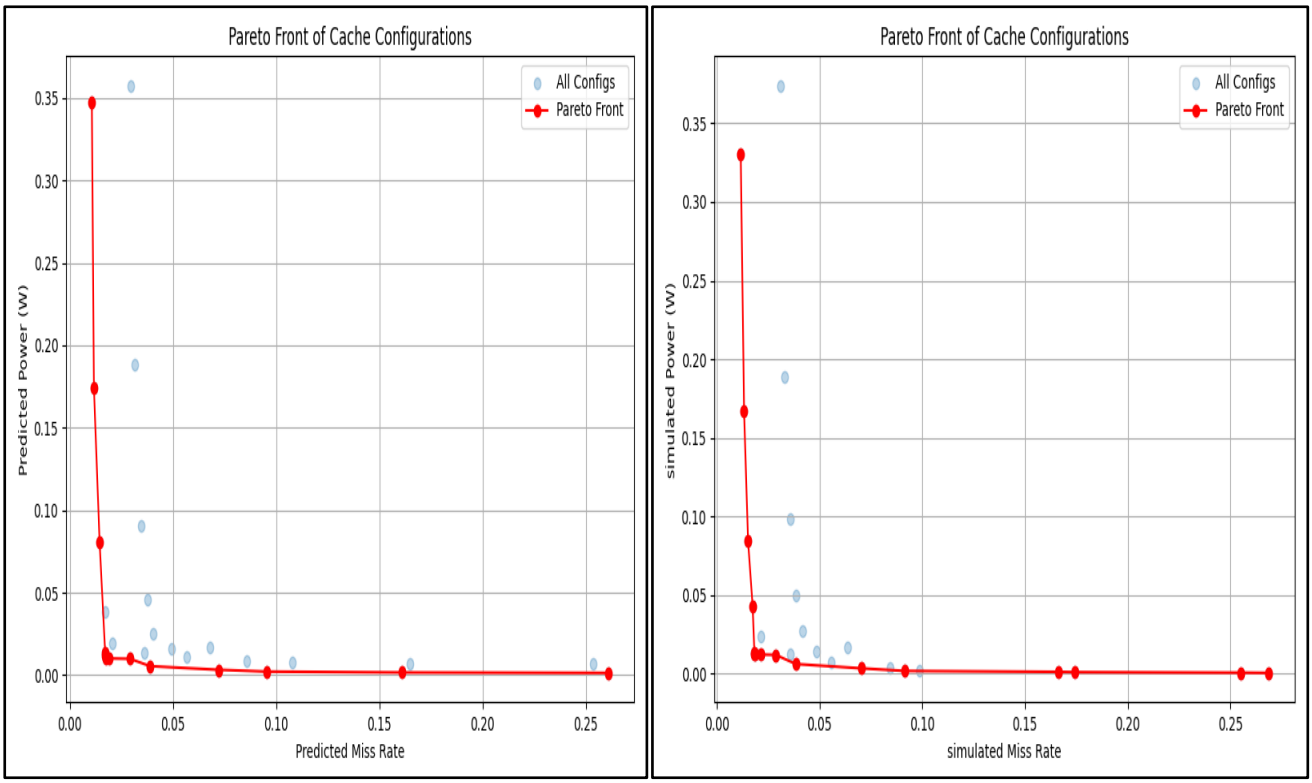
Applications	Method	Balanced cache configuration (Size-Line-Assoc.)	Miss rate	Power(W)
dc	Predicted	32KB-64B-4-way	0.0180	0.0110
	Simulated	32KB-64B-4-way	0.0176	0.0114
gu	Predicted	32KB-64B-4-way	0.016	0.011
	Simulated	32KB-64B-4-way	0.017	0.012
sssp	Predicted	32KB-64B-8-way	0.0174	0.0125
	Simulated	32KB-64B-8-way	0.0180	0.0130
ut	Predicted	32KB-64B-4-way	0.017	0.011
	Simulated	32KB-64B-4-way	0.018	0.0123
pr	Predicted	32KB-64B-8-way	0.018	0.012
	Simulated	32KB-64B-8-way	0.026	0.013
qsort	Predicted	32KB-64B-4-way	0.002	0.011
	Simulated	32KB-64B-4-way	0.005	0.012


Pareto Front of Cache Configurations
All Configs
Pareto Front
Predicted Power (W)
Predicted Miss Rate
Pareto Front of Cache Configurations
All Configs
Pareto Front
simulated Power (W)
simulated Miss Rate

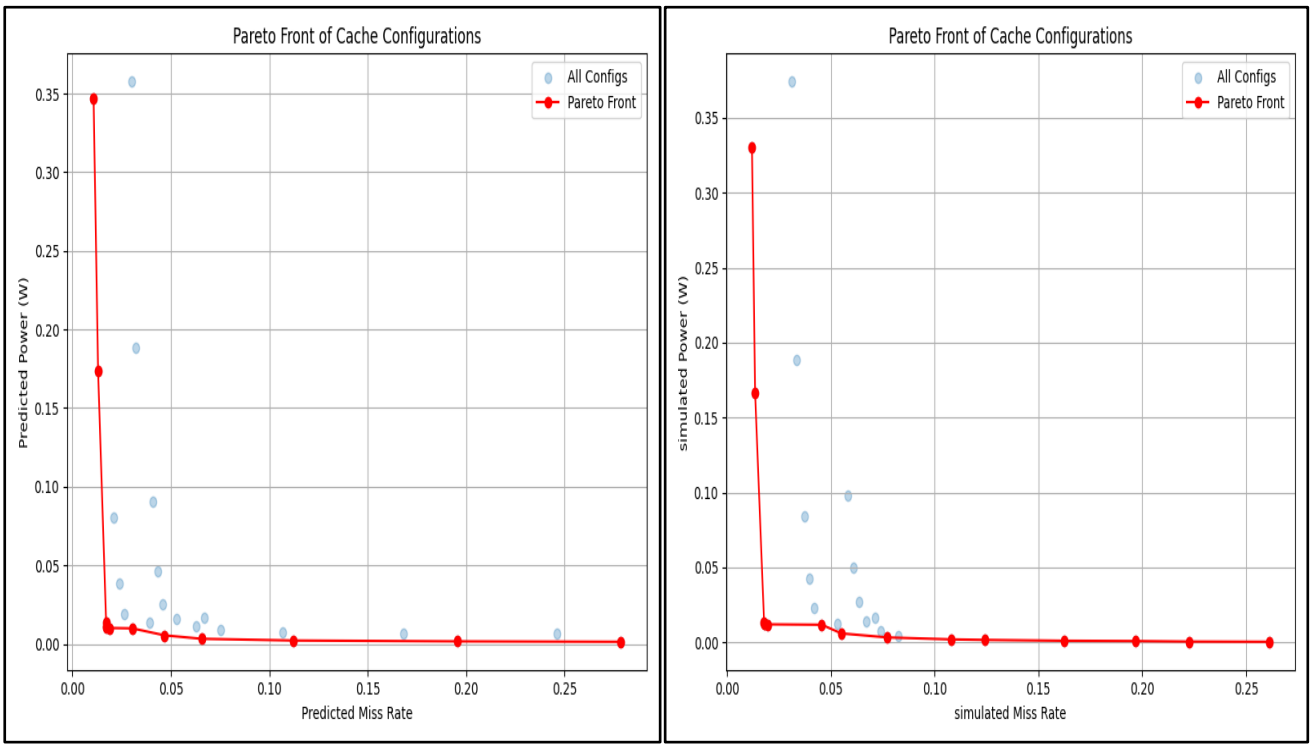
(a) dc


Pareto Front of Cache Configurations
All Configs
Pareto Front
Predicted Power (W)
Predicted Miss Rate
Pareto Front of Cache Configurations
All Configs
Pareto Front
simulated Power (W)
simulated Miss Rate

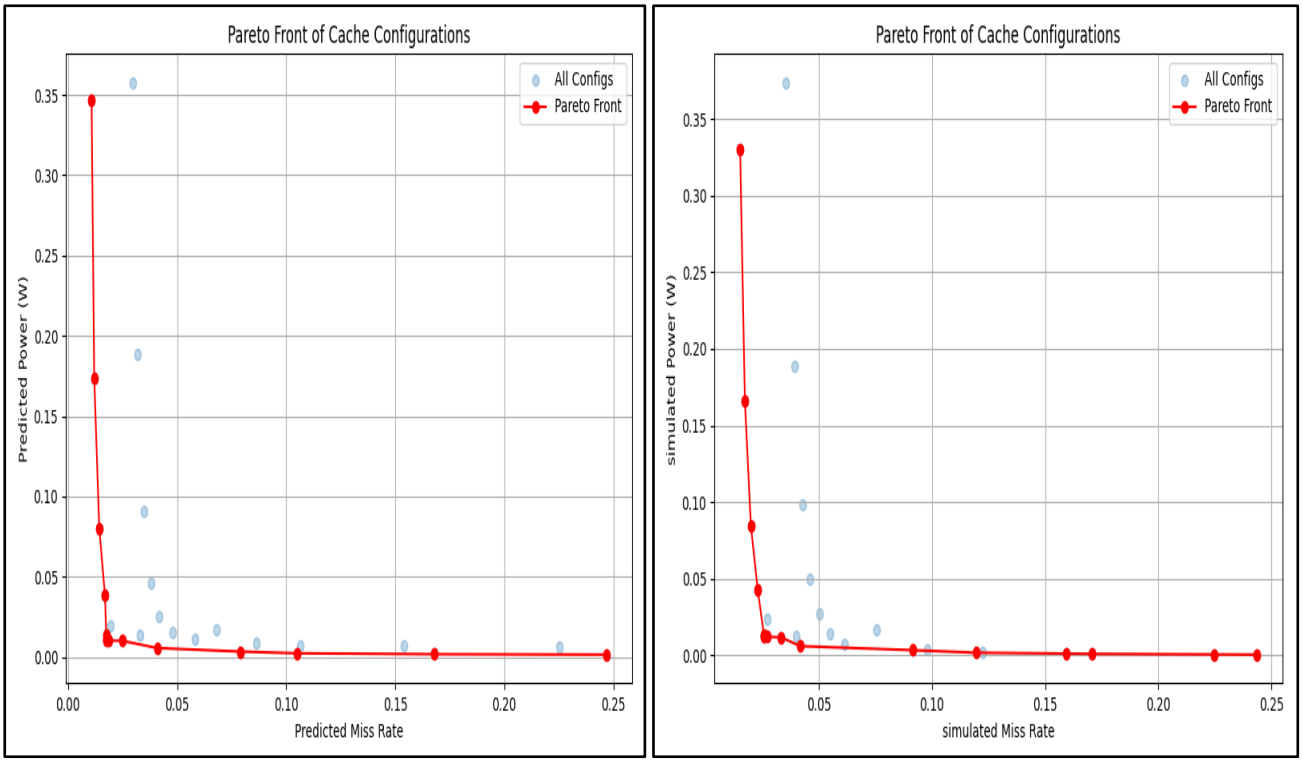
(b) gu


Pareto Front of Cache Configurations
All Configs
Pareto Front
Predicted Power (W)
Predicted Miss Rate
Pareto Front of Cache Configurations
All Configs
Pareto Front
simulated Power (W)
simulated Miss Rate

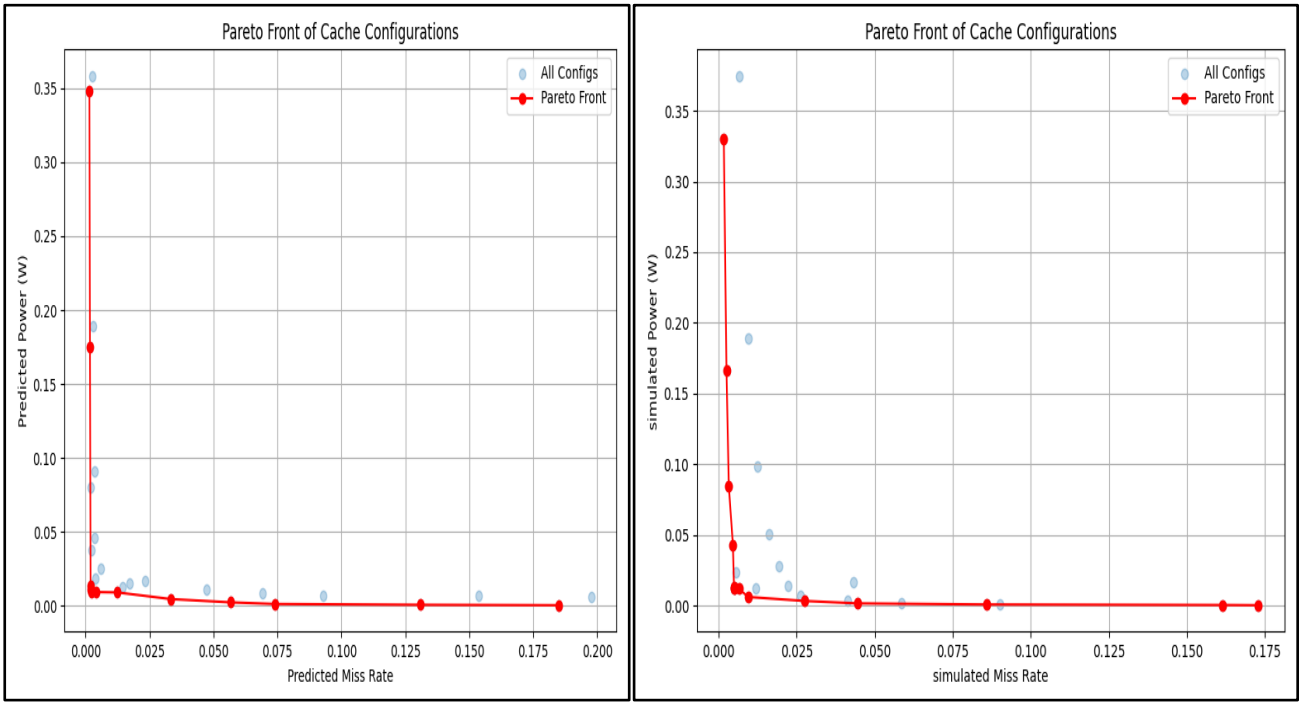
(c) sssp


Pareto Front of Cache Configurations
All Configs
Pareto Front
Predicted Power (W)
Predicted Miss Rate
Pareto Front of Cache Configurations
All Configs
Pareto Front
simulated Power (W)
simulated Miss Rate

(d) ut



(e) pr



(f) qsort

FIGURE 6.7 Predicted and simulated Pareto front graphs of applications for L1-D (a) dc, (b) gu, (c) sssp, (d) ut, (e) pr, and (f) qsort

TABLE 6.11 Predicted & simulated miss-rate-power-balanced L1-D cache configurations for remaining applications

Application	Predicted balanced cache config. [Size (KB)-Line (B)-Assoc. (way)]	Simulated balanced cache config. [Size (KB)-Line (B)-Assoc. (way)]	Application	Predicted balanced cache config. [Size (KB)-Line (B)-Assoc. (way)]	Simulated balanced cache config. [Size (KB)-Line (B)-Assoc. (way)]
atax	64-64-1	64-64-1	gemver	64-64-1	32-64-8
bfs	32-64-8	64-64-1	gesummv	32-64-8	32-64-8
bicg	32-64-2	32-64-8	mvt	32-64-8	32-64-8
cc	32-64-4	32-64-4	pca	32-64-4	32-64-4
dfs	32-64-4	32-64-4	patricia	32-64-4	4-64-1
dijkshatra	32-64-8	32-64-4	tc	32-64-4	32-64-4
gc	32-64-4	32-64-4	ufind	32-64-4	32-64-4

6.4. Runtime and Computational Complexity Analysis

This section examines runtime by reporting the achieved speedup relative to the architecture simulator and the computational complexity of the proposed framework.

6.4.1 Runtime and Speedup Analysis

The gem5 simulator can simulate only one cache configuration at a time, making it slow to analyze all 28 cache configurations for an application. i.e., 60 to 942 host clock minutes are required to simulate pca and gemver, respectively. At the same time, the proposed method simultaneously predicts the target values for all 28 cache configurations in an application in under two seconds. Some extra time is required to prepare the feature set for a new application to serve as input to the proposed model, e.g., collecting the application's characteristics from the Pintool, which takes approximately 5 minutes [67].

Thus, the overall speedup ranges from 12x to 148x (speedup was calculated as the ratio of simulation time to prediction time) compared to the simulator, enabling the target value to be achieved across all 28 cache configurations. Table 6.12 presents the time required to obtain target values using the gem5 simulator and the proposed method for all evaluated applications [67]. The ML methodology considerably decreases analysis time from hours (in

simulation) to minutes, with minimal loss in accuracy, aiding its practical adoption in the initial cache design phase.

TABLE 6.12 Time taken to get target values using the gem5 and the proposed method [67]

Sr. no.	Applications	Simulation time in minutes	Prediction time in minutes
1	atax	662	< 5 for all applications
2	bfs	540	
3	bicg	661	
4	cc	140	
5	dc	288	
6	dfs	140	
7	dijkshatra	180	
8	gc	298	
9	gemver	942	
10	gesummv	311	
11	gu	242	
12	mvt	645	
13	pca	60	
14	pitrica	140	
15	pr	180	
16	qsort	150	
17	sssp	285	
18	tc	304	
19	ufind	270	
20	ut	254	

6.4.2 Computational Complexity Analysis

The results of the runtime and speedup analysis indicate performance improvements in the proposed methodology's speed, as discussed in Subsection 6.4.1. This subsection presents a computational complexity analysis of the proposed method, further supporting the experimental observations.

Table 6.13 presents the computational complexity analysis of the proposed methodology's stages, from training dataset generation to ML model training, target prediction, and Pareto-optimal analysis. The computational cost of generating the training dataset and training the ML model was high (several hours to days) and moderate, respectively. To evaluate the cache performance of a new application, the trained ML model simultaneously predicts the target values for all 28 cache configurations at very low computational cost, enabling rapid

prediction of the L1 cache performance metric. Thus, once the training dataset is generated for the cache design space, the prediction cost remains negligible.

TABLE 6.13 Computational complexity analysis

Methodology stage	Computational cost	Execution frequency	Remarks
Training dataset generation (gem5, CACTI, Pintool)	High (several hours)	One-time	Required for training dataset generation
ML model training	Moderate	One-time	ML model training on the generated dataset
Target prediction	Very low	Whenever a new application is evaluated	Rapid prediction of the L1 cache performance metric for all 28 cache configurations simultaneously
Pareto-optimal analysis	$O(N^2)$, N=28	Once per application	Negligible computational overhead

6.5. Comparison with Prior ML-based Cache DSE Work

Among the past work discussed in Chapter 2, Vazquez et al. [16] shared the same aims as this thesis: evaluating L1 cache energy consumption and optimizing rapidly and accurately using their proposed regression-based ML model. The authors trained their ML model on application execution characteristics, gathered using the SimpleScalar simulator, as features to predict L1 cache energy. In this thesis, the models were trained using application characteristics (collected with Pintool) and cache configurations as features, and label values (miss rate and power consumption) gathered with a modern gem5 + CACTI setup. The authors [16] tested 15 EEMBC embedded benchmark applications across 18 L1 cache configurations, varying cache size, line size, and associativity. In contrast, this thesis work experimented with 20 applications from 5 widely used benchmark suites across diverse domains, including graph processing, linear algebra, embedded computing, and ML. The work [16] targeted specifically the cache energy prediction by training a single ANN model; this thesis work targets two metrics to be predicted, such as miss rate and power consumption, and trains metric-centric ML models; ARRF is employed for L1-D and L1-I miss rate prediction, and an MLP is employed for L1-D and L1-I power consumption

prediction. Thus, the proposed framework involves a multi-objective cache evaluation and optimization.

As mentioned in Chapter 2, the literature review, one of the early efforts to employ an ML regression model to predict the L1-D miss rate was reported in [66], demonstrating the possibility of using a trained ML model for cache performance prediction, and thereby reducing reliance on time-consuming simulation. However, their work primarily targets miss-rate prediction across 15 applications in 3 benchmark suites and does not consider power consumption or cache DSE. In contrast, this work covers the scope of L1 cache DSE by predicting miss rate and power consumption using ML models, based on 20 diverse applications from 5 benchmark suites. The quantitative comparison between the proposed model's average RMSE and the reported average RMSE in [66] for L1-D miss rate prediction is shown in Figure 6.8.

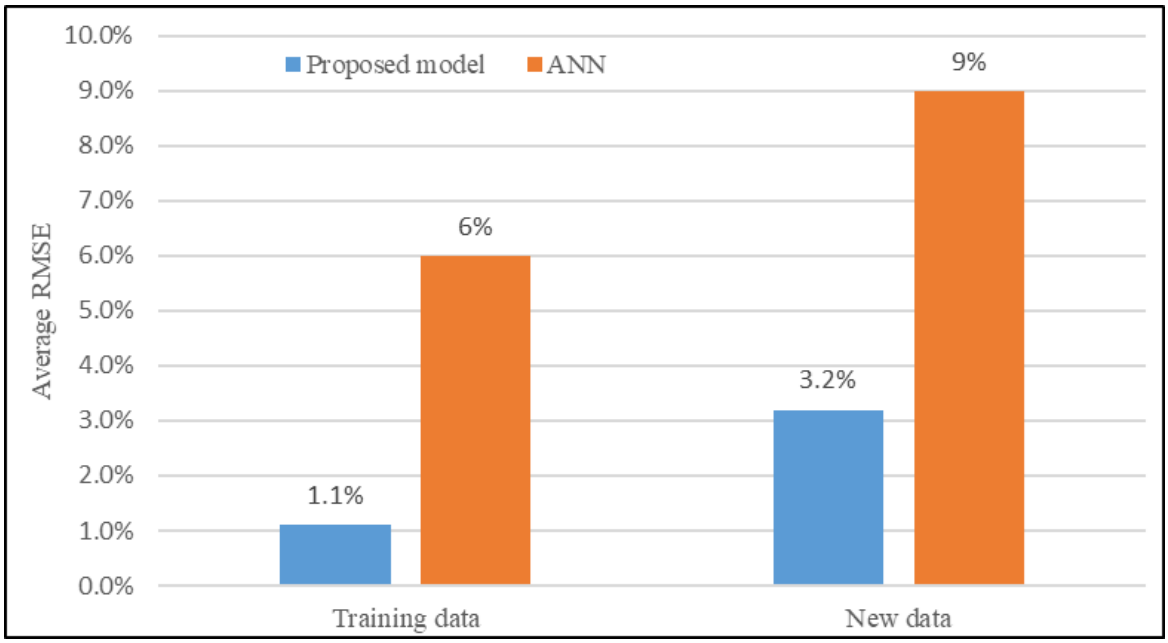


FIGURE 6.8 Comparison of average RMSE between the proposed ARRF and ANN for L1-D miss rate prediction [67]

The author [66] reported an average RMSE of less than 6% for the self-predicted L1-D miss rate of their trained ANN and, for new applications, around 9%. The average RMSE for the proposed trained model, ARRF, in self-predicting L1-D miss rate values is 1.1%, and for new applications, it is around 3.25%.

6.6. Discussion

This section discusses the key insights from the experimental results and general observations from the entire work on cache-miss rate and power prediction.

Based on the proposed model's experimental results, the L1-D cache's general behavior can be discussed in the same way as in the simulation results, with effects of associativity, line size, and cache size for the cc application, as shown in Table 5.8.

1. Associativity: Cache configurations 1-5 relate to associativity. For configurations 1 to 3, the miss rate decreases as associativity increases; thereafter, it remains constant and provides no significant benefit, indicating that increasing associativity benefits the cc application up to 4-way associativity. Higher associativity means more lines per set, which reduces the likelihood that data is not in the cache and thus reduces the miss rate. Still, it increases power consumption, as shown in Table 5.8.

2. Line size: Configurations 6-8 and configuration 1 are related to line size (fixed cache size = 32KB, associativity = 1, with line sizes varying from 8 to 64 bytes). As line size increases, miss rate decreases. Hence, a 64-byte line size is beneficial for achieving a reasonable miss rate in this case.

   Here, it can also be observed that line size depends on cache size. Let us consider the larger cache size configurations 7 (32KB-16B-1Way) and 1 (32KB-64B-1Way). By increasing the line size from 16B to 64B, the miss rate decreases (from 0.055 to 0.033) because a larger cache size reduces conflict misses and benefits from improved spatial locality. In contrast, the small-cache-size configurations 9 (1 KB-16B-1Way) and 19 (1 KB-64B-1Way), by increasing the line size from 16B to 64B, increase the miss rate (0.226 to 0.244), it is due to the number of lines decreasing with a 64B line size and a small cache, resulting in increased capacity and conflict misses at larger line sizes. It shows that the effect of line size on miss rate depends on the cache size and that this relationship is nonlinear.

   The following observation from Table 5.8 concerns the effect of line size on power consumption. Generally, a larger line size can increase dynamic energy due to more energy per access, but the results show that increasing the line size reduces overall power

consumption. This is due to its lower miss rate; fewer accesses resulted in lower power consumption, indicating that the cc application exhibits better spatial locality.

3. Cache size: To assess the effect of cache size on miss rate and power consumption, cache configurations 9-28 are considered for cc applications, as they are cache-size-related. As shown in Table 5.8, the miss rate decreased, while power consumption increased, as cache size increased.

The observations below discuss the results from Section 6.1.

1. The predictive ability of ARRF, an ensemble-based model, shows that the relationship between the feature set (cache design parameters and application characteristics) and the target value (cache performance) is nonlinear yet structured.

The observations below are discussed in light of the results from Section 6.2.

1. Some applications, such as bfs and gemver, exhibited higher prediction errors, as shown in Table 6.5. For miss-rate prediction, the L1-D MAE is 18.13% and 6.71% for bfs and gemver, respectively, due to their irregular memory access patterns and higher instruction counts. A bfs is a graph-handling application that navigates all vertices in the dataset. Each iteration interacts with or modifies other vertices, which are not necessarily adjacent, resulting in random and unequal access patterns [64][78]. The gemver application also differs in implementation and has more total dynamic instructions than other evaluated applications. As a result, the applications' characteristics are more complex than those of the other evaluated applications, making their cache behavior more challenging for the trained model to learn.

2. As shown in Table 6.6, the L1-D miss rate prediction errors are higher than those for L1-I. It shows that L1-D's overall miss rate is higher than L1-I's across all applications [78]. This difference may be due to the applications' access patterns: the former is more irregular, and it has more data transfer instructions, whereas the latter is more regular and has fewer ALU and control instructions. Yet, L1-D's error remains within the acceptable range for cache DSE. It does not indicate the method's weakness but shows the inherent variability of cache behavior.

3. As shown in Table 6.6, the difference between MAE and RMSE values is minor for both miss rate and power, indicating that the prediction errors are uniformly distributed without any significant outliers.

The observations below are discussed in light of the results from Section 6.3.

1. The Brute-force method is used in the Pareto-optimal analysis rather than well-known multi-objective evolutionary algorithms such as NSGA-II [96], as Brute-force does not use approximations; it is an exact method and is suitable here due to the limited design space (28 cache configurations) for generating a complete Pareto front.

2. The WSM method for selecting a balanced miss-rate-power cache configuration from a Pareto-optimal set is easy to use. Still, the final design decision depends on the weight values according to the design priorities. (This work set 0.5 weight for both miss rate and power as shown in Appendix C).

3. L1-I and L1-D cache sizes and their associativity are the most significant cache parameters in the miss-rate-power trade-off analysis, as they define most Pareto-optimal configurations.

4. Comparing predicted and simulated Pareto-optimal cache configurations across applications provides insights into the usefulness and limitations of the proposed ML-based cache performance prediction and DSE. It highlights the importance of feature selection and the need for a diverse set of datasets, including more cache configurations and applications from different benchmark suites.

Below is a discussion of general observations from the entire work.

1. After generating the training dataset for the L1 cache power label, it is observed that, across applications, L1-I power consumption is higher than L1-D because the processor accesses L1-I on every cycle. It is also observed that the L1-I power consumption is 0.427W for the bfs application with a 1024KB-16B-1-way cache. For the same application and cache configuration, L1-D power consumption is 0.376 W.

2. Training datasets are vital for ML, and related datasets are rarely available or too exclusive to find in the computer architecture and cache domains. Thus, the main challenge in training an ML model for cache-related prediction is collecting sufficient data to build a model that fits the data well and generalizes well. Also, prepare the training dataset, which contains diverse patterns and relevant features.

3. In this work, 28 L1 cache configurations (a design space) are evaluated to predict their performance rapidly. Expanding this cache design space is beneficial for preparing a large, diverse training dataset, as the proposed model can learn better. Moreover, for the performance prediction of the cache hierarchy (e.g., L2/L3) and the multicore processor, the relevant training dataset must be generated, and the corresponding architectural features must be included. In all cases, Pareto-optimal analysis based on predicted values would remain unchanged. In all cases, the simulation cost can be increased; however, once the training dataset is generated for the ample cache design space, the prediction cost remains negligible, demonstrating the proposed model's scalability for future work on expanded design spaces, cache hierarchies, and multiprocessors.

4. In ML, how well the model is trained depends on the training dataset, which represents the real task. Although the training dataset in the proposed work comprised a diverse set of applications and cache configurations, if applications have very different characteristics and memory access patterns, the proposed model may not generalize well, as this is a natural limitation of data-driven models. It can be overcome by continually adding datasets and periodically retraining the model.

5. In particular, the ML-based approach significantly reduces analysis time from hours (in simulation) to seconds, with minimal loss of accuracy, enabling its practical adoption in early design phases for pruning cache configurations. Thus, a key finding is the need for a hybrid exploration strategy that leverages ML-based models to identify candidate configurations and to refine the design space rapidly. Then, cycle-accurate simulations can be used for final validation. Thus, it balances exploration accuracy and the assurance of efficiency, making it valuable for large-scale or multilevel cache design studies.

6.7. Summary

This chapter presents results concerning the performance and power consumption predictions obtained from the proposed models. It compares various ML models using well-known metrics such as CC, MAE, and RMSE. The proposed model's generalization ability is evaluated by comparing it with two other models using MAE and RMSE.

Subsequently, the concept of Pareto-optimal analysis has been introduced to identify optimal cache configurations for an application based on predicted target values, thereby further reducing the overall design space, and to compare them with their actual values. From optimal configurations, a balanced miss-rate-power configuration is found as a case study. This chapter also shows the achieved speedup analysis and compares the proposed model with prior ML-based work. At last, the discussion section presents the main observations.

CHAPTER – 7

Conclusions and Future Work

7.1. Conclusion

This work presents an approach to accelerate the prediction of level-1 cache performance metrics using an ML method. Rapidly predicting L1 cache performance and power consumption for an application can accelerate DSE during the early cache design phase. The proposed approach is validated using 20 benchmark applications from 5 different benchmark suites. 28 cache configurations of L1-I and L1-D are considered by varying their design parameters, including size, line size, and associativity. Two ML regression models are trained for prediction: an ARRF for miss-rate prediction and an MLP for power prediction. These models are trained on a dataset that includes independent variables (cache configuration and the application's characteristics) and a dependent variable (miss rate and calculated power consumption). The label values are collected by simulating an application with all 28 cache configurations using the gem5 simulator. The CACTI tool is also used for label generation. An application's characteristics are gathered using Intel Pintool, and ML models are trained using the Weka platform. The proposed models are trained using 10-fold cross-validation to ensure high accuracy and low error.

The trained models are evaluated using well-known metrics such as the correlation coefficient, MAE, and RMSE. The trained model achieved a correlation coefficient of 0.987 and 0.992 for miss rate and power, respectively, for L1-I. For L1-D, correlation coefficients are 0.989 and 0.994 for miss rate and power, respectively. Once models are trained, they demonstrate good generalization by quickly predicting miss rates and power consumption for new (test) applications across 28 cache configurations without requiring additional simulations, with low MAE and RMSE. The average MAE is 0.61% (RMSE = 0.87%) for the miss rate, and 0.36% (RMSE = 0.48%) for power consumption prediction on new data for L1-I. The average MAE is 2.76% (RMSE = 3.25%) for the miss rate, and 0.39% (RMSE

= 0.58%) for power consumption prediction on new data for L1-D. The proposed method achieves speedups of 12x to 148x over gem5 simulations. This approach significantly reduces the designer's effort in exploring the cache design space by minimizing time-consuming simulations.

The proposed regression-based models enable the selection of Pareto-optimal cache configurations, similar to those obtained via simulation. The Pareto-optimal analysis can identify the trade-off between performance and power and select an optimal cache configuration from the entire design space, thereby reducing the cache design space (cache configurations) to be analyzed. For example, using the proposed model's predicted values for the cc application, a Pareto-optimal analysis identified 10 optimal cache configurations from the entire cache design space (28 cache configurations), representing a 64.29% reduction in the design space. The findings demonstrate the potential of ML-assisted cache DSE, which reduces simulation overhead and supports early design decisions with reliable predictions.

This research represents a significant step toward developing an ML-driven approach to evaluating cache performance parameters and DSE across various cache configurations, with the ultimate goal of designing an efficient cache with new capabilities for the next generation of applications.

7.2. Future Work

The mismatches in the predictions highlight opportunities for future work, including adding more features and applications to expand the training dataset and explore diverse cache configurations. That can improve the generalization ability. One can use approaches to include more closely similar samples in the training dataset, such as data augmentation (synthetic data generation) or generative models.

The framework can be further extended to predict additional cache performance metrics, such as execution time and IPC. It can also be applied to multilevel caches (e.g., L2 and L3) to optimize cache hierarchies to balance speed, size, and cost. Another promising area of future work is integrating transfer learning or reinforcement learning techniques to adapt predictive models to new architectures and workloads. Finally, for Pareto-optimal analysis

of a large number of cache configurations, well-known multi-objective evolutionary algorithms such as NSGA-II or advanced non-dominated sorting methods can be adopted.

The method accelerates cache design and serves as a practical tool for early-stage architectural decisions. In high-performance computing, the approach can help to select application-specific caches, thereby improving performance and reducing energy consumption at scale. In addition, for embedded systems, where energy efficiency is a key constraint, the ability to identify optimal cache configurations quickly helps design low-power systems with improved battery life. Overall, this research advances data-driven hardware design methodologies by combining machine learning with architectural simulation. Its ability to accelerate design space exploration while maintaining accuracy makes it highly applicable to current industrial practices.

List of References

[1] D. Cho, “A Technique for Improving the Performance of Cache Memories,” Int. J. Internet, Broadcast. Commun., vol. 13, no. 3, pp. 104–108, 2021, doi: 10.7236/IJIBC.2021.13.3.104.

[2] J. Díaz Álvarez, J. L. Risco-Martín, and J. M. Colmenar, “Multi-objective optimization of energy consumption and execution time in a single level cache memory for embedded systems,” J. Syst. Softw., vol. 111, pp. 200–212, 2016, doi: 10.1016/j.jss.2015.10.012.

[3] D. Efnusheva, A. Cholakoska, and A. Tentov, “A Survey of Different Approaches for Overcoming the Processor-Memory Bottleneck,” Int. J. Comput. Sci. Inf. Technol., vol. 9, no. 2, pp. 151–163, 2017, doi: 10.5121/ijcsit.2017.9214.

[4] A. Biswas, “Energy-Efficient Smart Embedded Memory Design for IoT and AI,” PhD Thesis, Massachusetts Institute of Technology, 2018.

[5] D. Reis et al., “Modeling and Benchmarking Computing-in-Memory for Design Space Exploration,” in Proc. of the ACM on Great Lakes Symposium on VLSI (GLSVLSI ’20). , New York, NY, USA, 2020, pp. 39–44. doi: https://doi.org/10.1145/3386263.3407580.

[6] J. L. Hennessy and D. A. Patterson, “A New Golden Age for Computer Architecture,” Communications of the ACM, vol. 62, no. 2, pp. 48–60, 2019. doi: https://doi.org/10.1145/3282307.

[7] M. V Wilkes, “The memory gap and the future of high performance memories,” ACM SIGARCH Comput. Archit. News, vol. 29, no. 1, pp. 2–7, 2001, doi: 10.1145/373574.373576.

[8] T. Ahmed and E. M. Elamin, “Design Strategy of Cache Memory for Computer Performance Improvement,” Int. J. Res. Stud. Electr. Electron. Eng., vol. 4, no. 3, pp. 12–17, 2018, doi: http://dx.doi.org/10.20431/2454-9436.0403002.

[9] H. V Dave and N. A. Kotak, “Critical analysis of cache memory performance concerning miss rate and power consumption,” Int. J. Embed. Syst., vol. 15, no. 6, pp. 516–524, 2022, doi: 10.1504/IJES.2022.129810.

[10] D. A. Patterson and J. L. Hennessy, Computer Organization and Design : The Hardware / Software Interface, 5th ed. Morgan Kaufmann, Waltham, USA, 2014.

[11] J. Schneider, J. Peddersen, and S. Parameswaran, “A Scorchingly Fast FPGA-Based Precise L1 LRU Cache Simulator,” in 19th Asia and South Pacific Design Automation Conference, Singapore, 2014, pp. 412–417. doi: 10.1109/ASPDAC.2014.6742926.

[12] “Pentium (original).” https://en.wikipedia.org/wiki/Pentium_(original).

[13] “Apple M2.” https://en.wikipedia.org/wiki/Apple_M2.

[14] D. Cho, “Technology of the next generation low power memory system,” Int. J. Internet, Broadcast. Commun., vol. 10, no. 4, pp. 6–11, 2018, doi: http://dx.doi.org/10.7236/IJIBC.2018.10.4.6.

[15] Y. Liang and T. Mitra, “An Analytical Approach for Fast and Accurate Design Space exploration of instruction caches,” ACM Trans. Embed. Comput. Syst., vol. 13, no. 3, p. 29 pages, 2013, doi: http://dx.doi.org/10.1145/2539036.2539039.

[16] R. Vazquez, A. Gordon-Ross, and G. Stitt, “Energy Prediction for Cache Tuning in Embedded Systems,” in IEEE 37th International Conference on Computer Design, Abu Dhabi, United Arab Emirates, 2019, pp. 630–637. doi: 10.1109/ICCD46524.2019.00091.

[17] R. Gonzalez-Alberquilla, F. Castro, L. Pinuel, and F. Tirado, “Stack filter : Reducing L1 data cache power consumption,” J. Syst. Archit., vol. 56, no. 12, pp. 685–695, 2010, doi: 10.1016/j.sysarc.2010.10.002.

[18] C. Zhang, F. Vahid, and W. Najjar, “A Highly Configurable Cache for Low Energy Embedded Systems,” ACM Trans. Embed. Comput. Syst., vol. 4, no. 2, pp. 363–387, 2005, doi: https://doi.org/10.1145/1067915.10679.

[19] I. Nawinne, J. Schneider, H. Javaid, and S. Parameswaran, “Hardware-Based Fast Exploration of Cache Hierarchies in Application Specific MPSoCs,” in Proceedings of the conference on Design, Automation & Test in Europe, Leuven, BEL, 2014, pp. 1–6. doi: https://dl.acm.org/doi/10.5555/2616606.2617020.

[20] G. Jain, Memory Models for Embedded Multicore Architecture, First Edit. Elsevier Inc., 2013. doi: https://doi.org/10.1016/B978-0-12-416018-7.00004-3.

[21] M. Tawada, M. Yanagisawa, T. Ohtsuki, and N. Togawa, “Exact, Fast and Flexible L1 Cache Configuration Simulation for Embedded Systems with FIFO/PLRU cache

replacement policies," in Proceedings of International Symposium on VLSI Design, Automation and Test, Hsinchu, Taiwan, 2011, vol. 4, pp. 1–4. doi: 10.1109/VDAT.2011.5783622.

[22] K. Balasubadra, A. P. Shanthi, and V. P. Srinivasan, "Hybrid Design Space Exploration Methodology for Application Specific System Design," Int. J. New Comput. Archit. their Appl., vol. 7, no. 3, pp. 102–111, 2017, doi: link.gale.com/apps/doc/A528198018/AONE?

[23] M. Kooli, G. Di, and N. Alberto, "Memory-Aware Design Space Exploration for Reliability Evaluation in Computing Systems," J. Electron. Testing, Springer, vol. 35, pp. 145–162, 2019, doi: https://doi.org/10.1007/s10836-019-05785-0.

[24] A. D. Pimentel, "Methodologies for Design Space Exploration," in Handbook of Computer Architecture. Springer, Singapore, A. Chattopadhyay, Ed. 2022, pp. 1–31. doi: https://doi.org/10.1007/978-981-15-6401-7_23-1.

[25] W. Zhang, M. Goli, A. Mahzoon, and R. Drechsler, "ANN-based Performance Estimation of Embedded Software for RISC-V Processors," 2022. doi: 10.1109/RSP57251.2022.10039004.

[26] I. Nawinne and S. Parameswaran, "A Survey on Exact Cache Design Space Exploration Methodologies for Application Specific SoC Memory Hierarchies," in IEEE 8th International Conference on Industrial and Information Systems, Peradeniya, Sri Lanka, 2013, pp. 332–337. doi: 10.1109/ICIInfS.2013.6732005.

[27] A. D. Pimentel, "Exploring Exploration : A Tutorial Introduction to Embedded Systems Design Space Exploration," IEEE Des. Test, vol. 34, no. 1, pp. 77–90, 2017, doi: 10.1109/MDAT.2016.2626445.

[28] S. Sen and N. Imam, "Machine learning based design space exploration for hybrid main-memory design," in Proceedings of the International Symposium on Memory Systems, Washington, DC, USA, 2019, pp. 480–489. doi: 10.1145/3357526.3357544.

[29] C. Tsai and R.-S. Tsay, "A Fast-and-Effective Early-Stage Multi-level Cache Optimization Method Based on Reuse-Distance Analysis," 2021. doi: https://doi.org/10.48550/arXiv.2109.04614.

[30] Q. Guo, T. Chen, Y. Chen, and F. Franchetti, "Accelerating Architectural Simulation Via Statistical Techniques : A Survey," IEEE Trans. Comput. Des. Integr. Circuits Syst., vol. 35, no. 3, pp. 433–446, 2016, doi: 10.1109/TCAD.2015.2481796.

[31] N. Wu and Y. Xie, “A Survey of Machine Learning for Computer Architecture and Systems,” ACM Comput. Surv., vol. 1, no. 1, pp. 1–37, 2021, doi: https://doi.org/10.1145/3494523 ACM.

[32] A. Akram and L. Sawalha, “A Survey of Computer Architecture Simulation Techniques and Tools,” IEEE Access, vol. 7, pp. 78120–78145, 2019, doi: 10.1109/ACCESS.2019.2917698.

[33] D. Steen et al., “Micro-Architecture Independent Analytical Processor Performance and Power Modeling,” in IEEE International Symposium on Performance Analysis of Systems and Software, Philadelphia, PA, USA, 2015, pp. 32–41. doi: 10.1109/ISPASS.2015.7095782.

[34] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Machine learning towards intelligent systems : applications, challenges, and opportunities,” Artif. Intell. Rev., vol. 54, pp. 3299–3348, 2021, doi: https://doi.org/10.1007/s10462-020-09948-w.

[35] I. K. Nti, J. A. Quarcoo, J. Aning, and G. K. Fosu, “A Mini-Review of Machine Learning in Big Data Analytics : Applications, Challenges, and Prospects,” Big Data Min. Anal., vol. 5, no. 2, pp. 81–97, 2022, doi: 10.26599/BDMA.2021.9020028.

[36] R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, “Machine Learning and Manycore Systems Design: A Serendipitous Symbiosis,” Computer, vol. 51, no. 7, pp. 66–77, 2018, doi: 10.1109/MC.2018.3011040.

[37] T. S. Ajani, A. L. Imoize, and A. A. Atayero, “An overview of machine learning within embedded and mobile devices-optimizations and applications,” Sensors, vol. 21, no. 13, pp. 1–44, 2021, doi: https://doi.org/10.3390/ s21134412.

[38] R. Akula, K. Jain, and D. Jigar, “System Performance with varying L1 Instruction and Data Cache Sizes : An Empirical Analysis,” 2019. doi: arXiv:1911.11642v1.

[39] X. Pan and B. Jonsson, “A Modeling Framework for Reuse Distance-based Estimation of Cache Performance,” in IEEE International Symposium on Performance Analysis of Systems and Software, Philadelphia, PA, USA, 2015, pp. 62–71. doi: 10.1109/ISPASS.2015.7095785.

[40] J. M. Sabarimuthu and T. G. Venkatesh, “Analytical Miss Rate Calculation of L2 Cache from the RD Profile of L1 Cache,” IEEE Trans. Comput., vol. 67, no. 1, pp. 9–15, 2017, doi: https://doi.org/10.1109/TC.2017.2723878.

[41] V. C. Chijindu et al., “Modeling cache performance for embedded systems,” Bull. Electr. Eng. Informatics, vol. 10, no. 5, pp. 2910–2920, 2021, doi: 10.11591/eei.v10i5.2459.

[42] M. D. Hill and A. J. Smith, “Evaluating Associativity in CPU Caches,” IEEE Trans. Comput., vol. 38, no. 12, pp. 1612–1630, 1989, doi: 10.1109/12.40842.

[43] A. Janapsatya, A. Ignjatovic, and S. Parameswaran, “Finding Optimal L1 Cache Configuration for Embedded Systems,” in Proceedings of Asia and South Pacific Design Automation Conference, Yokohama, Japan, IEEE Press, 2006, pp. 796–801. doi: https://doi.org/10.1145/1118299.1118482.

[44] N. Tojo, N. Togawa, M. Yanagisawa, and T. Ohtsuki, “Exact and Fast L1 Cache Simulation for Embedded Systems,” in Proceedings of the Asia and South Pacific Design Automation Conference, IEEE Press, Yokohama, Japan, 2009, pp. 817–822.

[45] M. Alipour, K. Moshari, and M. Reza, “Performance per Power Optimum Cache Architecture for Embedded Applications, a Design Space Exploration,” in IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications, Perth, WA, Australia, 2011, pp. 1–6. doi: 10.1109/NESEA.2011.6144938.

[46] M. Alipour, H. Taghdisi, and S. H. Sadeghzadeh, “MultiObjective Design Space Exploration of Cache for Embedded Applications,” in 25th IEEE Canadian Conference on Electrical and Computer Engineering, Montreal, QC, Canada, 2012, pp. 25–28. doi: 10.1109/CCECE.2012.6334940.

[47] N. Saeed and S. Arshad, “Design Space Exploration Using Cycle Accurate Simulator,” Int. J. Sci. Eng. Res., vol. 8, no. 8, pp. 903–906, 2017, doi: 10.14299/ijser.2017.08.006.

[48] R. Patel and A. Rajawat, “Instruction Cache Design Space Exploration for Embedded Software Applications,” in 19th International Symposium on VLSI Design and Test, Ahmedabad, India, 2015, pp. 1–5. doi: 10.1109/ISVDAT.2015.7208127.

[49] G. Yessin, A. A. Badawy, V. Narayana, D. Mayhew, and T. El-ghazawi, “‘ CERE ’: a CachE Recommendation Engine Efficient Evolutionary Cache Hierarchy Design Space,” in IEEE International Conference on High Performance Computing and Communications, 2014, pp. 566–573. doi: 10.1109/HPCC.2014.97.

[50] O. Navarro, T. Leiding, and H. Michael, “Configurable Cache Tuning with a Victim

Cache," in 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip, Bremen, Germany, 2015, pp. 1–6. doi: 10.1109/ReCoSoC.2015.7238080.

[51] S. Gianelli, E. Richter, D. Jimenez, H. Valdez, T. Adegbija, and A. Akoglu, "Application-Specific Autonomic Cache Tuning for General Purpose GPUs," in International Conference on Cloud and Autonomic Computing, Tucson, AZ, USA, 2017, pp. 104–113. doi: 10.1109/ICCAC.2017.17.

[52] N. Frid, V. Sruk, and D. Jakobovic, "Design Space Exploration of Clustered Sparsely Connected MPSoC Platforms," Sensors, vol. 22, no. 7803, pp. 1–27, 2022, doi: https://doi.org/10.3390/ s22207803.

[53] M. Santos, E. Barros, and A. Aziz, "A MPSoC Cache Design Space Exploration Approach Based on ABC Algorithm to Optimize Energy Consumption and Performance," in IEEE 27th International Conference on Application-specific Systems, Architectures and Processors, London, United Kingdom, 2016, pp. 153–158. doi: 10.1109/ASAP.2016.7760785.

[54] D. Sam and M. Agyeman, "An Overview of Design Space Exploration of Cache Memory," in Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control, Stockholm, Sweden, 2018, pp. 1–6. doi: 10.1145/3284557.3284558.

[55] S. M. Hasan, N. Imam, R. Kannan, S. Yoginath, and K. Kurte, "Design Space Exploration of Emerging Memory Technologies for Machine Learning Applications," in IEEE International Parallel and Distributed Processing Symposium Workshops, Portland, OR, USA, 2021, pp. 439–448. doi: 10.1109/IPDPSW52791.2021.00075.

[56] P. Elakkumanan, L. Liu, V. K. Vankadara, and R. Sridhar, "CHIDDAM : A Data Mining based Technique for Cache Hierarchy Determination in Commercial Applications," in 48th Midwest Symposium on Circuits and Systems, Covington, KY, USA, 2005, vol. 2, pp. 1888–1891. doi: 10.1109/MWSCAS.2005.1594493.

[57] S. Khakhaeng and C. Chantrapornchai, "On the finding proper cache prediction model using neural network," in 8th International Conference on Knowledge and Smart Technology, IEEE, Chiang Mai, Thailand, 2016, pp. 146–151. doi: 10.1109/KST.2016.7440514.

[58] O. Navarro, J. Yudi, J. Hoffmann, H. Gerardo, M. Hernandez, and M. Hübner, "A

machine learning methodology for cache memory design based on dynamic instructions," ACM Trans. Embed. Comput. Syst., vol. 19, no. 2, p. 20, 2020, doi: 10.1145/3376920.

[59] V. Thasneema and S. Bhaskaran, "A Classification Model for Predicting Suitable Cache Level in a Multi-core Architecture," in International Conference on Communication, Control and Information Sciences, Idukki, India, 2021, pp. 1–6. doi: 10.1109/ICCISc52257.2021.9484900.

[60] P. J. Joseph and K. Thazhuthaveetil, Matthew, Vaswani, "Construction and Use of Linear Regression Models for Processor Performance Analysis," in Proc. HPCA, Austin, TX, USA, 2006, pp. 99–108. doi: 10.1109/HPCA.2006.1598116.

[61] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "A Predictive Performance Model for Superscalar Processors," in 39th Annual IEEE/ACM International Symposium on Microarchitecture, Orlando, FL, USA, 2006, pp. 161–170. doi: 10.1109/MICRO.2006.6.

[62] A. Shahid, M. Y. Qadri, M. Fleury, H. Waris, A. Ahmad, and N. N. Qadri, "AC-DSE : Approximate Computing for the Design Space Exploration of Reconfigurable MPSoCs," J. Circuits, Syst. Comput., vol. 27, no. 9, pp. 1–25, 2018, doi: 10.1142/S0218126618501451.

[63] E. Ipek, S. A. Mckee, B. Supinski, M. Schulz, and R. Caruana, "Efficiently Exploring Architectural Design Spaces via Predictive Modeling," ACM SIGOPS Oper. Syst. Rev., vol. 40, no. 5, pp. 195–206, 2006, doi: https://doi.org/10.1145/1168917.1168882.

[64] G. Singh et al., "NAPEL: Near-memory computing application performance prediction via ensemble learning," in 56th ACM/IEEE Design Automation Conference, Las Vegas, NV, USA, 2019, pp. 1–6. doi: 10.1145/3316781.3317867.

[65] E. Jelačić, C. Seceleanu, N. Xiong, P. Backeman, S. Yaghoobi, and T. Seceleanu, "Machine learning-based cache miss prediction," Int. J. Softw. Tools Technol. Transf., vol. 27, pp. 53–80, 2025, doi: 10.1007/s10009-025-00800-6.

[66] K. Ji, M. Ling, Y. Zhang, and L. Shi, "An artificial neural network model of LRU-cache misses on out-of-order embedded processors," Microprocess. Microsyst., vol. 50, pp. 66–79, 2017, doi: 10.1016/j.micpro.2017.02.005.

[67] H. V Dave, N. A. Kotak, and J. B. Teraiya, "Hybrid Machine Learning Model for

Cache Performance Prediction and Design Space Exploration," Rev. Inform. Teor. e Apl., vol. 32, no. 3, pp. 11–23, 2025, doi: http://dx.doi.org/10.22456/2175-2745.146689.

[68] N. Binkert et al., "The gem5 Simulator," SIGARCH Comput. Arch. News, vol. 39, no. 2, pp. 1–7, 2011, doi: https://doi.org/10.1145/2024716.2024718.

[69] K. Datta et al., "CASPER : Embedding Power Estimation and Hardware-Controlled Power Management in a Cycle-Accurate Micro-Architecture Simulation Platform for Many-Core Multi-Threading Heterogeneous Processors," J. Low Power Electron. Appl., vol. 2, pp. 30–68, 2012, doi: 10.3390/jlpea2010030.

[70] X. Li, H. Negi, T. Mitra, and A. Roychoudhury, "Design Space Exploration of Caches Using Compressed Traces," in Proceedings of the 18th annual international conference on Supercomputing, New York, NY, USA, 2004, pp. 116–125. doi: 10.1145/1006209.1006227.

[71] T. Austin, E. Larson, and D. Ernst, "SimpleScalar : An Infrastructure for Computer System Modeling," Computer, vol. 35, no. 2, pp. 59–67, 2002, doi: 10.1109/2.982917.

[72] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6 . 0 : A Tool to Model Large Caches," in International Symposium on Microarchitecture, Chicago, 2009, pp. 0–24. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.1426&rep=rep 1 &type=pdf

[73] L. Nai, Y. Xia, I. Tanase, H. Kim, and C. Lin, "GraphBIG : Understanding Graph Computing in the Context of Industrial Solutions," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, USA, 2015, vol. 69, pp. 1–12. doi: 10.1145/2807591.2807626.

[74] M. Á. Abella-González, P. Carollo-fernández, L. Pouchet, F. Rastello, and G. Rodríguez, "PolyBench / Python : benchmarking Python environments with polyhedral optimizations," in Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction, Seoul, South Korea, 2021, vol. 1, no. 1, pp. 59–70. doi: 10.1145/3446804.3446842.

[75] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in Proceedings of

the Fourth Annual IEEE International Workshop on Workload Characterization, Austin, TX, USA, 2001, pp. 3–14. doi: 10.1109/WWC.2001.990739.

[76] S. Thomas et al., “CortexSuite: A synthetic brain benchmark suite,” in International Symposium on Workload Characterization, IEEE, Raleigh, NC, USA, 2014, pp. 76–79. doi: 10.1109/IISWC.2014.6983043.

[77] S. Che et al., “Rodinia: A benchmark suite for heterogeneous computing,” in Proceedings of the IEEE International Symposium on Workload Characterization, IEEE, Austin, TX, USA, 2009, pp. 44–54. doi: 10.1109/IISWC.2009.5306797.

[78] H. V. Dave and N. A. Kotak, “An Analysis of Cache Configuration's Impacts on the Miss Rate of Big Data Applications Using Gem5,” Serbian J. Electr. Eng., vol. 21, no. 2, pp. 217–234, 2024, doi: https://doi.org/10.2298/SJEE2402217D.

[79] U. Otozi, C.-O. Chioma, E. Peter, and M. Raymond, “Virtual and Cache Memory : Implications for Enhanced Performance of the Computer System,” Int. J. Comput. Appl., vol. 181, no. 20, pp. 6–13, 2018, doi: 10.5120/ijca2018917533.

[80] A. O. Abayomi, A. A. Olukayode, and G. O. Olakunle, “An Overview of Cache Memory in Memory Management,” Autom. Control Intell. Syst., vol. 8, no. 3, pp. 24–28, 2020, doi: 10.11648/j.acis.20200803.11.

[81] J. Lowe-Power et al., “The gem5 Simulator: Version 20.0+,” 2020. doi: arXiv:2007.03152v2.

[82] “gem5 simulator.” https://www.gem5.org/

[83] G. Singh et al., “A Review of Near-Memory Computing Architectures : Opportunities and Challenges,” in 21st Euromicro Conference on Digital System Design, Prague, Czech Republic, 2018, pp. 608–617. doi: 10.1109/DSD.2018.00106.

[84] C. K. Luk et al., “Pin: Building customized program analysis tools with dynamic instrumentation,” ACM SIGPLAN Not., vol. 40, no. 6, pp. 190–200, 2005, doi: 10.1145/1064978.1065034.

[85] J. Ilemobayo et al., “Hyperparameter Tuning in Machine Learning : A Comprehensive Review,” J. Eng. Res. Reports, vol. 26, no. 6, pp. 388–395, 2024, doi: https://doi.org/10.9734/jerr/2024/v26i61188.

[86] C. Aliferis and G. Simon, “Overfitting, Underfitting and General Model Overconfidence and Under- - Performance Pitfalls and Best Practices in Machine

Learning and AI," in Artificial Intelligence and Machine Learning in Health Care and Medical Sciences, 2024, pp. 477–524. doi: https://doi.org/10.1007/978-3-031-39355-6_10.

[87] "Understanding MAE, MSE, and RMSE: Key Metrics in Machine Learning." https://medium.com/@mondalsabbha/understanding-mae-mse-and-rmse-key-metrics-in-machine-learning-eeeff8bd1fac

[88] W. Wang, S. Ranka, and P. Mishra, "Energy-aware dynamic reconfiguration algorithms for real-time multitasking systems," Sustain. Comput. Informatics Syst., vol. 1, no. 1, pp. 35–45, 2011, doi: 10.1016/j.suscom.2010.10.006.

[89] M. D. Powell, S. Yang, B. Falsafi, K. Roy, and V. TN, "An Energy-Efficient High-Performance Deep-Submicron Instruction Cache," IEEE TVLSI Spec. issue low-power Des., vol. 9, no. 1, pp. 1–13, 2001.

[90] "Pin 3.31 - A dynamic binary instrumentation." http://software.intel.com/En-Us/Articles/Pintool-Downloads

[91] "Weka- 3.8.6." https://sourceforge.net/projects/weka/files/weka-3-8/3.8.6/

[92] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software : An Update Mark," ACM SIGKDD Explor. Newsl., vol. 11, no. 1, pp. 10–18, 2009, doi: 10.1145/1656274.1656278.

[93] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, "Ensemble learning," in Data Mining, Morgan Kaufmann, Waltham, USA, 2017, pp. 479–501. doi: 10.1016/B978-0-12-804291-5.00012-X.

[94] I. I. Baskin, G. Marcou, D. Horvath, and A. Varnek, "Bagging and Boosting of Regression Models -," in Tutorials in Chemoinformatics, First Edition. A. Varnek, Ed. John Wiley & Sons Ltd, 2017, pp. 249–255.

[95] J. Michanan, R. Dewri, and M. J. Rutherford, "Understanding the power-performance tradeoff through Pareto analysis of live performance data," in International Green Computing Conference, Dallas, Texas, 2014, pp. 1–8. doi: 10.13140/2.1.2475.2322.

[96] K. Deb, "Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction," in Multi-objective Evolutionary Optimisation for Product Design and Manufacturing, Springer- Verlag London, L. Wang, A. Ng, and K. Deb, Eds. 2011, pp. 3–32. doi: 10.1007/978-0-85729-652-8_1.

[97] X. B. Hu, M. Wang, and E. Di Paolo, “Calculating complete and exact pareto front for multiobjective optimization: A new deterministic approach for discrete problems,” IEEE Trans. Cybern., vol. 43, no. 3, pp. 1088–1101, 2013, doi: 10.1109/TSMCB.2012.2223756.

[98] H. Jafaryeganeh, M. Ventura, and C. G. Soares, “Effect of normalization techniques in multi-criteria decision making methods for the design of ship internal layout from a Pareto optimal set,” Struct. Multidiscip. Optim., vol. 62, pp. 1849–1863, 2020, doi: https://doi.org/10.1007/s00158-020-02581-9.

[99] N. Kamble, A. Phatak, A. Joshi, R. Adhao, and V. Pachghare, “Exploring the efficiency : A comprehensive analysis of machine learning algorithms in WEKA,” J. Stat. Manag. Syst., vol. 27, no. 5, pp. 1009–1019, 2024, doi: 10.47974/JSMS-1299.

[100] A. Cutler, D. R. Cutler, and J. R. Stevens, “Random Forests,” in Ensemble Machine Learning: Methods and Applications, Springer, New York, NY, C. Zhang and Y. Ma, Eds. 2012, pp. 157–175. doi: 10.1007/978-1-4419-9326-7 5.

[101] A. F. Jadama and M. K. Toray, “Ensemble Learning : Methods, Techniques, Application,” 2024. doi: 10.13140/RG.2.2.28017.08802.

[102] A. N. Tripathi and A. Rajawat, “An Accurate and Quick ANN based System-Level Dynamic Power Estimation Model using LLVM IR Profiling for FPGA Designs,” IEEE Embed. Syst. Lett., vol. 12, no. 2, pp. 58–61, 2019, doi: 10.1109/LES.2019.2935052.

[103] Sebastian Raschka, “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning,” 2018. doi:arXiv:1811.12808v3.

List of Publications

[1] Hetal Dave, Nirali Kotak, "Critical analysis of cache memory performance concerning miss rate and power consumption", International Journal of Embedded Systems, ISSN: 1741-1076, Pg no. 516–524, Volume 15, Issue 6, March 2022. Doi: 10.1504/IJES.2022.129810. (Indexed by: SCOPUS, Web of Science, EI Compendex, DBLP). (Status: Published).

[2] Hetal Dave, Nirali Kotak, "An Analysis of Cache Configuration's Impacts on the Miss Rate of Big Data Applications Using Gem5", Serbian Journal of Electrical Engineering, ISSN: 2217-7183, Pg. no. 217–234, Volume 21, Issue 2, July 2024. Doi: https://doi.org/10.2298/SJEE2402217D. (Indexed by: SCOPUS). (Status: Published).

[3] Hetal Dave, Nirali Kotak, Jay Teraiya, "Hybrid Machine Learning Model for Cache Performance Prediction and Design Space Exploration", Revista de Informática Teórica e Aplicada – RITA, ISSN: 2175-2745, Pg. no. 11–23, Volume 32, Issue 3, August 2025. Doi: 10.22456/2175-2745.146689. (Indexed by: SCOPUS, DBLP, Google Scholar). (Status: Published).

Appendix – A

Simulation Command Line Interface Setup

Below is an example of a prepared command-line configuration to simulate the cc application's executable for the L1 cache (cache configuration: 1 KB-16B-1-way) in the gem5 simulator.

```
./build/X86/gem5.opt configs/example/se.py  \
-c graphBig/CC/cc \
-o "--dataset /home/administrator/gem5_22.1/benchmark/X86/graphBig/dataset/ccinput" \
--cpu type=TimingSimpleCPU –l1d_size=1 KB –l1i_size=1 KB –l1d_assoc=1 \
--l1i_assoc=1 --cacheline_size=16  --caches
```

Where,

- ./build/X86/gem5.opt : is the gem5 binary compiled for the x86 ISA.
- configs/example/se.py: is the simulation script for system call emulation mode.
- -c graphBig/CC/cc: is used to pass the executable of an application to se.py.
- -o "--dataset /": is used to pass the input file to an application's executable.
- --cpu type=TimingSimpleCPU : defines CPU type.
- --l1d_size and –l1d_assoc : specify the L1-D size and its associativity.
- --l1i_size and –l1i_assoc : specify the L1-I size and its associativity.
- --cacheline_size : defines the L1-D and L1-I line size
- --caches: enable classic memory in the cache model

Appendix – B

Feature and Target Relevance

TABLE B.1 Theoretical relevance between selected features and target

Feature	Relevance to the target	Rationale
cache_size	Affects capacity miss, leakage power, and dynamic-energy-per-access	Fundamental L1-D and L1-I design parameters affect both miss rate and power consumption
line_size	Supports spatial locality for miss rate, and for a lower miss rate, power consumption is reduced	
associativity	Affects the conflict miss and the access-energy	
data_transfer	Related to the conflict/capacity miss rate of L1-D and the L1-I miss rate due to a larger instruction working set. Affects both L1-D and L1-I dynamic power consumption	Application's memory access characteristics and related to L1 cache overall activity
control_flow	Mainly affects the miss rate and power consumption of L1-I and indirectly affects the L1-D performance and power consumption.	Related to program structural behavior
ALU_instruction	Indicates the computational strength, affects the L1-I miss rate, and power	Complementary application's characteristics for the regression model to learn the pattern
input_file_size	Allied with the L1-D miss rate and power of L1-D and L1-I	Indicates the application's data working set size
dynamic_instruction	The power consumption is mainly affected. It also affects the capacity L1-D & L1-I miss rate	Related to cache activity
basic_block	Affects more on the L1-I cache miss rate by affecting the spatial locality. Indirect effect on the L1-D miss rate. Effects on the L1-D and L1-I dynamic power	Indicates control flow-branches, loops, and function calls

Appendix – C

Pareto-Optimal Program and Algorithm

C.1 Python Program for Pareto-Optimal Analysis

==========This program does the following tasks: ==========

1. Accepts 28 cache configurations, predicted/simulated miss rate, and predicted/simulated power consumption values.
2. Compares each cache configuration with every other configuration (Brute-Force Non--Dominated Sorting Algorithm) to find a set of predicted/simulated Pareto-optimal cache configurations.
3. Calculates the min-max score for the obtained Pareto-optimal cache configurations using the weighted-sum method.
4. As an output, this program generates predicted/simulated Pareto-optimal cache configurations from 28 configurations with their min-max score, and generates the Pareto-front graph.

==========

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

==== **Step 1:** Enter 28 cache configurations in the form of ('cache_size', 'line_size', 'associativity') and create a data frame - using Pandas Library, organize input data into proper rows and columns for the analysis purpose ==========

```
cache_configs = [[config 1], [config2], [config3], [config4], ……. [config27], [config28],]
#predicted or simulated miss rates and power consumption for each configuration of an application
predicted_miss_rates = [M1, M2, M3, M4, …… M27, M28,]
```

```
predicted_powers      =  [P1, P2, P3, P4,  …..…… P27, P28,]
assert len(cache_configs) == 28
assert len(predicted_miss_rates) == 28
assert len(predicted_powers) == 28
config_names = [f'config{i+1}' for i in range(28)]
df = pd.DataFrame(cache_configs, columns=['cache_size', 'line_size', 'associativity'])
df['config_name'] = config_names
df['predicted_miss_rate'] = predicted_miss_rates
df['predicted_power'] = predicted_powers

# === Step 2: Find a set of predicted/simulated Pareto-optimal cache configurations – Brute Force algorithm
==========
def find_pareto(configs):
    pareto_caches = []
    for i, config in enumerate(configs):
        dominated = False
        for j, other_config in enumerate(configs):
            if all(other_config <= config) and any(other_config < config):
                dominated = True
                break
        if not dominated:
            pareto_caches.append(i)
    return pareto_caches
configs = df[['predicted_miss_rate', 'predicted_power']].to_numpy()
pareto_caches = find_pareto(configs)
pareto_df = df.iloc[pareto_caches].copy()
pareto_df = pareto_df.sort_values(by='predicted_miss_rate')

# ========== Step 3: Calculates the min-max score for obtained Pareto-optimal cache configurations using the weighted-sum method.==========
```

```
pareto_df['norm_miss_rate'] =
(pareto_df['predicted_miss_rate'] - pareto_df['predicted_miss_rate'].min()) / \
(pareto_df['predicted_miss_rate'].max() - pareto_df['predicted_miss_rate'].min())
pareto_df['norm_power'] =
(pareto_df['predicted_power'] - pareto_df['predicted_power'].min()) / \
 (pareto_df['predicted_power'].max() - pareto_df['predicted_power'].min())
# Compute weighted score
# a1 and a2: weights for miss rate and power consumption, respectively
a1 = 0.5   a2 = 0.5
pareto_df['score'] = a1 * pareto_df['norm_miss_rate'] + a2 * pareto_df['norm_power']

# ========== Step 4: Predicted/simulated Pareto-front graph generation and save Pareto-optimal cache configurations with their min-max calculation as an output ==========
plt.figure(figsize=(8, 6))
plt.scatter(df['predicted_miss_rate'], df['predicted_power'], alpha=0.3, label='All Configs')
plt.plot(pareto_df['predicted_miss_rate'], pareto_df['predicted_power'], color='red', marker='o', label='Pareto Front')
plt.xlabel('Predicted Miss Rate')
plt.ylabel('Predicted Power (W)')
plt.title('Pareto Front of Cache Configurations')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
pareto_df.to_csv("predicted_pareto_optimal_cache_configs.csv", index=False)
```

C.2 Brute Force Non-Dominated Sorting Algorithm

In the above program, in step 2, the Pareto-dominance is checked using the Brute Force Non-Dominated Sorting algorithm. The procedure is explained below.

Let C denote the cache configuration, M the miss rate, and P the power consumption.

1. Initialize an empty set P (Pareto-optimal configuration).
2. For each configuration, *Ci* is compared with *Cj* (i = 1 to 28, and i is not equal to j).

3. If any configuration exists *Cj* such that M (*Cj*) <= M (*Ci*) and P (*Cj*) <= P (*Ci*), then *Ci* is dominated and discarded from the Pareto-optimal set P.
4. If no such *Cj* is found, then *Ci* is non-dominated and is added to the Pareto-optimal set P.
5. Final Pareto-optimal configuration set P consists of all non-dominated configurations.

To understand the Pareto-dominance condition, Table C.1 shows the Pareto-dominance status, indicating which configurations are dominated by others after applying the Brute-Force algorithm for the cc application based on the predicted values. (Found Pareto-optimal cache configurations for the cc application are shown in Table 6.7, Chapter 6).

TABLE C.1 Pareto-dominance status for the cc application from prediction

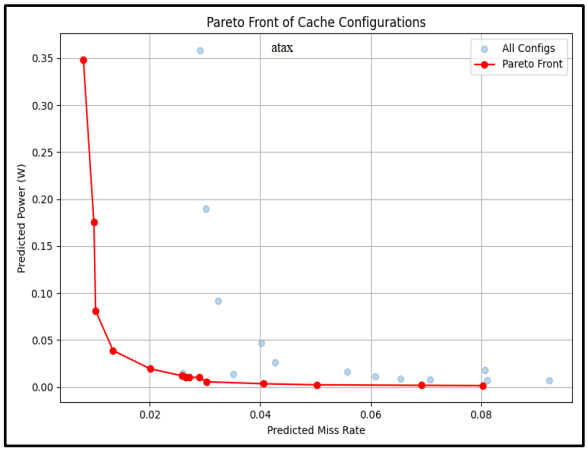
Cache config.	Miss rate	Power	Status	Cache config.	Miss rate	Power	Status
Config 1	0.033	0.010	Dominated by config2	Config 15	0.046	0.046	Dominated by config25
Config 2	0.027	0.010	Optimal	Config 16	0.043	0.091	Dominated by config26
Config 3	0.026	0.011	Optimal	Config 17	0.039	0.188	Dominated by config27
Config 4	0.026	0.012	Dominated by config3	Config 18	0.035	0.358	Dominated by config28
Config 5	0.026	0.014	Dominated by config3	Config 19	0.244	0.001	Optimal
Config 6	0.075	0.017	Dominated by config3	Config 20	0.171	0.002	Dominated by config21
Config 7	0.055	0.016	Dominated by config3	Config 21	0.119	0.002	Optimal
Config 8	0.040	0.013	Dominated by config3	Config 22	0.091	0.003	Optimal
Config 9	0.226	0.007	Dominated by config10	Config 23	0.043	0.006	Optimal
Config 10	0.160	0.007	Dominated by config11	Config 24	0.027	0.019	Dominated by config3
Config 11	0.122	0.007	Dominated by config21	Config 25	0.023	0.038	Optimal
Config 12	0.099	0.009	Dominated by config22	Config 26	0.020	0.080	Optimal
Config 13	0.062	0.011	Dominated by config23	Config 27	0.017	0.174	Optimal
Config 14	0.050	0.025	Dominated by config24	Config 28	0.015	0.347	Optimal

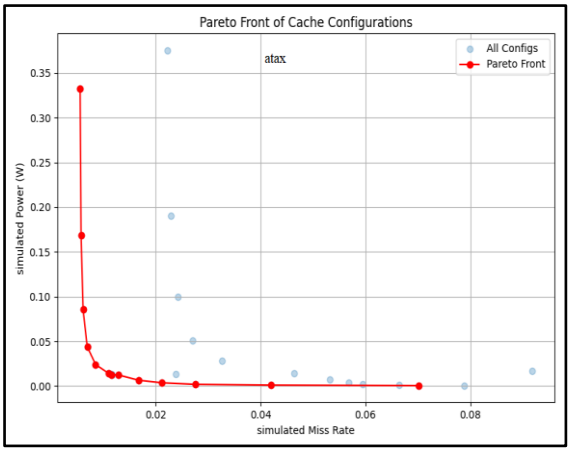
Appendix – D

Pareto Front Graph

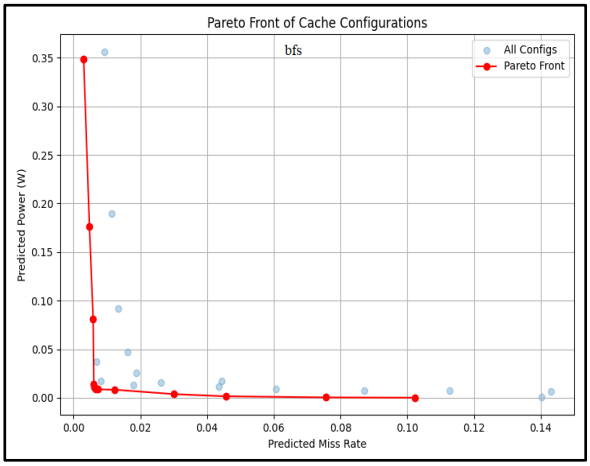
D.1 Pareto Front Graphs for L1-D

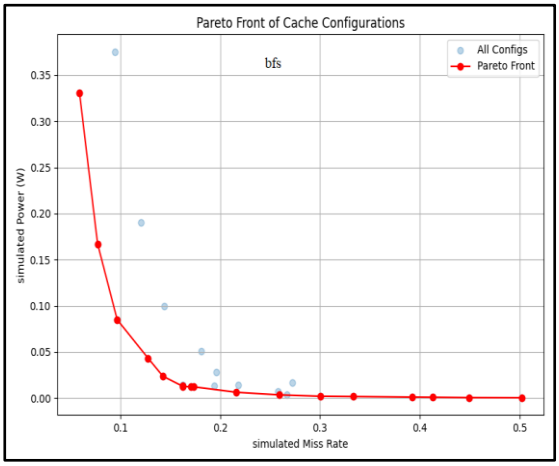
The predicted and simulated Pareto front graphs for the remaining applications of L1-D are shown in Figure D.1.



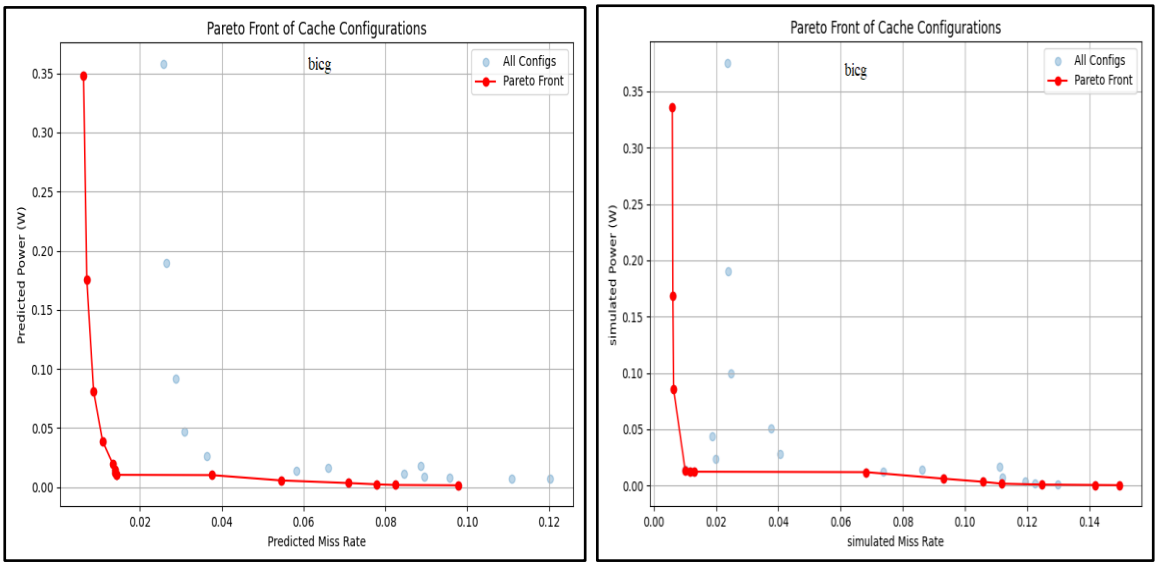


a) atax

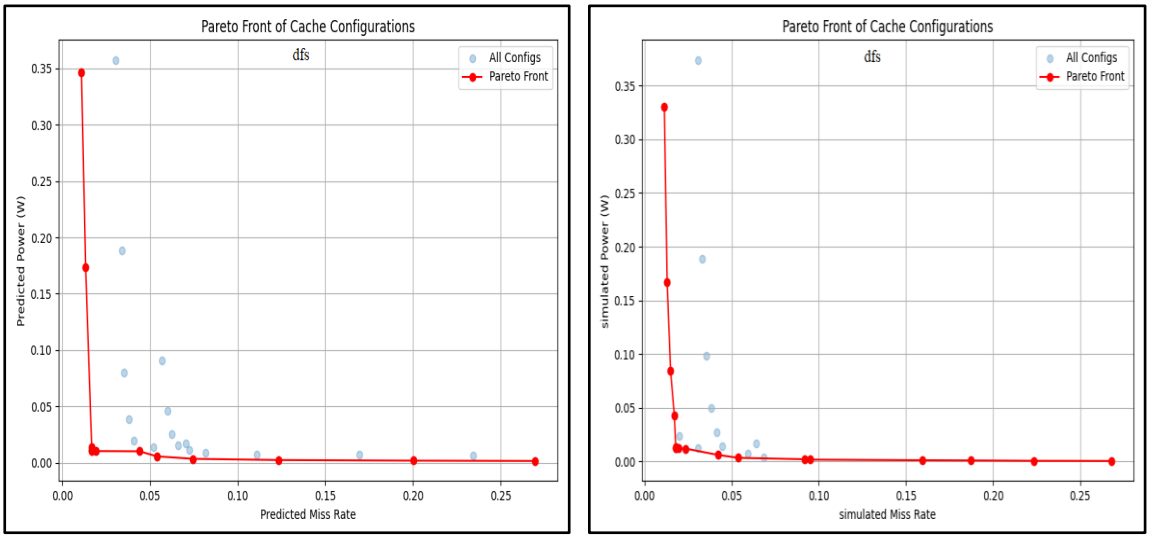




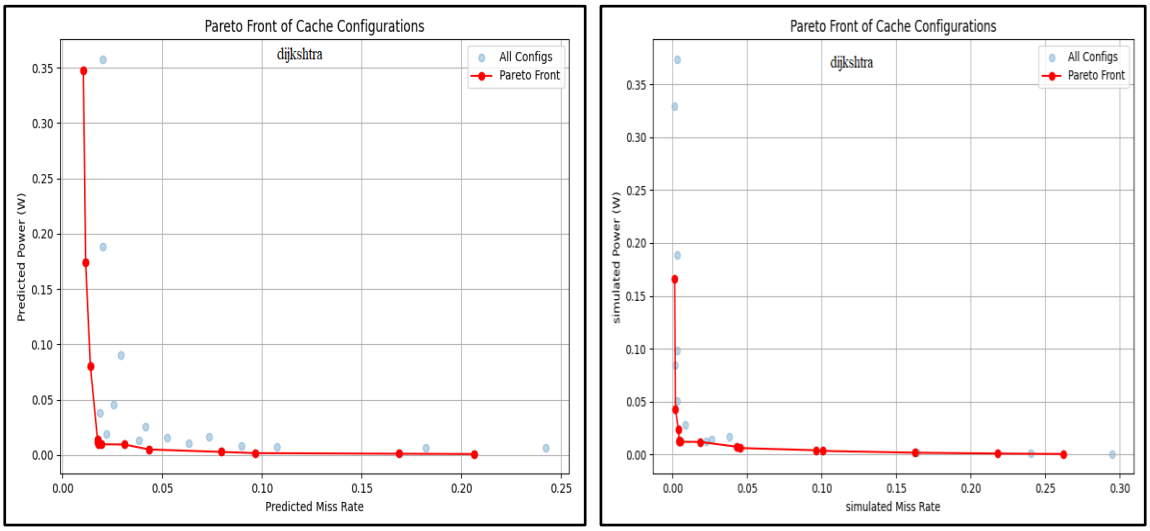
(b) bfs



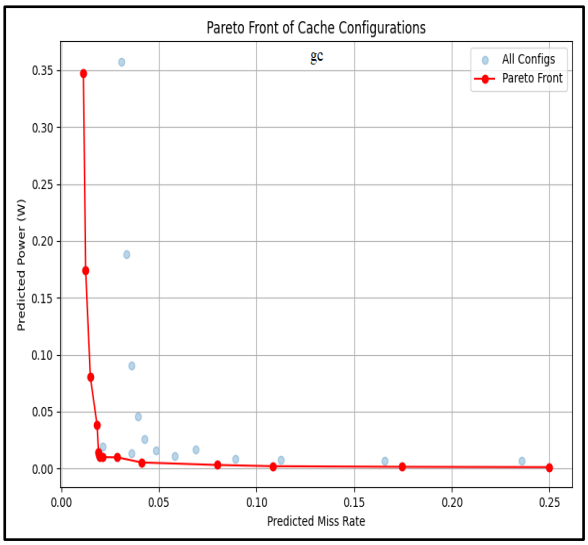
(c) bicg

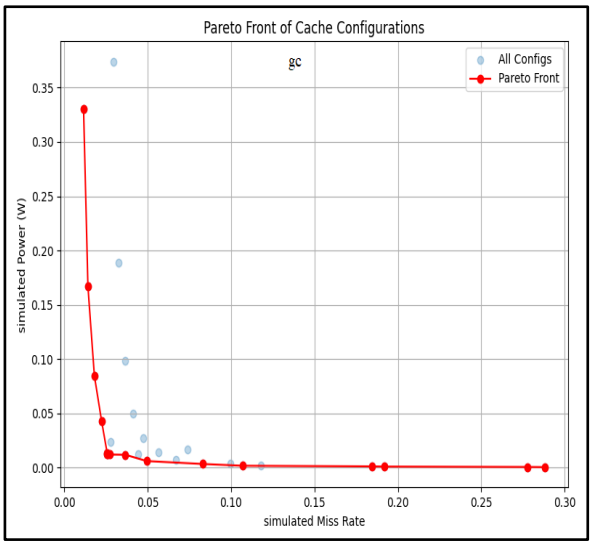


(d) dfs

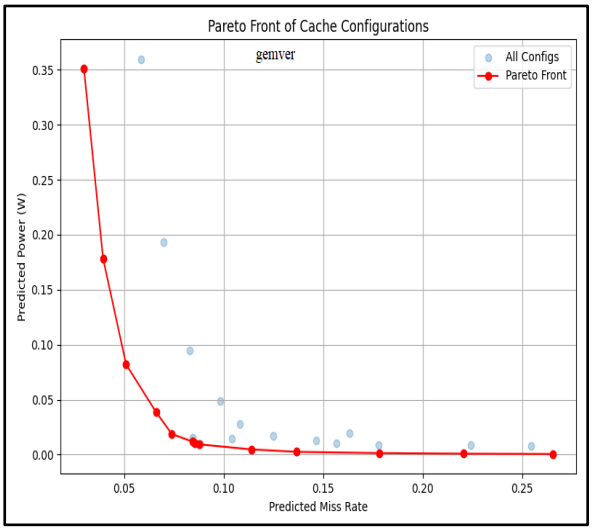


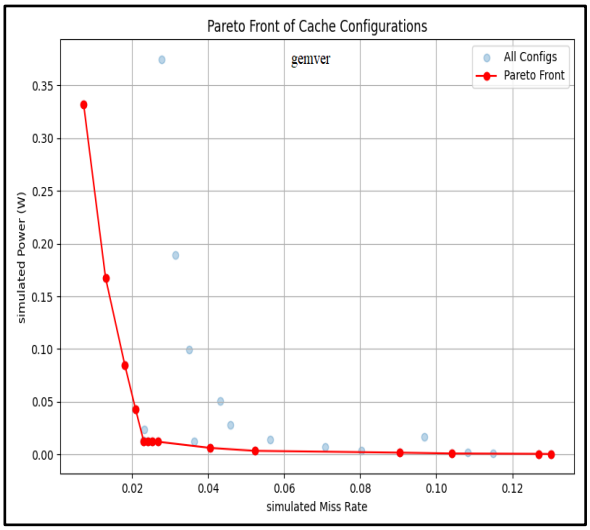
(e) dijkshtra



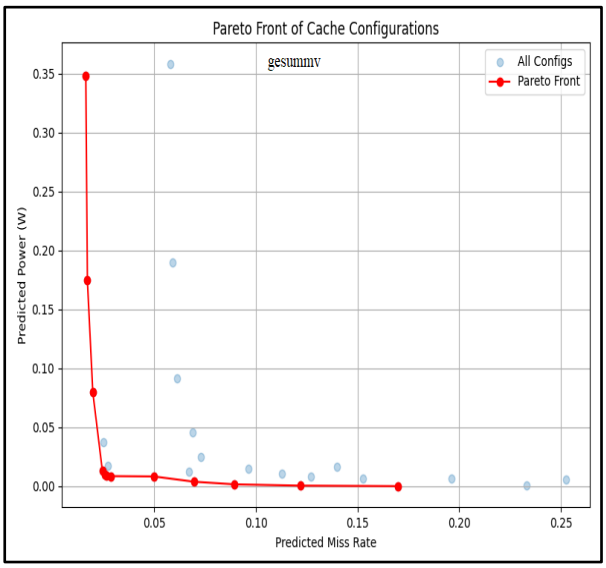


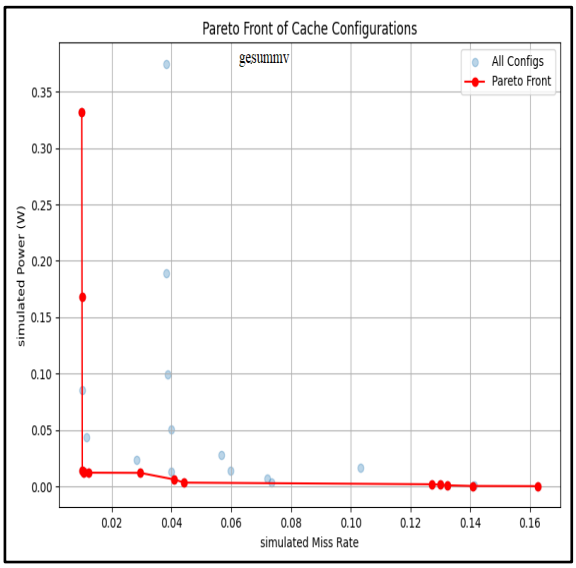
(f) gc



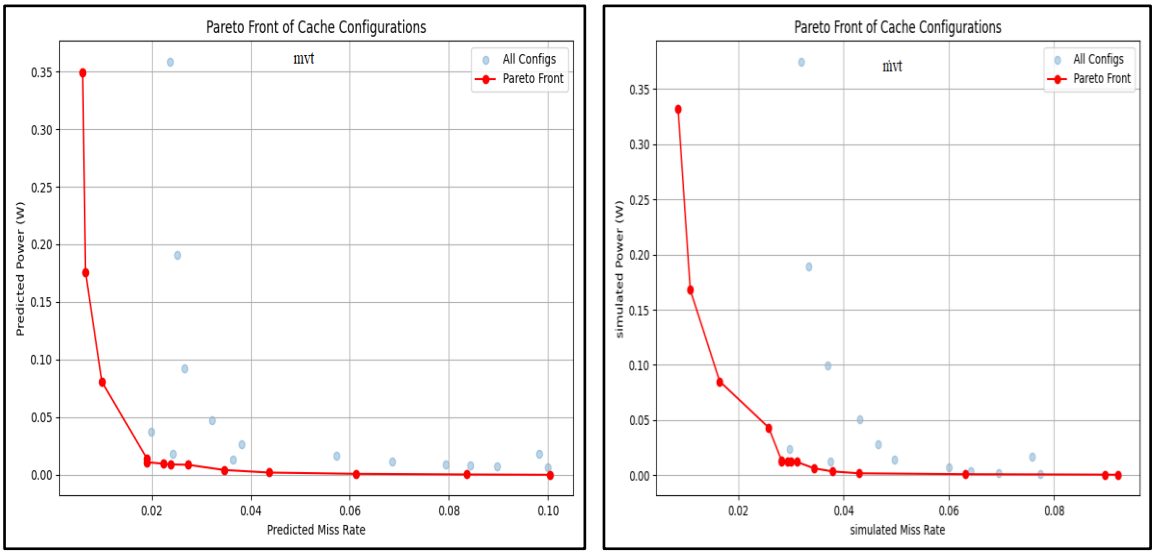


(g) gemver

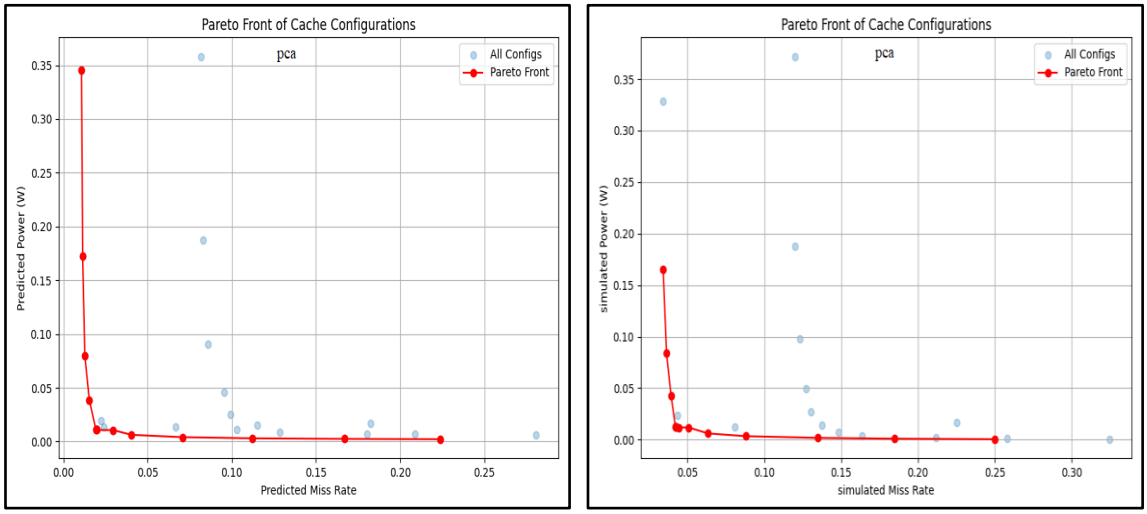




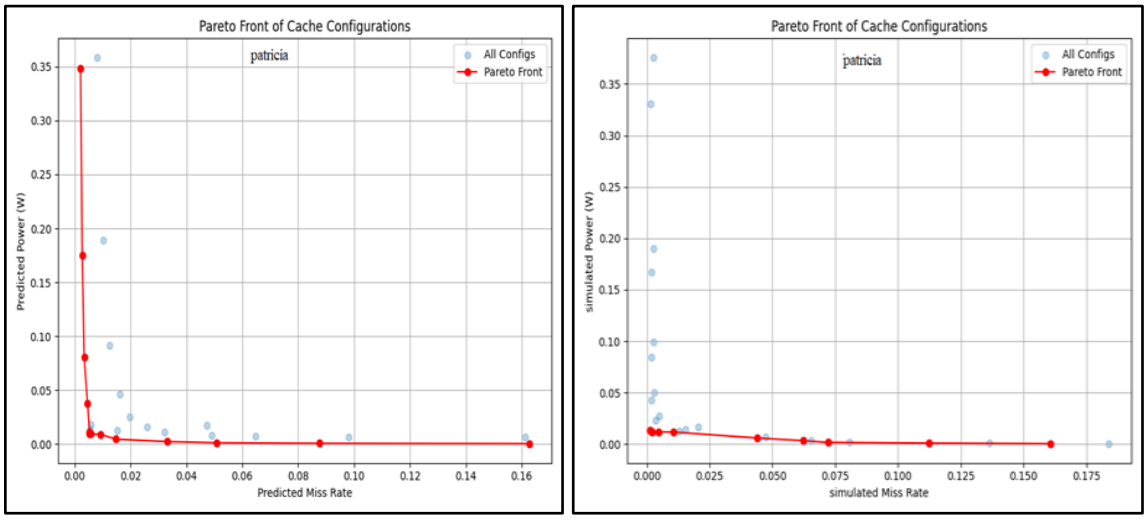
(h) gesummv



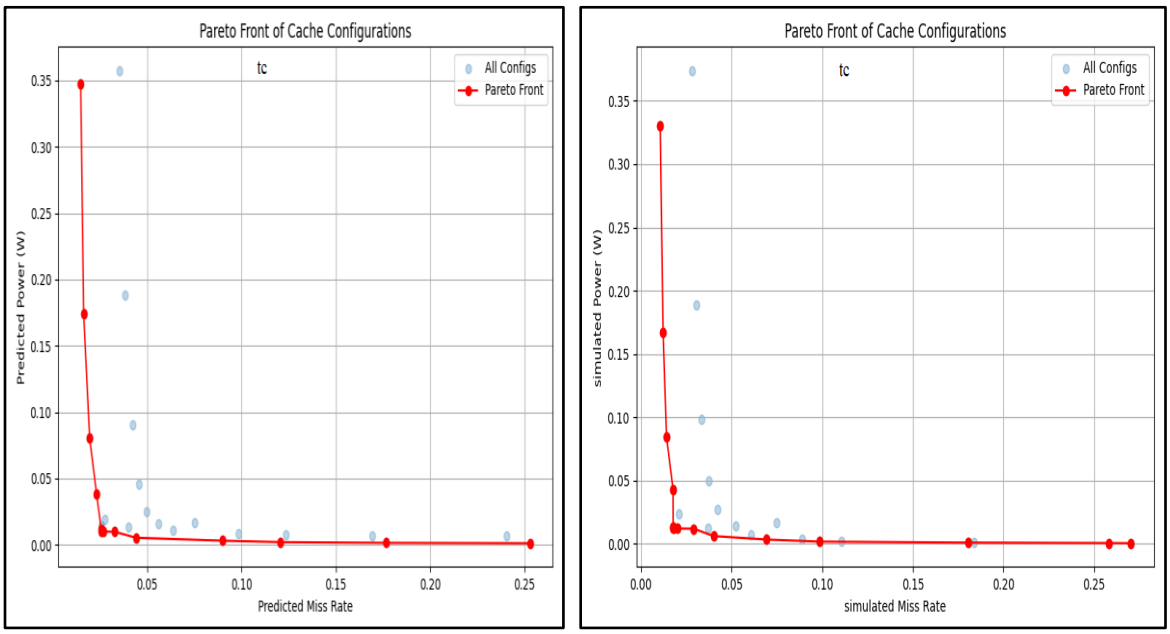
(i) mvt



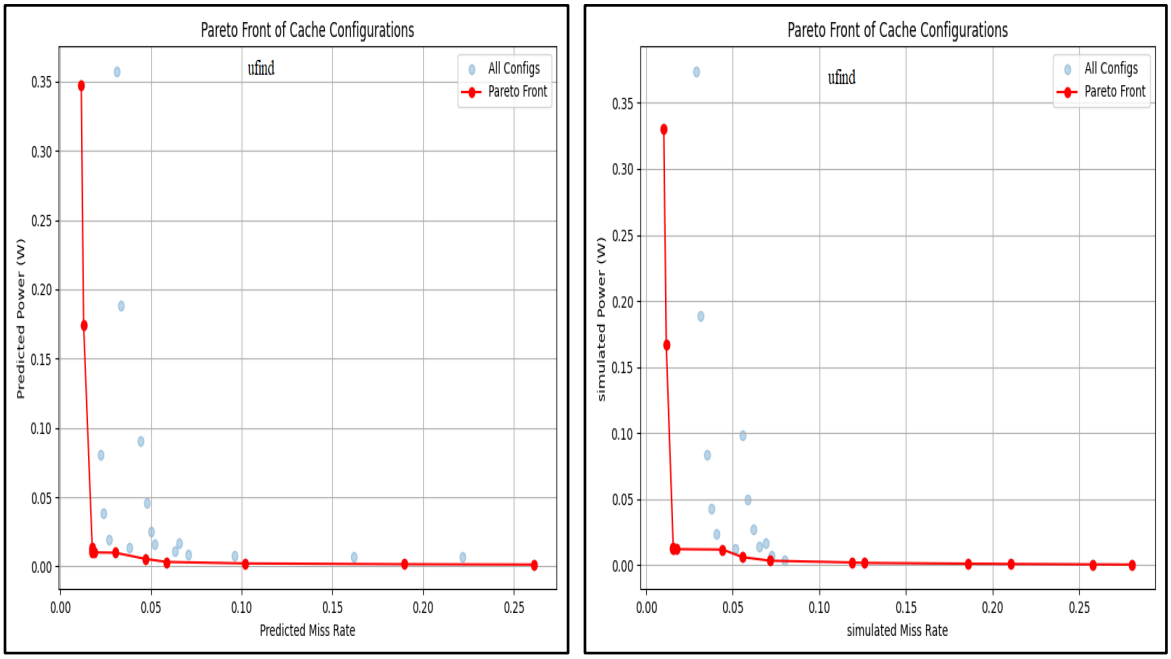
(j) pca



k) patricia



(l) tc

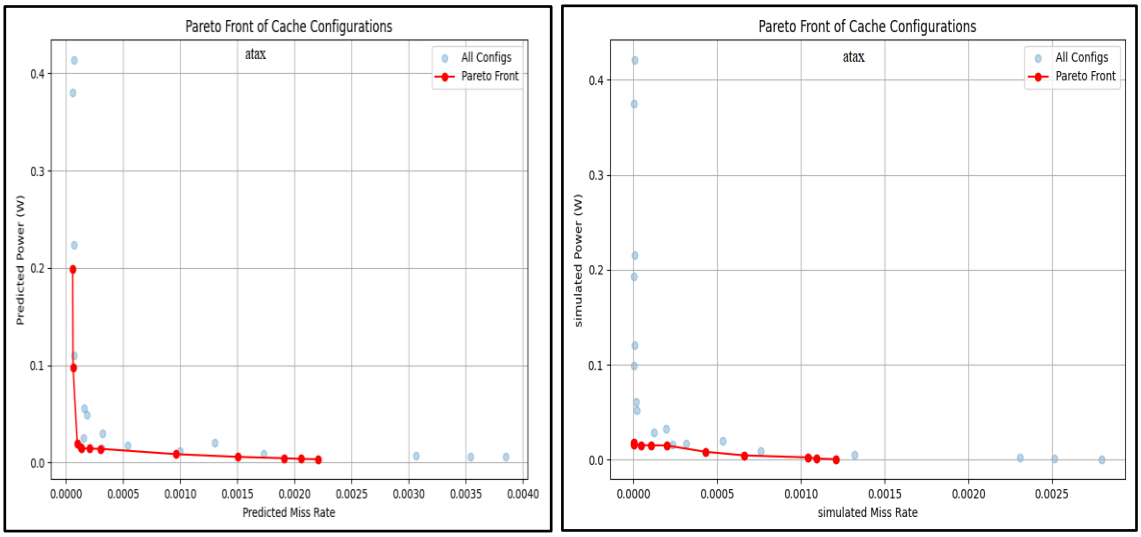


(m) ufind

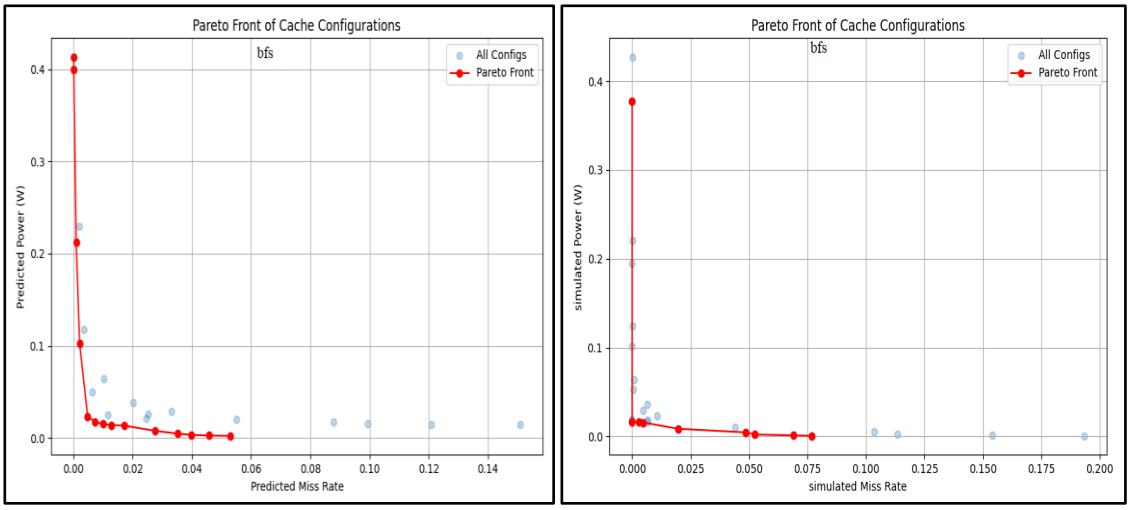
FIGURE D.1 Predicted and simulated Pareto front graphs for L1-D

D.2 Pareto Front Graphs for L1-I

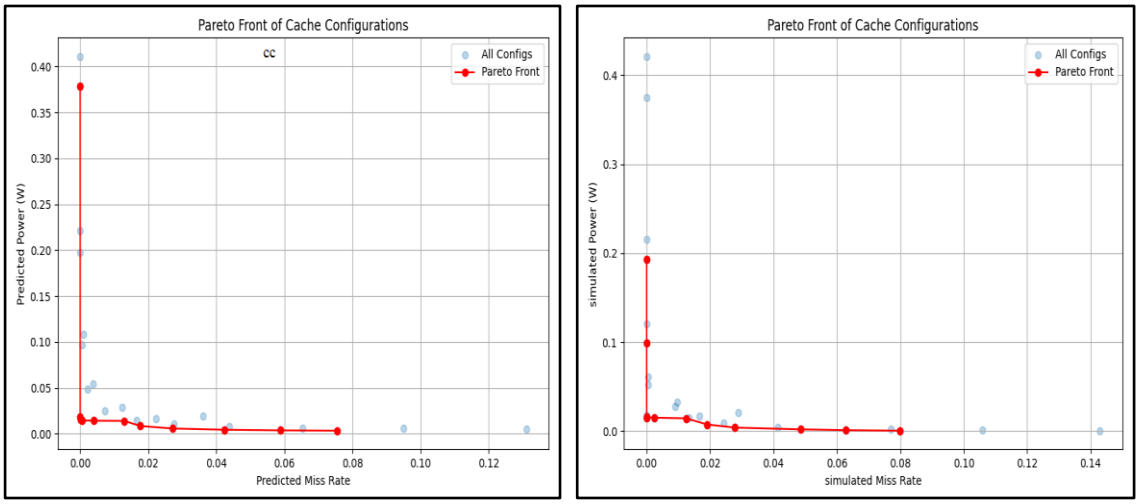
The predicted and simulated Pareto front graphs for the remaining applications of L1-I are shown in Figure D.2.



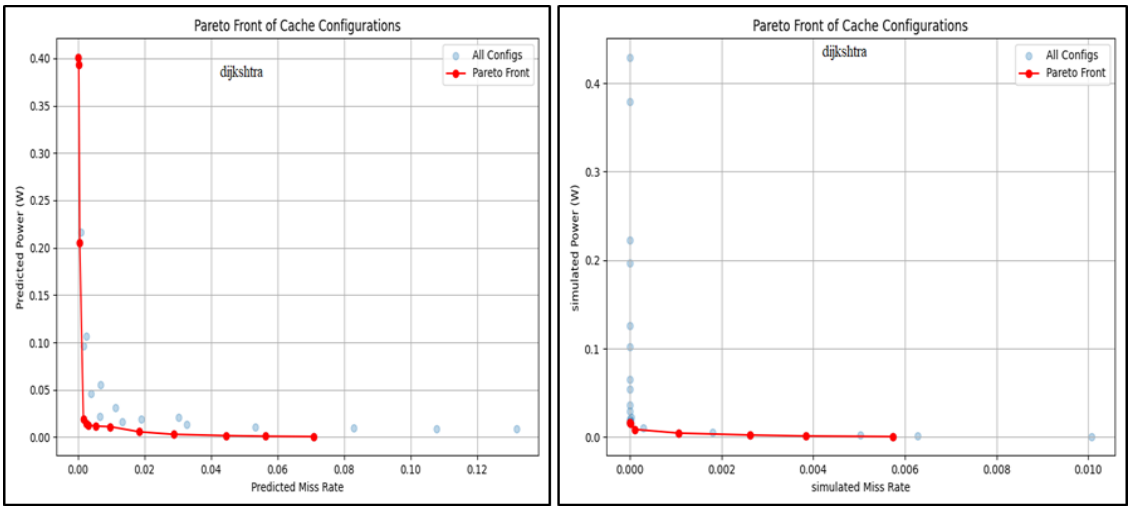
(a) atax



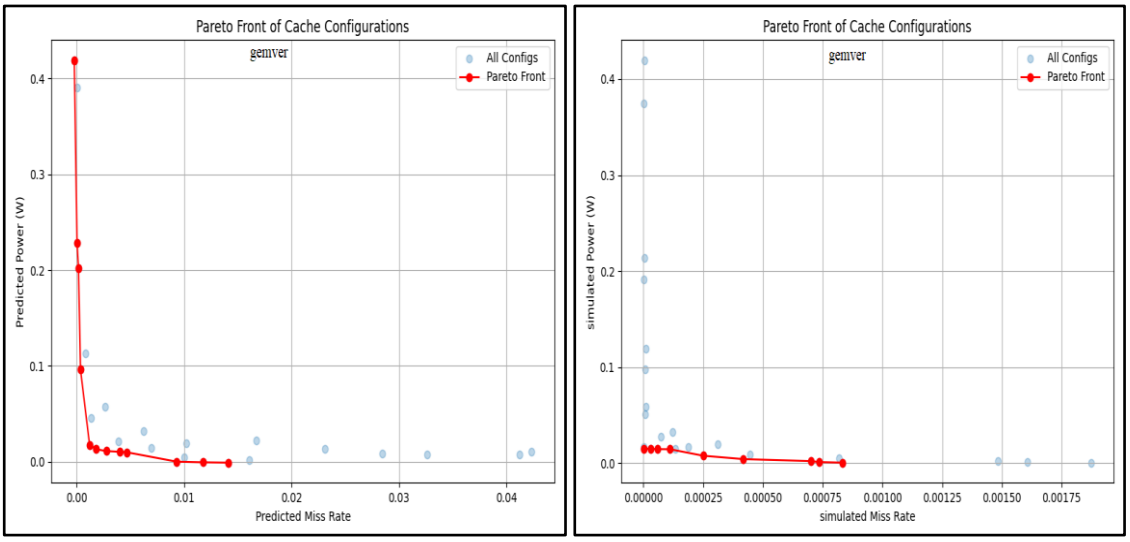
(b) bfs



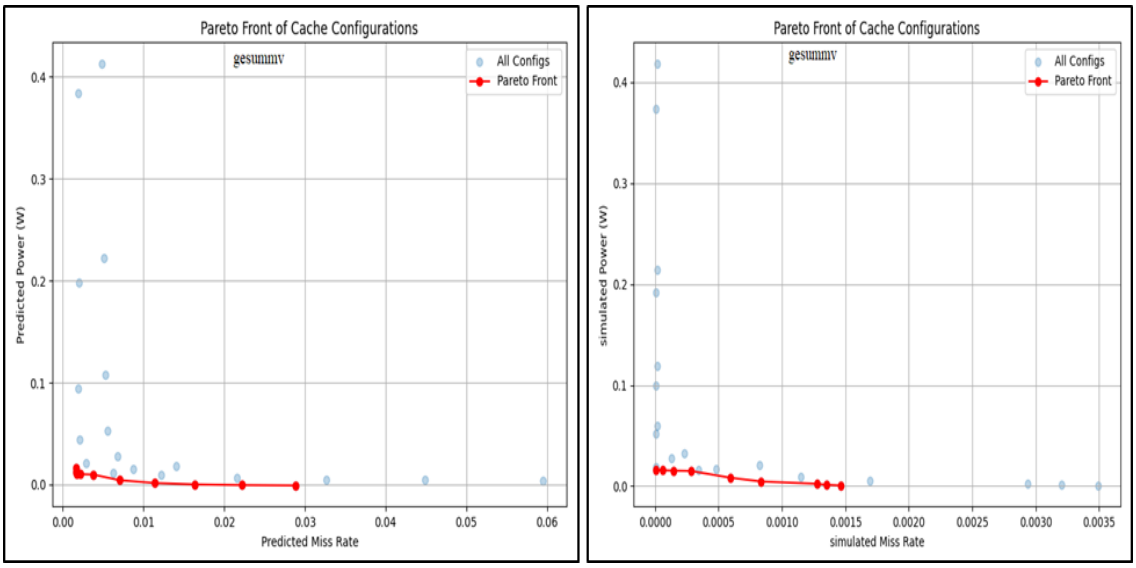
(c) cc



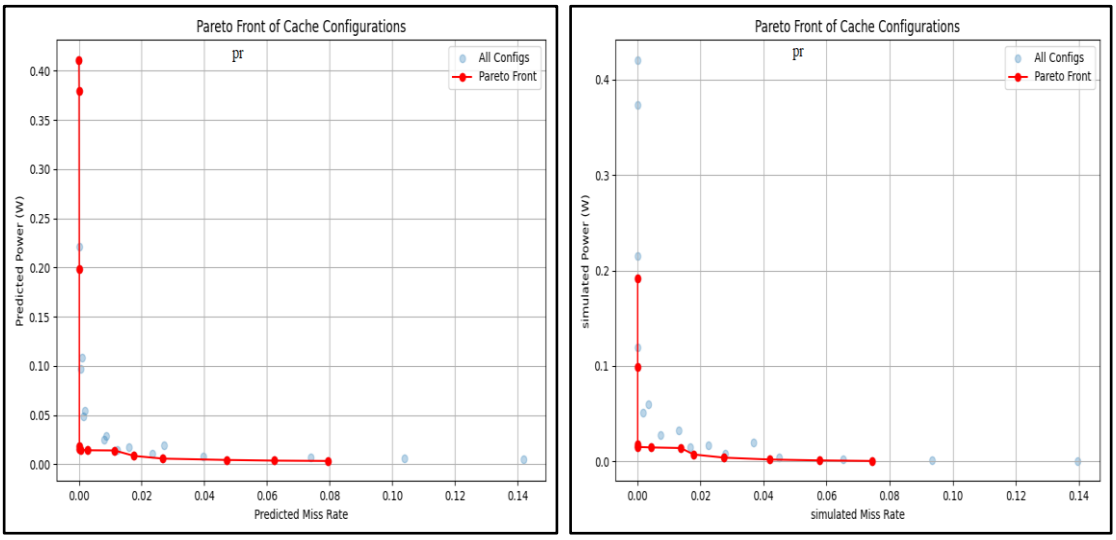
(d) dijkshtra



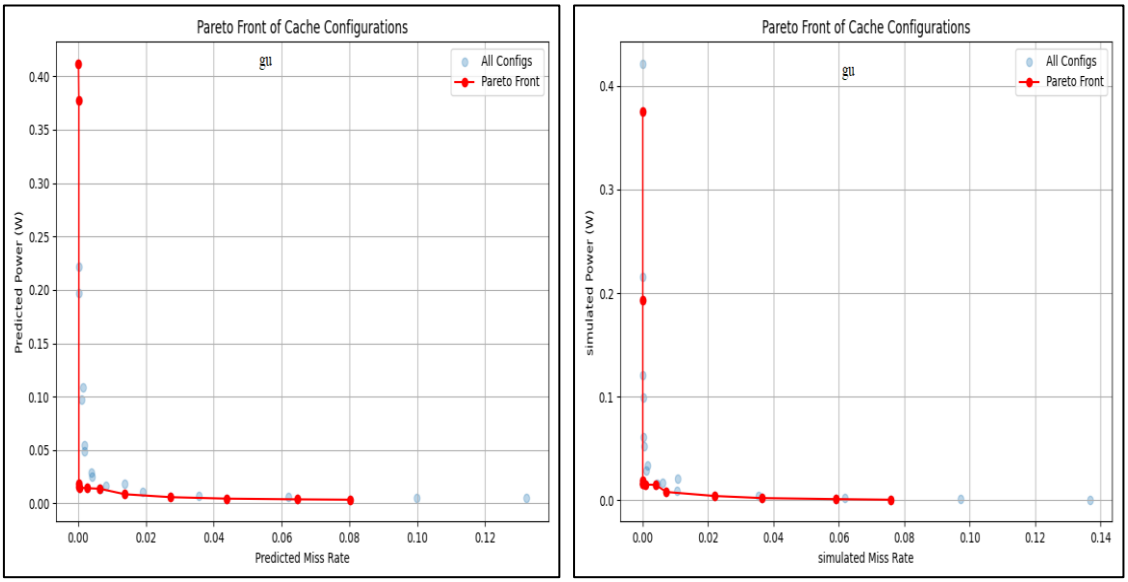
(e) gemver



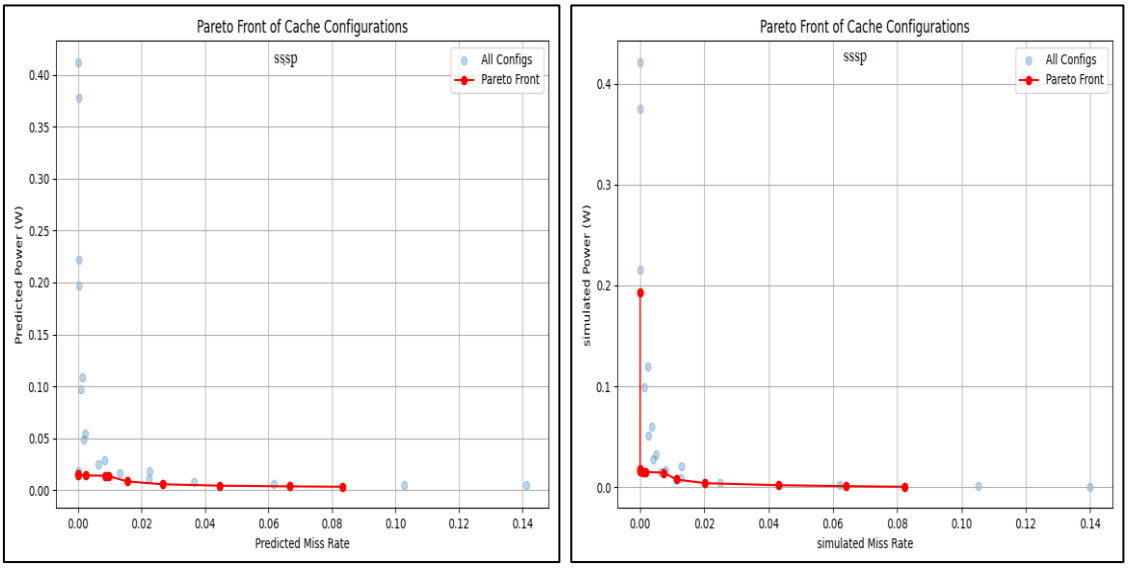
(f) gesummv



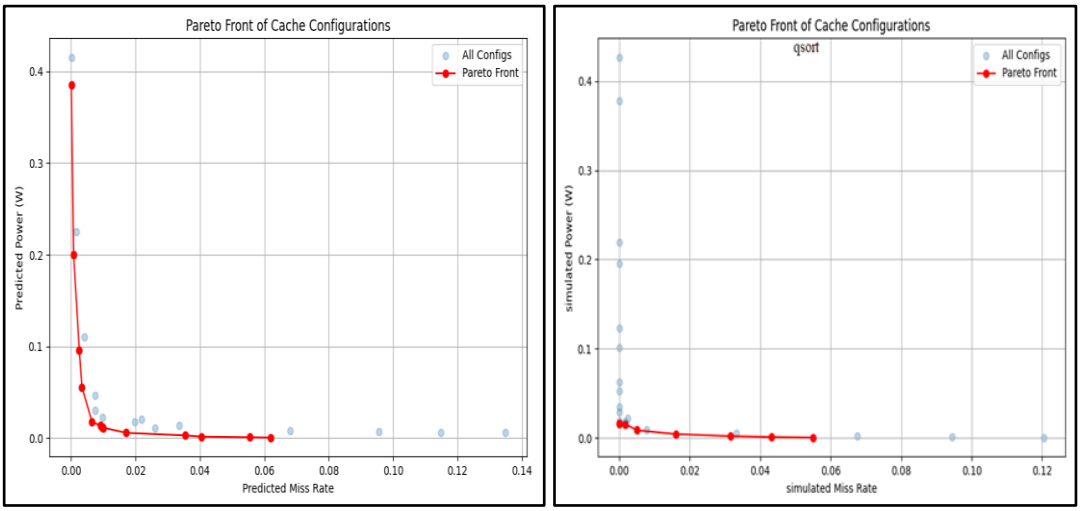
(g) pr



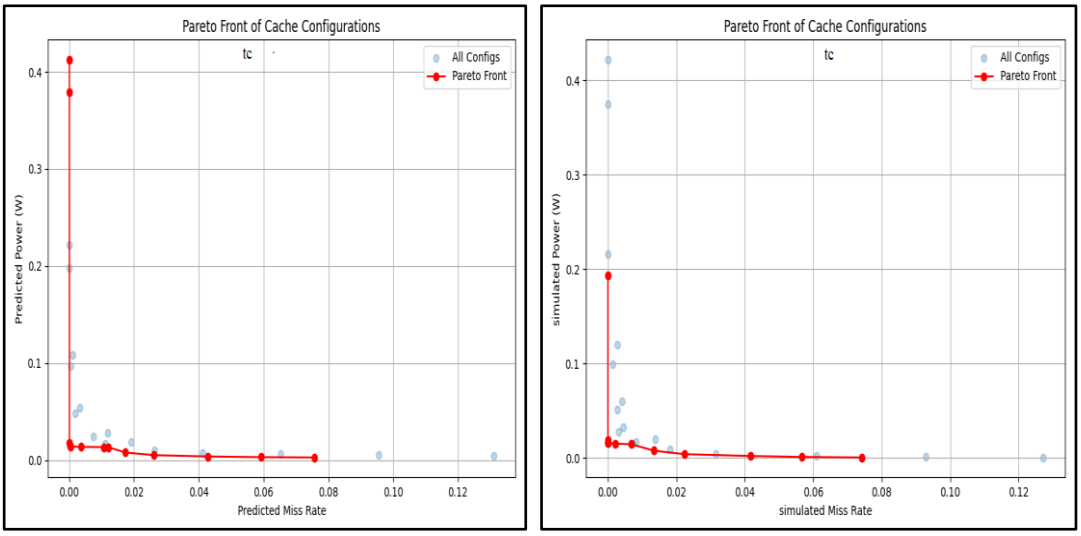
(h) gu



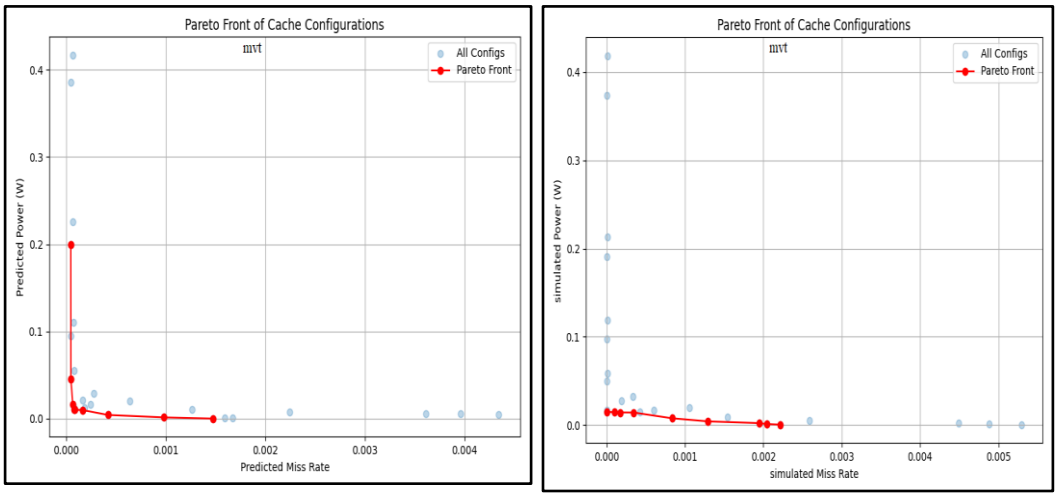
(i) sssp



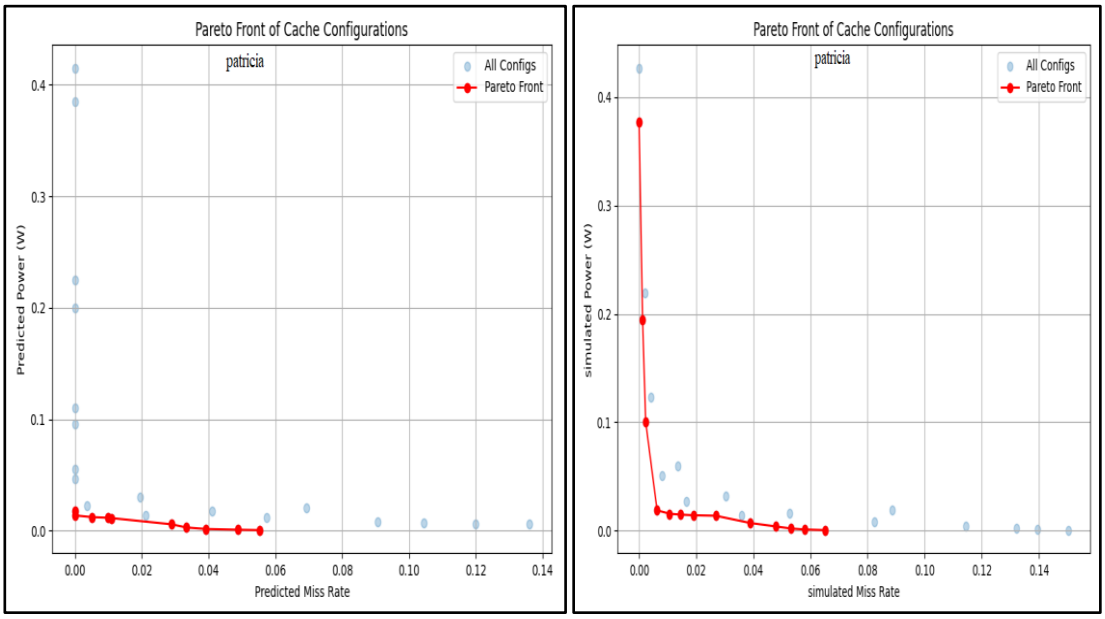
(j) qsort



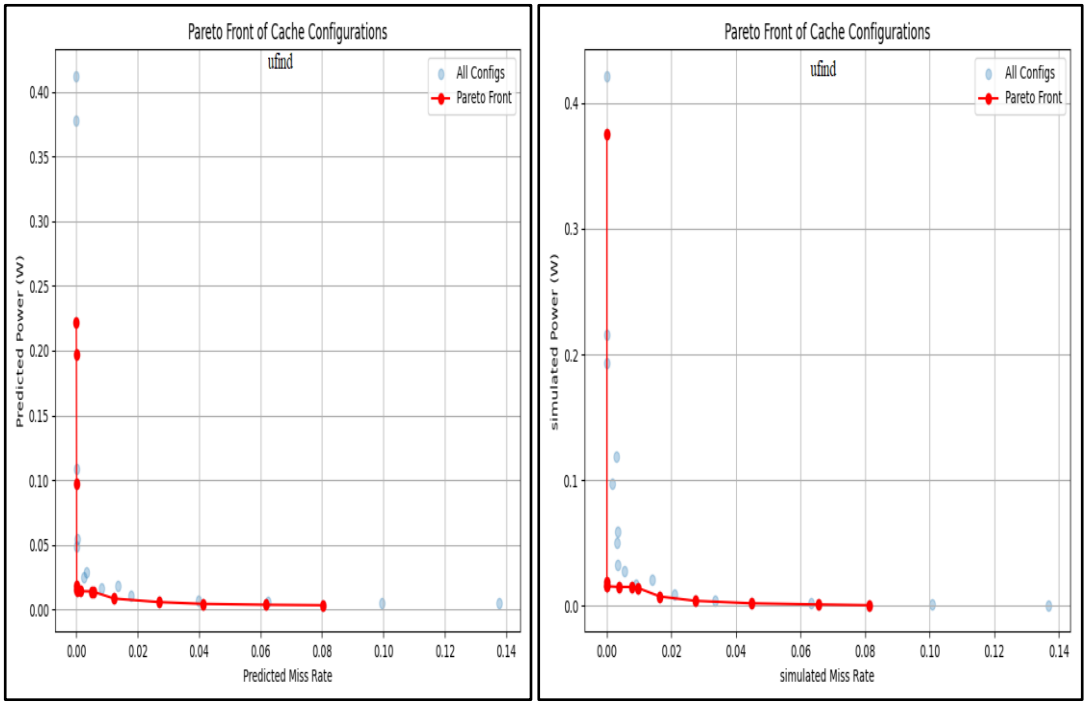
(k) tc



(l) mvt



(m) patricia



(n) ufind

FIGURE D.2 Predicted and simulated Pareto front graphs for L1-I