



NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF COMPUTER SCIENCE

Cache miss estimator

Predicting cache misses of different cache architectures

DIPLOMA THESIS

of

MARKOS-GEORGIOS RAMOS



Supervisor: Nectarios Koziris
Professor N.T.U.A.

Athens, April 2024



NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF COMPUTER SCIENCE

Cache miss estimator

Predicting cache misses of different cache architectures

DIPLOMA THESIS

of

MARKOS-GEORGIOS RAMOS

Supervisor: Nectarios Koziris
Professor N.T.U.A.

Approved by the examination committee on 5 April 2024.

(Signature)

(Signature)

(Signature)

.....
Nectarios Koziris
Professor N.T.U.A.

.....
Dionisios N. Pnevmatikatos
Professor N.T.U.A.

.....
Georgios I. Goumas
Associate Professor N.T.U.A.

Athens, April 2024



Copyright © - All rights reserved.

Markos-Georgios Ramos, 2024.

The copying, storage and distribution of this diploma thesis, exall or part of it, is prohibited for commercial purposes. Reprinting, storage and distribution for non - profit, educational or of a research nature is allowed, provided that the source is indicated and that this message is retained.

The content of this thesis does not necessarily reflect the views of the Department, the Supervisor, or the committee that approved it.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

(Signature)

.....

Markos-Georgios Ramos

Graduate Electrical and Computer Engineer at N.T.U.A.

5 April 2024

Περίληψη

Δεδομένης της αυξανόμενης διαφοράς στην ταχύτητα επεξεργασίας μεταξύ της κύριας μνήμης και της ΚΜΕ, ο ρόλος των κρυφών μνημών της ΚΜΕ στην επίτευξη της μέγιστης δυνατής ταχύτητας επεξεργασίας είναι τόσο σημαντικός όσο ποτέ άλλοτε. Κάθε πρόσβαση στην κρυφή μνήμη που δεν βρίσκει τα δεδομένα πρέπει να καλεί την κύρια μνήμη, χρησιμοποιώντας δεκάδες, αν όχι εκατοντάδες, κύκλους της μηχανής. Η πρόβλεψη της συμπεριφοράς μιας διεργασίας πριν από την εκτέλεσή της σε διαφορετικές αρχιτεκτονικές κρυφής μνήμης είναι ένα σημαντικό έργο. Θα βοηθήσει να καθοριστεί ποιος επεξεργαστής θα λειτουργήσει καλύτερα για ένα πρόγραμμα ή πώς να κατανεμηθεί βέλτιστα η υπολογιστική ισχύς. Ο άμεσος στόχος αυτής της διατριβής είναι να προτείνει ένα μοντέλο μηχανικής μάθησης που θα προβλέπει με όσο το δυνατόν μεγαλύτερη ακρίβεια τις αστοχίες στην κρυφή μνήμη ενός προγράμματος, ανάλογα αποκλειστικά με την αρχιτεκτονική της κρυφής μνήμης, τον κώδικα του προγράμματος και τα ιστογράμματα των αποστάσεων επαναχρησιμοποίησης. Για την επίτευξη αυτού του στόχου, προτείνουμε έναν αριθμό μοντέλων μετασχηματισμών που συνδυάζουν τον μετασχηματισμένο κώδικα εισόδου με πληροφορίες μέσα στα ιστογράμματα αποστάσεων επαναχρησιμοποίησης και εκτιμούν το ποσοστό αστοχιών κρυφής μνήμης. Εκπαιδεύσαμε και δοκιμάσαμε αυτά τα μοντέλα με προσομοιωμένα δεδομένα και παρήγαμε προβλέψεις υψηλής ακρίβειας για μεγάλο αριθμό πολιτικών αντικατάστασης και μεγεθών κρυφής μνήμης LLC. Αυτά τα δίκτυα μπορούν να λειτουργήσουν ως ένα χρήσιμο εργαλείο στην πρόβλεψη της κρυφής μνήμης και μπορούν να χρησιμοποιηθούν σε πραγματικές μηχανές σε μελλοντική έρευνα.

Λέξεις κλειδιά: Αρχιτεκτονική υπολογιστών, Αστοχίες κρυφής μνήμης, Μηχανική μάθηση, Νευρωνικά δίκτυα

Abstract

Given the increasing difference in processing speed between the main memory and the CPU, the role of CPU caches in achieving the maximum possible processing speed is as big as ever. Every cache access that doesn't find the data has to invoke the main memory, using tens, if not hundreds, of machine cycles. Predicting the behavior of a process prior to execution on different cache architectures is an important task. It will help determine what processor will work best for a program or how to distribute computing power optimally. The immediate goal of this thesis is to propose a machine learning model that will predict as accurately as possible the cache misses of a program depending solely on the cache architecture, the program code, and the reuse-distance histograms. To achieve this goal, we propose a number of transformer models that combine the transformed input code with information within the reuse-distance histograms and lead to an output cache miss ratio. We trained and tested these models with simulated data and produced high-accuracy predictions for a large number of replacement policies and LLC cache sizes. These networks can act as a useful tool in cache prediction and may be used on real machines in future research.

Key words: Computer architecture, cache misses, machine learning, neural networks

Contents

Περίληψη	1
Abstract	3
1 Introduction	11
1.1 Problem introduction	11
1.2 Suggested solution	12
1.3 Thesis structure	13
2 Related Work	15
2.1 Cache hierarchy	15
2.2 StatCache	16
2.3 Neural Networks	17
2.3.1 Other usages	17
2.3.2 Multi-Layered Perceptron	17
2.3.3 Long Short-Term Memory	20
2.3.4 Convolutional Neural Networks	21
2.4 Embeddings	22
3 Approach and Methods	25
3.1 Overview	25
3.2 Experimental setup	26
3.2.1 Simulator	26
3.2.2 Dataset	26
3.2.3 Metrics	29
3.3 StatCache calculation	30
3.4 Shallow Multi-Layered Perceptron	32
3.5 LSTM Network	32
3.6 Convolutional Neural Network	36
3.7 Deep Neural Network	37
4 Experimental evaluation	39
4.1 Predicting the miss ratio of an unknown application	39
4.1.1 Predicting for LRU replacement policy	39
4.1.2 Predicting any cache size with LRU replacement policy	41
4.1.3 Predicting other replacement policies	46

4.1.4	Benchmarks	49
4.1.5	Conclusion	53
4.2	Predicting the miss ratio of a known application	53
5	Conclusion	59
5.1	Concluding thoughts	59
5.2	Discussion	60
	Bibliography	66
	List of Abbreviations	67

List of Figures

2.1	The figure illustrates the reuse distances. The arrows indicate reuse of cache lines, and the numbers next to the arrows are the corresponding reuse distances assigned to the memory references pointed at by the arrows [1]	16
2.2	One perceptron neuron. X_i are the inputs to the neuron. Each input is multiplied by the weights W_i on the connecting arrow. Each neuron has one bias b . The output y is derived by applying the activation function $\phi()$ over the sum of weighted inputs and the bias.	18
2.3	Multi-Layered perceptron with 3 hidden layers	19
2.4	LSTM Unit	21
2.5	Convolution Operation [2]	22
3.1	Average miss ratio of all benchmarks with LRU replacement policy	28
3.2	StatCache’s absolute prediction errors for LRU	31
3.3	StatCache’s absolute absolute prediction errors.	32
3.4	LSTM network’s structure. input_1 are the embeddings, input_2 is the reuse distance histogram and input_3 is the reuse distance histogram reversed (from largest to smallest)	34
3.5	Convolutional neural network’s structure. input_1 are the embeddings and input_2 are the reuse distances.	36
3.6	Deep Neural Network for predicting other cache sizes. input_4 is the size of the cache, input_5 are the reuse distances and input_6 are the embeddings.	38
4.1	Prediction errors of the MLP, LSTM, CNN, DNN networks and StatCache for LLCs with LRU replacement policy	40
4.2	StatCache, LSTM, CNN, and DNN absolute prediction errors for LLC of size 4MB with LRU replacement policy	42
4.3	DNN network’s absolute prediction errors, trained on the 1MB, 2MB and 8MB cache sizes.	43
4.4	DNN network’s absolute prediction errors, trained on the 1MB, 2MB, 4MB and 8MB cache sizes.	44
4.5	DNN network’s absolute prediction errors, trained on the 0.75, 1, 2, 4, 6, 8 MB cache sizes.	45
4.6	Prediction errors of the MLP, LSTM, CNN networks for LLCs with SHiP replacement policy	46

4.7	Prediction errors of the MLP, LSTM, CNN networks for LLCs with SRRIP replacement policy	47
4.8	Prediction errors of the MLP, LSTM, CNN networks for LLCs with Mockingjay replacement policy	48
4.9	Network predictions for 416.gamess with LRU replacement policy.	49
4.10	Some of the worst benchmark to predictions.	51
4.11	Some of the best benchmark predictions.	52
4.12	LSTM Network's absolute prediction errors	54
4.13	DNN absolute prediction errors for 1 cache size, trained with 7 cache sizes' miss ratios	55
4.14	DNN absolute prediction errors for 2 cache sizes, trained with 6 cache sizes' miss ratios	56
4.15	DNN absolute prediction errors for 4 cache sizes, trained with 4 cache sizes' miss ratios	57

List of Tables

3.1	LLC cache configurations	26
4.1	Geometric mean of absolute prediction errors for the caches with LRU replacement policy.	41
4.2	Geometric mean of DNN network's absolute prediction errors, trained on different cache sizes. Together with the LSTM network's Geometric mean of absolute prediction errors, for comparison.	43
4.3	Geometric mean of prediction errors for the caches with SHiP replacement policy.	46
4.4	Geometric mean of prediction errors for the caches with SRRIP replacement policy.	48
4.5	Geometric mean of prediction errors for the caches with Mockingjay replacement policy.	49
4.6	Geometric mean of LSTM network's absolute prediction errors	53
4.7	Geometric mean of DNN network's absolute prediction errors.	56

Chapter **1**

Introduction

1.1 Problem introduction

Let's delve into the dynamic realm where the architecture of a computer isn't just a blueprint but a catalyst, wielding immense influence over performance outcomes. From the intricate dance of processors to the labyrinthine pathways of memory, every architectural facet orchestrates a symphony that determines the speed, efficiency, and transformative power of computing. Understanding this intricate interplay opens doors to unlocking unparalleled performance potentials, reshaping the digital landscape one architectural innovation at a time.

When establishing the pivotal role of computer architecture in shaping the performance landscape, it's imperative to recognize the intricate dynamics behind this technological symphony. Computer architecture goes beyond being a mere blueprint; it serves as the cornerstone dictating the efficiency and capabilities of modern computing. To comprehend its profound impact, it's essential to examine the intricate orchestration of processors and memory pathways, exploring how each architectural facet harmonizes to define the speed, efficacy, and transformative potential within the digital realm.

The relentless pursuit of faster and more efficient CPUs has been a tale of continual innovation and technological advancement. Since the inception of computing, the quest for speed has driven engineers and scientists to push the boundaries of what's possible. Initially, CPUs were constructed using basic electronic components. However, as technology advanced, the late 20th century witnessed a monumental shift with the advent of integrated circuits and microprocessors. Moore's Law, formulated by Intel co-founder Gordon Moore in 1965, predicted the doubling of transistor counts on a chip approximately every two years. This principle became a guiding force, driving the industry's aspirations for rapid advancements. Over time, the manufacturing process underwent significant enhancements, transitioning from larger transistors to smaller, more densely packed ones, leading to increased processing speeds and improved efficiency. Innovations in semiconductor technology, such as the development of silicon-based chips, the introduction of multicore processors, and the refinement of architectures through pipelining and parallel processing, have been pivotal in the relentless march towards faster CPUs. Additionally, advances in materials science, nanotechnology, and chip design methodologies have collectively contributed to the ongoing evolution of CPUs, fueling an

era of computational power that continues to redefine the boundaries of technological possibility.

CPU caches serve as a crucial mechanism to alleviate bottlenecks in performance by minimizing the time it takes for the processor to access frequently used data. When a CPU performs operations, it constantly fetches data and instructions from memory. However, accessing data directly from the main memory can be relatively slow due to the speed difference between the CPU and the memory. Caches act as a high-speed intermediary between the CPU and the main memory, storing frequently accessed data and instructions. By doing so, they reduce the latency in retrieving information required by the CPU. When the processor needs data, it first checks the cache. If the required data is found in the cache (cache hit), it can be accessed much faster than fetching it from the slower main memory. This avoids the need to wait for data from the main memory, thus alleviating the performance bottleneck caused by memory access latency. Effectively utilized caches optimize the CPU's efficiency by reducing idle time, allowing for quicker access to data and enhancing overall system performance across various computational tasks.

Cache misses induce substantial time penalties in computational workflows. When the CPU seeks data that isn't stored in the cache, it triggers a cache miss, prompting the processor to pause its execution while retrieving the required information from the slower main memory. This transition between the cache and main memory incurs a notable delay due to the significant speed gap between these memory tiers. This delay, termed the cache miss penalty, results in tens if not hundreds of idle processor cycles, hindering the smooth execution of instructions and impeding overall system performance. Therefore, identifying and elucidating performance bottlenecks that result from cache misses is crucial.

1.2 Suggested solution

The diversity in cache designs, sizes, mapping strategies, replacement policies, and technological advancements among different CPUs results in varying cache miss behaviors, making it essential to consider the specific characteristics of each CPU architecture when analyzing cache performance.

Cache miss behavior can fluctuate significantly across different CPUs due to variations in cache architecture, size, organization, and access policies. Different CPUs may employ diverse cache designs, such as different levels of cache (L1, L2, L3), cache sizes, associativity, replacement policies, and prefetching strategies. These architectural discrepancies can impact how cache misses occur and their subsequent penalties.

Moreover, architectural advancements and innovations in newer CPU generations often introduce optimizations aimed at reducing cache misses. Improved prefetching techniques, smarter prediction algorithms, or changes in cache hierarchy can impact how cache misses manifest on newer CPUs compared to older ones.

Simulation of a CPU cache serves as an indispensable tool in modern computing, offering crucial insights and optimizations. By mimicking the behavior of caching mechanisms, simulations enable in-depth analysis of cache performance, aiding in the design,

evaluation, and optimization of cache architectures. These simulations facilitate the exploration of diverse cache configurations, replacement policies and associativity allowing engineers to pick the most efficient setups for specific applications. Moreover, cache simulations offer a deeper understanding of how different workloads interact with the cache and impact the cache misses.

The process of simulating CPU caches can be time-consuming due to various factors. The complexity involved in emulating cache behavior requires significant computational resources and time. Handling large traces of memory access patterns for accurate simulations adds to the time needed for processing. Furthermore, incorporating intricate cache designs, such as multiple cache levels (L1, L2, L3), diverse cache sizes, associativity schemes and advanced cache algorithms, contributes to the computational complexity and duration of the simulations. Hence, these simulations prove to be extremely time consuming and a better way of computing miss ratio is needed.

The solution to this problem are prediction mechanisms. The importance of a model that can accurately predict the miss ratio of a workload without having the burden of simulating each aspect of it is immense. Almost all of the existing models for predicting miss ratios are created for LRU or random replacement policies [3, 4, 1]. The only existing model [3] that can use other replacement policies than LRU, needs to know, prior to execution, analytically all the configuration details. This leaves a large part of the problem unanswered, since most modern CPU processors don't use the LRU replacement policy in their caches and, in most cases, we don't know what replacement policy is being used.

1.3 Thesis structure

In this thesis we are tackling this problem of predicting cache misses on any given architecture by using machine learning. We propose a number of different networks that have the goal of estimating cache miss ratios as accurately as possible. Inspired by StatCache 2.2, the existing probabilistic approach to this question, we decided to use the same data that it uses for its prediction to train and evaluate a machine learning network. The current rise in popularity of NLP networks motivated us to add a transformer branch to the networks, that uses this new technology of understanding the source code, to obtain better results.

We created a total of three different network architectures that predict the cache miss ratio of any program for set cache sizes with any replacement policy. These networks perform really well for any replacement policy. We also introduced one that predicts the miss ratios for any cache size and any replacement policy, whose function we showcased on LRU. In the case of LRU, all of these networks predict the cache miss ratios significantly better than the existing probabilistic approach to the question.

This thesis is structured into five main chapters, each contributing uniquely to the exploration and analysis of miss ratios. Chapter 1 introduces the fundamental concepts and the theoretical framework that supports the study, presenting an overview of the historical context and the significance of predicting cache misses. Chapter 2 delves into the existing literature, critically examining prior research that will help understand what

this thesis is trying to address. Chapter 3 outlines the research methodology adopted, detailing the chosen approach, data collection methods and network architectures used. It also holds individually the results for each one of the networks that we will propose. Chapter 4 presents the collected results derived from our networks, offering an analysis and comparison of the data collected. Finally, Chapter 5 synthesizes the findings, discusses their implications, and offers conclusions along with recommendations for future research cache miss ratio prediction.

In summary, this thesis aims to use the current advancements in machine learning in order to predict the cache miss ratios of any program on any cache architecture. By exploring possible solutions to this problem such as the probabilistic approach or NLP algorithms, this study intends to shed light on the difficulty and the many aspects of predicting cache misses accurately. The following chapters will examine these concepts in detail. Chapter 2 will focus on related work that has been done, while Chapter 3 will delve into our proposal to solve this issue. This structured approach will provide a comprehensive understanding of predicting cache misses. Now, let's proceed to delve deeper into these areas, starting with the StatCache probabilistic approach.

Chapter 2

Related Work

2.1 Cache hierarchy

Back in the history of computer development, CPU speeds were outpacing memory access speeds. This discrepancy led to idle CPUs waiting for data from main memory. Enter cache memory, a concept first proposed by British computer scientist Maurice Wilkes in 1965 [5]. Early cache models improved data access latency, but making main memory entirely high-speed was prohibitively expensive. Researchers explored better designs, eventually leading to the idea of multi-level caches. These multi-level cache models, such as the three-level caches found in Intel's Core i7 products, strike a balance between cost and performance. Now, CPUs can tap into a hierarchy of caches, each level serving as a buffer between the processor and main memory, ensuring efficient data flow and faster execution.

Each one of these caches has its own size, and inner structure, independently of caches of other levels. There are many ways to configure a cache as to the structure. The important ones that we are going to discuss are the size, associativity and replacement policy. For the size, its only dependency is to be larger than the size of the cache in the lower levels. Generally meaning that the $\text{size}(L3) > \text{size}(L2) > \text{size}(L1)$. This is because the speed of lower level caches is supposed to be higher and therefore a smaller cache is required. Associativity is referring to the inner structure of this cache, specifically, it deals with the methods used to determine where data can be stored within the cache. In our case, we will be using set-associative caches, but there also exist fully-associative and direct mapping caches.

Lastly, it is important to define the replacement policy of the cache. Replacement policies are algorithms or strategies used to decide which cache entries to replace or evict when new data needs to be loaded into a full cache. These policies are crucial for maintaining the efficiency and performance of the cache system, as they directly influence the hit rates of cache accesses. LRU is the most known replacement policy that removes the cache entry that has not been accessed for the longest time. Modern processors use more modern replacement policies that are often not known to the user of a CPU. Therefore, we will be using a range of different replacement policies in this thesis, to show that the predictions consistently exhibit high quality across various machines. SHiP [6], SRRIP [7] and Mockingjay [8] are modern replacement policies that aim at predicting the

reuse of a cache entry and replace the least likely to be reused.

2.2 StatCache

There have been statistical attempts of estimating cache-miss ratios for the LRU replacement policy. Namely, StatCache [1] offers a probabilistic approach towards this goal using the reuse distances of said program.

Firstly, it is necessary to state that the data is stored in cache-line-sized pieces of data within the memory. We denote as a memory access the access to a cache-line-sized block. Let N be the number of memory accesses that occur during the execution of the program. We can enumerate those accesses from 1 to N . Suppose $i < j < N$ where i and j accesses to the same block and no accesses to the same block have occurred between i and j . We can say that the reuse distance of this is $i-j-1$, or, equally, the accesses that happened between 2 consecutive accesses to this block are $i-j-1$. We gather these reuse distances into a histogram h , where $h(i)$ denotes the accesses that have a reuse distance of i .

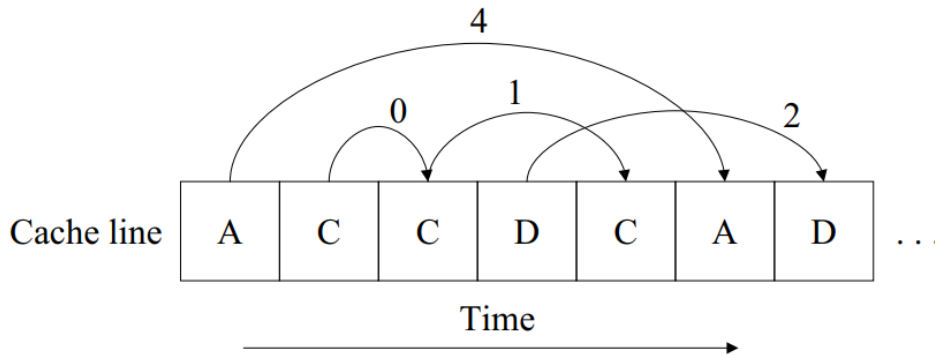


Figure 2.1. The figure illustrates the reuse distances. The arrows indicate reuse of cache lines, and the numbers next to the arrows are the corresponding reuse distances assigned to the memory references pointed at by the arrows [1]

Using those reuse distances, a mathematical equation is derived:

$$RN \approx h(1)f(R) + h(2)f(2R) + h(3)f(3R) + \dots \quad (2.1)$$

where R is the miss ratio, N is the highest possible reuse distance, h is the aforementioned histogram where $h(i)$ tells us that there are $h(i)$ references with reuse distance i .

$F(n)$ denotes the probability of the cache line not staying in cache after n misses. So, assuming the cache is fully associative and has L cache lines, the probability that a line will not remain in the cache after n misses is:

$$f(n) = 1 - (1 - 1/L)^n$$

This is basically StatCache's probabilistic approach to the question. Each reuse distance together with its probability of staying in the cache computes a number of cache misses that, summed up, give the total misses of the execution. Benefits of this method of

estimation are that it's simple and quick to calculate. The main downside of this method is that it only works for the LRU replacement policy.

2.3 Neural Networks

2.3.1 Other usages

Deep neural networks have shown their capabilities of analyzing and understanding complex patterns in multiple kinds of problems, such as image classification [9, 10]. In our case, helpful are researches that are concerned with computer code [11, 12].

In parallel with these, there is also a rise in popularity of neural networks within the contexts of computer architectures. For example, many compiler optimizations have resulted from the implementation of machine learning techniques. In instruction scheduling, the preference function of one scheduling over another can be computed by the RL algorithm's temporal difference [13]. The LSTM-based model [14], circumvents manual feature engineering, autonomously acquiring compiler heuristics from raw code. This enables the construction of appropriate embeddings for programs while simultaneously mastering the optimization process.

Many different machine learning networks have been used for predicting similar problems to ours [15]. Dong et al. [16] use artificial neural networks to predict higher-level features (like miss of cache read/write and instructions per cycle) from lower level features (cache associativity, capacity and latency) for non-volatile memory based cache hierarchies. Other machine learning networks have been introduced to help predict efficient resource allocation [17] and task scheduling [18], to always select the path of maximal instructions per cycle.

Numerous introductions of machine learning into the realm of computer architectures, both similar and distinct, have sparked our curiosity. This motivation led us to explore the cache miss ratio prediction problem-solving through the lens of machine learning.

2.3.2 Multi-Layered Perceptron

Via stimuli that happen on receptors all around the body, the human brain can reach several conclusions about the environment and on what actions to take accordingly. Inspired by the structure of the human brain, researchers proposed a structure that would process information similarly to the human brain, by breaking it down into smaller chunks and understanding the dependencies between those. These structures revolutionized the modern era of computing. Due to their similarity to the human brain, they were named neural networks and proved to have significant success in understanding massive data sets.

Just like a physical neural networks consists of neurons, neural networks in artificial intelligence consist out of many smaller parts, that cooperate to create a system. One of those many smaller parts is called a perceptron [19]. Perceptrons are so called neurons that execute one simple function each. They take as input a number of values, say x_i where $i \in [1, n]$. These values get multiplied by a weight, say w_i where $i \in [1, n]$. Then the

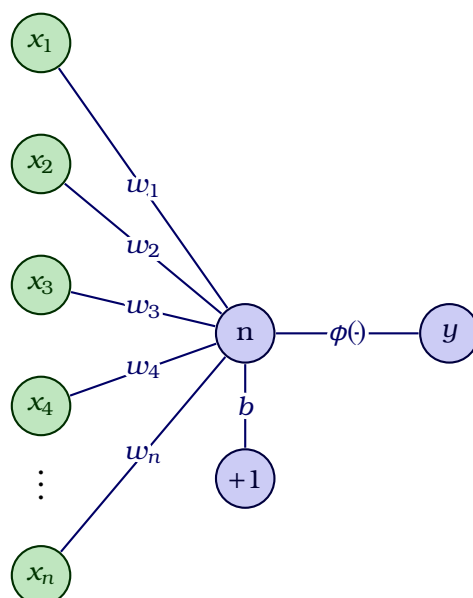


Figure 2.2. One perceptron neuron. x_i are the inputs to the neuron. Each input is multiplied by the weights w_i on the connecting arrow. Each neuron has one bias b . The output y is derived by applying the activation function $\phi()$ over the sum of weighted inputs and the bias.

output is computed by adding all those values together, adding an extra weight b that's known as a bias and applying a function φ onto the result, aka the activation function.

Meaning, each perceptron neuron derives it's output from the formula:

$$y = \varphi(\sum_{i=0}^n x_i * w_i + b)$$

This is a linear equation which can also be described as a multiplication of two vectors $x * w^T$, where $x = [1, x_1, x_2, \dots, x_n]$, $w = [b, w_1, w_2, \dots, w_n]$.

It is clearly visible that the parameters we can control are the weights of the vector w . These are the parameters that are changed according to the data set to compute the best possible output and are usually initialized to random float numbers and fitted in training.

Feeding the same data into a large number of perceptrons creates a group of perceptrons where each neuron will compute a different output according to the value of it's weights. All the outputs of those perceptrons are collected and fed into either another group of neurons or an output neuron that will calculate the final output of the whole structure, according to their outputs. This group of neurons is called a layer and since it's output is not visible to the outside, but fed into the next layer it's named a hidden layer. If more than 2 hidden layers exist in a network it is called a deep neural network [20]. Deep neural networks are particularly interesting in cases of difficult problems, since they detect more similarities and connections between the input data than a shallow perceptron can with only one hidden layer. This is because the modular function is applied more times to each data point, discarding for each perceptron layer more unimportant features of previous layers. Therefore, since the multi-layer perceptron is either way the multiple application of the above function, the understanding of the data will be more accurate and more efficient.

Fitting such a network to data is called training and it is the process of changing

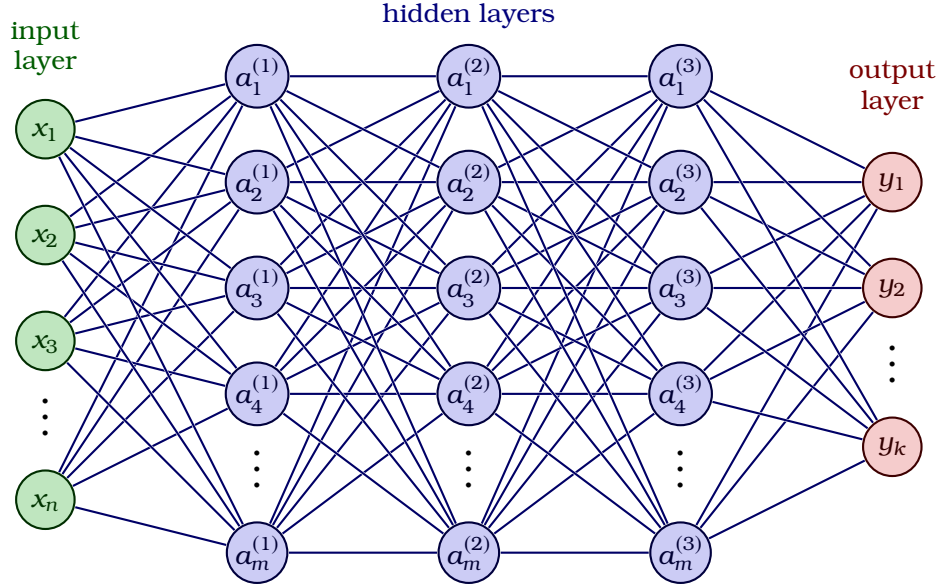


Figure 2.3. Multi-Layered perceptron with 3 hidden layers

the parameters w_k^j for each of the perceptrons k in each layer j of our system with the goal of achieving the least amount of possible error. To train a MLP a process called backpropagation is used.

Forward propagation is the calculation of the output. Since we are using a MLP where each neuron applies the function $y_i = \varphi(\sum_{i=-0}^n x_i * w_i + b)$, we just have to substitute x_i and w_i with the values of the previous layer. Therefore, if the intermediate values of the layer j are:

$$a_k^j = \varphi(\sum_{i=1}^m a_i^{j-1} * w_{k,i}^j + b), k \in [1, layer_size], m = \text{neurons of layer } k-1.$$

Or, equivalently:

$$a_k^j = \varphi(a^{j-1} w_k^j), w = [b, w_1, \dots, w_n], a^{j-1} = [1, a_1^{j-1}, \dots, a_n^{j-1}]$$

The first layer a^1 relies solely on the input layer x and the known weights, therefore we can calculate it's values. Then the next layer can be computed and so on, until we reach an output y (aka a^L where L denotes the last hidden layer). The output of the forward propagation, has a deviation from the real value of $y - v$ and a loss of $C(y - v)$, where C is a loss function and v the real value.

In the backpropagation step derivatives are being used to fit the values of the weights.

Firstly, if we declare the weighted input of a layer l as z^l , a derivative of the loss towards the input can be expanded to:

$$\frac{\partial C}{\partial x} = \frac{dC}{da^L} \cdot \frac{da^L}{dz^L} \cdot \frac{dz^L}{da^{L-1}} \cdot \dots \cdot \frac{da^1}{dz^1} \cdot \frac{\partial z^1}{\partial x}$$

The derivative of $\frac{dz^l}{da^{l-1}}$ equals to the weights that are used for this layer w^l . Additionally, the derivative $\frac{da^l}{dz^l}$ denotes the derivative of the activation function φ . Hence, the previous equation can be denoted as:

$$\nabla_x C = \nabla_{a^L} C \cdot (\varphi^L)' \cdot (w^L)^T \cdot (\varphi^{L-1})' \cdot \dots \cdot (\varphi^1)' \cdot (w^1)^T$$

Here, we declare an additional parameter δ^l which stores the values after the layer l :

$$\delta^l = \nabla_{a^L} C \cdot (\varphi^L)' \cdot (w^L)^T \cdot (\varphi^{L-1})' \cdot \dots \cdot (\varphi^{l+1})' \cdot (w^{l+1})^T \cdot (\varphi^l)'$$

Clearly, δ^l has a size of the neurons in layer l . Each one of those neurons gets a value

and that value is interpreted as the contribution of this neuron towards the computed output and therefore loss. Also this helps in the recursive computation, since we can express δ^{l-1} as :

$$\delta^{l-1} = (\varphi^{l-1})' \cdot (w^l)^T \cdot \delta^l, \text{ and}$$

$$\delta^L = (\varphi)' \cdot \nabla_{a^L} C$$

which is computable. Lastly, due to the layer output of layer l-1 being left when we differentiate over the weights of layer l, the gradient of the weights of layer l is:

$$\nabla_{w^L} C = \delta^l (a^{l-1})^T$$

As previously mentioned, this whole process is called backpropagation and the objective is tweaking the weights of our network after each iteration over an object in the training dataset to give a better result. That is how a neural network learns.

2.3.3 Long Short-Term Memory

Recurrent neural networks are another type of neural network, different from the previously mentioned MLP. RNNs consist of neurons that perform a feedback operation. This means that neurons, together with passing their result to the next layer to create the forward passing, just like the aforementioned MLP networks, they re-feed the output as input to themselves. This concept has been applied to many areas of interest, such as handwriting recognition [21] or acoustic modeling [22]. Similarly, we hypothesize that this method could help our model understand the reuse distances better since the reuse distances are not incoherent data-points that just exist; they are a sequence where consecutive buckets have similar access points.

The most known type of RNNs are the Long Short-Term Memory models. These were introduced in 1977 [23] and has since been used in multiple applications. Goal of these networks is combating the vanishing gradient problem. This is a problem that occurs in RNNs where one gradient is applied multiple times and if it's small it will multiply itself into insignificance. This is a problem which will cause parts of the network not being able to fit [24].

LSTM units take as input 3 parameters one being the input step from the input sequence and the other 2 a cell state and a hidden state. The cell state is the important one, which holds the information passed to our unit from previous time steps. The hidden state is the state that holds the output of the previous timestep and that will hold the output of this timestep once finished.

Each unit of a LSTM network goes through 3 stages in it's execution.

Firstly, the forget state determines how much of the input state C_{t-1} will be forgotten. It executes the formula:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Basically, this takes into account the h_{t-1} and x_t , then, due to the sigma function, computes a number between in the range [0, 1] and multiplies the C_{t-1} state with it.

Second phase is the input gate. This is when the new information is being added to the cell state $f_t * C_{t-1}$ to compute it's new value C_t . For this purpose, we need to calculate a \tilde{C}_t value from the inputs x, h_{t-1} and a multiplier i_t which is going to decide how big the

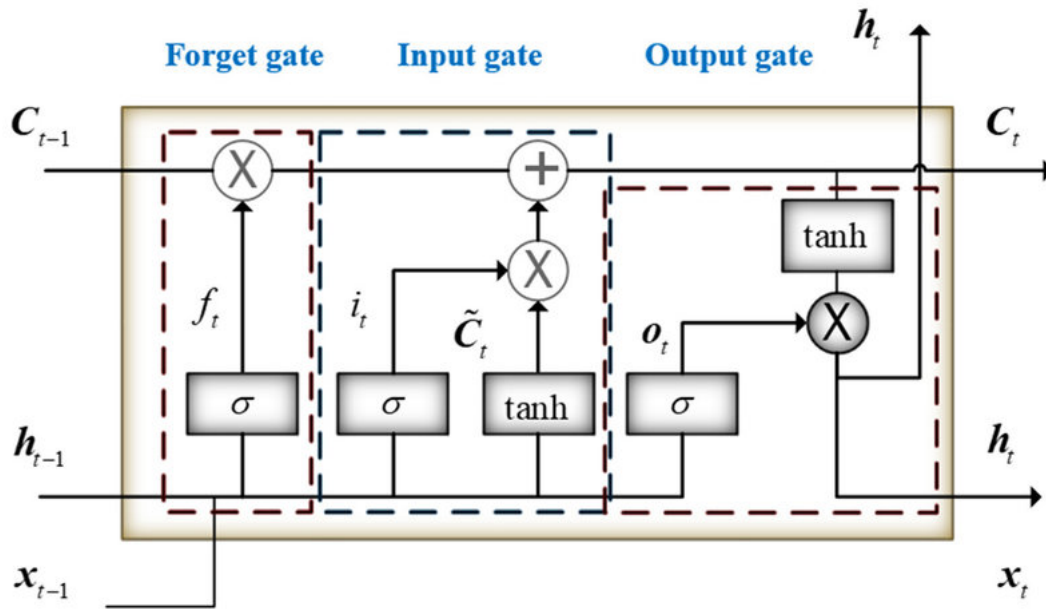


Figure 2.4. LSTM Unit

impact of the new state will be. The way to do that is just as before, using the respective functions and weights.

$$\tilde{C}_t = \tanh(W_t \cdot [h_{t-1}, x_t] + b_t)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Consequently, the current state is computed as:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Lastly, we have the output phase. To compute an output given an input we'll just apply tanh to the cell state and multiply it with a factor that results from the previous hidden state.

$$h_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \cdot \tanh(C_t)$$

This is the output h_t that will be forwarded to next layers.

2.3.4 Convolutional Neural Networks

Convolutional Neural Networks are a fundamental component of image recognition and classification. They appear in all sorts of problems involving image classification and or recognition [25, 26], but they appear in other sectors as well such as stock market analysis [27] or NLP [28]. They use a simple yet effective method of tackling the problem of having too many parameters and limited memory. This method involves convolution from which the name is derived.

First step, is creating a kernel of a specific size, which holds the weights that we want to create a convolution with. Then, this kernel is moved over every possible position of the input matrix and make a convolution between it's values and the input matrix's values. A convolution is simply a sum of the multiplication of each value that is covered by the kernel times the value of the kernel in it's respective cell.

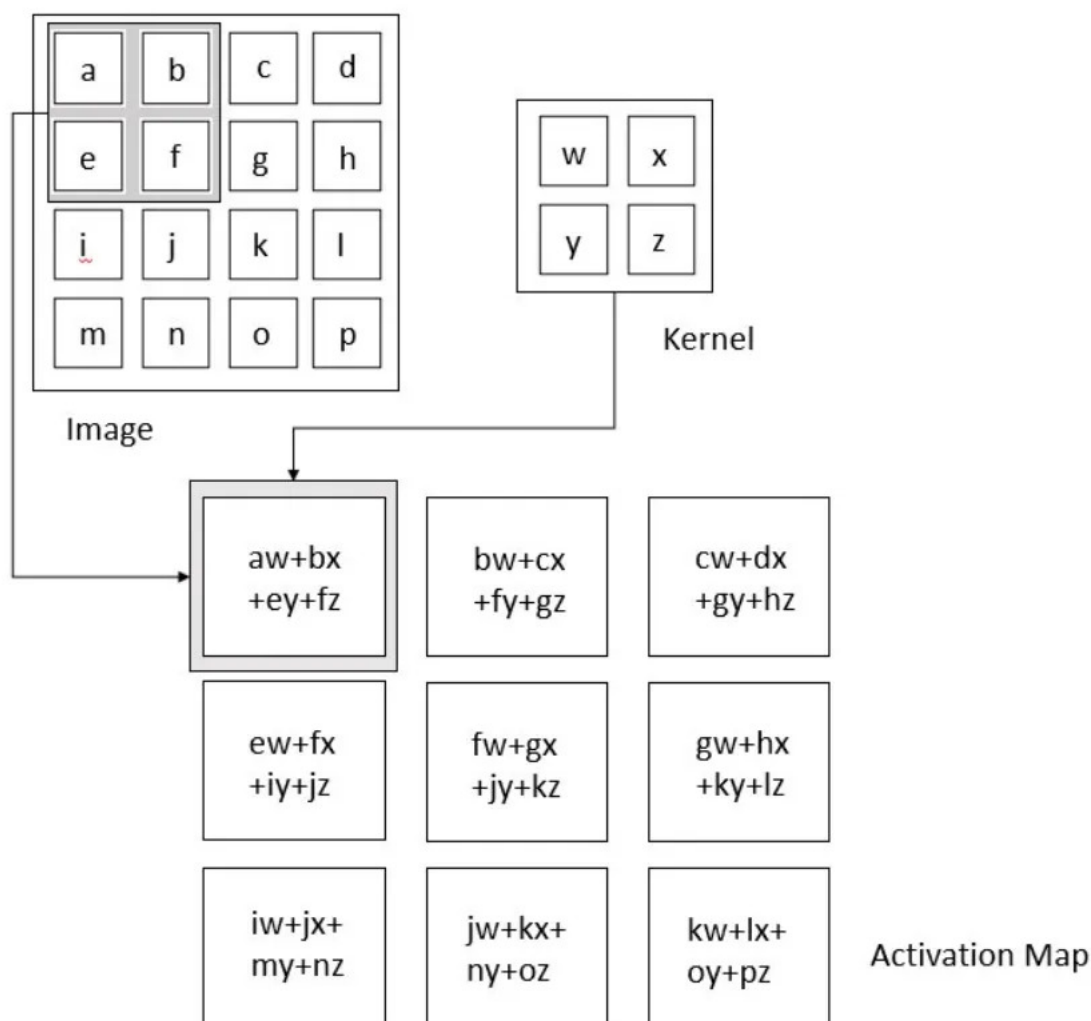


Figure 2.5. Convolution Operation [2]

After a CNN layer a pooling layer is often added [29]. Max and average pooling layers are the most frequently used. It has been shown that max-pooling layers accelerate convergence, enhance generalization as well as selection of better invariant features by [30]. This has also been mathematically proven by [31].

These pooling layers, unlike other layers that have been discussed, have no weight parameters and aim at lowering the data dimensions. They construct a Kernel of shape (K_h, K_v) and apply it to the input matrix in contiguous rectangular areas. To each one of those clusters the function from which the layer takes its name is applied, i.e. a max-pooling layer takes the max from each cluster it covers and an average-pooling layer the average. This reduces the dimensions of the CNN's output by (K_h, K_v) respectively.

2.4 Embeddings

To use the program code as input for our network we will need to convert it into usable data by a neural network. Embeddings are these representation of objects as vectors. A function converts an object which is recognizable to a human into a vector of numbers,

which can be recognized by a program with the least possible loss of data. This method of representation is used frequently in NLP where each word in the dictionary is converted into a number and fed into a neural network.

A similar technique has been developed for creating embeddings of code [32, 33] where the symbols of code are being used to convert a sequence of code symbols into the embedding vector. Each symbol takes a value from 0 to N where N is the arbitrary number of symbols we want to consider (the rest we discard) and this is known as a vocabulary. Then this conversion of symbols will be used by some mechanism to transform those arrays of symbols into a vector with the least possible loss of data.

In our case we will be using the IR2Vec [32] for the creation of our embeddings. IR2Vec is an embedding calculator that has the goal of creating a vector representation as accurate as possible of an IR file. IR file stands for intermediate representation file. It is the file that the input code is transformed into by the compiler before it gets transformed to machine code. In this stage of compilation the code has already been optimized and all the flags have been applied which makes it the ideal starting point to create an accurate vector representation of the program.

Via unsupervised learning the authors of IR2Vec created a network that identifies and represents these IR files accurately. Their network was trained and tested on the SPEC CPU 17 benchmarks and Boost library. Their experimental outcomes ensure the practical viability of this network in our problem, since our experimental data also stems from the SPEC benchmark suites as we'll discuss later on. Thus, we are going to use this network and vocabulary as a black box to transform the source code of our dataset. Output of this procedure is a vector of size 300 which is supposed to give our model vital information about the proceedings of the program and how to understand the reuse distances.

Chapter **3**

Approach and Methods

3.1 Overview

Our goal with this thesis is to predict the cache miss ratios given a program and a cache architecture. To use a program as input, we need to convert its features into usable data for our network to predict from. For this purpose, according to StatCache 2.2, we need the reuse distances of each program in our benchmark set. Reuse distances, as mentioned previously, are the distances between consecutive accesses to the same address in memory. These distances depend mainly on the program and its inputs; they vary very little between different architectures and therefore will only be captured once from each trace of a program. They are going to be stored in a histogram, and one branch of the network will try to predict the miss ratios using them.

Next, we need to transform the input code into a vector (embedding) so that the network can retrieve even more information from the input code about the structure of the program, the flow of data, etc. To predict the cache miss ratios accurately, we need to interpret the transformed data appropriately. The embeddings vector can be fed into a dense neural network, so this is what we do for the transformer branch. The reuse distance histograms, on the other hand, are more complex to understand; a simple dense neural network is not the most well-suited network to fully understand the complexity of this problem.

For this purpose, we introduce three different models that understand reuse distances differently. The first one is a shallow multi-layered perceptron that computes the miss ratio solely based on reuse distances. Next, we introduce a double LSTM layer and a CNN network that has the purpose of better understanding the reuse distances given the neighboring reuse distances. These two networks compute the cache miss ratio better, using the reuse distances as well as the transformer branch.

Lastly, we introduce a deep MLP that takes as input the reuse distances and embeddings, introduces the LLC size as a parameter, and computes the miss ratios for other cache architectures with the same replacement policy.

3.2 Experimental setup

3.2.1 Simulator

The initial input data for our system consists of the SPEC 2007 and SPEC 2016 benchmark suites [34, 35]. SPEC are benchmark suites designed to provide a comparative measure of compute-intensive performance across the widest practical range of hardware using workloads developed from real-user applications. These suites are perfect to showcase our goal since they provide a large variety of problems and algorithms covering a wide range of computing tasks.

To simulate these programs, we use ChampSim, a processor simulator aimed primarily at simulating the memory subsystem and branch prediction as accurately as possible [36]. We use this simulation tool to collect the runtime data that we will need in the next section. It executes the trace of a program on a simulated machine based on a given architecture. For this purpose, we use the traces given by the 3rd Data Prefetching Championship. These traces consist of 2 billion instructions each and cover large parts of the execution of a benchmark. Each benchmark has anywhere from one to six traces to its name, each representing a given percentage of its execution. Later on, we'll use this percentage as a weight to compute the cache misses of a benchmark as a weighted average of the miss ratios of its traces.

The cache that suffers the longest miss latency is the LLC. This makes the prediction for it far more valuable than for other-level caches. Therefore, in our approach, we keep the lower-level caches invariant. L1D is of size 48KB, associativity of 12 and LRU replacement policy, and the L2 cache has a size of 512KB, with associativity 16 and LRU replacement policy. We conducted the simulations mentioned in the next chapters with a wide range of cache sizes and four replacement policies for the LLC to cover a large set of real-world architectures. We used the cache sizes mentioned in table 3.1. The LLCs with sizes of 768KB, 1536KB, 3072KB, and 6144KB are 12-way associative, whereas the 1024, 2048, 4096, and 8192KB-sized caches have an associativity of 16.

Replacement policy	LLC size [KB]
LRU	768, 1024, 1536, 2048, 3072, 4096, 6144, 8192
SHiP	1024, 2048, 4096, 8192
SRRIP	1024, 2048, 4096, 8192
Mockingjay	1024, 2048, 4096, 8192

Table 3.1. LLC cache configurations

3.2.2 Dataset

Collecting reuse distances

The collection of reuse distances during run time is pretty simple. We implemented a reuse distance profiler to be used within ChampSim. It is given an address and a state, and it calculates the reuse distance. We did this via a simple function that checks the

last time this block was accessed and increments the reuse distance in the histogram. Afterward, the data address gets stored for future reuse distance calculations, and we continue. This implementation is lightweight, since nothing but a few instructions are added on execution, and effective, since efficient data structures are used, such as hash maps.

To use this reuse distance profiler, we added it to the replacement policy of L1D that executes this aforementioned process every n -th time the cache is accessed. This n has to be small enough to get a representative view of the reuse distances in our dataset, but not too small, since we don't want to run the reuse distance profiler for each data access because it increases processing time. In our case, we use $n = 16$ accesses. This is a bit too meticulous for 512k accesses, but we wanted to ensure that the outputs don't get affected by this and thus sacrificed some processing time.

The fluctuation of cache misses throughout the execution of the program can be an issue that negatively influences our estimation. As the program goes through the different execution phases, the cache miss ratio changes over time. According to the StatCache paper [1], dividing the profilers' output into windows with a smaller number of accesses is the solution to this issue. The rationale behind it is that the number of accesses will be small enough so that each window's cache miss ratio is likely to be constant over time.

To accomplish this, we simulated the traces, and for every 512k accesses, we returned the state of the reuse distance histogram. We noticed that this size works best in our case since it gives the simulation time to fill part of the caches and still sample the program's execution frequently enough to not notice the fluctuation. Now, for each benchmark trace, we have a number of windows with 512k accesses each. Each window consists of the reuse distances that occurred within those 512k accesses and the LLC miss ratio that those correspond to. Let it be known that some traces have far fewer than 512k accesses within their execution. We used a window for each one of these traces regardless, so that their data doesn't get lost, but some of them will result in some problems later on due to their different scale.

To calculate the overall miss ratio of a benchmark, one has to average the miss ratio of every window in a trace. This miss ratio is calculated by *cache misses/cache accesses* to the LLC over each window's period of 512k accesses. Then, each trace's miss ratio is multiplied by a weight, which represents the percentage of the total execution of the benchmark that this trace simulates. The sum of these miss ratios multiplied by the trace's weight constructs a weighted average, which will be the miss ratio of the whole benchmark.

As we can see, the average cache miss ratio of all the benchmarks is at 90% for the smallest cache and keeps gradually getting lower, until 55%. This is an immediate consequence of the fact that, on average, larger caches result in lower miss ratios since they can store more information.

To summarize and be more precise, our dataset contains 1 to 373 windows for each one of the 186 traces that result from 47 benchmarks. Each window contains the histogram of reuse distances and the cache miss ratios that were captured during 512k continuous memory accesses during the benchmark's execution. The cache miss ratios are numbers

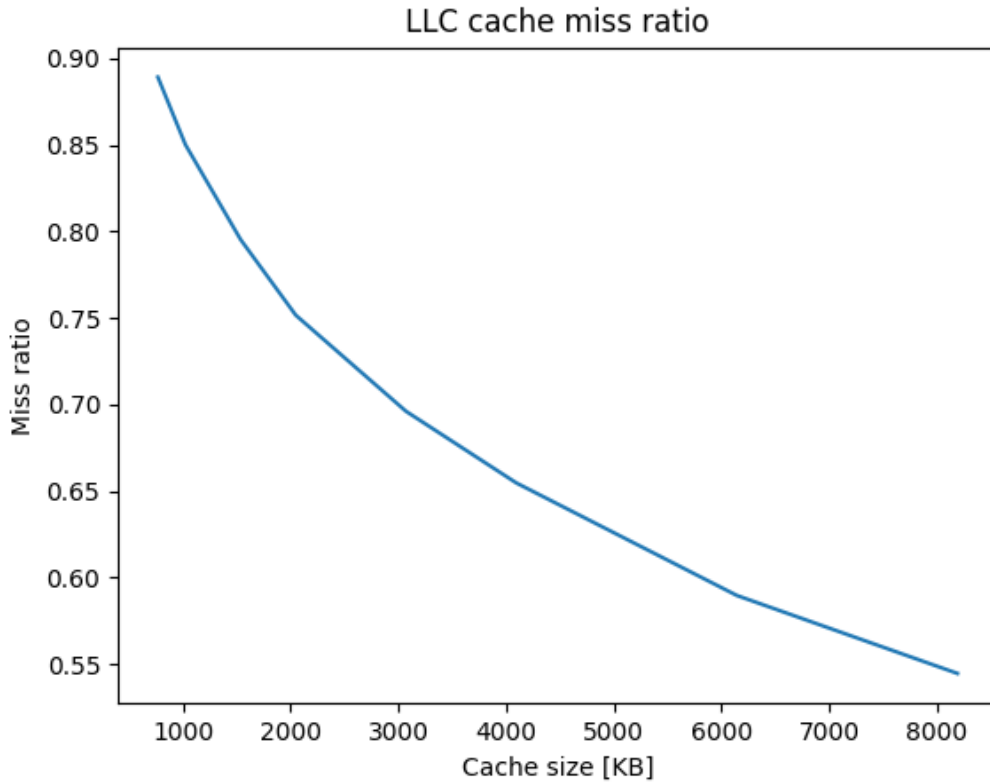


Figure 3.1. Average miss ratio of all benchmarks with LRU replacement policy

in the range $[0, 1]$ based on the simulated values of ChampSim for each one of the LLC configurations mentioned in Table 3.1. These reuse distances will be the input for the first branch of our network, and the miss ratios are the real output onto which the output of the network will be fitted in training and compared against in testing.

Embeddings

Each benchmark’s code needs to be transformed into a form that can later be converted into an embedding. For this reason, we compiled the SPEC files via clang-16 into IR files. LLVM IR files are essentially intermediate assembly instructions that are used to create the compiled code. The compiler options we used to generate these IR files are the default ones suggested by the SPEC toolchains.

We will be using the pre-trained model of IR2Vec for the transformation of the code into vectors. With it, we create 47 vectors (one representing each benchmark) of 300 values. This is the transformer branch of our networks. This helps the rest of the network, via the program code, understand program properties, which will help compute a more accurate miss ratio.

IR2Vec generates either one embedding for each function in the program or one embedding for the whole program at once. The benefit of having one embedding for each function is the precision as to what part of the execution we want to focus on. The program-wide embedding will convert the whole benchmark into one vector of 300 di-

mensions. In our case, we do not know the function or the part of the program that is being executed in each trace; thus, we use one vector that's the embedding of the whole benchmark and one only with the main function's vector for each benchmark.

Train-test split

Another concern is the split of the dataset, into a train and test set, which is not trivial at all. Our dataset consists of tuples. As described previously, each tuple contains a window's reuse distance histogram, miss ratios, as well as the embeddings of the benchmark that it stems from.

The reuse distances have been selected from one trace file at a time, so all the items that result from the same trace file will have a similar reuse distance structure. The same is true for the embeddings; we have collected them from benchmark-compiled code, and all the trace files that have been produced by the same benchmark with different configurations will have the same embedding file as input. For example, the traces 435.gromacs-111B, 435.gromacs-134B, 435.gromacs-226B, and 435.gromacs-228B have all been produced by the 435.gromacs benchmark from SPEC and thus have the same embedding in each item of the dataset. Due to this similarity, we can't have items of a benchmark in the train set as well as in the test set; it would skew our data since the network will have been trained on this structure.

Our initial thought for the train-test split was to separate 20%-30% of the dataset's benchmarks for the test set. But, since our dataset is pretty small, only consisting of 47 benchmarks, separating 20%-30% of the benchmarks for a test set wouldn't be optimal. It would result in a test set that is very small and, most likely, not indicative of all possible programs that the network could face. For this reason, we decided to use a method called leave-one-out cross-validation [37]. We separate one benchmark at a time and try to predict its miss ratios with the network trained on the other 46 benchmarks. This way, we cycle through the 47 benchmarks and get 47 results, one for each benchmark of the dataset.

3.2.3 Metrics

For training and testing of the networks, we used the TensorFlow libraries in Python. The processing times for the networks are from our training and testing procedures that were conducted using an NVIDIA T4 GPU.

We used a mean squared error metric for the training process. This works because, regardless of the window's weight in the final outcome, we want the least possible deviation from its estimated value. It works better than a weighted average (where the weight of a window will be $trace_weight * 1/windows_per_trace$) or a mean absolute error function because it forces the network to reduce the largest deviations. This, in turn, forces the overall average deviation to be smaller.

A normalization layer can boost the accuracy of our model. There are enormous deviations between the numbers in the reuse distance histogram buckets since they are reflective of real cache accesses. A simple standard scaler that is being fit on the training

set proves very helpful for resolving this issue. This one performs a simple transformation of $z = (x - u)/s$, where x is our row, u is a row of means, and s is the standard deviation of each column. Then, we apply this scaler to the train and the test set and continue. Note that this scaler has to be trained only on the train set to not falsely create better results. Calculating the standard deviation on the whole dataset could introduce a bias that would cause the result to be better than the real one in the test set.

The training was done with an Adam optimizer [38], a stochastic method for tweaking the weights of the networks for optimization. Our learning rate was 0.001, batch size was 16, and we trained the network for 50 epochs. For these variables, we tried lots of different configurations and kept this one since it provided the most accurate results.

To evaluate how good a prediction of a network is, we are going to evaluate its absolute error from the real miss ratio. To compute this, we need to average the output of the network for each window in a trace. Afterwards, a weighted average over a benchmark has to be calculated with the trace weight and subtracted from the real value. We'll refer to the absolute value of this subtraction as the absolute prediction error. For each one of the 47 benchmarks in our dataset, an error will be computed with the training set for the other 46 benchmarks, as explained in 3.2.2. To compare these sets of errors with each other and with StatCache predictions, we will use a geometric mean. It has to be mentioned that this way of calculating the overall result of a benchmark loses some information in the data in the analysis. Some traces' differences between prediction and real value vary significantly in some benchmarks, up to 70%. Also, the windows' difference between prediction and real value can vary within a trace. These differences are supposed to even out over the multiple windows and traces.

3.3 StatCache calculation

The outputs of StatCache need to be calculated. It is important to have a competitive, state-of-the-art predictive approach that also relies on reuse distance histograms to compare our results with. Comparing the two will allow us to quantify the effect of using neural networks as opposed to an analytic approach, given the same input data.

StatCache is a probabilistic approach to the problem for LRU replacement policy, and its values are obtained by the formula 2.1. When computing this formula, we get a miss ratio for each one of the windows. After averaging them over each trace and computing the weighted average over each benchmark, just like before, we compute the predicted miss ratio for each benchmark. The absolute error deviations of the predictions from the real values, as depicted in figure 3.2 for the 47 benchmarks, seem to be following a folded normal distribution. This is expected since the approach is trying to predict the value of the real miss ratio. Hence, the absolute error of this prediction should be the absolute of a normal distribution around zero. We'll use the geometric mean to summarize the distributions as one number and compare the outputs with each other. The geometric mean of absolute prediction errors from the StatCache model is 5.4%, 8.3%, 8.7%, and 8.5% for the caches of sizes 1MB, 2MB, 4MB, and 8MB, respectively.

This folded normal distribution results in boxplots such as the ones shown in figure

3.3. The orange line represents the median. The interquartile range (IQR) represents 25% of the values on each side of the median. The whiskers have a maximum length of $1.5 * IQR$ and all values outside of this range are considered outliers. All the boxplots of absolute prediction errors that we'll discuss in this thesis will have a similar structure for this problem. The mean is very low and thus we'll have little boxes and some outliers in the boxplot.

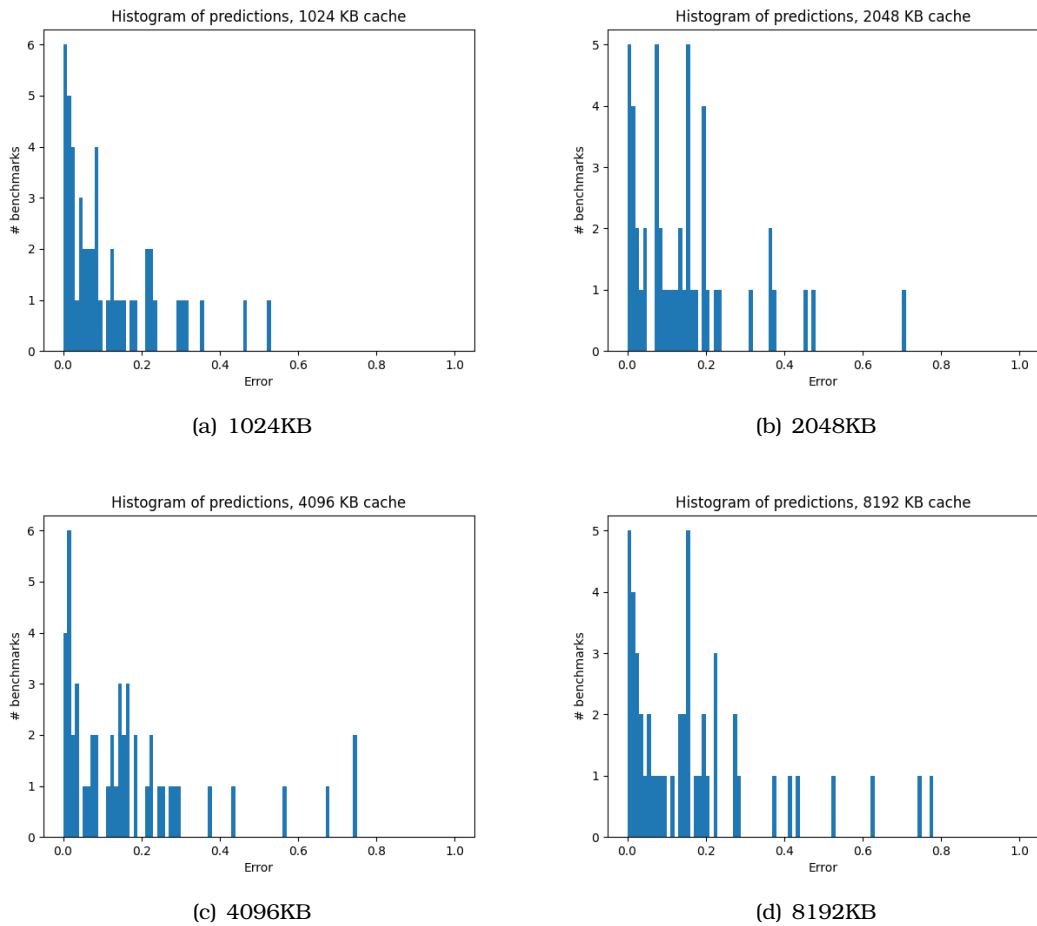


Figure 3.2. *StatCache's absolute prediction errors for LRU*

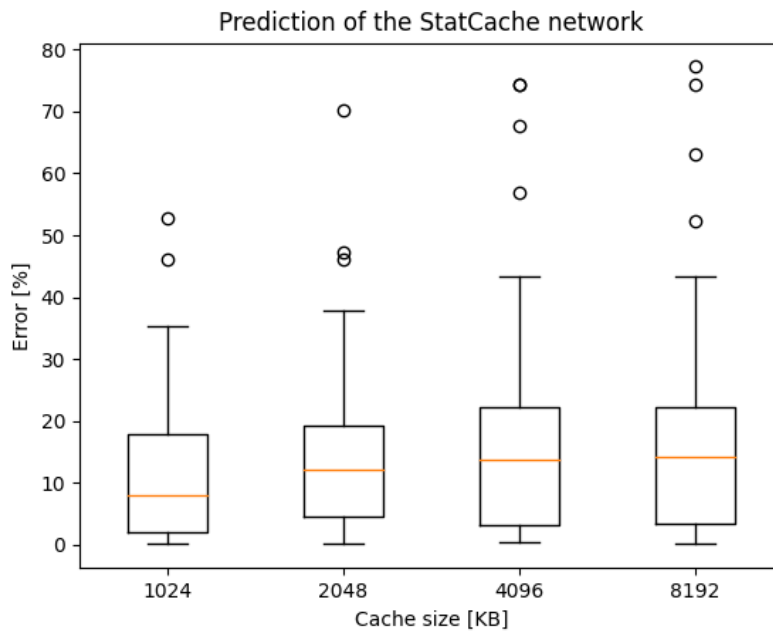


Figure 3.3. *StatCache’s absolute absolute prediction errors.*

3.4 Shallow Multi-Layered Perceptron

First, we created a simple MLP to predict the miss ratios solely from the reuse distances. Reuse distances are enough data to predict such values with high accuracy, especially for the LRU replacement policy. StatCache predicts the miss ratios with a low error, only knowing those; therefore, we suppose that there’s enough data to make an initial prediction and create a first model for it.

The model we created is an MLP with one hidden layer of 512 neurons. It takes as input the reuse distance histogram and produces four outputs, one for each cache size (1MB, 2MB, 4 MB, and 8MB). These four outputs are the cache miss predictions of the network for how the machine with the specific architecture will respond to the given problem.

Since this is the simplest of the networks that we are going to examine, its training time is also the lowest. It only needs 1ms/step, resulting in 1 second per epoch.

3.5 LSTM Network

Our quest for a superior solution has lead us down this path of maximizing our understanding of the reuse distances. When looking at a reuse distance, it is necessary to take into consideration the reuse distances larger and smaller than it to reach a conclusion. For this purpose we will try using an LSTM network.

An LSTM takes as input a series of consecutive values, iterates over them, and computes for each value an output, keeping in a hidden state parameter the values that have already passed. For our problem, we converted our array of reuse distances into a series

and fed it into an LSTM network. Thus, the network can compute a value for each one of the reuse distance histogram's buckets given the previous values that have passed from smaller reuse distances' buckets.

Similarly, we want the network to be able to compute a value for a certain histogram value given the values that are after it from larger reuse distances' buckets. For this reason, we reversed the reuse distance histogram and fed it into a second layer of LSTM units. For this second layer to act as a continuation of the first layer, the output cell and hidden state have to be initialized as the final ones from the first one. Now, the second layer will act as a continuation of the first layer with separate weights.

Originally, the thought behind this network was to utilize attention. This would benefit us in terms of understanding how important a feature is and where to focus when looking at the data. With this reasoning, we implemented an attention layer right after the two LSTM layers to combine them. Attention layers are attention mechanisms inspired by the human brain's cognitive attention. They detect the importance of each datapoint within the outputs via a softmax function [39].

Even though the intuition of using such a network seems correct, the training and output did not perform as well as expected. Its predictions were slightly worse than StatCache's, and therefore we won't show them. This is probably a consequence of not having enough training data. The attention matrix is computed from every output of every LSTM unit within the LSTM layers. This puts an enormous weight on every one of them and includes too many weights to optimize with training.

As an alternative to that, as depicted in figure 3.4, we opted to use a dense perceptron layer to understand the outputs of the LSTM layers. A concatenate and a flatten layer transform the output shapes from the LSTM layers into shapes acceptable to the dense layer. This dense network understands the problem better and can predict miss ratios with more accuracy.

For this network, in addition to the reuse distances, we implemented the transformer branch. We added the embeddings of the code into a parallel layer of perceptrons, `input_1` in figure 3.4. The embeddings resulting from IR2Vec are not normalized, so a normalization with a similar scaler as we used for the reuse distances provides us with better results. Then, we connected all those to the last hidden perceptron layer and, lastly, to the output layer. All of the dense neural networks use a relu activation function except the output layer, which uses the sigmoid function since the miss ratio is a probability in the range [0, 1].

We added some normalization layers in between the small dense layers of the inputs and the big dense layer of the output. These seem to have a positive impact on the outcome of the model. Each one trains on the data of the training set and normalizes the outputs created by the dense neural networks. Layer normalization is beneficial, even though the input data is normalized, because it enables smoother gradients, faster training, and better generalization accuracy. We tested all possible placements of these layers in the network, and the best-performing one was right after the small, dense layers.

As to the specifics of the network, our LSTM layers are of size 4 units each. We experimented with lots of values for the network in the range 1-16 LSTM units and

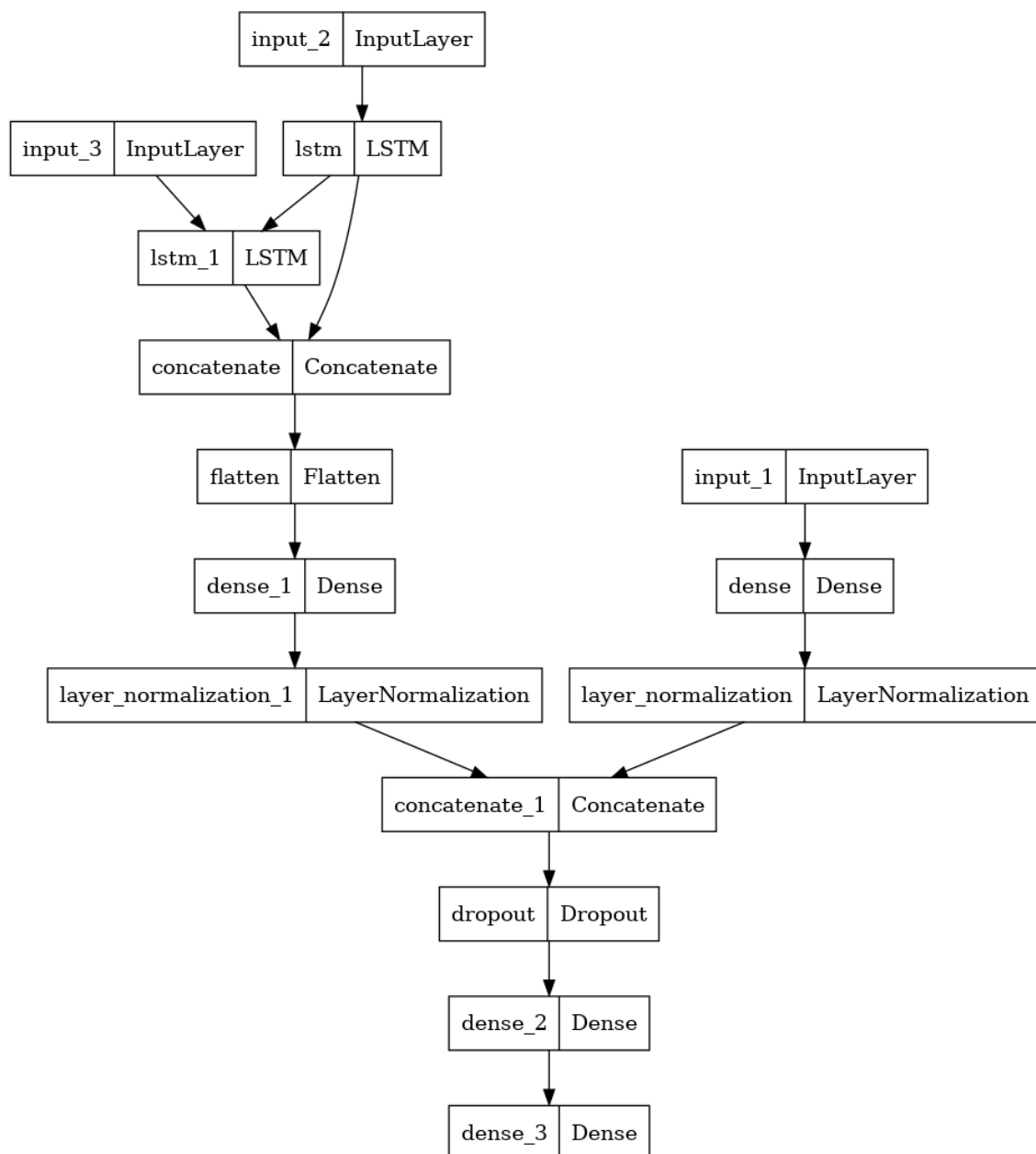


Figure 3.4. LSTM network's structure. *input_1* are the embeddings, *input_2* is the reuse distance histogram and *input_3* is the reuse distance histogram reversed (from largest to smallest)

discovered that 4 units per layer seems to be computing best. The dense layers right after (dense and dense_1 in figure 3.4) are of size 128 neurons, whereas the last dense layer (dense_2) is larger with a size of 1024. The size of the smaller layers doesn't influence the precision too much; sizes of 128, 256, and 512 all have similar results. This size of the larger layer, on the other hand, worked better than other values such as 512 and 2048.

In addition, we use a drop-out layer. This layer randomly sets 20% of its inputs to zero to prevent over-fitting.

This network is the slowest and most difficult to train network that we'll introduce. It trains with an average of 50ms/step or 18 seconds per epoch.

3.6 Convolutional Neural Network



Figure 3.5. Convolutional neural network's structure. *input_1* are the embeddings and *input_2* are the reuse distances.

Convolutional layers exist to help the network identify similarities between features close to each other that might get lost or overlooked if we use a dense network. Convoluting neighboring reuse distances could result in better performance since the input to the dense layer will be more informative.

Then, immediately after, a pooling layer has the purpose of reducing the dimensionality of the previous output. The convolutional layer has n filters, thus producing n results per histogram input. The histogram input is a matrix of shape $(896, 1)$; therefore, the output matrix will be $((896 - n)/stride, n)$. We're using our max-pooling layer to reduce those to dimensions horizontally to $((896 - n)/(stride * K), n)$, where K is the kernel size. There are alternatives to the pooling layer for reducing dimensions, some of the more

popular being max and average pooling or using a larger stride. In this problem, a small stride and max-pooling computed the best outputs, so we used those.

Again, just like in the LSTM implementation, the output of this pooling layer is connected to a dense MLP layer with relu and a normalization layer. The rest of the network is the same as before.

It is worth noting that, with this network, unscaled reuse distances work best. The StandardScaler on the unscaled reuse distances that we used previously doesn't improve our prediction at all. Quite the opposite, the prediction gets worse because the scaling influences the output of the convolution; this results in smaller differences between the resulting values of the convolution and consequently worse understanding. The network prefers unscaled reuse distances passed through it, which will then be passed by the first perceptron layer. Then, a normalization layer will normalize the results of this dense layer, rather than having pre-scaled data. All the possible places to insert normalization layers, with all the possible combinations, were tried, and simply one layer after the perceptron produces the best predictions.

As to the specifics of the network, our CNN network consists of filters that iterate over the input reuse distances with a stride and compute the convolution over a kernel. We experimented with lots of combinations for these three numbers (filter number, stride, and kernel size) in the ranges [1, 20] for each number. This testing concluded that the best combination is 6 filters, kernel size 8, and stride 4. The dense layers right after (dense and dense_1 in figure 3.5) are of size 512 neurons, whereas the last dense layer (dense_2) is larger with a size of 1024. The size of the smaller layers influences the precision a little bit; sizes 128 and 256 have a geometric mean of absolute prediction errors of about 0.5% less. This size of the larger layer, on the other hand, worked better than other values such as 512 and 2048.

This network's training needs on average 6 ms/step or 3 seconds/epoch, which is far quicker when compared to the LSTM network's 50 ms/step or 15 seconds/epoch.

3.7 Deep Neural Network

Lastly, we wanted to see if we could create a network that predicts the miss ratio on any LLC cache size. This has the goal of accurately estimating the miss ratio of each cache size and, ultimately, showing which is the best cache architecture for a specific problem. For this purpose, we simulated the traces again, using in-between cache sizes to create a bigger dataset. This contains the data for LLC cache sizes [768, 1024, 1536, 2048, 3152, 4096, 6044, 8192] KB with the LRU replacement policy.

This network can be used in two separate ways. If we have a program for which we have the data on one or more machines, we can add it to the training set and predict how high of a miss ratio it will have with other cache sizes. The second way of using it is for a new program that hasn't been seen by the network. This second way will in general be more useful than the previous three networks since it will be able to predict the miss ratio for any cache size and not just be restricted to the ones that it has been trained on.

Inspired by the MLP network 3.4, we implemented a deep neural network that is a bit

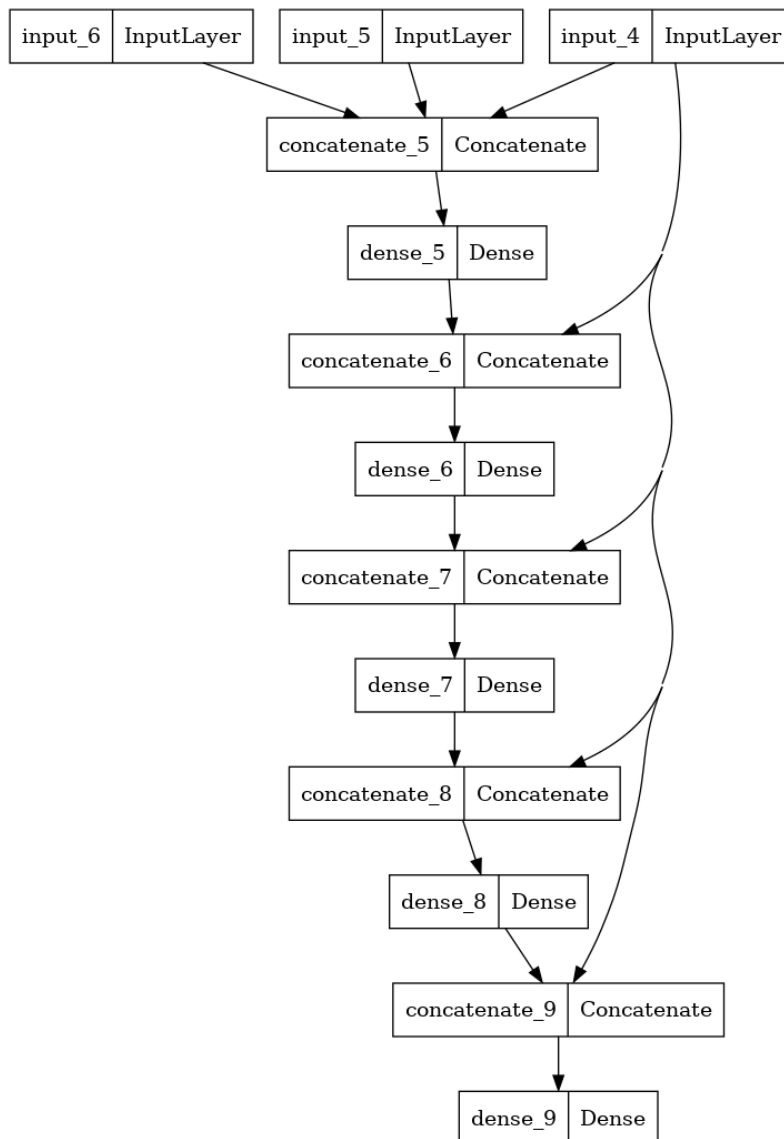


Figure 3.6. Deep Neural Network for predicting other cache sizes. *input_4* is the size of the cache, *input_5* are the reuse distances and *input_6* are the embeddings.

more complicated. For this network, as shown in figure 3.6, we use 4 hidden layers. We want each layer to make predictions according to the size of the cache; that's why we feed the input size to each hidden layer by concatenating it with the output of the previous layer. Each one of the hidden layers has 512 neurons; the first four use the relu activation function, and the last one and the output layer use the sigmoid activation function. This configuration was tested and seemed to have the best results. More hidden layers or more neurons didn't compute a better prediction, perhaps due to creating too many variables.

All three of the inputs contain a normalized version of reuse distances, embeddings, and cache sizes. The reuse distances and embeddings are normalized with the same method as in other networks. The input size was divided by 512KB, resulting in values ranging from 1.5 - 16.0 for each one of the caches.

Chapter 4

Experimental evaluation

4.1 Predicting the miss ratio of an unknown application

4.1.1 Predicting for LRU replacement policy

The problem that we are about to discuss is very straight forward to describe. We are trying to predict the miss ratios of any application that our networks are met with. To show that we use the aforementioned method of leave-one-out cross-validation where we separate one benchmark, train the network on the other benchmarks and try to predict it. This way we will have the best possible prediction for every workload without having seen it previously.

The search for the most accurate miss ratio prediction has led us to introduce four models in total: the MLP, CNN, LSTM, and DNN networks. Each of these models represents a distinct approach to predicting cache miss ratios as accurately as possible. Let's compare and discuss the overall predictions of our networks. For this purpose, we will choose a replacement policy and compare the prediction errors that we have computed with each one of our networks.

The LRU is the only replacement policy for which we can compare StatCache and the DNN networks as well. StatCache was invented and works only on the LRU replacement policy. Our DNN network is probably capable of predicting any replacement policy, but we would need more data on other replacement policies to confirm that it actually predicts any cache size. The DNN network that we are going to show results from is the DNN that has been trained on the 4 cache sizes (1MB, 2MB, 4MB, and 8MB) with the same method of leave-one-out cross-validation as the other networks.

It is clear that all of our networks outperform significantly the StatCache predictions. The outliers, the boxes, and the medians of absolute prediction errors in the boxplot of 4.1 are noticeably higher for the StatCache prediction. It is also clearly noticeable that the geometric means of prediction errors in table 4.1 are clearly worse than any of the other networks in the three lower cache sizes and equal to the worst for the 8MB sized cache.

It is also clearly visible that for larger cache sizes, the absolute prediction errors become larger. For larger cache sizes, the average miss ratio drops drastically. This leads us to believe that for larger caches, even though it is easier to predict a range where the

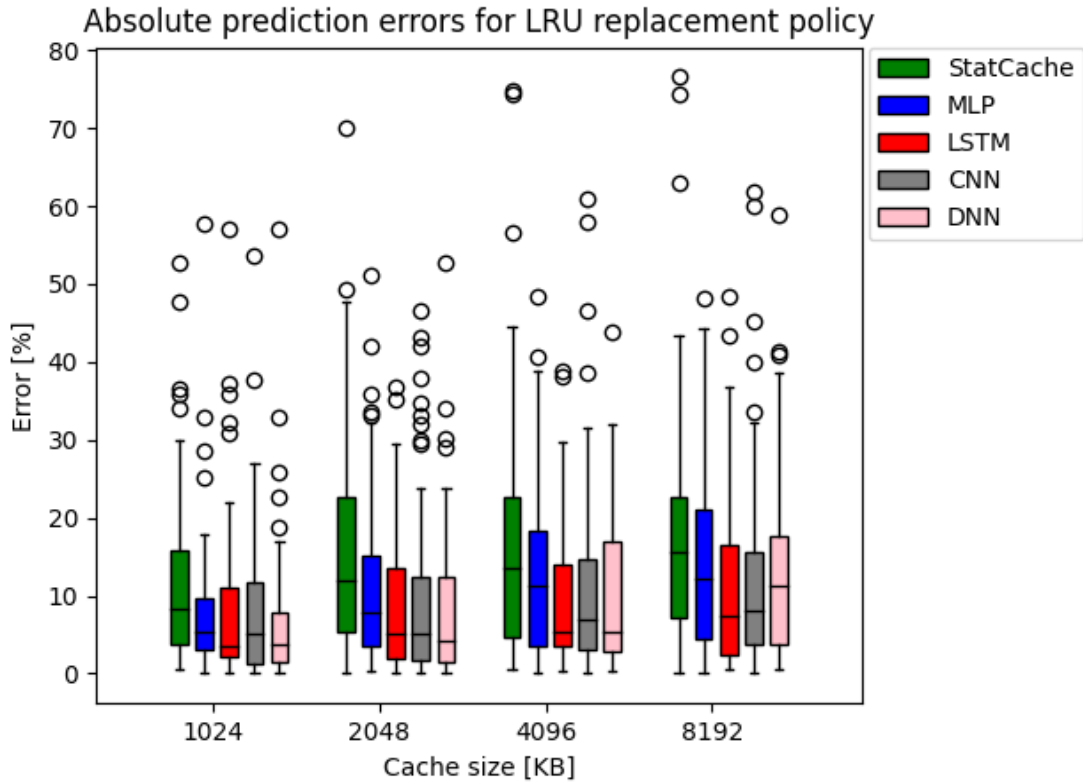


Figure 4.1. Prediction errors of the MLP, LSTM, CNN, DNN networks and StatCache for LLCs with LRU replacement policy

miss ratios will be, it is harder to predict the exact miss ratios.

The shallow MLP is overall the weakest of our networks. The fact that it is not getting as much information as other networks, due to it not having the transformer branch, as well as the worse understanding of the reuse distances, result in larger errors than the other networks. Its prediction errors have higher medians, larger boxes, and many outliers when compared to the CNN and LSTM networks' prediction errors. We also see that by the fact that it consistently produces the second largest prediction error in table 4.1. The goal of this network was to show that it is possible with a simple MLP to produce similar, if not better, results than StatCache only using reuse distances. The results indicate that we have successfully reached this goal.

The other three networks have access to the transformed code of the program, so their predictions are undeniably better than the ones from the MLP. To analyze their predictions more precisely, we will inspect the histograms of their absolute prediction errors 4.2. Since the histograms are similar for other cache sizes as to the shape, we will only show the ones from the 4MB cache size that are representative.

It is obvious that the prediction of StatCache is worse than any of the other predictions. Many large outliers, fewer benchmarks close to zero error, and an almost even distribution in the range of its values make it clear why the predictions of Statcache are worse. The highest column for all other histograms is the smallest one. The LSTM and the CNN

Network	Cache size [MB]			
	1	2	4	8
StatCache	5.4%	8.3%	8.7%	8.5%
MLP	4.1%	7.2%	6.3%	8.5%
LSTM	3.6%	3.9%	5.3%	6.4%
CNN	3.9%	4.6%	5.5%	6.2%
DNN	3.6%	4.0%	6.0%	8.1%

Table 4.1. Geometric mean of absolute prediction errors for the caches with LRU replacement policy.

networks absolute prediction errors have very similar shapes close to zero, with many benchmarks very close and less further away. What differentiates them is that CNN has many outliers. This is also visible in 4.1, where the CNN network’s absolute prediction errors have far more outliers than the LSTM network’s.

Assessing the DNN network’s prediction in comparison to the others is a bit more difficult. Its absolute prediction errors seem to have a more even distribution within the range of 0 to 0.2 with very few outliers. This is slightly worse than the LSTM network’s absolute prediction errors. When looking at the cumulative boxplot 4.1, its predictions seem to be better than LSTM’s for smaller values of cache size. The geometric means for both of these are almost equal. This means that the network has the capability of predicting cache miss ratios as well as the LSTM network, if not better, for small cache sizes. This also means that this DNN-structured network has a weakness for larger cache sizes.

Overall, the best-performing network for all cache sizes with an LRU replacement policy is the LSTM network, closely followed by CNN. The DNN network has a certain weakness for larger values, for which it doesn’t predict as well as the other networks.

4.1.2 Predicting any cache size with LRU replacement policy

After analyzing the results of all networks on these four cache sizes, we want to present the versatility of the DNN network. The DNN network takes as input the cache size parameter and can understand any cache size’s miss ratio. Therefore, we have the miss ratios of more cache sizes in the LLC to show that this DNN network can actually predict them.

In this section, we try to predict any unknown program for any cache size with the given replacement policy. We have the LLC sizes [768, 1024, 1536, 2048, 3152, 4096, 6044, 8192] KB and replacement policy LRU. We are going to use the method of cross-validation for our test set, just like in chapter 3.4, where we’ll iterate through the benchmarks and predict them with a network trained on the other 46 benchmarks. In the end, we will have 47 predictions, one for each benchmark, with the network trained on all other benchmarks each time.

For less than 3 cache sizes in the training set, the prediction holds no value since the network can’t understand the size parameter. When the network is trained on one

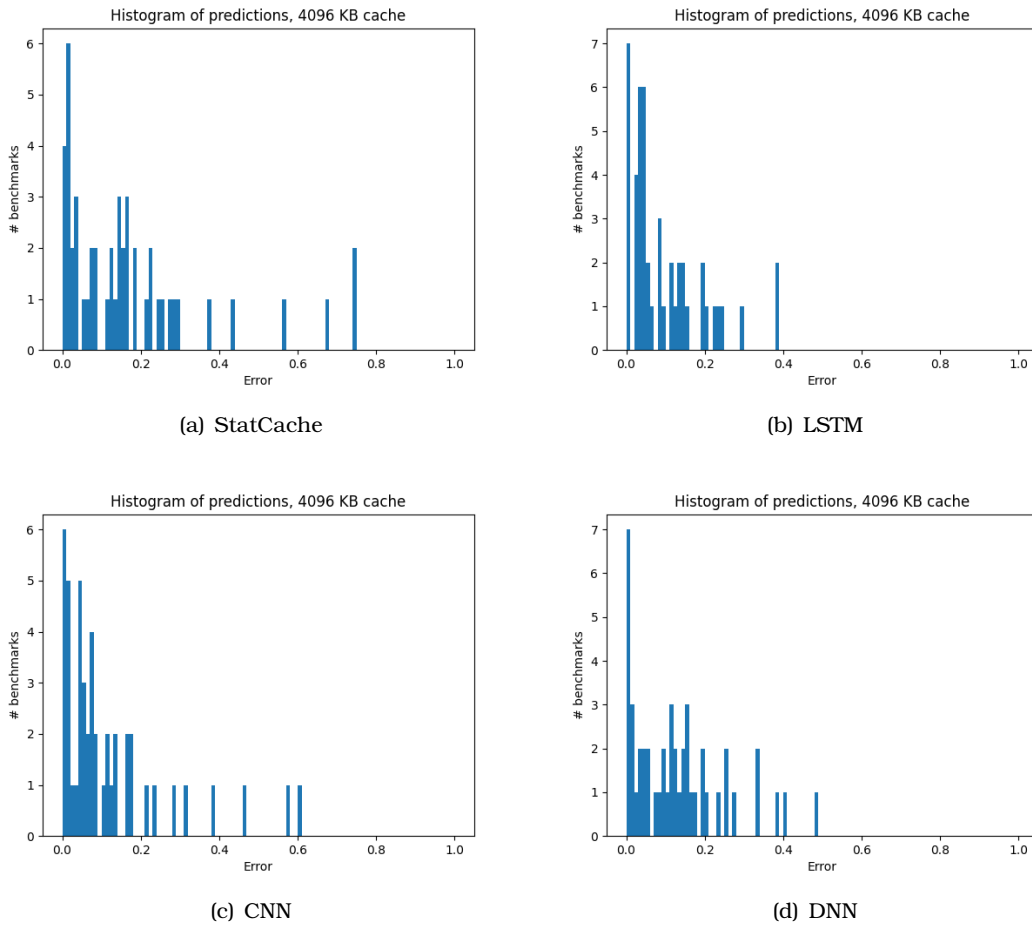


Figure 4.2. *StatCache, LSTM, CNN, and DNN absolute prediction errors for LLC of size 4MB with LRU replacement policy*

cache size, the size parameter is not changing in the training set, so the network will just assign random values to it. When two cache sizes are used in the training set, the network understands it as a binary problem and tries to apply a sigmoid function to it. This makes the predictions unreliable. So we will show the predictions with 3, 4, and 6 cache sizes in the training set.

The network has been trained with the 1MB, 2MB, and 8MB cache sizes in the training set. As we can see in figure 4.3, the absolute prediction errors for the lower caches are very good. The medians are low and steady. There is mostly only one outlier for these caches, and that is the prediction for the 416.gamess benchmark; its miss ratio is always predicted to be higher, and it will be miss-predicted in the future as well, for good reason. This benchmark consists of one trace, which only has 17k accesses to the LLC, which means that the network reads one window of reuse distances with 17k accesses and tries to predict the miss ratio of it. The majority of reused distance windows that our network is trained on consist of 512k accesses. Thus, the network sees the low number of reuse distances and assumes lots of accesses that only happen once and therefore don't have reuse distances. These result in misses (cold misses), and therefore the network computes a higher miss ratio. We can safely ignore this output since, with a 30x longer simulation,

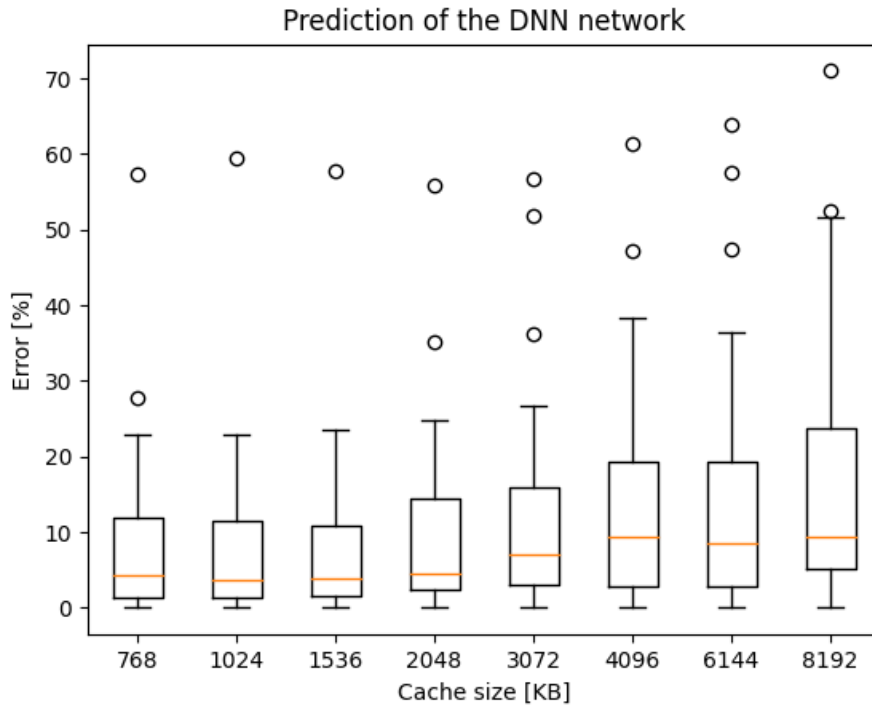


Figure 4.3. DNN network’s absolute prediction errors, trained on the 1MB, 2MB and 8MB cache sizes.

the output would most likely get closer to the real value.

As for the larger LLCs, the predictions are gradually getting worse. The precision of the network for these caches is significantly worse than the ones from the LSTM network, as suggested by their geometric means 4.2. The geometric means of prediction errors are close to those predicted by the LSTM for smaller cache sizes and get drastically worse for larger cache sizes.

Train-set cache sizes [MB]	Cache size [KB]							
	768	1024	1536	2048	3072	4096	6144	8192
1, 2, 8	3.0%	3.1%	3.8%	5.0%	5.8%	6.2%	6.5%	8.5%
1, 2, 4, 8	3.8%	3.6%	3.1%	4.0%	5.1%	6.0%	6.9%	8.1%
0.75, 1, 2, 4, 6, 8	2.5%	3.4%	4.1%	3.5%	4.2%	5.3%	7.2%	9.7%
LSTM Network	-	3.6%	-	3.9%	-	5.3%	-	6.4%

Table 4.2. Geometric mean of DNN network’s absolute prediction errors, trained on different cache sizes. Together with the LSTM network’s Geometric mean of absolute prediction errors, for comparison.

Next, we added the 4MB-sized cache's data to the training set. Now we have the data for 1, 2, 4, and 8 MB-sized LLCs in the training set. This means that the network trains on the exact same data as the previous networks that we introduced.

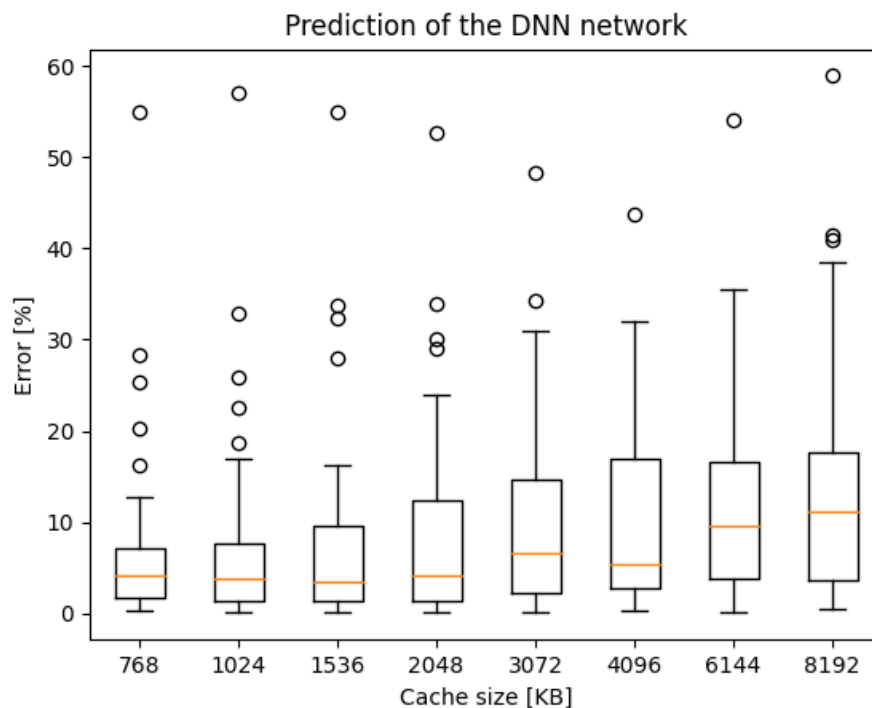


Figure 4.4. DNN network's absolute prediction errors, trained on the 1MB, 2MB, 4MB and 8MB cache sizes.

It is interesting how this addition influenced the prediction of the 4MB-sized cache very little, in the figure 4.4 or in the table 4.2. Apparently, the network could already predict the miss ratios for the 4MB cache size well enough when trained on three cache sizes, therefore, the addition of it changed it very little.

This addition helped our network understand some of the smaller caches better. When first training a network, we trained it only on the 2MB cache size. Only knowing the 2 MB cache gave us a prediction with a geometric mean of absolute prediction errors of 4.7% for the 2MB column. It is interesting how the geometric mean of the 2MB-sized cache (row 2 of table 4.2) is lower than the one predicted with only this cache in the training set. This means that the network gained some insight into the miss ratios of the 2MB-sized cache from the addition of the 4MB-sized cache to the training set.

Lastly, we trained the network on six cache sizes: 0.75, 1, 2, 4, 6, and 8 MB-sized LLCs. This is just to see how the network will behave with such an addition of cache sizes.

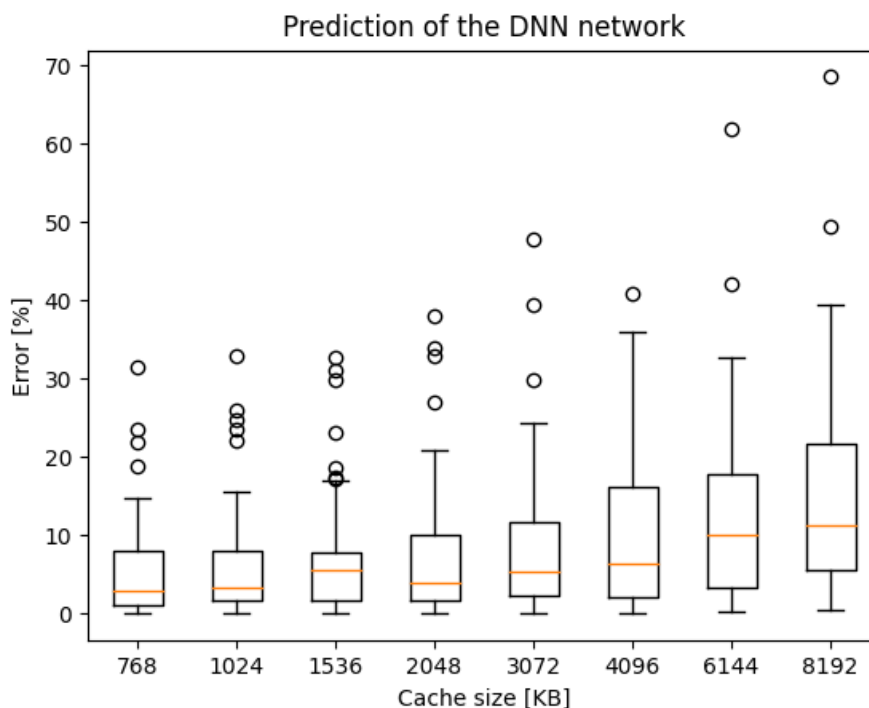


Figure 4.5. DNN network’s absolute prediction errors, trained on the 0.75, 1, 2, 4, 6, 8 MB cache sizes.

The predictions of this training set in figure 4.5 seem very similar to the previous prediction’s errors. The sizes and shapes of the boxes in the boxplot are fairly similar to the one trained with 4 cache sizes. The only difference is that 416.gamess’s predictions have gotten a bit better for smaller cache sizes, but not enough.

In the third row of the table 4.2 we can see that the prediction of the 768 KB cache got even better, as well as the prediction for the 2MB-sized cache. Now, the geometric mean of absolute prediction errors for the 2MB-sized cache is lower than the LSTM network’s prediction that we mentioned at the start. This means that most likely for any cache size lower than 4MB this DNN network will have similar results to the LSTM network. It is also interesting that the geometric mean of absolute prediction errors for the 6 and 8 MB-sized caches has risen. The network focuses more on predicting the small cache sizes exactly and apparently cannot understand the larger ones as precisely.

4.1.3 Predicting other replacement policies

SHiP replacement policy

For the other three replacement policies, we only have the prediction errors from the MLP, CNN, and LSTM networks. The StatCache’s prediction is designed to describe the behavior of LRU caches, and therefore can’t be applied to other cache replacement policies. The DNN network is only trained and tested on the LRU replacement policy; for other replacement policies, we don’t have enough data for enough cache sizes to accurately evaluate its performance in between cache sizes.

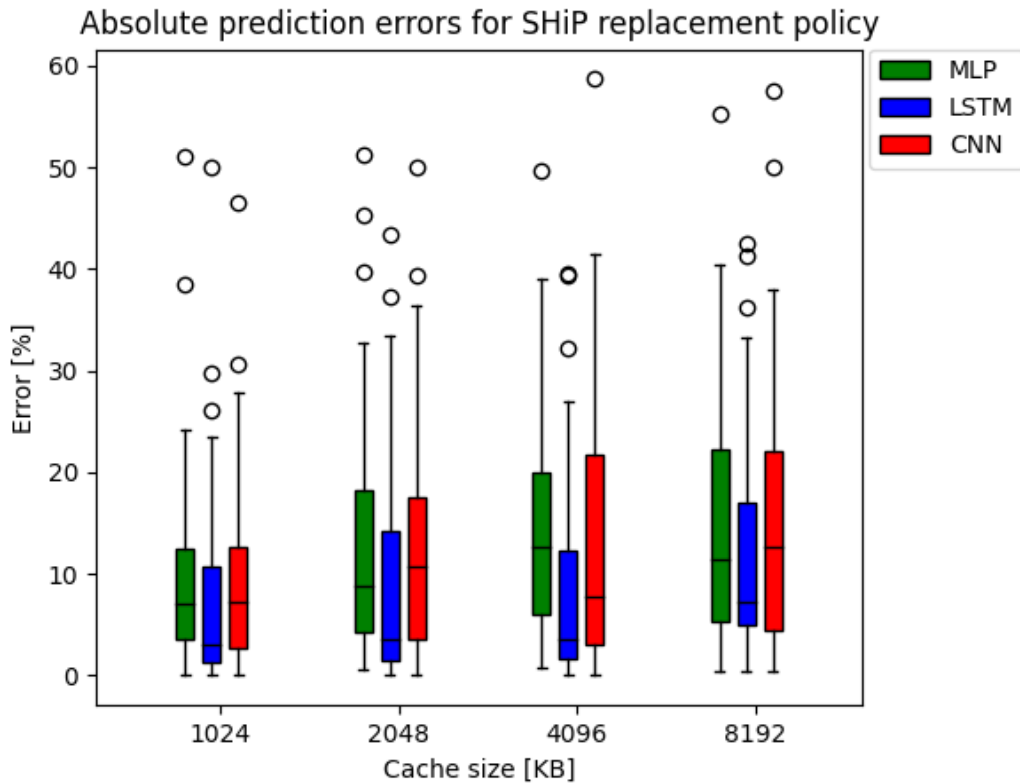


Figure 4.6. Prediction errors of the MLP, LSTM, CNN networks for LLCs with SHiP replacement policy

Network	Cache size [MB]			
	1	2	4	8
MLP	5.6%	7.7%	9.8%	8.6%
LSTM	2.7%	4.1%	3.7%	7.6%
CNN	5.3%	7.3%	5.8%	9.4%

Table 4.3. Geometric mean of prediction errors for the caches with SHiP replacement policy.

The best prediction when looking at the prediction errors, figure 4.6, or their geometric means, table 4.3, is clearly the LSTM prediction. All the means are closer to zero than any of the others, and the boxplots look better for all cache sizes. The geometric means

of the network are at least 2% better than the geometric means of the smallest of the other two networks. They also don't deviate more than 1% from the geometric means of prediction errors of LSTM for LRU replacement policy. This means that they have the same range as the ones predicted for LRU and show the stability of this network despite the change in replacement policy. The prediction errors from the CNN network for this replacement policy are very similar to the MLP network's prediction errors, which indicates that the CNN network shows some weakness for this replacement policy. This still raises the question, what makes the LSTM predict so much better than the other two in this replacement policy.

SRRIP replacement policy

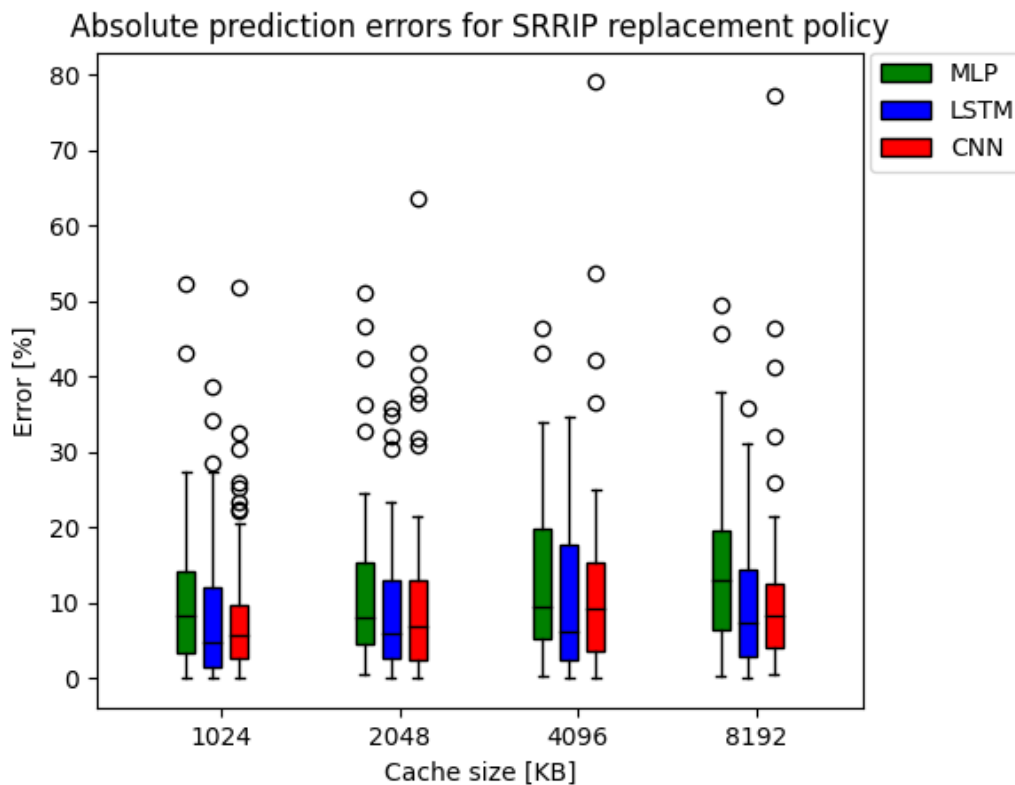


Figure 4.7. Prediction errors of the MLP, LSTM, CNN networks for LLCs with SRRIP replacement policy

Again, the predictions of the LSTM network seem to be the strongest ones. The prediction errors from the CNN network are better than the ones that we had for SHiP, looking similar to the LSTM network's prediction errors. They have smaller boxes in the boxplot, meaning that the prediction error is pretty stable. Still, the LSTM network's prediction errors are better regarding means and geometric means. What is also worth mentioning is that this is the lowest geometric mean of prediction errors that we have had for the 8MB-sized cache.

Network	Cache size [MB]			
	1	2	4	8
MLP	5.7%	8.0%	8.6%	9.3%
LSTM	3.1%	4.6%	5.5%	5.7%
CNN	4.7%	5.2%	6.4%	7.4%

Table 4.4. Geometric mean of prediction errors for the caches with SRRIP replacement policy.

Mockingjay replacement policy

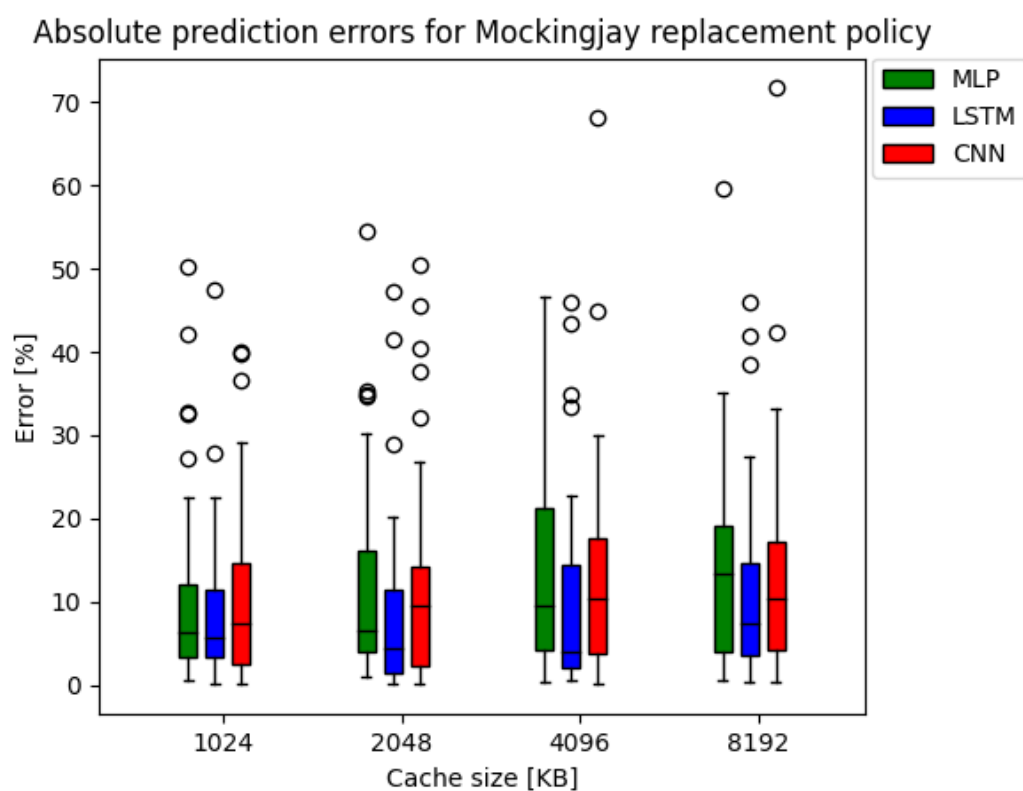


Figure 4.8. Prediction errors of the MLP, LSTM, CNN networks for LLCs with Mockingjay replacement policy

Lastly, there is the Mockingjay replacement policy. Just like before, the MLP network is clearly worse than the other two networks in predicting the caches with this replacement policy. The LSTM network outperforms the other networks significantly. Every box in the boxplot of prediction errors from the LSTM network is smaller, lower, and has a smaller mean than the others.

Mockingjay is a cache replacement policy that relies on reuse distances. It tracks them, calculates the likelihood of reuse, and then decides if an object will stay in the cache. This would lead us to the conclusion that a process such as a double LSTM layer helps the network understand reuse distances better and how the cache will behave with

Network	Cache size [MB]			
	1	2	4	8
MLP	6.2%	7.8%	9.0%	8.8%
LSTM	4.9%	3.6%	5.3%	6.2%
CNN	5.3%	6.2%	7.6%	8.3%

Table 4.5. Geometric mean of prediction errors for the caches with Mockingjay replacement policy.

this replacement policy. But the LSTM network doesn't predict Mockingjay significantly better than any of the other replacement policies. This leads us to believe that there is still room for improvement when it comes to this network.

4.1.4 Benchmarks

This section of the thesis aims to dissect and compare the outcomes of the aforementioned networks. We are going to compare and discuss the strengths and weaknesses of our networks, on individual benchmarks. Let's begin this result comparison by assessing the predictions of our networks for one specific benchmark.

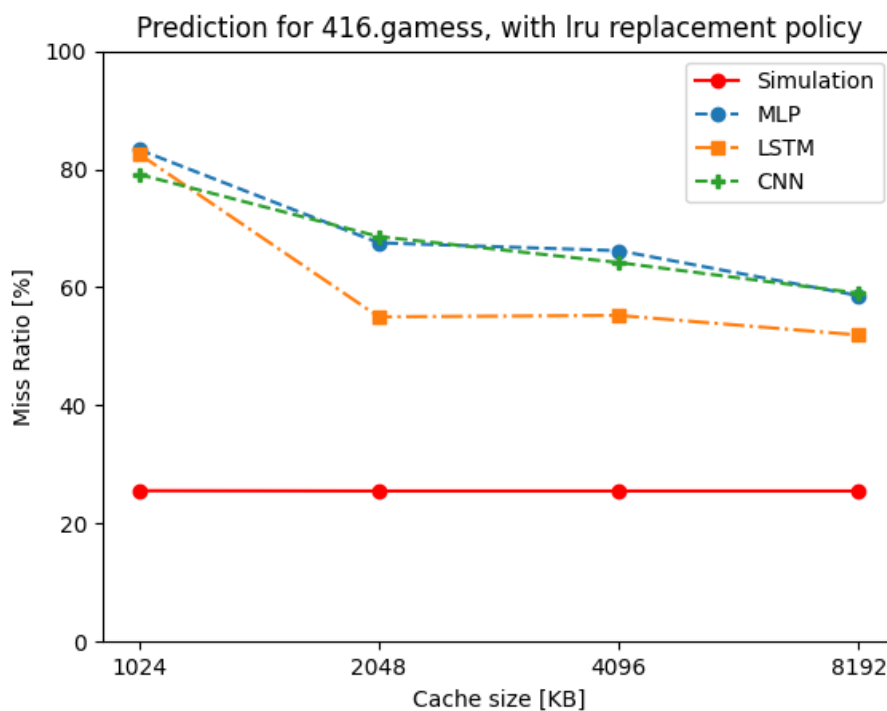


Figure 4.9. Network predictions for 416.gamess with LRU replacement policy.

Foremost, let's address the most miss-predicted benchmark in our dataset. The 416.gamess benchmark consists of one trace with 17k accesses within it. The majority of windows that our network is trained on consist of 512k accesses. When confronted with a window of this size, our network assumes that most of the accesses that happen

are cold misses, and as a result, they don't have reuse distances. Therefore, the prediction is much higher than the actual value. As we can see above, in figure 4.9, the three first networks predict a much higher miss rate than the actual. This type of benchmark can be safely ignored since, with a longer simulation, the result will be more accurate.

Moving onto some of the worst-predicted benchmarks by all of our networks. In figure 4.10 we are presenting three of the benchmarks for which we have the largest deviations between the predicted and the simulated value. These three benchmarks are 401.bzip2, 434.zeusmp, and 456.hmmmer. All of them are highly intensive programs with millions of accesses each. We show the outputs of our networks for the four replacement policies. For LRU, since we depict separately the predictions of StatCache, MLP, CNN, and LSTM from the DNN networks to preserve clarity. The DNN predictions (plots d, e, and f) are labeled as DNN-i, where i is the number of training cache sizes of the network. We also show the predictions from the DNN-1 and DNN-2 networks, to show that their predictions indeed are unreliable. It is clear to see that all the DNN-1 networks are random, and that is because the network doesn't understand the size parameter, since it was trained on only one size parameter.

The 434.zeusmp is a program that simulates astrophysical phenomena based on the ZEUS-MP computational fluid dynamics code. It is clear that the prediction from StatCache for this benchmark is the only one that is lower than the actual simulated miss ratios. All the predictions from the networks that we proposed seem to agree that the miss ratio should be significantly higher than the simulated one. Even though this holds true for most networks, LSTM seems to have made a valiant attempt at closely predicting the miss ratio for the Mockingjay replacement policy (plot m).

401.bzip2 is a compression algorithm that compresses a range of files throughout its execution. StatCache's prediction for this benchmark is very accurate, beating our networks by quite a bit. The misprediction of our networks for this benchmark is not as bad as for the other two benchmarks. The prediction of miss ratios is always accurate for the smallest cache size, and then as the cache sizes get larger, it deviates.

Lastly, the 456.hmmmer file is a computation of profile Hidden Markov Models to do sensitive database searching using statistical descriptions of a sequence family's consensus. Again, the prediction of Statcache is far better for this benchmark. The steep drop-off in miss ratio for large cache sizes seems to confuse our networks, since they always predict higher for the larger cache sizes, even though for the smallest ones they get it correct. It is interesting how the DNN network (plot f) creates seemingly the same prediction shape when it is trained with 1, 2, or 3 cache sizes, upwards of the fourth. On the other hand, it seems to understand the shape of the benchmarks better and move to overcome it. The LSTM also seems to have the correct shape when compared to the other two networks, but it doesn't quite seem to understand the steepness of the curve.

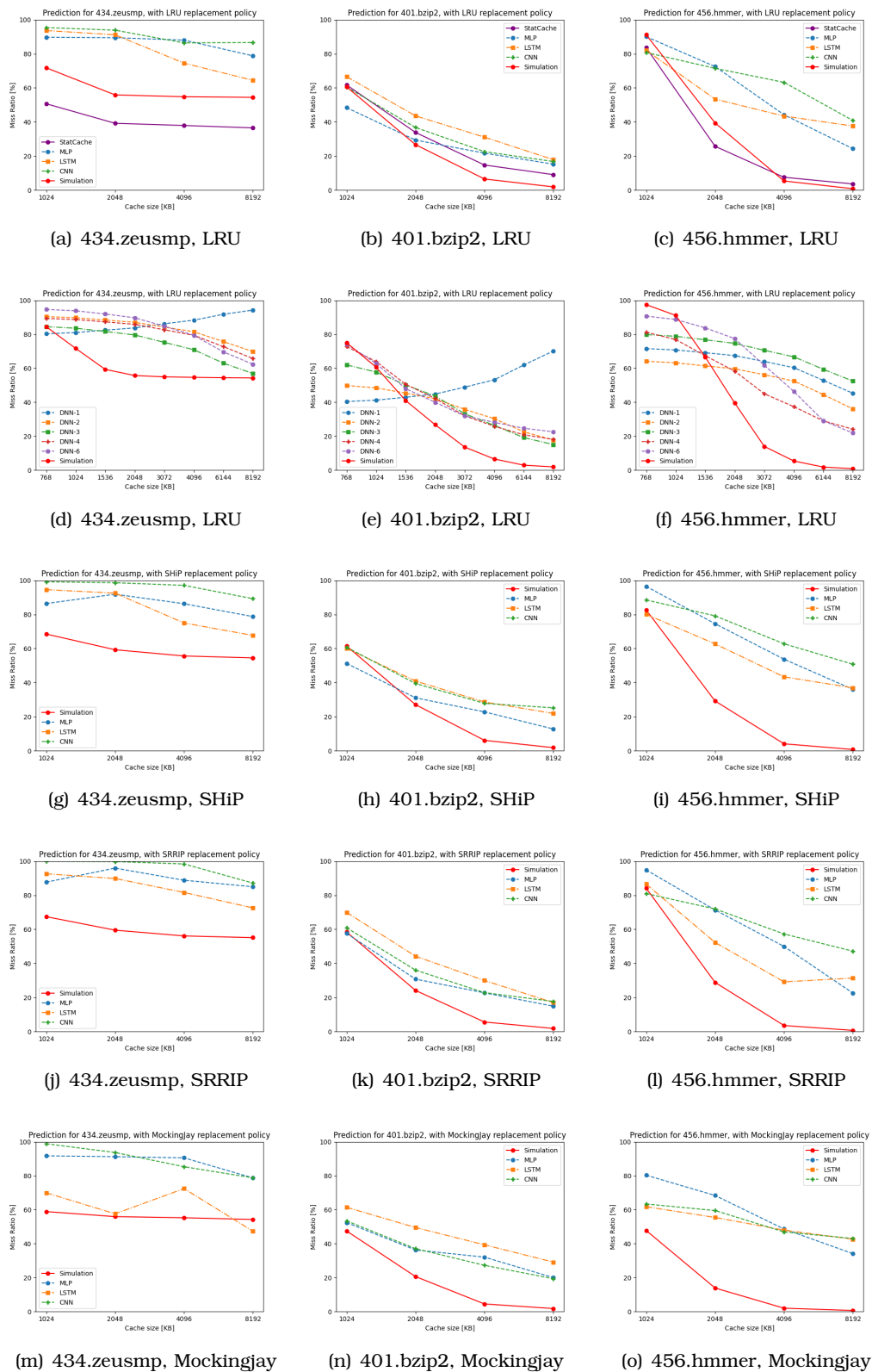


Figure 4.10. Some of the worst benchmark to predictions.

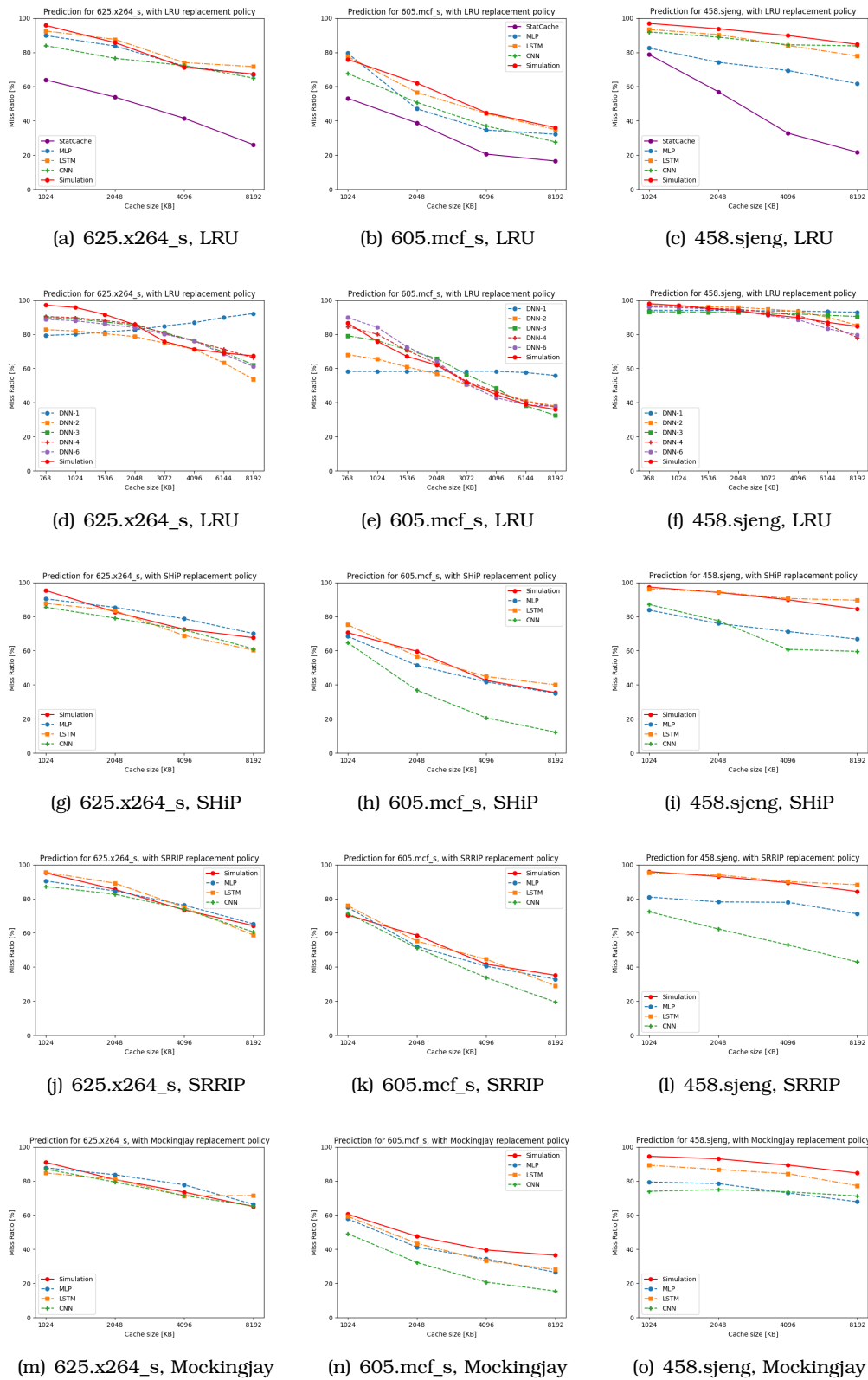


Figure 4.11. Some of the best benchmark predictions.

Some of the best predicted files are shown in figure 4.11. This time, the predictions from the networks are very close to the simulation’s cache miss ratios. The three benchmarks that we show have some of the best predictions from our networks. These are

obviously not all the benchmarks with good predictions, just some that are indicative. It is interesting how, for almost all of these predictions, the best prediction is the one from the LSTM network. It understands these problems almost perfectly, having minimal deviations from the real values. Furthermore, it really shows how a better understanding of the problem applies to the prediction.

The CNN and MLP overall seem to have small deviations from the simulation's values, larger than the ones of the LSTM network. The DNN network seems to understand the problem as well, and with more data, the predictions get better as well. Again, we have the untrained DNN-1 plot that visibly doesn't understand the size parameter.

Lastly, StatCache is outperformed significantly on those predictions. The predictions of StatCache are not bad for all of those benchmarks. They may be bad for the 458.sjeng benchmark, but for the other ones, the predictions from our networks are far more accurate.

4.1.5 Conclusion

In conclusion, the LSTM network consistently predicts the most precise results of the networks we proposed. As we can see in figure 4.12 its absolute prediction errors are just as good as for any replacement policy. Its results showed that it can predict any cache size that it is trained on consistently with very low geometric means of prediction errors. It also showed that it is very consistent in its predictions, as the geometric means of prediction errors in table 4.6 don't vary too much on different replacement policies.

Replacement policy	1MB	2MB	4MB	8MB
LRU	3.6%	3.9%	5.3%	6.4%
SHiP	2.7%	4.1%	3.7%	7.6%
SRRIP	3.1%	4.6%	5.5%	5.7%
Mockingjay	4.9%	3.6%	5.3%	6.2%

Table 4.6. Geometric mean of LSTM network's absolute prediction errors

A simpler and less costly alternative to it is the CNN network, which has the capability to predict very accurately and is much faster to train, with about 20% of the training time of the LSTM. But it is also inconsistent for some replacement policies, with similar predictions to the MLP, which is not optimal. The DNN network shows the most potential, since it has the best predictions for lower cache sizes within the LRU replacement policy, but still has room for improvement on the larger caches. Overall, these results emphasize the different intricacies of the problem and show that it is indeed possible to predict it accurately.

4.2 Predicting the miss ratio of a known application

Lastly, there is another usage that we can have with this network. Suppose we have a few machines in place that have a specific architecture. We want to predict how a

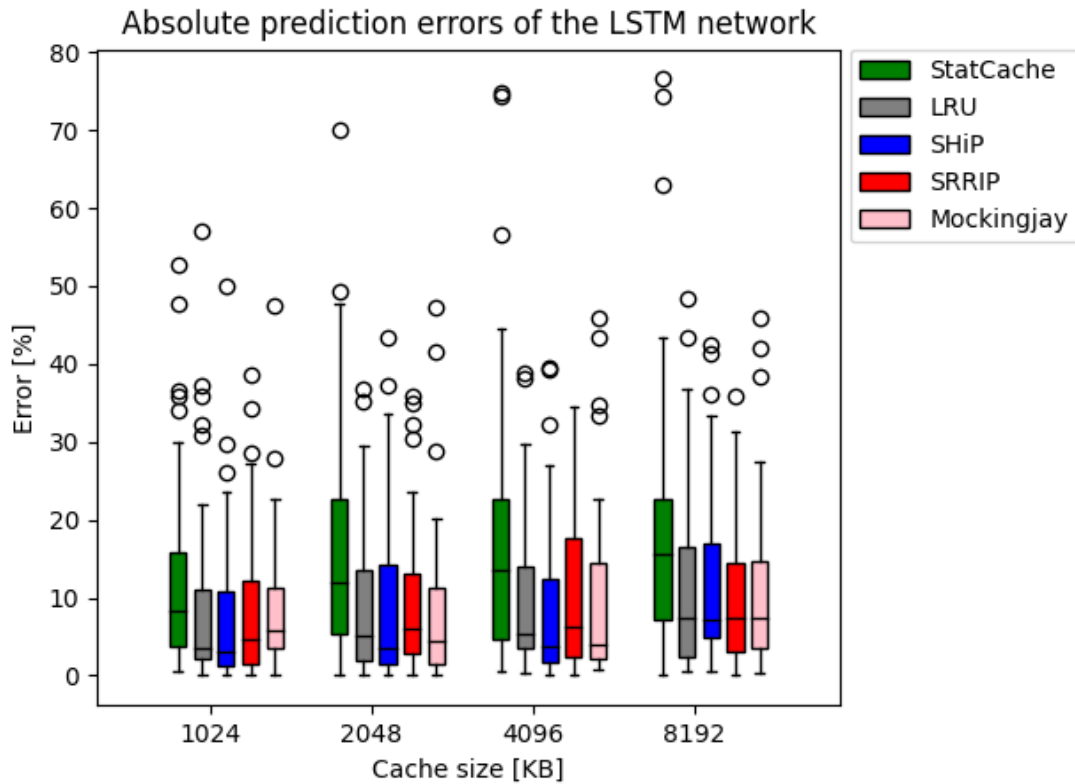


Figure 4.12. LSTM Network’s absolute prediction errors

program that has been executed on these machines will behave when being executed by machines with the same replacement policy and other LLC sizes. To model this use case, we will use the simulations of the benchmark suite for some cache sizes and move the others to the test set. This way, it will be like having machines with the LLC cache sizes of the training set and LRU replacement policy and predicting the miss ratios for the cache sizes within the test set.

For this problem we will be keeping all the benchmarks for some cache sizes within the training set. Then, the network will be trained on these cache sizes and try to predict the miss ratios of all other cache sizes.

Firstly, we moved the data for one cache size from the train to the test set to see if and how well the network can predict it based on the rest of the benchmarks. More precisely, the test set consists of the column for cache size 3MB and the train set of the other 7 cache sizes [768, 1024, 1536, 2048, 4096, 6044, 8192] KB.

The predictions are way better than any other network could make. The geometric mean of the above errors is 0.9%, as shown in the last row of table 4.7. The geometric mean of LSTM’s absolute prediction errors previously predicted for LRU are 3.9% and 5.3% for the two neighboring cache sizes of 2MB and 4MB, respectively. This means that this prediction is undeniably far better than the best prediction we had until now. It was expected that this result would be good, since the network has a very large dataset to train on and can understand the problem very well.

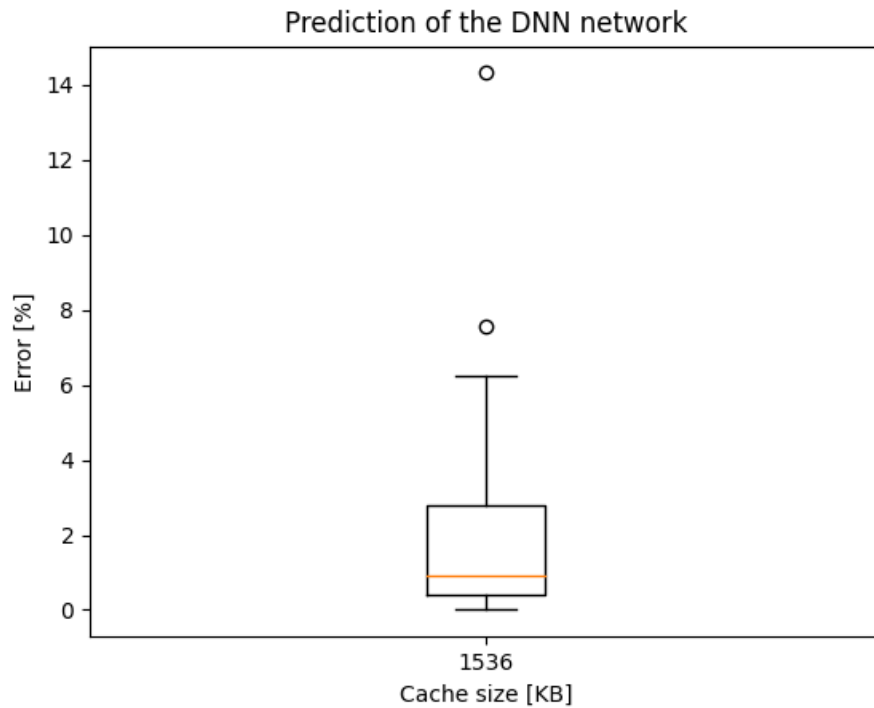


Figure 4.13. DNN absolute prediction errors for 1 cache size, trained with 7 cache sizes' miss ratios

This result is a good indicator that our network is suitable to predict this kind of problem. It also begged the question of how few cache sizes are needed in the training set to have an accurate prediction, since simulating seven cache sizes to predict the eighth is not a realistic problem.

Train-set cache sizes [MB]	Cache sizes [KB]							
	768	1024	1536	2048	3072	4096	6144	8192
1, 2, 4, 8	1.4%	-	0.8%	-	0.9%	-	1.1%	-
0.75, 1, 2, 4, 6, 8	-	-	0.8%	-	0.8%	-	-	-
0.75,1,1.5,2,4,6,8	-	-	-	-	0.9%	-	-	-
LSTM Network	-	3.6%	-	3.9%	-	5.3%	-	6.4%

Table 4.7. Geometric mean of DNN network’s absolute prediction errors.

The immediate next step towards answering this question is to remove another cache size and observe how the predictions’ error behaves.

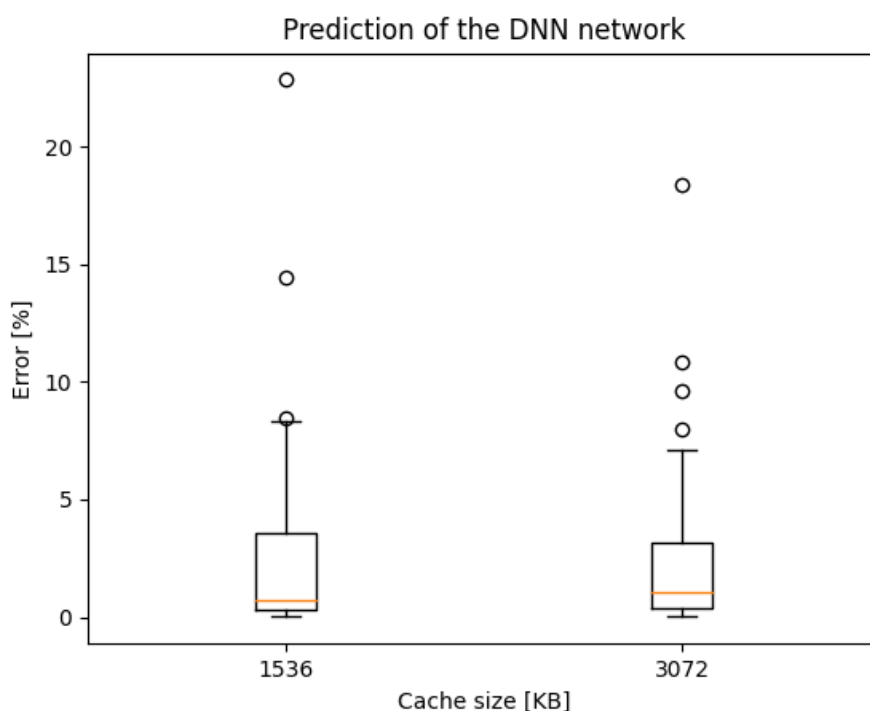


Figure 4.14. DNN absolute prediction errors for 2 cache sizes, trained with 6 cache sizes’ miss ratios

The geometric mean for the two columns of sizes 1.5MB and 3MB is 0.8% and 0.8%, respectively. We can see that these geometric means are just as good as the ones we observed before.

It is clearly visible that, for each cache size, there is one by far worst predicted benchmark. This is the prediction for the benchmark 416.gamess; its miss ratio is always predicted to be higher, and it will be miss-predicted in the future as well, for good reason. This benchmark consists of one trace, which only has 17k accesses to the LLC, which means that the network reads one window of reuse distances with 17k accesses and tries to predict the miss ratio of it. The majority of reused distance windows that our network is trained on consist of 512k accesses. Thus, the network sees the low number of reuse distances and assumes lots of accesses that only happen once and therefore don’t

have reuse distances. These result in misses (cold misses), and therefore the network computes a higher miss ratio. We can safely ignore this output since, with a 30x longer simulation, the output would most likely get closer to the real value.

Lastly, we keep the four most likely LLC caches that a real architecture will have and try to predict the other four values. Namely, we keep the LLC caches of 1024, 2048, 4096, and 8192 KB in the training set and try to predict the behavior of the benchmarks that were simulated with 768, 1536, 3152, and 6044 KB sized LLCs. With this input, we compute similar results as previously. The scale of the geometric means in the third row of table 4.7 is similar to the one for the six cache sizes in the training set.

We can observe in 4.15 that the outliers' errors still persist. It is also worth mentioning that the prediction for 768 KB LLC is clearly, but not a lot worse than the other three predictions. This is a result of it not being in between the cache sizes that have been seen by the network. For example, the network knows the output for 1MB and 2MB sized caches, and therefore it can better understand how a machine would behave for a 1536 KB sized cache. For the 768KB, it is more difficult to extrapolate the behavior of the program, and therefore the errors rise. This error is still much lower than the best error we had by predicting the 1MB-sized cache previously.

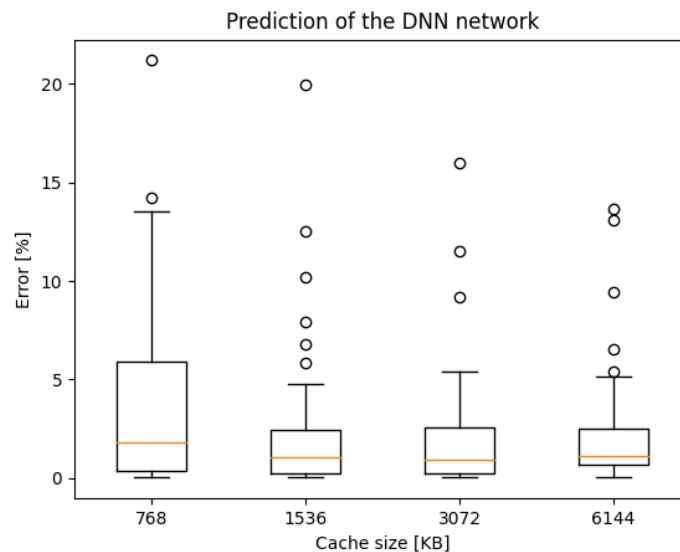


Figure 4.15. DNN absolute prediction errors for 4 cache sizes, trained with 4 cache sizes' miss ratios

Again, having less cache sizes in the training set doesn't make sense, therefore we won't show them.

Overall, it is clear that with this usage, the network predicts the problem with utmost precision. The predictions are undeniably better than any other predictions made before without knowing the problem. The downside of this network is that it needs to be executed and trained on all of these different cache architectures. So it is application-specific if this is worth it.

Chapter **5**

Conclusion

5.1 Concluding thoughts

In this thesis, we introduce a number of machine learning networks that offer solutions for the problem of predicting miss ratios on different cache architectures. Knowing the cache miss ratios of a program on different architectures is important since it lets the user know what is the best machine to execute said program on. Simulating a process is a difficult and time-consuming process, and skipping it through a prediction is a very efficient solution to this problem.

We introduce four machine learning networks that understand the problem in their own unique way. Inspired by StatCache's already-existing probabilistic approach to this question, we decided to use the same data that it uses for its predictions to train and evaluate a machine learning network. First, a simple MLP network shows that it is quite simple to predict the network as well as the StatCache probabilistic method. This network is based on the reuse distances of one execution of the program and applies to any cache size and any replacement policy. It predicts the miss ratios of any program with any replacement policy with a geometric mean similar to the one that StatCache introduces.

The path to understanding the input data as accurately as possible led us to introduce two new networks. Together with the NLP embeddings of the programs as input, these try to understand the reuse distances differently and thus compute more accurate predictions of the problem. The first one is the LSTM network, which, as the name suggests, understands reuse distances via an LSTM layer. This network is the most time-consuming to train, but it produces the best results of all our networks by far. It produces stable, accurate predictions of miss ratios for any replacement policy and cache size that it is trained on. The CNN network takes the reuse distances as an input to a CNN layer that has the purpose of understanding the reuse distances via a convolution of the values within proximity. This network is significantly less costly to train but has the downside of not being as accurate as the LSTM network. Its predictions are mostly slightly less accurate than the LSTM network's predictions, but for some replacement policies, such as the SHiP, they are even worse, similar to the MLP network's predictions. These two networks are the essential steps that this thesis takes towards understanding this problem accurately.

Lastly, we introduce a deep neural network that works for any cache size. This one,

just like the previous ones, uses reuse distances and the embeddings of the code to understand the problem. The difference is that this one uses the cache size as input and, as we have shown, is able to understand the problem for any cache size. This network that we propose is very strong for smaller cache sizes, but has a slight weakness for larger ones. This network also produces highly accurate results if it already knows the real value of a problem for some cache sizes, and then it can predict any cache size's miss ratio with unseen accuracy. This network can be a stepping stone for further research in the direction of similar architectures that understand the problem better.

5.2 Discussion

The preceding result section, introduced a comprehensive analysis of the empirical data gathered in this study. This discussion section aims to get a deeper understanding of the significance of these results in connection with the broader context of machine architectures. By examining these findings through various lenses, this section seeks to elucidate their meaning, significance, and potential impacts within the realm of computer architectures. Furthermore, this discussion will critically analyze the implications of these results in relation to existing literature, aiming to contribute new perspectives and avenues for further exploration within the field.

The models that we propose in this thesis challenge the outputs of previous prediction mechanisms that have been the best estimations of this problem. They prove to be better than the predictions of StatCache by a significant margin, and also predict the output of any replacement policy. This means that the prediction of cache misses on a machine can now be described as a matter of training a machine learning model on a machine's outputs and letting it predict the miss ratios. The introduction of such a network would not only simplify the process of deciding what is the best architecture for a problem; it would also skip lots of hours of processing or simulation time.

To further contextualize these findings, it's imperative to consider the broader factors that might influence our networks. The introduction of custom-made traces for the benchmarks, and together with that, the embeddings of the specific part, could influence our results a lot. A network's prediction, that knows what part of the program is executing in each trace, can improve the predictions from these networks significantly. Additionally, avenues for further exploration could be made towards data that results from parallel processing or data from real machines. Predicting the miss ratio of processes executing on multiple cores within a machine will introduce much higher difficulty to the problem and should be a subject for further exploration. The data that we used for this thesis is the data produced by a simulation program. It would be interesting to see how these results compare with the networks trained on a real machine, since each has its own difficulties.

Additionally, there is potential for improving our networks. As discussed previously, the LSTM and CNN networks show potential to have even better results. An introduction of more benchmarks to the dataset or a significant change in their architecture could improve their predictions even more. The DNN network is a network that hasn't been explored to the same depth as the other networks by us. Different network architectures

and implementations could reduce its errors significantly. It could even be expanded to have an additional input, the cache miss ratio of one execution on a cache, and then make estimations for other cache sizes.

In summary, the in-depth analysis and interpretation of the findings presented in this discussion underscore the complexities inherent in predicting cache misses. Ultimately, this study stands as a stepping stone in the continual quest to unravel the complexities of computer architecture, encouraging further scholarly inquiry to advance our understanding of this field.

Bibliography

- [1] E. Berg και E. Hagersten. *StatCache: a probabilistic approach to efficient and accurate data locality analysis*. *IEEE International Symposium on - ISPASS Performance Analysis of Systems and Software*, 2004.
- [2] Ian Goodfellow, Yoshua Bengio και Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [3] N. Beckmann και D. Sanchez. *Modeling cache performance beyond LRU*. *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016.
- [4] X. Liu Q. Wang και M. Chabby. *Featherlight Reuse-distance Measurement*. *IEEE International Symposium on High Performance Computer Architecture*, 2019.
- [5] Maurice V. Wilkes. *Slave Memories and Dynamic Storage Allocation*. *IEEE Trans. Electron. Comput.*, 14(2):270–271, 1965.
- [6] Carole Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely και Joel Emer. *SHiP: Signature-based Hit Predictor for high performance caching*. *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, σελίδες 430–441, 2011.
- [7] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely και Joel Emer. *High performance cache replacement using re-reference interval prediction (RRIP)*. *ISCA '10*, σελίδα 60–71, New York, NY, USA, 2010. Association for Computing Machinery.
- [8] Ishan Shah, Akanksha Jain και Calvin Lin. *Effective Mimicry of Belady's MIN Policy*. *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, σελίδες 558–572, 2022.
- [9] Alex Krizhevsky, Ilya Sutskever και Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. *Advances in Neural Information Processing Systems* F. Pereira, C.J. Burges, L. Bottou και K.Q. Weinberger, επιμελητές, τόμος 25. Curran Associates, Inc., 2012.
- [10] Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun. *Deep Residual Learning for Image Recognition*. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, σελίδες 770–778, 2016.
- [11] Miltiadis Allamanis, Earl T. Barr, Christian Bird και Charles Sutton. *Learning Natural Coding Conventions*. *FSE 2014*, New York, NY, USA, 2014. Association for Computing Machinery.

- [12] Miltiadis Allamanis και Charles Sutton. *Mining Idioms from Source Code*. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014*, New York, NY, USA, 2014. Association for Computing Machinery.
- [13] Amy McGovern. *Building a Basic Block Instruction Scheduler with Reinforcement Learning and Rollouts*. σελίδες 141–160, 2002.
- [14] C. Cummins, P. Petoumenos, Z. Wang και H. Leather. *End-to-End Deep Learning of Optimization Heuristics*. *26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017.
- [15] Nan Wu και Yuan Xie. *A Survey of Machine Learning for Computer Architecture and Systems*. *ACM Comput. Surv.*, 55(3), 2022.
- [16] Xiangyu Dong, Norman P. Jouppi και Yuan Xie. *A circuit-architecture co-optimization framework for exploring nonvolatile memory hierarchies*. *ACM Trans. Archit. Code Optim.*, 10(4), 2013.
- [17] Ramazan Bitirgen, Engin Ipek και Jose F. Martinez. *Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach*. *2008 41st IEEE/ACM International Symposium on Microarchitecture*, σελίδες 318–329, 2008.
- [18] Daniel Nemirovsky, Tugberk Arkose, Nikola Markovic, Mario Nemirovsky, Osman Unsal και Adrian Cristal. *A Machine Learning Approach for Performance Prediction and Scheduling on Heterogeneous CPUs*. σελίδες 121–128, 2017.
- [19] Rosenblatt F. *The perceptron: a probabilistic model for information storage and organization in the brain*. *CCM Information Corporation. First working Deep Learners with many layers, learning internal representations*.
- [20] A. G. Ivakhnenko και V. G. Lapa. *Cybernetic Predicting Devices*. *CCM Information Corporation. First working Deep Learners with many layers, learning internal representations*.
- [21] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke και Jürgen Schmidhuber. *A Novel Connectionist System for Unconstrained Handwriting Recognition*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.
- [22] Hasim Sak, Andrew W. Senior και Françoise Beaufays. *Long short-term memory recurrent neural network architectures for large scale acoustic modeling*. *INTERSPEECH*, σελίδες 338–342, 2014.
- [23] Qianji Zhao και Zequn Shang. *Deep learning and Its Development*. *Journal of Physics: Conference Series*, 1948(1):012023, 2021.

- [24] Sunitha Basodi, Chunyan Ji, Haiping Zhang και Yi Pan. *Gradient amplification: An efficient way to train deep neural networks*. *Big Data Mining and Analytics*, 3(3):196–207, 2020.
- [25] Joshua Peeples, Weihuang Xu και Alina Zare. *Histogram Layers for Texture Analysis*. *IEEE Transactions on Artificial Intelligence*, 3(4):541–552, 2022.
- [26] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev και N.I. Chervyakov. *Application of the residue number system to reduce hardware costs of the convolutional neural network implementation*. *Mathematics and Computers in Simulation*, 177:232–243, 2020.
- [27] Hamid Sadeghi και Abolghasem A. Raie. *HistNet: Histogram-based convolutional neural network with Chi-squared deep metric learning for facial expression recognition*. *Information Sciences*, 608:472–488, 2022.
- [28] Ronan Collobert και Jason Weston. *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*. New York, NY, USA, 2008. Association for Computing Machinery.
- [29] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella και Jürgen Schmidhuber. *Flexible, High Performance Convolutional Neural Networks for Image Classification*. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, σελίδα 1237–1242. AAAI Press, 2011.
- [30] Dominik Scherer, Andreas Müller και Sven Behnke. *Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition*. σελίδες 92–101, 2010.
- [31] Y Lan Boureau, J. Ponce και Yann Lecun. *A Theoretical Analysis of Feature Pooling in Visual Recognition*. σελίδες 111–118, 2010.
- [32] S. VenkataKeerthy, Rohit Aggarwal, Shalini Jain, Maunendra Sankar Desarkar, Ramakrishna Upadrasta και Y. N. Srikant. *IR2VEC: LLVM IR Based Scalable Program Embeddings*. *ACM Trans. Archit. Code Optim.*, 17(4), 2020.
- [33] Yulei Sui, Xiao Cheng, Guanqin Zhang και Haoyu Wang. *Flow2Vec: Value-Flow-Based Precise Code Embedding*. *Proc. ACM Program. Lang.*, 4(OOPSLA), 2020.
- [34] John L. Henning. *SPEC CPU2006 Benchmark Descriptions*. *SIGARCH Comput. Archit. News*, 34(4):1–17, 2006.
- [35] James Bucek, Klaus Dieter Lange και Jóakimv. Kistowski. *SPEC CPU2017: Next-Generation Compute Benchmark*. *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, σελίδα 41–42, New York, NY, USA, 2018. Association for Computing Machinery.

- [36] Nathan Gober, Gino Chacon, Lei Wang, Paul V. Gratz, Daniel A. Jimenez, Elvira Teran, Seth Pugsley και Jinchun Kim. *The Championship Simulator: Architectural Simulation for Education and Competition*, 2022.
- [37] Geoffrey Webb, Claude Sammut, Claudia Perlich, Tamás Horváth, Stefan Wrobel, Kevin Korb, William Noble, Christina Leslie, Michail Lagoudakis, Novi Quadrianto, Wray Buntine, Lise Getoor, Galileo Namata, Jiawei Jin, Jo Anne Ting, Sethu Vijayakumar, Stefan Schaal και Luc De Raedt. *Leave-One-Out Cross-Validation*. 2010.
- [38] Diederik P. Kingma και Jimmy Ba. *Adam: A Method for Stochastic Optimization*. *CoRR*, abs/1412.6980, 2014.
- [39] Cem Subakan, Mirco Ravanelli, Samuele Cornell, Mirko Bronzi και Jianyuan Zhong. *Attention Is All You Need In Speech Separation*. *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, σελίδες 21-25, 2021.

List of Abbreviations

MSE	Mean Squared Error
IR	Intermediate representation
NLP	Natural Language Processing
LRU	Least Recently Used
MLP	Multi-Layered Perceptron
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
DNN	Deep Neural Network
RL	Reinforcement Learning



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Εκτίμηση αστοχιών κρυφής μνήμης

*Πρόβλεψη αστοχιών κρυφής μνήμης σε διάφορες
αρχιτεκτονικές κρυφών μνημών*

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΡΑΜΟΥ Γ. ΜΑΡΚΟΥ



Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π

Αθήνα, Απρίλιος 2024



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Εκτίμηση αστοχιών κρυφής μνήμης

Πρόβλεψη αστοχιών κρυφής μνήμης σε διάφορες
αρχιτεκτονικές κρυφών μνημών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΡΑΜΟΥ Γ. ΜΑΡΚΟΥ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2 Απριλίου 2024.

(Υπογραφή)

(Υπογραφή)

(Υπογραφή)

.....
Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π

.....
Διονύσιος Πνευματικάτος
Καθηγητής Ε.Μ.Π

.....
Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής Ε.Μ.Π

Αθήνα, Απρίλιος 2024



Copyright © – All rights reserved. Με την επιφύλαξη παντός δικαιώματος.
Μάρκος Γεώργιος Ραμός, 2024.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα.

Το περιεχόμενο αυτής της εργασίας δεν απηχεί απαραίτητα τις απόψεις του Τμήματος, του Επιβλέποντα, ή της επιτροπής που την ενέκρινε.

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Μάρκος Γεώργιος Ραμός

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

2 Απριλίου 2024

Περίληψη

Δεδομένης της αυξανόμενης διαφοράς στην ταχύτητα επεξεργασίας μεταξύ της κύριας μνήμης και της ΚΜΕ, ο ρόλος των κρυφών μνημών της ΚΜΕ στην επίτευξη της μέγιστης δυνατής ταχύτητας επεξεργασίας είναι τόσο σημαντικός όσο ποτέ άλλοτε. Κάθε πρόσβαση στην κρυφή μνήμη που δεν βρίσκει τα δεδομένα πρέπει να καλεί την κύρια μνήμη, χρησιμοποιώντας δεκάδες, αν όχι εκατοντάδες, κύκλους της μηχανής. Η πρόβλεψη της συμπεριφοράς μιας διεργασίας πριν από την εκτέλεσή της σε διαφορετικές αρχιτεκτονικές κρυφής μνήμης είναι ένα σημαντικό έργο. Θα βοηθήσει να καθοριστεί ποιος επεξεργαστής θα λειτουργήσει καλύτερα για ένα πρόγραμμα ή πώς να κατανεμηθεί βέλτιστα η υπολογιστική ισχύς. Ο άμεσος στόχος αυτής της διατριβής είναι να προτείνει ένα μοντέλο μηχανικής μάθησης που θα προβλέπει με όσο το δυνατόν μεγαλύτερη ακρίβεια τις αστοχίες στην κρυφή μνήμη ενός προγράμματος, ανάλογα αποκλειστικά με την αρχιτεκτονική της κρυφής μνήμης, τον κώδικα του προγράμματος και τα ιστογράμματα των αποστάσεων επαναχρησιμοποίησης. Για την επίτευξη αυτού του στόχου, προτείνουμε έναν αριθμό μοντέλων μετασχηματισμών που συνδυάζουν τον μετασχηματισμένο κώδικα εισόδου με πληροφορίες μέσα στα ιστογράμματα αποστάσεων επαναχρησιμοποίησης και εκτιμούν το ποσοστό αστοχιών κρυφής μνήμης. Εκπαιδεύσαμε και δοκιμάσαμε αυτά τα μοντέλα με προσομοιωμένα δεδομένα και παρήγαμε προβλέψεις υψηλής ακρίβειας για μεγάλο αριθμό πολιτικών αντικατάστασης και μεγεθών κρυφής μνήμης LLC. Αυτά τα δίκτυα μπορούν να λειτουργήσουν ως ένα χρήσιμο εργαλείο στην πρόβλεψη της κρυφής μνήμης και μπορούν να χρησιμοποιηθούν σε πραγματικές μηχανές σε μελλοντική έρευνα.

Λέξεις κλειδιά: Αρχιτεκτονική υπολογιστών, Αστοχίες κρυφής μνήμης, Μηχανική μάθηση, Νευρωνικά δίκτυα

Abstract

Given the increasing difference in processing speed between the main memory and the CPU, the role of CPU caches in achieving the maximum possible processing speed is as big as ever. Every cache access that doesn't find the data has to invoke the main memory, using tens, if not hundreds, of machine cycles. Predicting the behavior of a process prior to execution on different cache architectures is an important task. It will help determine what processor will work best for a program or how to distribute computing power optimally. The immediate goal of this thesis is to propose a machine learning model that will predict as accurately as possible the cache misses of a program depending solely on the cache architecture, the program code, and the reuse-distance histograms. To achieve this goal, we propose a number of transformer models that combine the transformed input code with information within the reuse-distance histograms and lead to an output cache miss ratio. We trained and tested these models with simulated data and produced high-accuracy predictions for a large number of replacement policies and LLC cache sizes. These networks can act as a useful tool in cache prediction and may be used on real machines in future research.

Key words: Computer architecture, cache misses, machine learning, neural networks

Περιεχόμενα

Περίληψη	1
Abstract	3
1 Εισαγωγή	13
I Θεωρητικό Μέρος	17
2 Θεωρητικό υπόβαθρο	19
2.1 Ιεραρχία κρυφών μνημών	19
2.2 StatCache	20
2.3 Άλλες χρήσεις Νευρωνικών Δικτύων	21
2.4 Ενσωματώσεις	21
II Πρακτικό Μέρος	23
3 Προσέγγιση και μέθοδοι	25
3.1 Επισκόπηση	25
3.2 Πειραματική οργάνωση	26
3.2.1 Προσομοίωση	26
3.2.2 Σύνολο δεδομένων	27
3.2.3 Μετρικές	30
3.3 Υπολογισμός StatCache	31
3.4 Ρηχό πολυεπίπεδο perceptron	32
3.5 Δίκτυο LSTM	34
3.6 Συνελκτικό νευρωνικό δίκτυο	37
3.7 Βαθύ νευρωνικό δίκτυο	38
4 Σύγκριση αποτελεσμάτων	41
4.1 Πρόβλεψη για άγνωστα προβλήματα	41
4.1.1 Πρόβλεψη για την πολιτική αντικατάστασης LRU	41
4.1.2 Πρόβλεψη οποιουδήποτε μεγέθους κρυφής μνήμης για την πολιτική αντικατάστασης LRU	43
4.1.3 Πρόβλεψη πολιτικής αντικατάστασης SHiP	49
4.1.4 Πρόβλεψη πολιτικής αντικατάστασης SRRIP	50

4.1.5 Πρόβλεψη πολιτικής αντικατάστασης Mockingjay	52
4.1.6 Ανάλυση συγκεκριμένων εφαρμογών	53
4.1.7 Συμπεράσματα	57
4.2 Πρόβλεψη γνωστού προγράμματος	58
5 Κατακλείδα	63
5.1 Συμπεράσματα	63
5.2 Συζήτηση	64
III Επίλογος	67
Παραρτήματα	69
Βιβλιογραφία	73
Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια	75
Απόδοση ξενόγλωσσων όρων	77

Κατάλογος Σχημάτων

3.1	Μέσος όρος λόγω αστοχίας όλων των αρχείων αναφοράς με πολιτική αντικατάστασης LRU	28
3.2	Τα απόλυτα σφάλματα πρόβλεψης της StatCache για την LRU	32
3.3	Τα απόλυτα σφάλματα πρόβλεψης της StatCache.	33
3.4	Δομή του δικτύου LSTM. input_1 είναι οι ενσωματώσεις, input_2 το ιστόγραμμα αποστάσεων επαναχρησιμοποίησης και input_3 το ιστόγραμμα αποστάσεων επαναχρησιμοποίησης με αντίστροφη σειρά (από μεγαλύτερη τιμή προς την μικρότερη).	36
3.5	Δομή του συνελκτικού νευρωνικού δικτύου. input_1 είναι οι ενσωματώσεις και input_2 είναι οι αποστάσεις επαναχρησιμοποίησης.	37
3.6	Βαθύ νευρωνικό δίκτυο για την πρόβλεψη άλλων μεγεθών κρυφής μνήμης. input_4 είναι το μέγεθος της κρυφής μνήμης, input_5 είναι οι αποστάσεις επαναχρησιμοποίησης και input_6 είναι οι ενσωματώσεις.	39
4.1	Σφάλματα πρόβλεψης των δικτύων MLP, LSTM, CNN, DNN και StatCache για LLC με πολιτική αντικατάστασης LRU	42
4.2	StatCache, LSTM, CNN, and DNN absolute prediction errors for LLC of size 4MB with LRU replacement policy	44
4.3	Απόλυτα σφάλματα πρόβλεψης του δικτύου DNN, εκπαιδευμένα σε μεγέθη κρυφής μνήμης 1MB, 2MB και 8MB.	45
4.4	Απόλυτα σφάλματα πρόβλεψης του δικτύου DNN, εκπαιδευμένα σε μεγέθη κρυφής μνήμης 1MB, 2MB, 4MB και 8MB.	47
4.5	Απόλυτα σφάλματα πρόβλεψης του δικτύου DNN, εκπαιδευμένα σε μεγέθη κρυφής μνήμης 0.75, 1, 2, 4, 6 και 8 MB.	48
4.6	Σφάλματα πρόβλεψης των δικτύων MLP, LSTM, CNN για LLC με πολιτική αντικατάστασης SHiP.	49
4.7	Σφάλματα πρόβλεψης των δικτύων MLP, LSTM, CNN για LLC με πολιτική αντικατάστασης SRRIP.	50
4.8	Σφάλματα πρόβλεψης των δικτύων MLP, LSTM, CNN για LLC με πολιτική αντικατάστασης Mockingjay.	52
4.9	Προβλέψεις δικτύου για 416.gamess με πολιτική αντικατάστασης LRU.	53
4.10	Μερικές από τις χειρότερες προβλέψεις αρχείων αναφοράς.	55
4.11	Μερικές από τις καλύτερες προβλέψεις αρχείων αναφοράς.	56
4.12	Απόλυτα σφάλματα πρόβλεψης του δικτύου LSTM	58

4.13	Απόλυτα σφάλματα πρόβλεψης του DNN για 1 μέγεθος κρυφής μνήμης, εκπαιδευμένο με ποσοστά αστοχίας 7 μεγεθών κρυφής μνήμης	59
4.14	Απόλυτα σφάλματα πρόβλεψης του DNN για 2 μεγέθη κρυφής μνήμης, εκπαιδευμένο με ποσοστά αστοχίας 6 μεγεθών κρυφής μνήμης	60
4.15	Απόλυτα σφάλματα πρόβλεψης του DNN για 4 μεγέθη κρυφής μνήμης, εκπαιδευμένο με ποσοστά αστοχίας 4 μεγεθών κρυφής μνήμης	61

Κατάλογος Εικόνων

Κατάλογος Πινάκων

3.1	Διαμορφώσεις κρυφής μνήμης LLC	27
4.1	Γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης για τις κρυφές μνήμες με πολιτική αντικατάστασης LRU.	43
4.2	Γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης του δικτύου DNN, εκπαιδευμένο σε διαφορετικά μεγέθη κρυφής μνήμης. Μαζί με τον γεωμετρικό μέσο όρο των απόλυτων σφαλμάτων πρόβλεψης του δικτύου LSTM, για σύγκριση.	46
4.3	Γεωμετρικός μέσος όρος των σφαλμάτων πρόβλεψης για τις κρυφές μνήμες με πολιτική αντικατάστασης SHiP.	49
4.4	Γεωμετρικός μέσος όρος των σφαλμάτων πρόβλεψης για τις κρυφές μνήμες με πολιτική αντικατάστασης SRRIP.	51
4.5	Γεωμετρικός μέσος όρος των σφαλμάτων πρόβλεψης για τις κρυφές μνήμες με πολιτική αντικατάστασης Mockingjay.	52
4.6	Γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης του δικτύου LSTM.	57
4.7	Γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης του δικτύου DNN.	60

Κεφάλαιο 1

Εισαγωγή

Ας εμβαθύνουμε στη δυναμική σφαίρα όπου η αρχιτεκτονική ενός υπολογιστή δεν είναι απλώς ένα σχέδιο αλλά ένας καταλύτης, που ασκεί τεράστια επιρροή στα αποτελέσματα των επιδόσεων. Από τον περίπλοκο χορό των επεξεργαστών έως τα δαιδαλώδη μονοπάτια της μνήμης, κάθε αρχιτεκτονική πτυχή εννοχρησιμεύει μια συμφωνία που καθορίζει την ταχύτητα, την αποδοτικότητα και τη μετασχηματιστική δύναμη των υπολογιστών. Η κατανόηση αυτής της περίπλοκης αλληλεπίδρασης ανοίγει τις πόρτες για την απελευθέρωση απaráμιλλων δυνατοτήτων απόδοσης, αναδιαμορφώνοντας το ψηφιακό τοπίο με μία αρχιτεκτονική καινοτομία τη φορά.

Κατά τη διαπίστωση του κομβικού ρόλου της αρχιτεκτονικής υπολογιστών στη διαμόρφωση του τοπίου των επιδόσεων, είναι επιτακτική ανάγκη να αναγνωρίσουμε τις περίπλοκες δυναμικές πίσω από αυτή την τεχνολογική συμφωνία. Η αρχιτεκτονική υπολογιστών υπερβαίνει το ρόλο ενός απλού σχεδίου- χρησιμεύει ως ο ακρογωνιαίος λίθος που υπαγορεύει την αποδοτικότητα και τις δυνατότητες της σύγχρονης πληροφορικής. Για να κατανοήσουμε τον βαθύ αντίκτυπό της, είναι απαραίτητο να εξετάσουμε την περίπλοκη εννοχρησίμευση των επεξεργαστών και των μονοπατιών μνήμης, διερευνώντας πώς κάθε αρχιτεκτονική πτυχή εναρμονίζεται για να καθορίσει την ταχύτητα, την αποτελεσματικότητα και τις μετασχηματιστικές δυνατότητες στο ψηφιακό πεδίο.

Η αδιάκοπη επιδίωξη ταχύτερων και αποδοτικότερων επεξεργαστών είναι μια ιστορία συνεχούς καινοτομίας και τεχνολογικής προόδου. Από την απαρχή της πληροφορικής, η αναζήτηση της ταχύτητας οδήγησε τους μηχανικούς και τους επιστήμονες να διευρύνουν τα όρια του εφικτού. Αρχικά, οι ΚΜΕ κατασκευάζονταν με τη χρήση βασικών ηλεκτρονικών εξαρτημάτων. Ωστόσο, καθώς η τεχνολογία εξελισσόταν, στα τέλη του 20ού αιώνα σημειώθηκε μια μνημειώδης αλλαγή με την έλευση των ολοκληρωμένων κυκλωμάτων και των μικροεπεξεργαστών. Ο νόμος του Μοορε, που διατυπώθηκε από τον συνιδρυτή της Ιντελ Γορδον Μοορε το 1965, προέβλεψε τον διπλασιασμό του αριθμού των τρανζίστορ σε ένα τσιπ περίπου κάθε δύο χρόνια. Η αρχή αυτή έγινε κατευθυντήρια δύναμη, οδηγώντας τις φιλοδοξίες της βιομηχανίας για ταχεία πρόοδο. Με την πάροδο του χρόνου, η διαδικασία κατασκευής υπέστη σημαντικές βελτιώσεις, μεταβαίνοντας από μεγαλύτερα τρανζίστορ σε μικρότερα, πιο πυκνά τοποθετημένα, οδηγώντας σε αυξημένες ταχύτητες επεξεργασίας και βελτιωμένη αποδοτικότητα. Οι καινοτομίες στην τεχνολογία των ημιαγωγών, όπως η ανάπτυξη τσιπ με βάση το πυρίτιο, η εισαγωγή πολυπύρηνων επεξεργαστών και η βελτίωση των αρχιτεκτονικών μέσω της επένδυσης σωλήνων και της παράλληλης επεξεργασίας, υπήρξαν καθοριστικής σημασίας

για την αδιάκοπη πορεία προς ταχύτερους επεξεργαστές. Επιπλέον, οι πρόοδοι στην επιστήμη των υλικών, τη νανοτεχνολογία και τις μεθοδολογίες σχεδιασμού των τσιπ συνέβαλαν συλλογικά στη συνεχή εξέλιξη των ΚΜΕ, τροφοδοτώντας μια εποχή υπολογιστικής ισχύος που συνεχίζει να επαναπροσδιορίζει τα όρια των τεχνολογικών δυνατοτήτων.

Οι κρυφές μνήμες επεξεργαστή χρησιμεύουν ως ένας κρίσιμος μηχανισμός για την άμβλυνση των σημείων συμφόρησης στις επιδόσεις, ελαχιστοποιώντας τον χρόνο που χρειάζεται ο επεξεργαστής για την πρόσβαση σε συχνά χρησιμοποιούμενα δεδομένα. Όταν μια CPU εκτελεί λειτουργίες, ανακτά συνεχώς δεδομένα και εντολές από τη μνήμη. Ωστόσο, η πρόσβαση σε δεδομένα απευθείας από την κύρια μνήμη μπορεί να είναι σχετικά αργή λόγω της διαφοράς ταχύτητας μεταξύ της ΚΜΕ και της μνήμης. Οι κρυφές μνήμες ενεργούν ως ενδιάμεσος υψηλής ταχύτητας μεταξύ της ΚΜΕ και της κύριας μνήμης, αποθηκεύοντας δεδομένα και εντολές στις οποίες γίνεται συχνή πρόσβαση. Με τον τρόπο αυτό, μειώνουν την καθυστέρηση στην ανάκτηση πληροφοριών που απαιτούνται από την ΚΜΕ. Όταν ο επεξεργαστής χρειάζεται δεδομένα, ελέγχει πρώτα την κρυφή μνήμη. Εάν τα απαιτούμενα δεδομένα βρεθούν στην κρυφή μνήμη (cache hit), μπορεί να γίνει πρόσβαση σε αυτά πολύ πιο γρήγορα από την ανάκτησή τους από την πιο αργή κύρια μνήμη. Με τον τρόπο αυτό αποφεύγεται η ανάγκη αναμονής για δεδομένα από την κύρια μνήμη, με αποτέλεσμα να αμβλύνεται η δυσχέρεια επιδόσεων που προκαλείται από την καθυστέρηση πρόσβασης στη μνήμη. Οι αποτελεσματικά χρησιμοποιούμενες κρυφές μνήμες βελτιστοποιούν την απόδοση της CPU μειώνοντας τον χρόνο αδράνειας, επιτρέποντας ταχύτερη πρόσβαση σε δεδομένα και βελτιώνοντας τη συνολική απόδοση του συστήματος σε διάφορες υπολογιστικές εργασίες.

Οι αστοχίες στην κρυφή μνήμη προκαλούν σημαντικές χρονικές ποινές στις υπολογιστικές ροές εργασίας. Όταν η CPU αναζητά δεδομένα που δεν είναι αποθηκευμένα στην κρυφή μνήμη, προκαλείται αστοχία στην κρυφή μνήμη, γεγονός που ωθεί τον επεξεργαστή να διακόψει την εκτέλεσή του, ενώ ανακτά τις απαιτούμενες πληροφορίες από την πιο αργή κύρια μνήμη. Αυτή η μετάβαση μεταξύ της κρυφής μνήμης και της κύριας μνήμης προκαλεί αξιοσημείωτη καθυστέρηση λόγω της σημαντικής διαφοράς ταχύτητας μεταξύ αυτών των επιπέδων μνήμης. Αυτή η καθυστέρηση, που ονομάζεται ποινή αστοχίας στη μνήμη (cache miss), έχει ως αποτέλεσμα δεκάδες, αν όχι εκατοντάδες, αδρανείς κύκλους επεξεργαστή, παρεμποδίζοντας την ομαλή εκτέλεση των εντολών και δυσχεραίνοντας τη συνολική απόδοση του συστήματος. Ως εκ τούτου, ο εντοπισμός και η διαλεύκανση των σημείων συμφόρησης των επιδόσεων που προκύπτουν από τις αστοχίες της κρυφής μνήμης είναι ζωτικής σημασίας.

Η ποικιλομορφία στα σχέδια, τα μεγέθη, τις στρατηγικές αντιστοίχισης, τις πολιτικές αντικατάστασης και τις τεχνολογικές εξελίξεις μεταξύ των διαφόρων επεξεργασιών έχει ως αποτέλεσμα διαφορετικές συμπεριφορές αστοχίας στην κρυφή μνήμη, γεγονός που καθιστά απαραίτητη την εξέταση των ιδιαίτερων χαρακτηριστικών κάθε αρχιτεκτονικής επεξεργαστή κατά την ανάλυση της απόδοσης της κρυφής μνήμης.

Η συμπεριφορά των αστοχιών της κρυφής μνήμης μπορεί να κυμαίνεται σημαντικά μεταξύ διαφορετικών ΚΜΕ λόγω των διαφοροποιήσεων στην αρχιτεκτονική, το μέγεθος, την οργάνωση και τις πολιτικές πρόσβασης της κρυφής μνήμης. Διαφορετικές ΚΜΕ μπορεί να χρησιμοποιούν διαφορετικές σχεδιάσεις κρυφής μνήμης, όπως διαφορετικά επίπεδα κρυφής μνήμης (L1, L2, L3), μεγέθη κρυφής μνήμης, συσχετιστικότητα, πολιτικές αντικατάστασης και στρατηγικές προφόρτωσης. Αυτές οι αρχιτεκτονικές αποκλίσεις μπορούν να επηρεάσουν

τον τρόπο με τον οποίο συμβαίνουν οι αστοχίες στην κρυφή μνήμη και τις επακόλουθες κυρώσεις τους.

Επιπλέον, οι αρχιτεκτονικές εξελίξεις και οι καινοτομίες σε νεότερες γενιές CPU συχνά εισάγουν βελτιστοποιήσεις με στόχο τη μείωση των αστοχιών της κρυφής μνήμης. Οι βελτιωμένες τεχνικές προφόρτωσης, οι εξυπνότεροι αλγόριθμοι πρόβλεψης ή οι αλλαγές στην ιεραρχία της κρυφής μνήμης μπορούν να επηρεάσουν τον τρόπο με τον οποίο εκδηλώνονται οι αστοχίες κρυφής μνήμης στις νεότερες CPU σε σύγκριση με τις παλαιότερες.

Η προσομοίωση μιας κρυφής μνήμης CPU χρησιμεύει ως απαραίτητο εργαλείο στη σύγχρονη πληροφορική, προσφέροντας κρίσιμες γνώσεις και βελτιστοποιήσεις. Μιμούμενοι τη συμπεριφορά των μηχανισμών προσωρινής αποθήκευσης, οι προσομοιώσεις επιτρέπουν τη σε βάθος ανάλυση των επιδόσεων της κρυφής μνήμης, βοηθώντας στο σχεδιασμό, την αξιολόγηση και τη βελτιστοποίηση των αρχιτεκτονικών κρυφής μνήμης. Αυτές οι προσομοιώσεις διευκολύνουν τη διερεύνηση ποικίλων διαμορφώσεων κρυφής μνήμης, πολιτικών αντικατάστασης και συσχετισμού, επιτρέποντας στους μηχανικούς να επιλέξουν τις πιο αποδοτικές ρυθμίσεις για συγκεκριμένες εφαρμογές. Επιπλέον, οι προσομοιώσεις κρυφής μνήμης προσφέρουν μια βαθύτερη κατανόηση του τρόπου με τον οποίο οι διάφοροι φόρτοι εργασίας αλληλεπιδρούν με την κρυφή μνήμη και επηρεάζουν τις αστοχίες της κρυφής μνήμης.

Η διαδικασία προσομοίωσης κρυφής μνήμης CPU μπορεί να είναι χρονοβόρα λόγω διαφόρων παραγόντων. Η πολυπλοκότητα που συνεπάγεται η προσομοίωση της συμπεριφοράς της κρυφής μνήμης απαιτεί σημαντικούς υπολογιστικούς πόρους και χρόνο. Ο χειρισμός μεγάλων ιχνών προτύπων προσπέλασης μνήμης για ακριβείς προσομοιώσεις αυξάνει τον χρόνο που απαιτείται για την επεξεργασία. Επιπλέον, η ενσωμάτωση περίπλοκων σχεδίων κρυφής μνήμης, όπως πολλαπλά επίπεδα κρυφής μνήμης (L1, L2, L3), ποικίλα μεγέθη κρυφής μνήμης, συστήματα συσχετισμού και προηγμένοι αλγόριθμοι κρυφής μνήμης, συμβάλλει στην υπολογιστική πολυπλοκότητα και τη διάρκεια των προσομοιώσεων. Ως εκ τούτου, αυτές οι προσομοιώσεις αποδεικνύονται εξαιρετικά χρονοβόρες και απαιτείται ένας καλύτερος τρόπος υπολογισμού του ποσοστού αστοχίας.

Η λύση σε αυτό το πρόβλημα είναι οι μηχανισμοί πρόβλεψης. Η σημασία ενός μοντέλου που μπορεί να προβλέψει με ακρίβεια το μισο ρατιο ενός φόρτου εργασίας χωρίς να έχει το βάρος της προσομοίωσης κάθε πτυχής του είναι τεράστια. Σχεδόν όλα τα υπάρχοντα μοντέλα πρόβλεψης του μισο ρατιο έχουν δημιουργηθεί για πολιτικές LRU ή τυχαίας αντικατάστασης [1, 2, 3]. Το μόνο υπάρχον μοντέλο [1] που μπορεί να χρησιμοποιήσει άλλες πολιτικές αντικατάστασης εκτός της LRU, πρέπει να γνωρίζει, πριν από την εκτέλεση, αναλυτικά όλες τις λεπτομέρειες διαμόρφωσης. Αυτό αφήνει ένα μεγάλο μέρος του προβλήματος αναπάντητο, αφού οι περισσότεροι σύγχρονοι επεξεργαστές CPU δεν χρησιμοποιούν την πολιτική αντικατάστασης LRU στις κρυφές μνήμες τους και, στις περισσότερες περιπτώσεις, δεν γνωρίζουμε ποια πολιτική αντικατάστασης χρησιμοποιείται.

Στην παρούσα διατριβή αντιμετωπίζουμε αυτό το πρόβλημα της πρόβλεψης των αστοχιών της κρυφής μνήμης σε οποιαδήποτε δεδομένη αρχιτεκτονική με τη χρήση μηχανικής μάθησης. Προτείνουμε έναν αριθμό διαφορετικών δικτύων που έχουν ως στόχο την όσο το δυνατόν ακριβέστερη εκτίμηση των αναλογιών αστοχίας στην κρυφή μνήμη. Εμπνευσμένοι από την ήδη υπάρχουσα πιθανολογική προσέγγιση σε αυτό το ζήτημα, γνωστή και ως Στατάτση 2.2, αποφασίσαμε να χρησιμοποιήσουμε τα ίδια δεδομένα που χρησιμοποιεί για την πρόβλεψη

του για να εκπαιδευσουμε και να αξιολογήσουμε ένα δίκτυο μηχανικής μάθησης. Η σημερινή αύξηση της δημοτικότητας των δικτύων NLP μας παρακίνησε να προσθέσουμε έναν κλάδο μετασχηματιστή στα δίκτυα, ο οποίος χρησιμοποιεί αυτή τη νέα τεχνολογία κατανόησης του πηγαιού κώδικα, για να επιτύχουμε καλύτερα αποτελέσματα.

Δημιουργήσαμε συνολικά τρεις διαφορετικές αρχιτεκτονικές δικτύων που προβλέπουν το ποσοστό αστοχίας στην κρυφή μνήμη οποιουδήποτε προγράμματος για καθορισμένα μεγέθη κρυφής μνήμης με οποιαδήποτε πολιτική αντικατάστασης. Αυτά τα δίκτυα αποδίδουν πολύ καλά για οποιαδήποτε πολιτική αντικατάστασης. Εισάγαμε επίσης ένα που προβλέπει τα ποσοστά αστοχίας για οποιοδήποτε μέγεθος κρυφής μνήμης και οποιαδήποτε πολιτική αντικατάστασης, τη λειτουργία του οποίου παρουσιάσαμε στο LRU. Στην περίπτωση του LRU, όλα αυτά τα δίκτυα κατανοούν το πρόβλημα σημαντικά καλύτερα από την υπάρχουσα πιθανολογική προσέγγιση του ζητήματος.

Η παρούσα διατριβή είναι δομημένη σε πέντε κύρια κεφάλαια, καθένα από τα οποία συμβάλλει μοναδικά στη διερεύνηση και ανάλυση των αναλογιών αστοχίας. Το Κεφάλαιο 1 εισάγει τις θεμελιώδεις έννοιες και το θεωρητικό πλαίσιο που υποστηρίζει τη μελέτη, παρουσιάζοντας μια επισκόπηση του ιστορικού πλαισίου και της σημασίας της πρόβλεψης των αστοχιών της κρυφής μνήμης. Το κεφάλαιο 2 εμβαθύνει στην υπάρχουσα βιβλιογραφία, εξετάζοντας κριτικά προηγούμενες έρευνες που θα βοηθήσουν στην κατανόηση των ζητημάτων που προσπαθεί να αντιμετωπίσει η παρούσα διατριβή. Το κεφάλαιο 3 περιγράφει την ερευνητική μεθοδολογία που υιοθετήθηκε, περιγράφοντας λεπτομερώς την επιλεγμένη προσέγγιση, τις μεθόδους συλλογής δεδομένων και τις αρχιτεκτονικές δικτύου που χρησιμοποιήθηκαν. Επίσης, κατέχει ξεχωριστά τα αποτελέσματα για κάθε ένα από τα δίκτυα που θα προτείνουμε. Στο κεφάλαιο 4 παρουσιάζονται τα συλλεχθέντα αποτελέσματα που προέκυψαν από τα δίκτυά μας, προσφέροντας μια ανάλυση και σύγκριση των δεδομένων που συλλέχθηκαν. Τέλος, το Κεφάλαιο 5 συνθέτει τα ευρήματα, συζητά τις επιπτώσεις τους και προσφέρει συμπεράσματα μαζί με συστάσεις για μελλοντική έρευνα πρόβλεψης του ποσοστού αστοχίας στην κρυφή μνήμη.

Συνοψίζοντας, η παρούσα διατριβή έχει ως στόχο να χρησιμοποιήσει τις τρέχουσες εξελίξεις στη μηχανική μάθηση προκειμένου να προβλέψει τα ποσοστά αστοχίας κρυφής μνήμης οποιουδήποτε προγράμματος σε οποιαδήποτε αρχιτεκτονική κρυφής μνήμης. Με τη διερεύνηση πιθανών λύσεων σε αυτό το πρόβλημα, όπως η πιθανολογική προσέγγιση ή οι αλγόριθμοι NLP, η παρούσα μελέτη σκοπεύει να ρίξει φως στη δυσκολία και τις πολλές πτυχές της ακριβούς πρόβλεψης των ζαση μισσες. Τα επόμενα κεφάλαια θα εξετάσουν λεπτομερώς αυτές τις έννοιες. Το κεφάλαιο 2 θα επικεντρωθεί σε σχετικές εργασίες που έχουν γίνει, ενώ το κεφάλαιο 3 θα εμβαθύνει στην πρότασή μας για την επίλυση αυτού του ζητήματος. Αυτή η δομημένη προσέγγιση θα παρέχει μια ολοκληρωμένη κατανόηση της πρόβλεψης των αστοχιών της κρυφής μνήμης. Τώρα, ας προχωρήσουμε στη βαθύτερη εμβάθυνση σε αυτές τις περιοχές, ξεκινώντας με την πιθανολογική προσέγγιση StatCache.

Μέρος I

Θεωρητικό Μέρος

Θεωρητικό υπόβαθρο

2.1 Ιεραρχία κρυφών μνημών

Στην ιστορία της ανάπτυξης των υπολογιστών, οι ταχύτητες της CPU ξεπερνούσαν τις ταχύτητες πρόσβασης στη μνήμη. Αυτή η διαφορά οδηγούσε σε αδρανείς ΚΜΕ που περίμεναν δεδομένα από την κύρια μνήμη. Εισάγεται η κρυφή μνήμη, μια ιδέα που προτάθηκε για πρώτη φορά από τον Βρετανό επιστήμονα υπολογιστών Maurice Wilkes το 1965 [4]. Τα πρώτα μοντέλα κρυφής μνήμης βελτίωσαν την καθυστέρηση πρόσβασης στα δεδομένα, αλλά το να γίνει η κύρια μνήμη εντελώς υψηλής ταχύτητας ήταν απαγορευτικά ακριβό. Οι ερευνητές διερεύνησαν καλύτερα σχέδια, οδηγώντας τελικά στην ιδέα των πολυεπίπεδων κρυφών μνημών. Αυτά τα πολυεπίπεδα μοντέλα κρυφής μνήμης, όπως οι κρυφές μνήμες τριών επιπέδων που βρίσκονται στα προϊόντα Core i7 της Ιντελ, επιτυγχάνουν μια ισορροπία μεταξύ κόστους και απόδοσης. Τώρα, οι CPUs μπορούν να αξιοποιήσουν μια ιεραρχία κρυφών μνημών, κάθε επίπεδο της οποίας λειτουργεί ως ρυθμιστικός χώρος μεταξύ του επεξεργαστή και της κύριας μνήμης, εξασφαλίζοντας αποτελεσματική ροή δεδομένων και ταχύτερη εκτέλεση.

Κάθε μία από αυτές τις κρυφές μνήμες έχει το δικό της μέγεθος και τη δική της εσωτερική δομή, ανεξάρτητα από τις κρυφές μνήμες άλλων επιπέδων. Υπάρχουν πολλοί τρόποι διαμόρφωσης μιας κρυφής μνήμης ως προς τη δομή της. Οι σημαντικοί που θα συζητήσουμε είναι το μέγεθος, η συσχετιστικότητα και η πολιτική αντικατάστασης. Για το μέγεθος, η μόνη εξάρτησή του είναι να είναι μεγαλύτερο από το μέγεθος της κρυφής μνήμης στα χαμηλότερα επίπεδα. Γενικά σημαίνει ότι το μέγεθος(L3 > μέγεθος(L2) > μέγεθος(L1)). Αυτό οφείλεται στο γεγονός ότι η ταχύτητα των κρυφών μνήμης χαμηλότερων επιπέδων υποτίθεται ότι είναι υψηλότερη και επομένως απαιτείται μικρότερη κρυφή μνήμη. Η συσχέτιση αναφέρεται στην εσωτερική δομή αυτής της κρυφής μνήμης, συγκεκριμένα, ασχολείται με τις μεθόδους που χρησιμοποιούνται για να καθοριστεί πού μπορούν να αποθηκευτούν τα δεδομένα εντός της κρυφής μνήμης. Στην περίπτωσή μας, θα χρησιμοποιήσουμε set-associative ζαζαρες, αλλά υπάρχουν επίσης fully-associative και direct mapping caches.

Τέλος, είναι σημαντικό να καθοριστεί η πολιτική αντικατάστασης της κρυφής μνήμης. Οι πολιτικές αντικατάστασης είναι αλγόριθμοι ή στρατηγικές που χρησιμοποιούνται για να αποφασιστεί ποιες εγγραφές της κρυφής μνήμης θα αντικατασταθούν ή θα εκδιωχθούν όταν πρέπει να φορτωθούν νέα δεδομένα σε μια πλήρη κρυφή μνήμη. Αυτές οι πολιτικές είναι ζωτικής σημασίας για τη διατήρηση της αποδοτικότητας και της απόδοσης του συστήματος κρυφής μνήμης, καθώς επηρεάζουν άμεσα τα ποσοστά επιτυχίας των προσπελάσεων της

κρυφής μνήμης. Η LRU είναι η πιο γνωστή πολιτική αντικατάστασης, η οποία αφαιρεί την καταχώρηση της κρυφής μνήμης που δεν έχει προσπελαστεί για το μεγαλύτερο χρονικό διάστημα. Οι σύγχρονοι επεξεργαστές χρησιμοποιούν πιο σύγχρονες πολιτικές αντικατάστασης που συχνά δεν είναι γνωστές στον χρήστη μιας CPU. Ως εκ τούτου, θα χρησιμοποιήσουμε μια σειρά από διαφορετικές πολιτικές αντικατάστασης σε αυτή τη διατριβή, για να δείξουμε ότι οι προβλέψεις παρουσιάζουν σταθερά υψηλή ποιότητα σε διάφορες μηχανές. Οι SHiP [5], SRRIP [6] και Mockingjay [7] είναι σύγχρονες πολιτικές αντικατάστασης που στοχεύουν στην πρόβλεψη της επαναχρησιμοποίησης μιας εγγραφής στην κρυφή μνήμη και αντικαθιστούν τη λιγότερο πιθανή να επαναχρησιμοποιηθεί.

2.2 StatCache

Έχουν γίνει στατιστικές προσπάθειες εκτίμησης των ποσοτών αστοχίας κρυφής μνήμης για πολιτικές αντικατάστασης LRU. Συγκεκριμένα, το StatCache [3] προσφέρει μια πιθανολογική προσέγγιση προς αυτόν τον στόχο χρησιμοποιώντας τις αποστάσεις επαναχρησιμοποίησης του εν λόγω προγράμματος.

Συμβολίζουμε ως προσπέλαση μνήμης τις διευθύνσεις ενός μπλοκ διευθύνσεων μεγέθους γραμμής κρυφής μνήμης. Έστω N ο αριθμός των προσπελάσεων μνήμης που συμβαίνουν κατά την εκτέλεση του προγράμματος. Μπορούμε να απαριθμήσουμε αυτές τις προσβάσεις από το 1 έως N . Για κάθε $i < j < N$ όπου i και j προσπελαίνουν τα ίδια δεδομένα και δεν έχουν συμβεί ενδιάμεσες προσπελάσεις, μπορούμε να πούμε ότι η απόσταση επαναχρησιμοποίησης αυτού είναι $i - j - 1$. Αυτό σημαίνει ότι οι προσπελάσεις που συνέβησαν μεταξύ δύο διαδοχικών προσπελάσεων μιας διεύθυνσης μνήμης σε αυτά τα δεδομένα είναι $i - j - 1$. Στη συνέχεια συγκεντρώνουμε αυτές τις προσβάσεις σε ένα ιστόγραμμα h . Σε αυτό το ιστόγραμμα $h(i)$ υποδηλώνει τις προσπελάσεις που έχουν απόσταση επαναχρησιμοποίησης i .

Χρησιμοποιώντας τις αποστάσεις επαναχρησιμοποίησης, μπορούμε να καταλήξουμε στην εξής εξίσωση:

$$RN \approx h(1)f(R) + h(2)f(2R) + h(3)f(3R) + \dots$$

όπου R είναι το ποσοστό αστοχίας, N είναι η μεγαλύτερη δυνατή απόσταση επαναχρησιμοποίησης, h είναι ένα ιστόγραμμα όπου το $h(i)$ μας λέει ότι υπάρχουν $h(i)$ αναφορές με απόσταση επαναχρησιμοποίησης i , και τέλος

Η συνάρτηση $F(n)$ δηλώνει την πιθανότητα η γραμμή κρυφής μνήμης να μην παραμείνει στην κρυφή μνήμη μετά από n αστοχίες. Έτσι, υποθέτοντας ότι η κρυφή μνήμη είναι πλήρως συσχετιστική και έχει L γραμμές κρυφής μνήμης, η πιθανότητα μια γραμμή να μην παραμείνει στην κρυφή μνήμη μετά από n αναδρομές είναι:

$$f(n) = 1 - (1 - 1/L)^n$$

Αυτή είναι ουσιαστικά η πιθανολογική προσέγγιση της StatCache στο ερώτημα. Κάθε απόσταση επαναχρησιμοποίησης μαζί με την πιθανότητα παραμονής της στην κρυφή μνήμη υπολογίζει έναν αριθμό αστοχιών στην κρυφή μνήμη που, αθροισόμενες, δίνουν τις συνολικές αστοχίες της εκτέλεσης. Τα πλεονεκτήματα αυτής της μεθόδου εκτίμησης είναι ότι ο υπολογισμός της είναι απλός και γρήγορος. Το κύριο μειονέκτημα αυτής της μεθόδου είναι ότι λειτουργεί μόνο για την πολιτική αντικατάστασης LRU.

2.3 Άλλες χρήσεις Νευρωνικών Δικτύων

Τα βαθιά νευρωνικά δίκτυα έχουν αποδείξει τις ικανότητές τους να αναλύουν και να κατανοούν σύνθετα μοτίβα σε πολλά είδη προβλημάτων, όπως η ταξινόμηση εικόνων [8, 9]. Στην περίπτωση μας, χρήσιμες είναι οι έρευνες που ασχολούνται με τον κώδικα υπολογιστών [10, 11].

Παράλληλα με αυτές, παρατηρείται επίσης μια αύξηση της δημοτικότητας των νευρωνικών δικτύων στα πλαίσια των αρχιτεκτονικών υπολογιστών. Για παράδειγμα, πολλές βελτιστοποιήσεις μεταγλωττιστών έχουν προκύψει από την εφαρμογή τεχνικών μηχανικής μάθησης. Στον προγραμματισμό εντολών, η συνάρτηση προτίμησης ενός προγραμματισμού έναντι ενός άλλου μπορεί να υπολογιστεί από τη χρονική διαφορά του αλγορίθμου RL [12]. Το μοντέλο που βασίζεται σε LSTM [13], παρακάμπτει τη χειροκίνητη μηχανική χαρακτηριστικών, αποκτώντας αυτόνομα ευρετικές λειτουργίες του μεταγλωττιστή από τον ακατέργαστο κώδικα. Αυτό επιτρέπει την κατασκευή κατάλληλων ενσωματώσεων για προγράμματα, ενώ ταυτόχρονα κατέχει τη διαδικασία βελτιστοποίησης.

Πολλά διαφορετικά δίκτυα μηχανικής μάθησης έχουν χρησιμοποιηθεί για την πρόβλεψη παρόμοιων προβλημάτων με το δικό μας [14]. Dong *et al.* [15] χρησιμοποιούν τεχνητά νευρωνικά δίκτυα για την πρόβλεψη χαρακτηριστικών υψηλότερου επιπέδου (όπως αστοχίες σε cache read/write και instructions per cycle) από χαμηλότερου επιπέδου χαρακτηριστικά (όπως cache associativity, capacity, latency) για ιεραρχίες κρυφών μνημών βασισμένες σε μη πιθητική μνήμη. Άλλα δίκτυα μηχανικής μάθησης έχουν εισαχθεί για να βοηθήσουν στην πρόβλεψη αποτελεσματικής κατανομής πόρων [16] και χρονοπρογραμματισμού εργασιών [17], ώστε να επιλέγεται πάντα η διαδρομή των μέγιστων οδηγιών ανά κύκλο.

Πολυάριθμες εισαγωγές της μηχανικής μάθησης στη σφαίρα των αρχιτεκτονικών υπολογιστών, τόσο παρόμοιες όσο και διαφορετικές, έχουν προκαλέσει την περιέργειά μας. Αυτό το κίνητρο μας οδήγησε να διερευνήσουμε την επίλυση του προβλήματος πρόβλεψης του ποσοστού αστοχίας στην κρυφή μνήμη μέσα από το πρίσμα της μηχανικής μάθησης.

2.4 Ενσωματώσεις

Οι ενσωματώσεις είναι αναπαράσταση των αντικειμένων ως διανύσματα. Μια συνάρτηση μετατρέπει ένα αντικείμενο που είναι αναγνωρίσιμο από τον άνθρωπο σε ένα διάνυσμα αριθμών, το οποίο μπορεί να αναγνωριστεί από ένα πρόγραμμα με τη μικρότερη δυνατή απώλεια δεδομένων. Αυτή η μέθοδος αναπαράστασης χρησιμοποιείται συχνά στο NLP, όπου κάθε λέξη του λεξικού μετατρέπεται σε αριθμό και τροφοδοτείται σε ένα νευρωνικό δίκτυο.

Μια παρόμοια τεχνική έχει αναπτυχθεί για τη δημιουργία ενσωματώσεων κώδικα [18, 19] όπου τα σύμβολα του κώδικα χρησιμοποιούνται για τη μετατροπή μιας ακολουθίας εντολών κώδικα στο διάνυσμα ενσωμάτωσης. Κάθε εντολή παίρνει μια τιμή από το 0 έως το N όπου N είναι ο αυθαίρετος αριθμός συμβόλων που θέλουμε να λάβουμε υπόψη (τα υπόλοιπα τα απορρίπτουμε) και αυτό είναι γνωστό ως λεξιλόγιο. Στη συνέχεια, αυτή η μετατροπή των συμβόλων θα χρησιμοποιηθεί από κάποιο μηχανισμό για να μετατρέψει αυτούς τους πίνακες συμβόλων σε ένα διάνυσμα ποθ αναπαρηστά το πρόγραμμα με τη μικρότερη δυνατή απώλεια δεδομένων.

Στην περίπτωση μας θα χρησιμοποιήσουμε το IR2Vec [18] για τη δημιουργία των ενσωματώσεων μας. Το IR2Vec δέχεται ως είσοδο την ενδιάμεση αναπαράσταση του κώδικα καθώς και ένα λεξιλόγιο, μετατρέπει το αρχείο IR σε έναν πίνακα αριθμών και στη συνέχεια τροφοδοτεί ένα δίκτυο, το οποίο έχει σχεδιαστεί για να καταγράφει τη σύνταξη και τη σημασιολογία. Έξοδος αυτής της διαδικασίας είναι ένα διάνυσμα μεγέθους 300, το οποίο υποτίθεται ότι θα δώσει στο μοντέλο μας ζωτικές πληροφορίες σχετικά με τις διαδικασίες του προγράμματος και τον τρόπο κατανόησης των αποστάσεων επαναχρησιμοποίησης.

Μέσω μάθησης χωρίς επίβλεψη οι συγγραφείς του IR2Vec δημιούργησαν ένα δίκτυο που αναγνωρίζει και αναπαριστά με ακρίβεια αυτά τα αρχεία IR. Το δίκτυό τους εκπαιδεύτηκε και δοκιμάστηκε στα βενσημαρκς SPEC CPU 17 και στη βιβλιοθήκη Boost. Τα πειραματικά τους αποτελέσματα εξασφαλίζουν την πρακτική βιωσιμότητα αυτού του δικτύου στο πρόβλημά μας, καθώς τα δικά μας πειραματικά δεδομένα προέρχονται επίσης από τις σουίτες συγκριτικών δοκιμών SPEC, όπως θα συζητήσουμε στη συνέχεια. Έτσι, θα χρησιμοποιήσουμε αυτό το δίκτυο και το λεξιλόγιο ως μαύρο κουτί για να μετατρέψουμε τον πηγαίο κώδικα του συνόλου δεδομένων μας. Η έξοδος αυτής της διαδικασίας είναι ένα διάνυσμα μεγέθους 300, το οποίο υποτίθεται ότι θα δώσει στο μοντέλο μας ζωτικές πληροφορίες σχετικά με τις διαδικασίες του προγράμματος και τον τρόπο κατανόησης των αποστάσεων επαναχρησιμοποίησης.

Μέρος 

Πρακτικό Μέρος

Προσέγγιση και μέθοδοι

3.1 Επισκόπηση

Ο στόχος μας σε αυτή τη διατριβή είναι να προβλέψουμε τα ποσοστά αστοχίας της κρυφής μνήμης δεδομένης ενός προγράμματος και μιας αρχιτεκτονικής κρυφής μνήμης. Για να χρησιμοποιήσουμε ένα πρόγραμμα ως είσοδο, πρέπει να μετατρέψουμε τα χαρακτηριστικά του σε χρήσιμα δεδομένα για να προβλέψει το δίκτυό μας. Για το σκοπό αυτό, σύμφωνα με το StatCache 2.2, χρειαζόμαστε τις αποστάσεις επαναχρησιμοποίησης κάθε προγράμματος στο σύνολο αναφοράς μας. Οι αποστάσεις επαναχρησιμοποίησης, όπως αναφέρθηκε προηγουμένως, είναι οι αποστάσεις μεταξύ διαδοχικών προσπελάσεων στην ίδια διεύθυνση στη μνήμη. Αυτές οι αποστάσεις εξαρτώνται κυρίως από το πρόγραμμα και τις εισόδους του-διαφέρουν πολύ λίγο μεταξύ διαφορετικών αρχιτεκτονικών και επομένως θα καταγραφούν μόνο μία φορά από κάθε ίχνος ενός προγράμματος. Θα αποθηκευτούν σε ένα ιστόγραμμα και ένας κλάδος του δικτύου θα προσπαθήσει να προβλέψει τα ποσοστά αστοχίας χρησιμοποιώντας τα.

Στη συνέχεια, πρέπει να μετατρέψουμε τον κώδικα εισόδου σε διάνυσμα (ενσωμάτωση), ώστε το δίκτυο να μπορεί να ανακτήσει ακόμη περισσότερες πληροφορίες από τον κώδικα εισόδου σχετικά με τη δομή του προγράμματος, τη ροή των δεδομένων κ.λπ. Για να προβλέψουμε με ακρίβεια τα ποσοστά αστοχίας στην κρυφή μνήμη, πρέπει να ερμηνεύσουμε κατάλληλα τα μετασχηματισμένα δεδομένα. Το διάνυσμα ενσωμάτωσης μπορεί να τροφοδοτηθεί σε ένα πυκνό νευρωνικό δίκτυο, οπότε αυτό κάνουμε για τον κλάδο του μετασχηματιστή. Από την άλλη πλευρά, τα ιστογράμματα της απόστασης επαναχρησιμοποίησης είναι πιο πολύπλοκα στην κατανόηση- ένα απλό πυκνό νευρωνικό δίκτυο δεν είναι το πιο κατάλληλο δίκτυο για την πλήρη κατανόηση της πολυπλοκότητας αυτού του προβλήματος.

Για τον σκοπό αυτό, εισάγουμε τρία διαφορετικά μοντέλα που κατανοούν με διαφορετικό τρόπο τις αποστάσεις επαναχρησιμοποίησης. Το πρώτο είναι ένα ρηχό πολυεπίπεδο περσεπτρον που υπολογίζει την αναλογία αστοχίας αποκλειστικά με βάση τις αποστάσεις επαναχρησιμοποίησης. Στη συνέχεια, εισάγουμε ένα διπλό στρώμα LSTM και ένα δίκτυο CNN που έχει ως σκοπό την καλύτερη κατανόηση των αποστάσεων επαναχρησιμοποίησης δεδομένων των γειτονικών αποστάσεων επαναχρησιμοποίησης. Αυτά τα δύο δίκτυα υπολογίζουν καλύτερα τον λόγο αστοχίας στην κρυφή μνήμη, χρησιμοποιώντας τις αποστάσεις επαναχρησιμοποίησης καθώς και τον κλάδο μετασχηματισμού.

Τέλος, εισάγουμε ένα βαθύ MLP που λαμβάνει ως είσοδο τις αποστάσεις επαναχρησι-

μοποίησης και τις ενσωματώσεις, εισάγει το μέγεθος LLC ως παράμετρο και υπολογίζει τα ποσοστά μισς για άλλες αρχιτεκτονικές κρυφής μνήμης με την ίδια πολιτική αντικατάστασης.

3.2 Πειραματική οργάνωση

3.2.1 Προσομοίωση

Τα αρχικά δεδομένα εισόδου για το σύστημά μας αποτελούνται από τις σουίτες συγκριτικών δοκιμών SPEC 2007 και SPEC 2016 [20, 21]. Τα SPEC είναι σύνολα δοκιμών αναφοράς που έχουν σχεδιαστεί για να παρέχουν ένα συγκριτικό μέτρο των επιδόσεων έντασης υπολογισμού σε ένα ευρύτατο πρακτικό φάσμα υλικού, χρησιμοποιώντας φόρτους εργασίας που έχουν αναπτυχθεί από εφαρμογές πραγματικών χρηστών. Αυτές οι σουίτες είναι ιδανικές για την παρουσίαση του στόχου μας, καθώς παρέχουν μια μεγάλη ποικιλία προβλημάτων και αλγορίθμων που καλύπτουν ένα ευρύ φάσμα υπολογιστικών εργασιών.

Για την προσομοίωση αυτών των προγραμμάτων, χρησιμοποιούμε το ChampSim, έναν προσομοιωτή επεξεργαστή που στοχεύει κυρίως στην όσο το δυνατόν ακριβέστερη προσομοίωση του υποσυστήματος μνήμης και της πρόβλεψης διακλαδώσεων [22]. Θα χρησιμοποιήσουμε αυτό το εργαλείο προσομοίωσης για τη συλλογή των δεδομένων χρόνου εκτέλεσης (αστοχίες και αποστάσεις επαναχρησιμοποίησης της κρυφής μνήμης, όπως εξηγείται στο επόμενο κεφάλαιο 3.2.2). Εκτελεί ένα πρόγραμμα σε μια προσομοιωμένη μηχανή που βασίζεται σε μια δεδομένη αρχιτεκτονική. Ως εκ τούτου, για να εκτελέσουμε το ChampSim, χρειαζόμαστε ίχνη από τα αρχεία αναφοράς του SPEC. Για το σκοπό αυτό, χρησιμοποιούμε τα ίχνη που δίνονται από το [3rd Data Prefetching Championship](#). Αυτά τα ίχνη αποτελούνται από 2 δισεκατομμύρια εντολές το καθένα και καλύπτουν μεγάλα τμήματα της εκτέλεσης ενός αρχείου αναφοράς. Κάθε αρχείο αναφοράς έχει από ένα έως έξι ίχνη στο όνομά του, καθένα από τα οποία είναι αντιπροσωπευτικό ενός συγκεκριμένου ποσοστού της εκτέλεσής του. Αργότερα, θα χρησιμοποιήσουμε αυτό το ποσοστό ως βάρος για να υπολογίσουμε τις αστοχίες κρυφής μνήμης ενός αρχείου αναφοράς ως σταθμισμένο μέσο όρο των ποσοστών αστοχίας των ιχνών του.

Η κρυφή μνήμη που παράγει τη μεγαλύτερη καθυστέρηση αστοχίας είναι η LLC. Αυτό καθιστά την πρόβλεψη για αυτήν πολύ πιο πολύτιμη από ό,τι για τις κρυφές μνήμες άλλου επιπέδου. Επομένως, στην προσέγγισή μας, διατηρούμε αμετάβλητες τις κρυφές μνήμες χαμηλότερου επιπέδου L1D και L2C με μεγέθη 48KB και 512KB, συσχετιστικότητα 12 και 16 αντίστοιχα και πολιτικές αντικατάστασης LRU. Πραγματοποιήσαμε τις προσομοιώσεις που αναφέρονται στα επόμενα κεφάλαια με ένα ευρύ φάσμα μεγεθών κρυφής μνήμης και τέσσερις πολιτικές αντικατάστασης για την LLC, ώστε να καλύψουμε ένα μεγάλο σύνολο αρχιτεκτονικών του πραγματικού κόσμου και να δείξουμε ότι είναι ευρέως χρησιμοποιήσιμη. Χρησιμοποιήσαμε τα μεγέθη κρυφής μνήμης που αναφέρονται στον πίνακα 3.1. Τα μεγέθη LLC των 768KB, 1536KB, 3072KB και 6144KB έχουν συσχετιστικότητα 12, ενώ οι κρυφές μνήμες μεγέθους 1024, 2048, 4096 και 8192KB έχουν συσχετιστικότητα 16.

Replacement policy	LLC μέγεθος [KB]
LRU	768, 1024, 1536, 2048, 3072, 4096, 6144, 8192
SHiP	1024, 2048, 4096, 8192
SRRIP	1024, 2048, 4096, 8192
Mockingjay	1024, 2048, 4096, 8192

Πίνακας 3.1: Διαμορφώσεις κρυφής μνήμης LLC

3.2.2 Σύνολο δεδομένων

Συλλογή αποστάσεων επαναχρησιμοποίησης

Η συλλογή των αποστάσεων επαναχρησιμοποίησης κατά τη διάρκεια του χρόνου εκτέλεσης είναι αρκετά απλή. Υλοποιήσαμε έναν αναλυτή αποστάσεων επαναχρησιμοποίησης που θα χρησιμοποιηθεί στο ChampSim. Του δίνεται μια διεύθυνση και μια κατάσταση και υπολογίζει την απόσταση επαναχρησιμοποίησης. Αυτό έγινε μέσω μιας απλής συνάρτησης που, κάθε φορά που καλείται, ελέγχει την τελευταία φορά που έγινε πρόσβαση σε αυτά τα δεδομένα και την αυξάνει στο ιστόγραμμα. Στη συνέχεια, η διεύθυνση των δεδομένων αποθηκεύεται για μελλοντικούς υπολογισμούς της απόστασης επαναχρησιμοποίησης και συνεχίζουμε. Αυτή η υλοποίηση είναι ελαφριά, αφού δεν προστίθενται παρά μόνο μερικές εντολές κάθε φορά που καλείται, και αποτελεσματική, αφού χρησιμοποιούνται αποδοτικές δομές δεδομένων, όπως οι χάρτες κατακερματισμού.

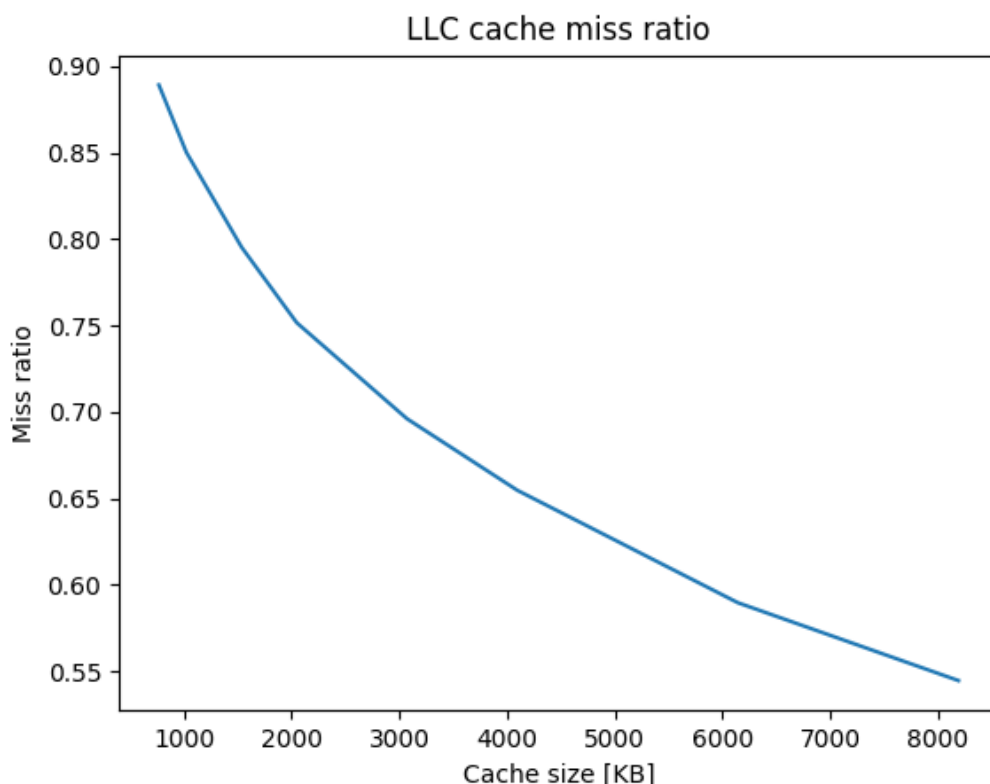
Για να χρησιμοποιήσουμε αυτόν τον αναλυτή της αποστάσεων επαναχρησιμοποίησης, τον προσθέσαμε στην πολιτική αντικατάστασης του L1D που εκτελεί αυτή την προαναφερθείσα διαδικασία κάθε n -οστή φορά που καλείται. Αυτό το n πρέπει να είναι αρκετά μικρό για να έχουμε μια αντιπροσωπευτική εικόνα των αποστάσεων επαναχρησιμοποίησης στο σύνολο δεδομένων μας, αλλά όχι πολύ μικρό, καθώς δεν θέλουμε να εκτελούμε τον αναλυτή αποστάσεων επαναχρησιμοποίησης με κάθε πρόσβαση σε δεδομένα, επειδή αυτό αυξάνει τον χρόνο επεξεργασίας. Στην περίπτωση μας, χρησιμοποιούμε $n = 16$ προσπελάσεις. Αυτό είναι υπερβολικά σχολαστικό για 512k προσπελάσεις, αλλά θέλουμε να διασφαλίσουμε ότι οι έξοδοι δεν επηρεάζονται από αυτό και έτσι θυσιάσαμε λίγο χρόνο επεξεργασίας.

Η διακύμανση των αστοχιών της κρυφής μνήμης καθ' όλη τη διάρκεια της εκτέλεσης του προγράμματος μπορεί να είναι ένα ζήτημα που επηρεάζει αρνητικά την εκτίμησή μας. Καθώς το πρόγραμμα περνάει από τις διάφορες φάσεις εκτέλεσης, ο λόγος των αστοχιών στην κρυφή μνήμη αλλάζει με την πάροδο του χρόνου. Σύμφωνα με το έγγραφο της StatCache [3], ο διαχωρισμός της εξόδου των αναλυτών σε παράθυρα με μικρότερο αριθμό προσπελάσεων είναι η λύση σε αυτό το ζήτημα. Η σκέψη πίσω από αυτό είναι ότι ο αριθμός των προσπελάσεων θα είναι αρκετά μικρός, έτσι ώστε ο λόγος αστοχίας κρυφής μνήμης κάθε παραθύρου να είναι πιθανότατα σταθερός με την πάροδο του χρόνου.

Για να το πετύχουμε αυτό, προσομοιώσαμε τα ίχνη και για κάθε 512k προσπελάσεις, επιστρέψαμε την κατάσταση του ιστογράμματος της απόστασης επαναχρησιμοποίησης. Παρατηρήσαμε ότι αυτό το μέγεθος λειτουργεί καλύτερα στην περίπτωση μας, καθώς δίνει στην προσομοίωση χρόνο για να γεμίσει ένα μέρος των κρυφών μνήμης και παρόλα αυτά να δειγματοληπτεί η εκτέλεση του προγράμματος αρκετά συχνά ώστε να μην παρατηρείται η διακύμανση. Τώρα, για κάθε ίχνος αναφοράς, έχουμε έναν αριθμό παραθύρων με 512k

προσπελάσεις το καθένα. κάθε παράθυρο αποτελείται από τις αποστάσεις επαναχρησιμοποίησης που εμφανίστηκαν μέσα σε αυτές τις 512k προσπελάσεις και τον λόγο αστοχίας LLC στον οποίο αντιστοιχούν αυτές. Ας είναι γνωστό ότι ορισμένα ίχνη έχουν πολύ λιγότερες από 512k προσπελάσεις εντός της εκτέλεσής τους. Χρησιμοποιήσαμε ένα παράθυρο για κάθε ένα από αυτά τα ίχνη ανεξάρτητα, έτσι ώστε να μην χαθούν τα δεδομένα τους, αλλά κάποια από αυτά θα οδηγήσουν σε κάποια προβλήματα αργότερα λόγω της διαφορετικής κλίμακας τους.

Για τον υπολογισμό του συνολικού λόγου αστοχίας μιας συγκριτικής αξιολόγησης, πρέπει να υπολογιστεί ο μέσος όρος του λόγου αστοχίας κάθε παραθύρου σε ένα ίχνος. Αυτός ο λόγος αστοχιών υπολογίζεται από το *αστοχίες κρυφής μνήμης/προσπελάσεις κρυφής μνήμης* στην LLC κατά τη διάρκεια της περιόδου 512k προσπελάσεων κάθε παραθύρου. Στη συνέχεια, ο λόγος αστοχιών κάθε ίχνους πολλαπλασιάζεται με ένα βάρος, το οποίο αντιπροσωπεύει το ποσοστό της συνολικής εκτέλεσης του αρχείου αναφοράς που προσομοιώνει αυτό το ίχνος. Το άθροισμα αυτών των αναλογιών αστοχίας πολλαπλασιαζόμενο με το βάρος του ίχνους κατασκευάζει έναν σταθμισμένο μέσο όρο, ο οποίος θα είναι η αναλογία αστοχίας ολόκληρου του αρχείου αναφοράς.



Σχήμα 3.1: Μέσος όρος λόγων αστοχίας όλων των αρχείων αναφοράς με πολιτική αντικατάστασης LRU

Όπως βλέπουμε, ο μέσος λόγος αστοχιών στην κρυφή μνήμη όλων των συγκριτικών δοκιμών βρίσκεται στο 90% για τη μικρότερη κρυφή μνήμη και συνεχίζει να μειώνεται σταδιακά, μέχρι το 55%. Αυτό είναι άμεση συνέπεια του γεγονότος ότι, κατά μέσο όρο, οι μεγαλύτε-

ρες κρυφές μνήμες οδηγούν σε χαμηλότερα ποσοστά αστοχίας, δεδομένου ότι μπορούν να αποθηκεύσουν περισσότερες πληροφορίες.

Για να συνοψίσουμε και να γίνουμε πιο ακριβείς, το σύνολο δεδομένων μας περιέχει από 1 έως 373 παράθυρα για κάθε ένα από τα 186 ίχνη που προκύπτουν από 47 συγκριτικά μέτρα. Κάθε παράθυρο περιέχει το ιστόγραμμα των αποστάσεων επαναχρησιμοποίησης και τα ποσοστά αστοχίας της κρυφής μνήμης που καταγράφηκαν κατά τη διάρκεια 512k συνεχών προσπελάσεων μνήμης κατά τη διάρκεια της εκτέλεσης του αρχείου αναφοράς. Οι λόγοι αστοχίας κρυφής μνήμης είναι αριθμοί στο εύρος $[0, 1]$ με βάση τις προσομοιωμένες τιμές του ChampSim για κάθε μία από τις διαμορφώσεις LLC που αναφέρονται στον πίνακα 3.1. Αυτές οι αποστάσεις επαναχρησιμοποίησης θα αποτελέσουν την είσοδο για τον πρώτο κλάδο του δικτύου μας και οι αναλογίες αστοχίας είναι η πραγματική έξοδος στην οποία θα προσαρμοστεί η έξοδος του δικτύου κατά την εκπαίδευση και θα συγκριθεί με αυτήν κατά τη δοκιμή.

Ενσωματώσεις

Όπως συζητήθηκε στην ενότητα 3.2, έχουμε μεταγλωττίσει τον κώδικα σε αρχεία IP. Θα χρησιμοποιήσουμε το προ-εκπαιδευμένο μοντέλο του IR2Vec για τη μετατροπή του κώδικα σε διανύσματα. Με αυτό, δημιουργούμε 47 διανύσματα 300 διαστάσεων, ένα που αντιπροσωπεύει κάθε σημείο αναφοράς. Αυτός είναι ο κλάδος μετασχηματισμού των δικτύων μας. Αυτό βοηθά το υπόλοιπο δίκτυο, μέσω του κώδικα του προγράμματος, να κατανοήσει τις ιδιότητες του προγράμματος, γεγονός που θα βοηθήσει στον υπολογισμό ενός πιο ακριβούς λόγου αστοχίας.

Το IR2Vec παράγει είτε μία ενσωμάτωση για κάθε συνάρτηση του προγράμματος είτε μία ενσωμάτωση για ολόκληρο το πρόγραμμα ταυτόχρονα. Το πλεονέκτημα της ύπαρξης μιας ενσωμάτωσης για κάθε συνάρτηση είναι η ακρίβεια ως προς το ποιο μέρος της εκτέλεσης θέλουμε να εστιάσουμε. Η ενσωμάτωση για όλο το πρόγραμμα θα μετατρέψει ολόκληρο το αρχείο αναφοράς σε ένα διάνυσμα 300 διαστάσεων. Στην περίπτωση μας, δεν γνωρίζουμε τη συνάρτηση ή το μέρος του προγράμματος που εκτελείται σε κάθε ίχνος-έτσι, χρησιμοποιούμε ένα διάνυσμα που είναι η ενσωμάτωση ολόκληρου του αρχείου αναφοράς και ένα μόνο με το διάνυσμα της κύριας συνάρτησης για κάθε αρχείο αναφοράς.

Train-test split

Μια άλλη μέριμνα είναι ο διαχωρισμός του συνόλου δεδομένων, το οποίο προκύπτει από τα τμήματα 3.2.2 και 3.2.2, σε ένα σύνολο εκπαίδευσης και ένα σύνολο δοκιμής, το οποίο δεν είναι καθόλου τειριμμένο. το σύνολο δεδομένων μας αποτελείται από πλειάδες. Όπως περιγράφεται στις ενότητες 3.2.2 και 3.2.2, κάθε πλειάδα περιέχει το ιστόγραμμα της απόστασης επαναχρησιμοποίησης ενός παραθύρου, τα ποσοστά αστοχίας, καθώς και τις ενσωματώσεις του αρχείου αναφοράς από το οποίο προέρχεται.

Οι αποστάσεις επαναχρησιμοποίησης έχουν επιλεγεί από ένα αρχείο ιχνών κάθε φορά, οπότε όλα τα στοιχεία που προκύπτουν από το ίδιο αρχείο ιχνών θα έχουν παρόμοια δομή απόστασης επαναχρησιμοποίησης. Το ίδιο ισχύει και για τις ενσωματώσεις- τις συλλέξαμε από τον κώδικα που έχει μεταγλωττιστεί από τον συγκριτικό δείκτη και όλα τα αρχεία

ιχνών που έχουν παραχθεί από τον ίδιο συγκριτικό δείκτη με διαφορετικές διαμορφώσεις θα έχουν ως είσοδο το ίδιο αρχείο ενσωμάτωσης. Για παράδειγμα, τα ίχνη 435.gromacs-111B, 435.gromacs-134B, 435.gromacs-226B & 435.gromacs-228B έχουν όλα παραχθεί από το αρχείο αναφοράς 435.gromacs από το SPEC και συνεπώς έχουν την ίδια ενσωμάτωση σε κάθε στοιχείο του συνόλου δεδομένων. Λόγω αυτής της ομοιότητας, δεν μπορούμε να έχουμε στοιχεία ενός αρχείου αναφοράς τόσο στο σύνολο εκπαίδευσης όσο και στο σύνολο δοκιμής- αυτό θα διαστρέβλωνε τα δεδομένα μας, αφού το δίκτυο θα έχει εκπαιδευτεί σε αυτή τη δομή.

Η αρχική μας σκέψη για το διαχωρισμό εκπαίδευσης-δοκιμής ήταν να διαχωρίσουμε το 20%-30% των αρχείων αναφοράς του συνόλου δεδομένων για το σύνολο δοκιμών. Όμως, δεδομένου ότι το σύνολο δεδομένων μας είναι αρκετά μικρό, αποτελούμενο μόνο από 47 αρχεία αναφοράς, ο διαχωρισμός του 20%-30% των σημείων αναφοράς για ένα σύνολο δοκιμής δεν θα ήταν βέλτιστος. Θα είχε ως αποτέλεσμα ένα σύνολο δοκιμών που θα ήταν πολύ μικρό και, πιθανότατα, δεν θα ήταν ενδεικτικό όλων των πιθανών προγραμμάτων που θα μπορούσε να αντιμετωπίσει το δίκτυο. Για το λόγο αυτό, αποφασίσαμε να χρησιμοποιήσουμε μια μέθοδο που ονομάζεται *leave-one-out cross-validation* [23]. Ξεχωρίζουμε ένα αρχείο αναφοράς κάθε φορά και προσπαθούμε να προβλέψουμε τα ποσοστά αστοχίας του με το δίκτυο που εκπαιδεύτηκε στα άλλα 46 σημεία αναφοράς. Με αυτόν τον τρόπο, διατρέχουμε τα 47 σημεία αναφοράς και λαμβάνουμε 47 αποτελέσματα, ένα για κάθε αρχείο αναφοράς του συνόλου δεδομένων.

3.2.3 Μετρικές

Για την εκπαίδευση και τη δοκιμή των δικτύων, χρησιμοποιήσαμε τις βιβλιοθήκες *TensorFlow* στην *Python*. Οι χρόνοι επεξεργασίας των δικτύων προέρχονται από τις διαδικασίες εκπαίδευσης και δοκιμής που πραγματοποιήσαμε με τη χρήση μιας *NVIDIA T4 GPU*.

Χρησιμοποιήσαμε μια μετρική μέσου τετραγωνικού σφάλματος για τη διαδικασία εκπαίδευσης. Αυτό λειτουργεί επειδή, ανεξάρτητα από τη βαρύτητα του παραθύρου στο τελικό αποτέλεσμα, θέλουμε τη μικρότερη δυνατή απόκλιση από την εκτιμώμενη τιμή του. Λειτουργεί καλύτερα από έναν σταθμισμένο μέσο όρο (όπου το βάρος ενός παραθύρου θα είναι $trace_weight * 1 / windows_per_trace$) ή μια συνάρτηση μέσου απόλυτου σφάλματος, επειδή αναγκάζει το δίκτυο να μειώσει τις μεγαλύτερες αποκλίσεις. Αυτό, με τη σειρά του, ωθεί τη συνολική μέση απόκλιση να είναι μικρότερη.

Η εκπαίδευση έγινε με έναν βελτιστοποιητή *Adam* [24], μια στοχαστική μέθοδο για τη ρύθμιση των βαρών των δικτύων για βελτιστοποίηση. Ο ρυθμός μάθησης ήταν 0,001, το μέγεθος της παρτίδας ήταν 16 και εκπαιδεύσαμε το δίκτυο για 50 εποχές. Για αυτές τις μεταβλητές, δοκιμάσαμε πολλές διαφορετικές διαμορφώσεις και κρατήσαμε αυτήν εδώ, καθώς παρείχε τα πιο ακριβή αποτελέσματα.

Για να αξιολογήσουμε πόσο καλή είναι η πρόβλεψη ενός δικτύου, θα αξιολογήσουμε το απόλυτο σφάλμα του από την πραγματική αναλογία αστοχίας που υπολογίσαμε στο 3.2.2. Για να το υπολογίσουμε αυτό, πρέπει να υπολογίσουμε το μέσο όρο της εξόδου του δικτύου για κάθε παράθυρο σε ένα ίχνος. Στη συνέχεια, πρέπει να υπολογιστεί ένας σταθμισμένος μέσος όρος σε ένα σημείο αναφοράς με το βάρος του ίχνους και να αφαιρεθεί από την

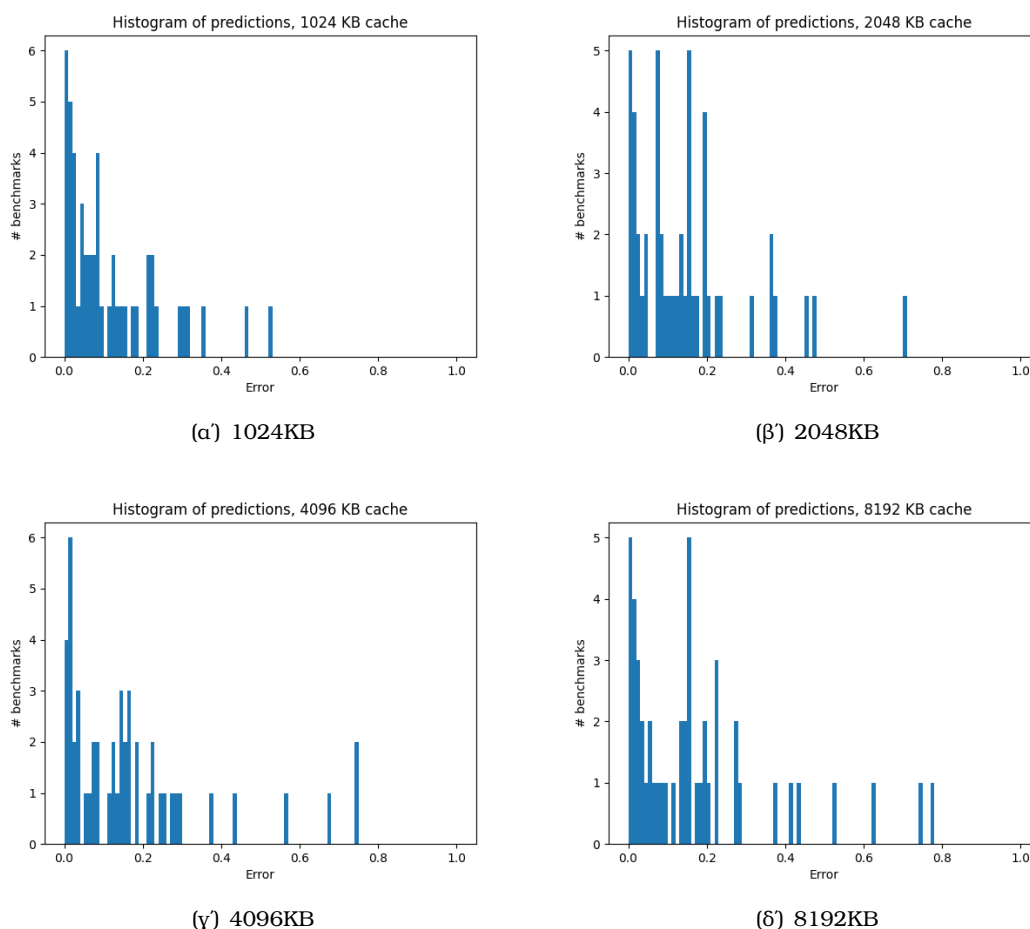
πραγματική τιμή. Η απόλυτη τιμή αυτής της αφαίρεσης θα αναφέρεται ως το απόλυτο σφάλμα πρόβλεψης. Για κάθε ένα από τα 47 σημεία αναφοράς στο σύνολο δεδομένων μας, θα υπολογιστεί ένα σφάλμα με το σύνολο εκπαίδευσης για τα άλλα 46 σημεία αναφοράς, όπως εξηγείται στο 3.2. Για να συγκρίνουμε αυτά τα σύνολα σφαλμάτων μεταξύ τους και με τις προαναφερθείσες προβλέψεις της StatCache, θα χρησιμοποιήσουμε έναν γεωμετρικό μέσο όρο. Πρέπει να αναφερθεί ότι με αυτόν τον τρόπο υπολογισμού του συνολικού αποτελέσματος μιας συγκριτικής αξιολόγησης χάνεται κάποια διακύμανση των δεδομένων στην ανάλυση. Οι διαφορές μεταξύ της πρόβλεψης και της πραγματικής τιμής ορισμένων ιχνών ποικίλλουν σημαντικά σε ορισμένα σημεία αναφοράς, έως και 70%. Επίσης, η διαφορά των παραθύρων μεταξύ πρόβλεψης και πραγματικής τιμής μπορεί να ποικίλλει εντός ενός ίχνους. Αυτές οι διαφορές εξισορροπούνται σε πολλαπλά παράθυρα και ίχνη.

3.3 Υπολογισμός StatCache

Οι έξοδοι της StatCache πρέπει να υπολογιστούν. Είναι σημαντικό να έχουμε μια ανταγωνιστική, σύγχρονη προσέγγιση πρόβλεψης που βασίζεται επίσης σε ιστογράμματα αποστάσεων επαναχρησιμοποίησης για να συγκρίνουμε τα αποτελέσματά μας. Η σύγκριση των δύο θα μας επιτρέψει να ποσοτικοποιήσουμε την επίδραση της χρήσης νευρωνικών δικτύων σε αντίθεση με μια αναλυτική προσέγγιση, δεδομένων των ίδιων δεδομένων εισόδου.

Το StatCache είναι μια πιθανολογική προσέγγιση του προβλήματος για την πολιτική αντικατάστασης LRU και οι τιμές του λαμβάνονται από τον τύπο που αναφέρεται στο κεφάλαιο 2.2. Κατά τον υπολογισμό αυτού του τύπου, λαμβάνουμε έναν λόγο αστοχίας για κάθε ένα από τα παράθυρα. Αφού τα υπολογίσουμε κατά μέσο όρο για κάθε ίχνος και υπολογίσουμε τον σταθμισμένο μέσο όρο για κάθε σημείο αναφοράς, όπως ακριβώς και προηγουμένως, υπολογίζουμε τον προβλεπόμενο λόγο αστοχίας για κάθε σημείο αναφοράς. Οι απόλυτες αποκλίσεις σφάλματος των προβλέψεων από τις πραγματικές τιμές, όπως απεικονίζονται στο σχήμα 3.2 για τα 47 αρχεία αναφοράς, φαίνεται να ακολουθούν μια αναδιπλωμένη κανονική κατανομή. Αυτό είναι αναμενόμενο αφού η προσέγγιση προσπαθεί να προβλέψει την τιμή του πραγματικού λόγου αστοχίας. Ως εκ τούτου, το απόλυτο σφάλμα αυτής της πρόβλεψης θα πρέπει να είναι το απόλυτο μιας κανονικής κατανομής γύρω από το μηδέν. Θα χρησιμοποιήσουμε τον γεωμετρικό μέσο όρο για να συνοψίσουμε τις κατανομές ως έναν αριθμό και να συγκρίνουμε τις εξόδους μεταξύ τους. Ο γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης από το μοντέλο StatCache είναι 5,4%, 8,3%, 8,7% και 8,5% για τις κρυφές μνήμες μεγέθους 1MB, 2MB, 4MB και 8MB, αντίστοιχα.

Αυτή η αναδιπλωμένη κανονική κατανομή οδηγεί σε θηκογράμματα όπως αυτά που φαίνονται στο σχήμα 3.3. Η πορτοκαλί γραμμή αντιπροσωπεύει τη διάμεσο. Το ενδοτεταρτημοριακό εύρος (IQR) αντιπροσωπεύει το 25% των τιμών σε κάθε πλευρά της διαμέσου. Τα μουστάκια έχουν μέγιστο μήκος $1,5 * IQR$ και όλες οι τιμές εκτός αυτού του εύρους θεωρούνται ακραίες. Όλα τα θηκογράμματα των απόλυτων σφαλμάτων πρόβλεψης που θα συζητήσουμε σε αυτή τη διατριβή θα έχουν παρόμοια δομή για αυτό το πρόβλημα. Ο μέσος όρος είναι πολύ χαμηλός και έτσι θα έχουμε μικρά κουτιά και κάποιες ακραίες τιμές στο θηκογράμματα.

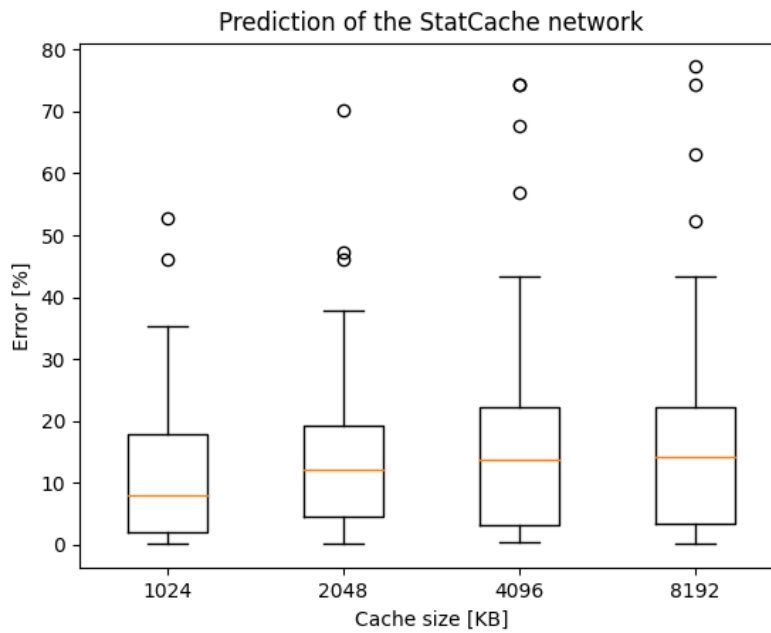


Σχήμα 3.2: Τα απόλυτα σφάλματα πρόβλεψης της StatCache για την LRU

3.4 Ρηχό πολυεπίπεδο perceptron

Πρώτον, δημιουργήσαμε ένα απλό MLP για να προβλέψουμε τα ποσοστά αστοχίας αποκλειστικά από τις αποστάσεις επαναχρησιμοποίησης. Οι αποστάσεις επαναχρησιμοποίησης είναι αρκετά δεδομένα για την πρόβλεψη τέτοιων τιμών με μεγάλη ακρίβεια, ειδικά για την πολιτική αντικατάστασης LRU. Το StatCache προβλέπει τα ποσοστά αστοχίας με χαμηλό σφάλμα, γνωρίζοντας μόνο αυτά- επομένως, υποθέτουμε ότι υπάρχουν αρκετά δεδομένα για να κάνουμε μια αρχική πρόβλεψη και να δημιουργήσουμε ένα πρώτο μοντέλο γι' αυτό.

Ένα στρώμα κανονικοποίησης μπορεί να ενισχύσει την ακρίβεια του μοντέλου μας, υπάρχουν τεράστιες αποκλίσεις μεταξύ των αριθμών στους κελιά του ιστογράμματος απόστασης επαναχρησιμοποίησης, καθώς αντικατοπτρίζουν τις πραγματικές προσπελάσεις στην κρυφή μνήμη. Μια απλή τυπική κλίμακα που προσαρμόζεται στο σύνολο εκπαίδευσης αποδεικνύεται πολύ χρήσιμη για την επίλυση αυτού του ζητήματος. Αυτός εκτελεί έναν απλό μετασχηματισμό $z = (x - u)/s$, όπου ξ είναι η σειρά μας, u είναι μια σειρά μέσων και s είναι η τυπική απόκλιση κάθε στήλης. Στη συνέχεια, εφαρμόζουμε αυτόν τον κλιμακωτή στο σύνολο εκπαίδευσης και στο σύνολο δοκιμής και συνεχίζουμε. Σημειώστε ότι αυτός ο κλιμακωτής πρέπει να εκπαιδευτεί μόνο στο σύνολο εκπαίδευσης για να μην δημιουργήσει ψευδώς καλύτερα αποτελέσματα. Ο υπολογισμός της τυπικής απόκλισης σε ολόκληρο το



Σχήμα 3.3: Τα απόλυτα σφάλματα πρόβλεψης της StatCache.

σύνολο δεδομένων θα μπορούσε να εισάγει μια μεροληψία που θα προκαλούσε καλύτερο αποτέλεσμα από το πραγματικό στο σύνολο δοκιμής.

Το μοντέλο που δημιουργήσαμε είναι ένα MLP με ένα κρυφό επίπεδο 512 νευρώνων. Λαμβάνει ως είσοδο το ιστόγραμμα της απόστασης επαναχρησιμοποίησης και παράγει τέσσερις εξόδους, μία για κάθε μέγεθος κρυφής μνήμης (1 MB, 2 MB, 4 MB και 8 MB). Αυτές οι τέσσερις εξοδοί είναι οι προβλέψεις του δικτύου για τις αστοχίες στην κρυφή μνήμη για το πώς θα ανταποκριθεί το μηχάνημα με τη συγκεκριμένη αρχιτεκτονική στο συγκεκριμένο πρόβλημα.

3.5 Δίκτυο LSTM

Ένα LSTM δέχεται ως είσοδο μια σειρά διαδοχικών τιμών, τις επαναλαμβάνει και υπολογίζει για κάθε τιμή μια έξοδο, διατηρώντας σε μια παράμετρο κρυφής κατάστασης τις τιμές που έχουν ήδη περάσει. Για το πρόβλημά μας, μετατρέψαμε τον πίνακα των αποστάσεων επαναχρησιμοποίησης σε σειρά και τον τροφοδοτήσαμε σε ένα δίκτυο LSTM. Έτσι, το δίκτυο μπορεί να υπολογίσει μια τιμή για κάθε έναν από τους κάδους του ιστογράμματος των αποστάσεων επαναχρησιμοποίησης, λαμβάνοντας υπόψη τις προηγούμενες τιμές που έχουν περάσει από τους κάδους μικρότερων αποστάσεων επαναχρησιμοποίησης.

Ομοίως, θέλουμε το δίκτυο να μπορεί να υπολογίζει μια τιμή για μια συγκεκριμένη τιμή του ιστογράμματος, δεδομένων των τιμών που ακολουθούν μετά από αυτήν από τους κάδους μεγαλύτερων αποστάσεων επαναχρησιμοποίησης. Για το λόγο αυτό, αντιστρέψαμε το ιστογράμμο της απόστασης επαναχρησιμοποίησης και το τροφοδοτήσαμε σε ένα δεύτερο επίπεδο μονάδων LSTM. Για να λειτουργήσει αυτό το δεύτερο στρώμα ως συνέχεια του πρώτου στρώματος, το κελί εξόδου και η κρυφή κατάσταση πρέπει να αρχικοποιηθούν ως οι τελικές από το πρώτο. Τώρα, το δεύτερο στρώμα θα λειτουργεί ως συνέχεια του πρώτου στρώματος με ξεχωριστά βάρη.

Αρχικά, η σκέψη πίσω από αυτό το δίκτυο ήταν να αξιοποιήσει τον μηχανισμό της προσοχής. Αυτό θα μας ωφελούσε όσον αφορά την κατανόηση του πόσο σημαντικό είναι ένα χαρακτηριστικό και πού να εστιάσουμε όταν εξετάζουμε τα δεδομένα. Με αυτό το σκεπτικό, υλοποιήσαμε ένα στρώμα προσοχής αμέσως μετά τα δύο στρώματα LSTM για να τα συνδυάσουμε. Τα στρώματα προσοχής είναι μηχανισμοί προσοχής εμπνευσμένοι από τη γνωστική προσοχή του ανθρώπινου εγκεφάλου. Ανιχνεύουν τη σημασία κάθε σημείου δεδομένων μέσα στις εξόδους μέσω μιας συνάρτησης softmax [25].

Παρόλο που η διαίσθηση της χρήσης ενός τέτοιου δικτύου φαίνεται σωστή, η εκπαίδευση και η έξοδος δεν απέδωσαν τόσο καλά όσο αναμενόταν. Οι προβλέψεις του ήταν ελαφρώς χειρότερες από αυτές του StatCache και για το λόγο αυτό δεν θα τις παρουσιάσουμε. Αυτό είναι πιθανότατα συνέπεια του ότι δεν είχαμε αρκετά δεδομένα εκπαίδευσης. Ο πίνακας προσοχής υπολογίζεται από κάθε έξοδο κάθε μονάδας LSTM εντός των στρωμάτων LSTM. Αυτό θέτει ένα τεράστιο βάρος σε κάθε μία από αυτές και περιλαμβάνει πάρα πολλά βάρη για να βελτιστοποιηθούν με την εκπαίδευση.

Εναλλακτικά, όπως απεικονίζεται στο σχήμα 3.4, επιλέξαμε να χρησιμοποιήσουμε ένα πυκνό στρώμα περσεπτρον για να κατανοήσουμε τις εξόδους των στρωμάτων LSTM. Ένα στρώμα ζονσατενατε και ένα στρώμα φλαπτεν μετατρέπουν τα σχήματα εξόδου από τα στρώματα LSTM σε σχήματα αποδεκτά από το πυκνό στρώμα. Αυτό το πυκνό δίκτυο κατανοεί καλύτερα το πρόβλημα και μπορεί να προβλέψει τα ποσοστά αστοχίας με μεγαλύτερη ακρίβεια.

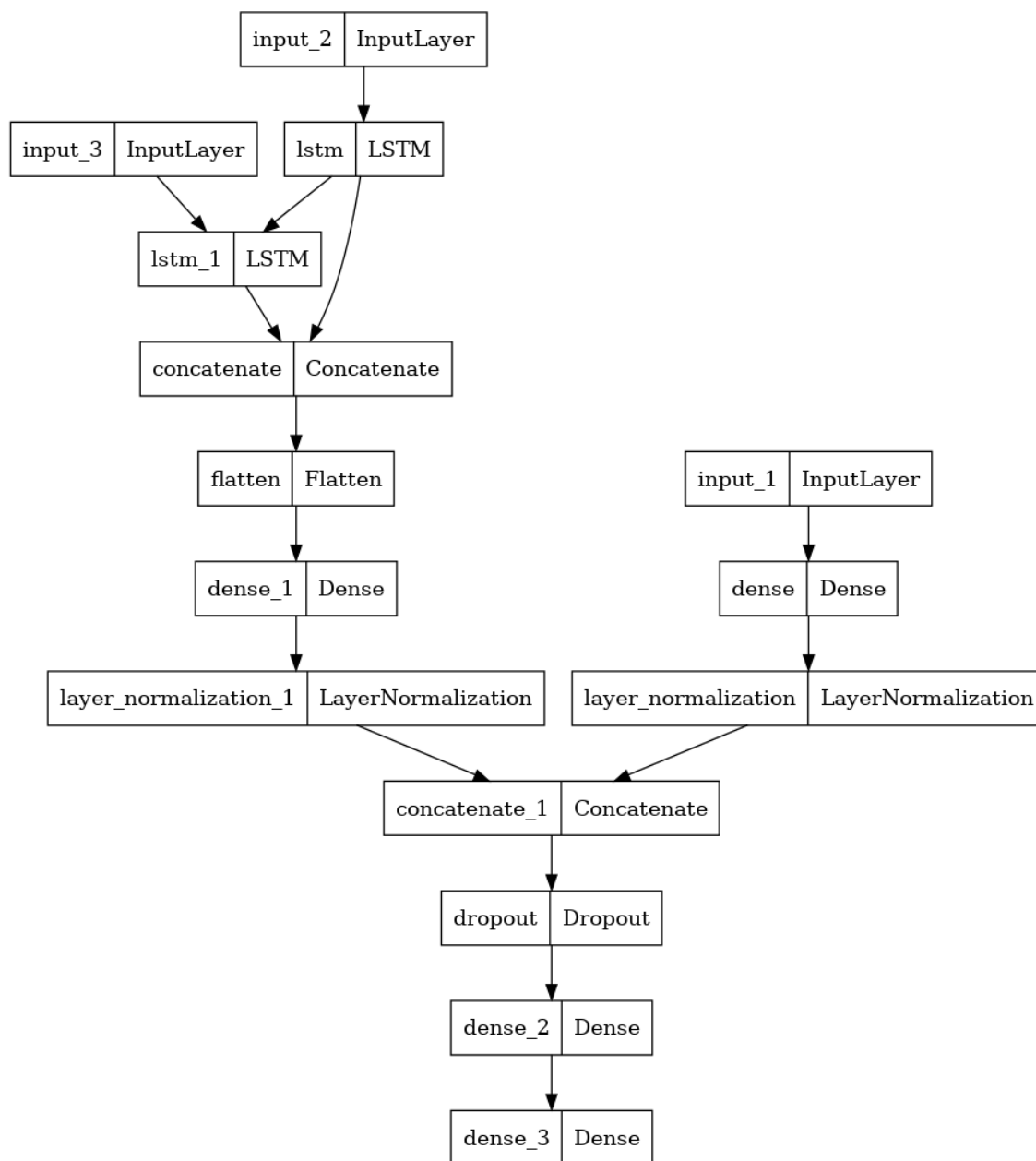
Για αυτό το δίκτυο, εκτός από τις αποστάσεις επαναχρησιμοποίησης, υλοποιήσαμε τον κλάδο μετασχηματισμού. Προσθήσαμε τις ενσωματώσεις του κώδικα σε ένα παράλληλο στρώμα περσεπτρονς, `input_1` στο σχήμα 3.4. Οι ενσωματώσεις που προκύπτουν από το `IR2Vec` δεν είναι κανονικοποιημένες, οπότε μια κανονικοποίηση με παρόμοιο κλιμακωτή όπως χρησιμοποιήσαμε για τις αποστάσεις επαναχρησιμοποίησης μας παρέχει καλύτερα αποτελέσματα. Στη συνέχεια, συνδέσαμε όλα αυτά στο τελευταίο κρυφό στρώμα `perceptron` και, τέλος, στο στρώμα εξόδου. Όλα τα πυκνά νευρωνικά δίκτυα χρησιμοποιούν μια συνάρτηση ενεργο-

ποίησης relu εκτός από το στρώμα εξόδου, το οποίο χρησιμοποιεί τη σιγμοειδή συνάρτηση, δεδομένου ότι ο λόγος αστοχίας είναι μια πιθανότητα στο εύρος $[0, 1]$.

Προσθέσαμε μερικά στρώματα κανονικοποίησης μεταξύ των μικρών πυκνών στρωμάτων των εισόδων και του μεγάλου πυκνού στρώματος της εξόδου. Αυτά φαίνεται να έχουν θετικό αντίκτυπο στο αποτέλεσμα του μοντέλου. Κάθε ένα από αυτά εκπαιδεύεται στα δεδομένα του συνόλου εκπαίδευσης και κανονικοποιεί τις εξόδους που δημιουργούνται από τα πυκνά νευρωνικά δίκτυα. Η κανονικοποίηση των στρωμάτων είναι ευεργετική, παρόλο που τα δεδομένα εισόδου είναι κανονικοποιημένα, διότι επιτρέπει ομαλότερες κλίσεις, ταχύτερη εκπαίδευση και καλύτερη ακρίβεια γενίκευσης. Δοκιμάσαμε όλες τις πιθανές τοποθεσίες αυτών των στρωμάτων στο δίκτυο και η καλύτερη απόδοση ήταν ακριβώς μετά τα μικρά, πυκνά στρώματα.

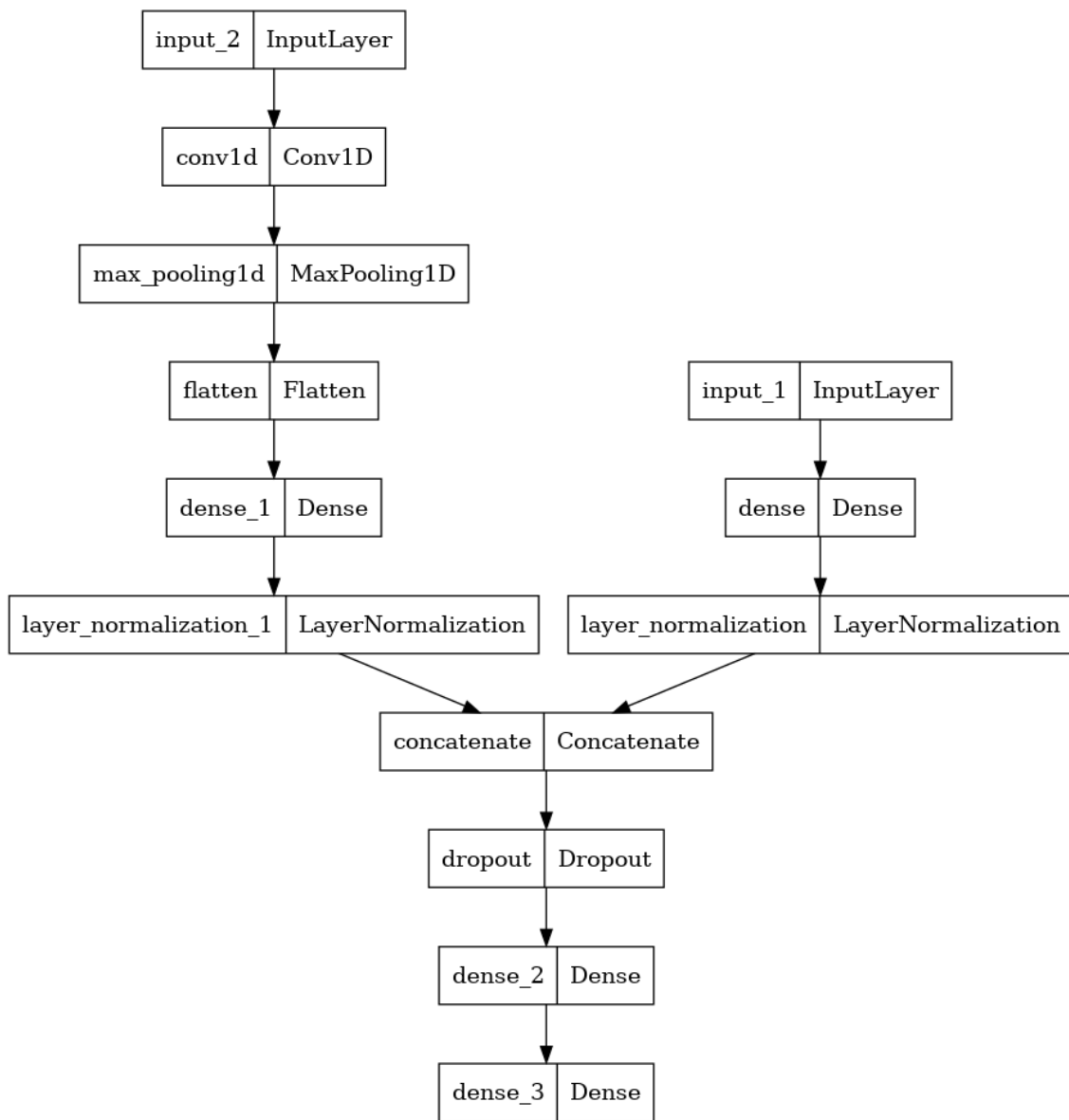
Όσον αφορά τις ιδιαιτερότητες του δικτύου, τα στρώματα LSTM μας είναι μεγέθους 4 μονάδων το καθένα. Πειραματιστήκαμε με πολλές τιμές για το δίκτυο στο εύρος 1-16 μονάδες LSTM και ανακαλύψαμε ότι οι 4 μονάδες ανά στρώμα φαίνεται να είναι η καλύτερη υπολογιστική τιμή. τα πυκνά στρώματα αμέσως μετά (dense και dense_1 στο σχήμα 3.4) έχουν μέγεθος 128 νευρώνων, ενώ το τελευταίο πυκνό στρώμα (dense_2) είναι μεγαλύτερο με μέγεθος 1024. Το μέγεθος των μικρότερων στρωμάτων δεν επηρεάζει πολύ την ακρίβεια-τα μεγέθη 128, 256 και 512 έχουν παρόμοια αποτελέσματα. Αυτό το μέγεθος του μεγαλύτερου στρώματος, από την άλλη πλευρά, λειτούργησε καλύτερα από άλλες τιμές όπως 512 και 2048.

Επιπλέον, χρησιμοποιούμε ένα στρώμα εγκατάλειψης. Αυτό το στρώμα θέτει τυχαία το 20% των εισόδων του στο μηδέν για να αποτρέψει την υπερβολική προσαρμογή.



Σχήμα 3.4: Δομή του δικτύου LSTM. *input_1* είναι οι ενσωματώσεις, *input_2* το ιστόγραμμα αποστάσεων επαναχρησιμοποίησης και *input_3* το ιστόγραμμα αποστάσεων επαναχρησιμοποίησης με αντίστροφη σειρά (από μεγαλύτερη τιμή προς την μικρότερη).

3.6 Συνελκτικό νευρωνικό δίκτυο



Σχήμα 3.5: Δομή του συνελκτικού νευρωνικού δικτύου. *input_1* είναι οι εσωματώσεις και *input_2* είναι οι αποστάσεις επαναχρησιμοποίησης.

Τα στρώματα συνέλιξης υπάρχουν για να βοηθούν το δίκτυο να εντοπίζει ομοιότητες μεταξύ χαρακτηριστικών που βρίσκονται κοντά το ένα στο άλλο, οι οποίες μπορεί να χαθούν ή να αγνοηθούν αν χρησιμοποιήσουμε ένα πυκνό δίκτυο. Οι αποστάσεις γειτονικής επαναχρησιμοποίησης με συνελίξεις θα μπορούσαν να οδηγήσουν σε καλύτερη απόδοση, καθώς η είσοδος στο πυκνό στρώμα θα είναι πιο κατατοπιστική.

Στη συνέχεια, αμέσως μετά, ένα στρώμα συγκέντρωσης έχει ως σκοπό τη μείωση της διαστατικότητας της προηγούμενης εξόδου. Το στρώμα συνελκτικού δικτύου έχει n φίλτρα, παράγοντας έτσι n αποτελέσματα ανά είσοδο ιστογράμματος. Η είσοδος ιστογράμματος είναι ένας πίνακας σχήματος $(896, 1)$ - επομένως, ο πίνακας εξόδου θα είναι $((896 - n)/step, n)$. Χρησιμοποιούμε το στρώμα συσσώρευσης μεγίστου για να μειώσουμε αυτές τις διαστάσεις

οριζόντια σε $((896 - n)/(step * K), n)$, όπου K είναι το μέγεθος του πυρήνα. Υπάρχουν εναλλακτικές λύσεις στο στρώμα συσσώρευσης για τη μείωση των διαστάσεων, μερικές από τις πιο δημοφιλείς είναι η συγκέντρωση μεγίστου και μέσου ή η χρήση μεγαλύτερου βήματος. Σε αυτό το πρόβλημα, ένα μικρό βήμα και το στρώμα συσσώρευσης μεγίστου υπολόγισαν τα καλύτερα αποτελέσματα, οπότε χρησιμοποιήσαμε αυτά.

Και πάλι, όπως ακριβώς και στην εφαρμογή LSTM, η έξοδος αυτού του στρώματος συγκέντρωσης συνδέεται με ένα πυκνό στρώμα MLP με relu και ένα στρώμα κανονικοποίησης. Το υπόλοιπο δίκτυο είναι το ίδιο όπως και προηγουμένως.

Αξίζει να σημειωθεί ότι, με αυτό το δίκτυο, οι μη κλιμακούμενες αποστάσεις επαναχρησιμοποίησης λειτουργούν καλύτερα. Ο κλιμακωτής στις μη κλιμακωτές αποστάσεις επαναχρησιμοποίησης που χρησιμοποιήσαμε προηγουμένως δεν βελτιώνει καθόλου την πρόβλεψή μας. Ακριβώς το αντίθετο, η πρόβλεψη χειροτερεύει επειδή η κλιμάκωση επηρεάζει την έξοδο της συνέλιξης- αυτό έχει ως αποτέλεσμα μικρότερες διαφορές μεταξύ των τιμών που προκύπτουν από τη συνέλιξη και κατά συνέπεια χειρότερη κατανόηση. Το δίκτυο προτιμά τις μη κλιμακωμένες αποστάσεις επαναχρησιμοποίησης που περνούν από αυτό, οι οποίες στη συνέχεια θα περάσουν από το πρώτο στρώμα perceptron. Στη συνέχεια, ένα στρώμα κανονικοποίησης θα κανονικοποιήσει τα αποτελέσματα αυτού του πυκνού στρώματος, αντί να έχει προ-κλιμακωμένα δεδομένα. Δοκιμάστηκαν όλα τα πιθανά σημεία για την εισαγωγή στρωμάτων κανονικοποίησης, με όλους τους δυνατούς συνδυασμούς, και απλά ένα στρώμα μετά το perceptron παράγει τις καλύτερες προβλέψεις.

Όσον αφορά τις ιδιαιτερότητες του δικτύου, το CNN δίκτυό μας αποτελείται από φίλτρα που επαναλαμβάνουν τις αποστάσεις επαναχρησιμοποίησης εισόδου με ένα βήμα και υπολογίζουν τη συνέλιξη πάνω σε έναν πυρήνα. Πειραματιστήκαμε με πολλούς συνδυασμούς για αυτούς τους τρεις αριθμούς (αριθμός φίλτρου, βήμα και μέγεθος πυρήνα) στις περιοχές $[1, 20]$ για κάθε αριθμό. Η δοκιμή αυτή κατέληξε στο συμπέρασμα ότι ο καλύτερος συνδυασμός είναι 6 φίλτρα, μέγεθος πυρήνα 8 και στριδε 4. Τα πυκνά στρώματα αμέσως μετά (dense και dense_1 στο σχήμα 3.5) έχουν μέγεθος 512 νευρώνων, ενώ το τελευταίο πυκνό στρώμα (dense_2) είναι μεγαλύτερο με μέγεθος 1024. Το μέγεθος των μικρότερων στρωμάτων επηρεάζει λίγο την ακρίβεια- τα μεγέθη 128 και 256 έχουν γεωμετρικό μέσο όρο απόλυτων σφαλμάτων πρόβλεψης περίπου 0,5% λιγότερο. Αυτό το μέγεθος του μεγαλύτερου στρώματος, από την άλλη πλευρά, λειτούργησε καλύτερα από άλλες τιμές όπως 512 και 2048.

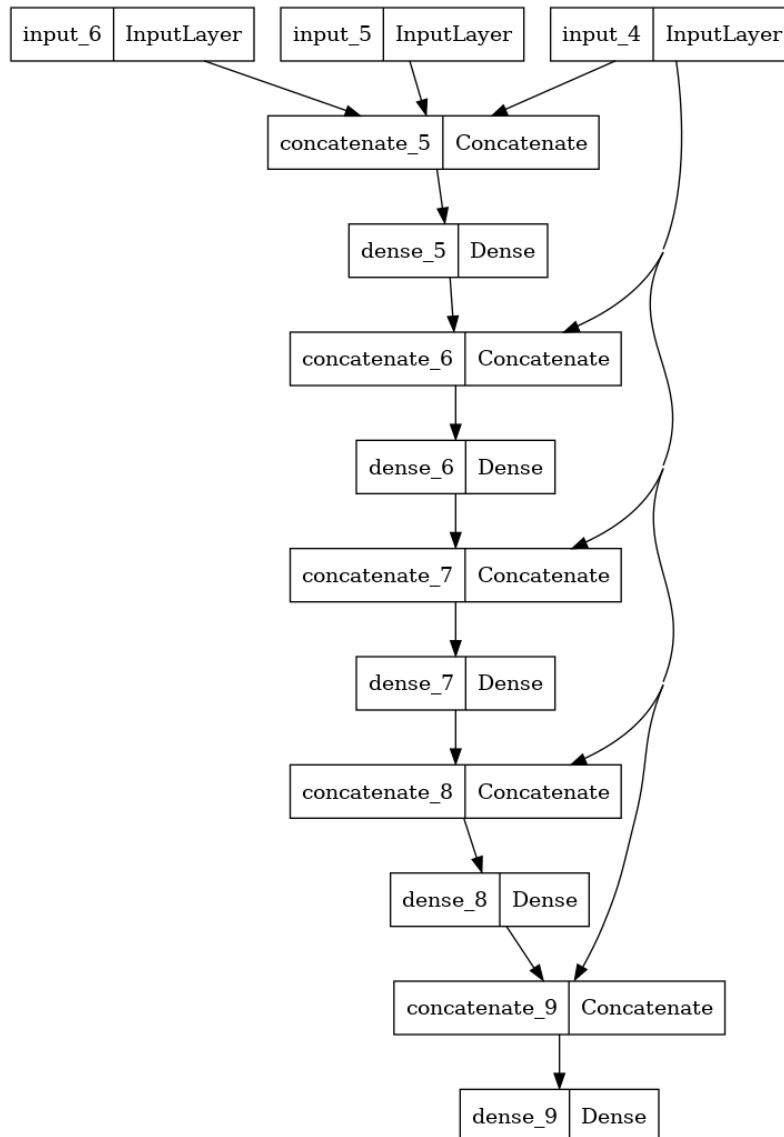
Η εκπαίδευση αυτού του δικτύου χρειάζεται κατά μέσο όρο 6 ms/step ή 3 δευτερόλεπτα/εποχή, κάτι που είναι πολύ πιο γρήγορο σε σύγκριση με τα 50 ms/step ή 15 δευτερόλεπτα/εποχή του δικτύου LSTM.

3.7 Βαθύ νευρωνικό δίκτυο

Αφού εκτιμήσαμε την ποιότητα των προηγούμενων αποτελεσμάτων και για τα 4 μεγέθη κρυφής μνήμης (1, 2, 4 και 8 MB), θελήσαμε να δούμε αν θα μπορούσαμε να δημιουργήσουμε ένα δίκτυο που να προβλέπει το ποσοστό αστοχίας σε οποιοδήποτε μέγεθος κρυφής μνήμης LLC. Αυτό έχει ως στόχο την ακριβή εκτίμηση του ποσοστού αστοχίας για κάθε μέγεθος κρυφής μνήμης και, τελικά, να δείξει ποια είναι η καλύτερη αρχιτεκτονική κρυφής

μνήμης για ένα συγκεκριμένο πρόβλημα. Για τον σκοπό αυτό, προσομοιώσαμε ξανά τα ίχνη, χρησιμοποιώντας ενδιάμεσα μεγέθη κρυφής μνήμης για να δημιουργήσουμε ένα μεγαλύτερο σύνολο δεδομένων. Αυτό περιέχει τα δεδομένα για τα μεγέθη κρυφής μνήμης LLC [768, 1024, 1536, 2048, 3152, 4096, 6044, 8192] KB με την πολιτική αντικατάστασης LRU.

Αυτό το δίκτυο μπορεί να χρησιμοποιηθεί με δύο διαφορετικούς τρόπους. Εάν έχουμε ένα πρόγραμμα για το οποίο έχουμε τα δεδομένα σε ένα ή περισσότερα μηχανήματα, μπορούμε να το προσθέσουμε στο σύνολο εκπαίδευσης και να προβλέψουμε πόσο υψηλό ποσοστό αστοχίας θα έχει με άλλα μεγέθη κρυφής μνήμης. Ο δεύτερος τρόπος χρήσης είναι για ένα νέο πρόγραμμα που δεν έχει δει το δίκτυο. Αυτός ο δεύτερος τρόπος θα είναι γενικά πιο χρήσιμος από τα τρία προηγούμενα δίκτυα, καθώς θα μπορεί να προβλέψει την αναλογία αστοχίας για οποιοδήποτε μέγεθος κρυφής μνήμης και δεν θα περιορίζεται μόνο σε αυτά στα οποία έχει εκπαιδευτεί.



Σχήμα 3.6: Βαθύ νευρωνικό δίκτυο για την πρόβλεψη άλλων μεγεθών κρυφής μνήμης. *input_4* είναι το μέγεθος της κρυφής μνήμης, *input_5* είναι οι αποστάσεις επαναχρησιμοποίησης και *input_6* είναι οι ενσωματώσεις.

Για αυτό το δίκτυο, όπως φαίνεται στο σχήμα 3.6, χρησιμοποιούμε 4 κρυφά στρώματα. Θέλουμε κάθε στρώμα να κάνει προβλέψεις ανάλογα με το μέγεθος της κρυφής μνήμης- γι' αυτό τροφοδοτούμε το μέγεθος εισόδου σε κάθε κρυφό στρώμα συνδιάζοντάς το με την έξοδο του προηγούμενου στρώματος. Κάθε ένα από τα κρυφά στρώματα έχει 512 νευρώνες- τα τέσσερα πρώτα χρησιμοποιούν τη συνάρτηση ενεργοποίησης ρελυ, ενώ το τελευταίο στρώμα και το στρώμα εξόδου χρησιμοποιούν τη σιγμοειδή συνάρτηση ενεργοποίησης. Αυτή η διαμόρφωση δοκιμάστηκε και φάνηκε να αποδίδει καλύτερα. Περισσότερα κρυφά στρώματα ή περισσότεροι νευρώνες δεν απέδιδαν καλύτερη πρόβλεψη, ίσως λόγω της δημιουργίας πολλών μεταβλητών.

Και οι τρεις εισοδοί περιέχουν μια κανονικοποιημένη έκδοση των αποστάσεων επαναχρησιμοποίησης, των ενσωματώσεων και των μεγεθών της κρυφής μνήμης. Οι αποστάσεις επαναχρησιμοποίησης και οι ενσωματώσεις κανονικοποιούνται με την ίδια μέθοδο όπως και στα άλλα δίκτυα. Το μέγεθος της εισόδου διαιρέθηκε με 512KB, με αποτέλεσμα να προκύψουν τιμές που κυμαίνονται από 1,5 - 16,0 για κάθε μία από τις κρυφές μνήμες.

Σύγκριση αποτελεσμάτων

4.1 Πρόβλεψη για άγνωστα προβλήματα

4.1.1 Πρόβλεψη για την πολιτική αντικατάστασης LRU

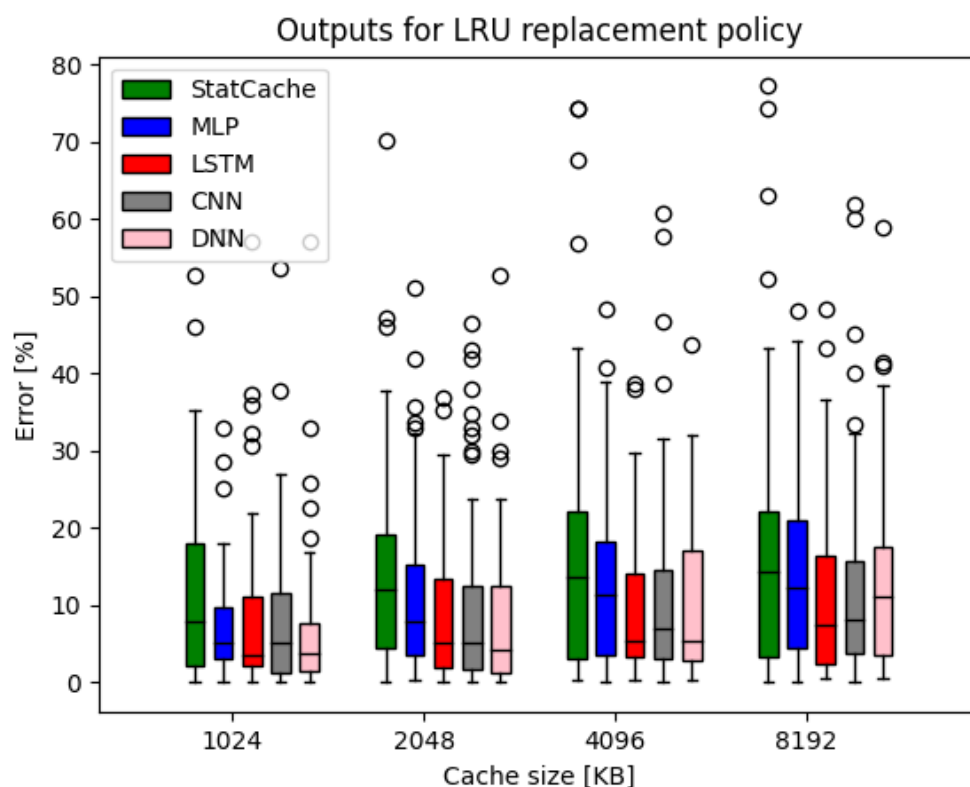
Το πρόβλημα που πρόκειται να συζητήσουμε είναι πολύ εύκολο να περιγραφεί. Προσπαθούμε να προβλέψουμε τα ποσοστά αστοχίας οποιασδήποτε εφαρμογής που συναντούν τα δίκτυά μας. Για να το δείξουμε αυτό χρησιμοποιούμε την προαναφερθείσα μέθοδο *leave-one-out cross-validation* όπου διαχωρίζουμε ένα βενσημαρκ, εκπαιδεύουμε το δίκτυο στα άλλα βενσημαρκς και προσπαθούμε να το προβλέψουμε. Με αυτόν τον τρόπο θα έχουμε την καλύτερη δυνατή πρόβλεψη για κάθε φόρτο εργασίας χωρίς να τον έχουμε δει προηγουμένως.

Η αναζήτηση της ακριβέστερης πρόβλεψης του ποσοστού αστοχίας μας οδήγησε στην εισαγωγή τεσσάρων συνολικά μοντέλων: των δικτύων MLP, CNN, LSTM και DNN. Κάθε ένα από αυτά τα μοντέλα αντιπροσωπεύει μια ξεχωριστή προσέγγιση για την όσο το δυνατόν ακριβέστερη πρόβλεψη των αναλογιών αστοχίας στην κρυφή μνήμη. Ας συγκρίνουμε και ας συζητήσουμε τις συνολικές προβλέψεις των δικτύων μας. Για το σκοπό αυτό, θα επιλέξουμε μια πολιτική αντικατάστασης και θα συγκρίνουμε τα σφάλματα πρόβλεψης που έχουμε υπολογίσει με κάθε ένα από τα δίκτυά μας.

Η LRU είναι η μόνη πολιτική αντικατάστασης για την οποία μπορούμε να συγκρίνουμε τόσο το Στατάτση όσο και τα δίκτυα DNN. Το Στατάτση επινοήθηκε και λειτουργεί μόνο με την πολιτική αντικατάστασης LRU. Το δίκτυό μας DNN είναι πιθανότατα ικανό να προβλέψει οποιαδήποτε πολιτική αντικατάστασης, αλλά θα χρειαζόμασταν περισσότερα δεδομένα για άλλες πολιτικές αντικατάστασης για να επιβεβαιώσουμε ότι πράγματι προβλέπει οποιοδήποτε μέγεθος κρυφής μνήμης. Το δίκτυο DNN από το οποίο θα παρουσιάσουμε αποτελέσματα είναι το DNN που έχει εκπαιδευτεί στα 4 μεγέθη κρυφής μνήμης (1MB, 2MB, 4MB και 8MB) με την ίδια μέθοδο διασταυρούμενης επικύρωσης *leave-one-out* όπως και τα άλλα δίκτυα.

Είναι σαφές ότι όλα τα δίκτυά μας ξεπερνούν σημαντικά τις προβλέψεις της StatCache. Οι ακραίες τιμές, τα πλαίσια και οι διάμεσοι των απόλυτων σφαλμάτων πρόβλεψης στο θηκόγραμμα του 4.1 είναι αισθητά υψηλότερα για την πρόβλεψη της StatCache. Είναι επίσης σαφώς αξιοσημείωτο ότι οι γεωμετρικοί μέσοι των σφαλμάτων πρόβλεψης στον πίνακα 4.1 είναι σαφώς χειρότεροι από οποιοδήποτε άλλο δίκτυο στα τρία μικρότερα μεγέθη κρυφής μνήμης και ίσοι με τους χειρότερους για την κρυφή μνήμη μεγέθους 8MB.

Είναι επίσης σαφώς ορατό ότι για μεγαλύτερα μεγέθη κρυφής μνήμης, τα απόλυτα σφάλ-



Σχήμα 4.1: Σφάλματα πρόβλεψης των δικτύων MLP, LSTM, CNN, DNN και StatCache για LLC με ποθιτική αντικατάσταση LRU

ματα πρόβλεψης γίνονται μεγαλύτερα. Για μεγαλύτερα μεγέθη κρυφής μνήμης, ο μέσος λόγος αστοχίας μειώνεται δραστικά. Αυτό μας οδηγεί στο συμπέρασμα ότι για μεγαλύτερες κρυφές μνήμες, παρόλο που είναι ευκολότερο να προβλέψουμε ένα εύρος όπου θα είναι τα ποσοστά αστοχίας, είναι δυσκολότερο να προβλέψουμε τα ακριβή ποσοστά αστοχίας.

Το ρηχό MLP είναι συνολικά το πιο αδύναμο από τα δίκτυά μας. Το γεγονός ότι δεν λαμβάνει τόσες πληροφορίες όσο τα άλλα δίκτυα, λόγω του ότι δεν έχει τον κλάδο του μετασηματιστή, καθώς και η χειρότερη κατανόηση των αποστάσεων επαναχρησιμοποίησης, έχουν ως αποτέλεσμα μεγαλύτερα σφάλματα από τα άλλα δίκτυα. Τα σφάλματα πρόβλεψης του έχουν υψηλότερες διαμέσους, μεγαλύτερα κουτιά και πολλές ακραίες τιμές σε σύγκριση με τα σφάλματα πρόβλεψης των δικτύων CNN και LSTM. Αυτό το βλέπουμε και από το γεγονός ότι παράγει σταθερά το δεύτερο μεγαλύτερο σφάλμα πρόβλεψης στον πίνακα 4.1. Ο στόχος αυτού του δικτύου ήταν να δείξει ότι είναι δυνατόν με ένα απλό ΜΛΠ να παράγει παρόμοια, αν όχι καλύτερα, αποτελέσματα από το StatCache χρησιμοποιώντας μόνο τις αποστάσεις επαναχρησιμοποίησης. Τα αποτελέσματα δείχνουν ότι επιτύχαμε με επιτυχία αυτόν τον στόχο.

Τα άλλα τρία δίκτυα έχουν πρόσβαση στον μετασηματισμένο κώδικα του προγράμματος, οπότε οι προβλέψεις τους είναι αναμφισβήτητα καλύτερες από αυτές του MLP. Για να αναλύσουμε τις προβλέψεις τους με μεγαλύτερη ακρίβεια, θα εξετάσουμε τα ιστογράμματα των απόλυτων σφαλμάτων πρόβλεψης 4.2. Δεδομένου ότι τα ιστογράμματα είναι παρόμοια

για άλλα μεγέθη κρυφής μνήμης ως προς το σχήμα, θα παρουσιάσουμε μόνο αυτά από το μέγεθος της κρυφής μνήμης 4MB που είναι αντιπροσωπευτικά.

Network	Cache size [MB]			
	1	2	4	8
StatCache	5.4%	8.3%	8.7%	8.5%
MLP	4.1%	7.2%	6.3%	8.5%
LSTM	3.6%	3.9%	5.3%	6.4%
CNN	3.9%	4.6%	5.5%	6.2%
DNN	3.6%	4.0%	6.0%	8.1%

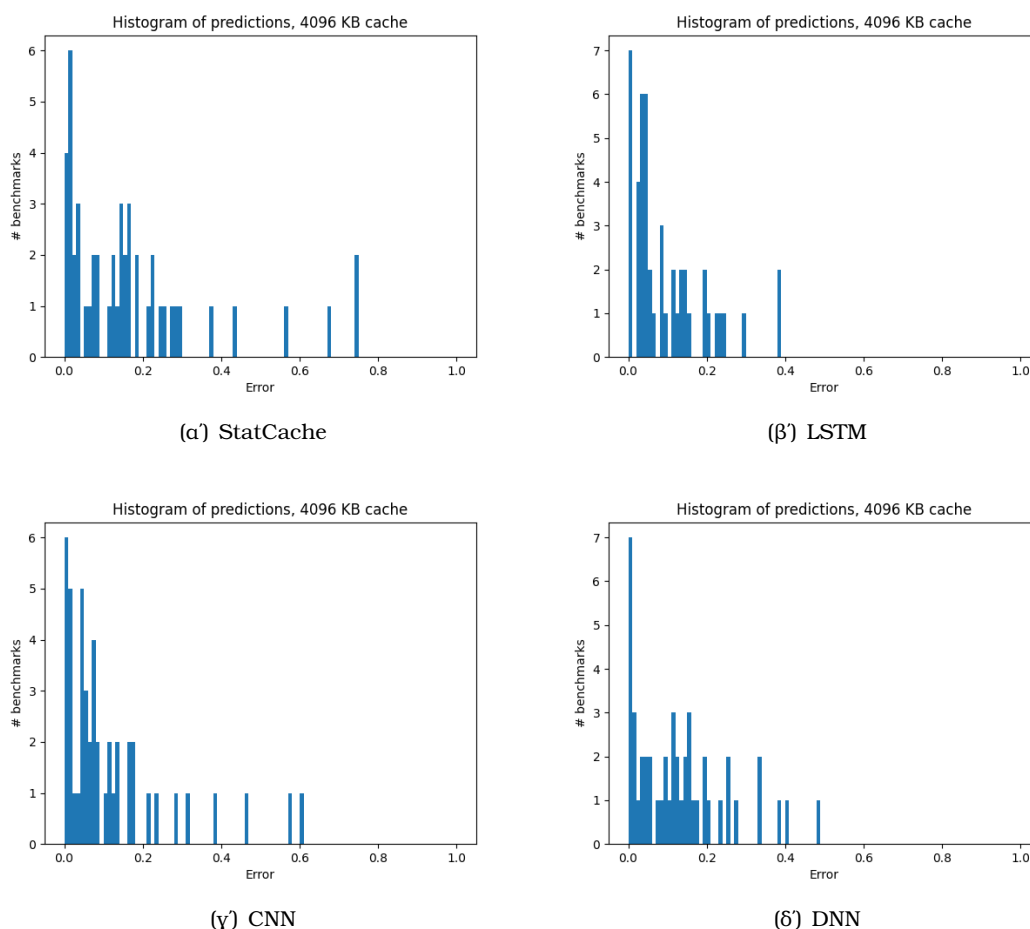
Πίνακας 4.1: Γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης για τις κρυφές μνήμες με πολιτική αντικατάστασης LRU.

Είναι προφανές ότι η πρόβλεψη της StatCache είναι χειρότερη από οποιαδήποτε άλλη πρόβλεψη. Πολλές μεγάλες ακραίες τιμές, λιγότερα σημεία αναφοράς κοντά στο μηδενικό σφάλμα και μια σχεδόν ομοιόμορφη κατανομή στο εύρος των τιμών της καθιστούν σαφές γιατί οι προβλέψεις του StatCache είναι χειρότερες. Η υψηλότερη στήλη για όλα τα άλλα ιστογράμματα είναι η μικρότερη. Τα απόλυτα σφάλματα πρόβλεψης των δικτύων LSTM και CNN έχουν πολύ παρόμοια σχήματα κοντά στο μηδέν, με πολλά σημεία αναφοράς πολύ κοντά και λιγότερα πιο μακριά. Αυτό που τα διαφοροποιεί είναι ότι το CNN έχει πολλές ακραίες τιμές. Αυτό είναι επίσης ορατό στο 4.1, όπου τα απόλυτα σφάλματα πρόβλεψης του δικτύου CNN έχουν πολύ περισσότερες ακραίες τιμές από τα σφάλματα πρόβλεψης του δικτύου LSTM.

Η αξιολόγηση της πρόβλεψης του δικτύου DNN σε σύγκριση με τα άλλα είναι λίγο πιο δύσκολη. Τα απόλυτα σφάλματα πρόβλεψής του φαίνεται να έχουν μια πιο ομοιόμορφη κατανομή στο εύρος από 0 έως 0,2 με πολύ λίγες ακραίες τιμές. Αυτό είναι ελαφρώς χειρότερο από τα απόλυτα σφάλματα πρόβλεψης του δικτύου LSTM. Όταν εξετάζουμε το αθροιστικό θηκόγραμμα 4.1, οι προβλέψεις του φαίνεται να είναι καλύτερες από αυτές του LSTM για μικρότερες τιμές του μεγέθους της κρυφής μνήμης. Οι γεωμετρικοί μέσοι όροι και για τις δύο αυτές περιπτώσεις είναι σχεδόν ίσοι. Αυτό σημαίνει ότι το δίκτυο έχει την ικανότητα να προβλέπει τα ποσοστά αστοχίας της κρυφής μνήμης εξίσου καλά με το δίκτυο LSTM, αν όχι καλύτερα, για μικρά μεγέθη κρυφής μνήμης. Αυτό σημαίνει επίσης ότι αυτό το δίκτυο με δομή DNN έχει αδυναμία για μεγαλύτερα μεγέθη κρυφής μνήμης.

4.1.2 Πρόβλεψη οποιουδήποτε μεγέθους κρυφής μνήμης για την πολιτική αντικατάστασης LRU

Αφού αναλύσαμε τα αποτελέσματα όλων των δικτύων σε αυτά τα τέσσερα μεγέθη κρυφής μνήμης από τα δίκτυά μας, θέλουμε να παρουσιάσουμε την ευελιξία του δικτύου DNN. Το δίκτυο DNN δέχεται ως είσοδο την παράμετρο του μεγέθους της κρυφής μνήμης και μπορεί να κατανοήσει την αναλογία αστοχίας οποιουδήποτε μεγέθους κρυφής μνήμης. Ως εκ τούτου, έχουμε τα μισά ρατίος περισσότερων μεγεθών κρυφής μνήμης στην LLC για να δείξουμε ότι αυτό το δίκτυο DNN μπορεί πράγματι να τα προβλέψει.

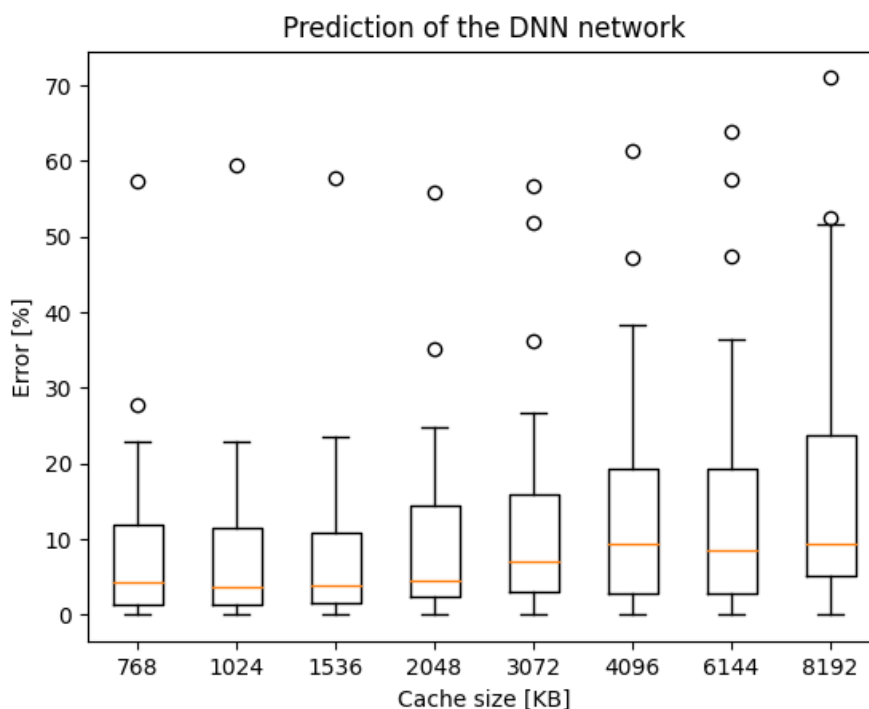


Σχήμα 4.2: *StatCache*, *LSTM*, *CNN*, and *DNN* absolute prediction errors for LLC of size 4MB with LRU replacement policy

Σε αυτή την ενότητα, προσπαθούμε να προβλέψουμε οποιοδήποτε άγνωστο πρόγραμμα για οποιοδήποτε μέγεθος κρυφής μνήμης με τη δεδομένη πολιτική αντικατάστασης. Έχουμε τα μεγέθη LLC [768, 1024, 1536, 2048, 3152, 4096, 6044, 8192] KB και πολιτική αντικατάστασης LRU. Θα χρησιμοποιήσουμε τη μέθοδο της διασταυρούμενης επικύρωσης (ζροσσ-αλιδατιον) για το σύνολο δοκιμών μας, όπως ακριβώς και στο κεφάλαιο 3.4, όπου θα διατρέξουμε τα αρχεία αναφοράς και θα τα προβλέψουμε με ένα δίκτυο που εκπαιδεύτηκε στα άλλα 46 αρχεία αναφοράς. Στο τέλος, θα έχουμε 47 προβλέψεις, μία για κάθε σημείο αναφοράς, με το δίκτυο να εκπαιδεύεται κάθε φορά σε όλα τα άλλα σημεία αναφοράς.

Για λιγότερα από 3 μεγέθη κρυφής μνήμης στο σύνολο εκπαίδευσης, η πρόβλεψη δεν έχει καμία αξία, καθώς το δίκτυο δεν μπορεί να κατανοήσει την παράμετρο του μεγέθους. Όταν το δίκτυο εκπαιδεύεται σε ένα μέγεθος κρυφής μνήμης, η παράμετρος μεγέθους δεν αλλάζει στο σύνολο εκπαίδευσης, οπότε το δίκτυο απλώς θα της αποδίδει τυχαίες τιμές. Όταν χρησιμοποιούνται δύο μεγέθη κρυφής μνήμης στο σύνολο εκπαίδευσης, το δίκτυο το κατανοεί ως δυαδικό πρόβλημα και προσπαθεί να εφαρμόσει σε αυτό μια σιγμοειδή συνάρτηση. Αυτό καθιστά τις προβλέψεις αναξιόπιστες. Έτσι, θα παρουσιάσουμε τις προβλέψεις με 3, 4 και 6 μεγέθη κρυφής μνήμης στο σύνολο εκπαίδευσης.

Το δίκτυο εκπαιδεύτηκε με τα μεγέθη της κρυφής μνήμης 1MB, 2MB και 8MB στο



Σχήμα 4.3: Απόλυτα σφάλματα πρόβλεψης του δικτύου DNN, εκπαιδευμένα σε μεγέθη κρυφής μνήμης 1MB, 2MB και 8MB.

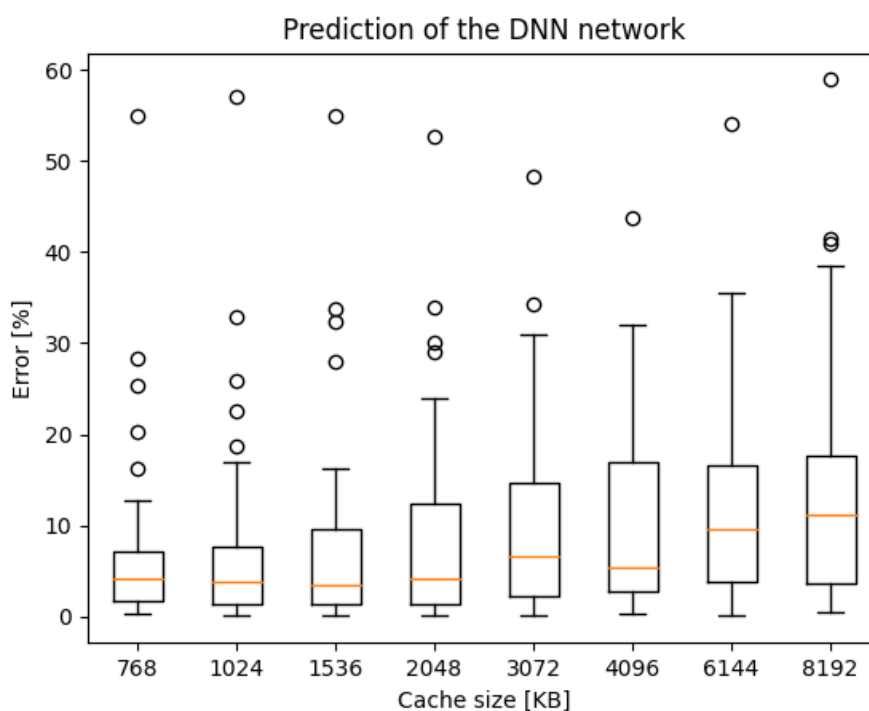
σύνολο εκπαίδευσης. Όπως βλέπουμε στο σχήμα 4.3, τα απόλυτα σφάλματα πρόβλεψης για τις χαμηλότερες κρυφές μνήμες είναι πολύ καλά. Οι διάμεσοι είναι χαμηλοί και σταθεροί. Κυρίως υπάρχει μόνο μία ακραία τιμή για αυτές τις κρυφές μνήμες, και αυτή είναι η πρόβλεψη για το σημείο αναφοράς 416.gamess- ο λόγος αστοχίας του προβλέπεται πάντα υψηλότερος και θα είναι και στο μέλλον, για καλό λόγο, η πρόβλεψη αστοχίας. Αυτό το σημείο αναφοράς αποτελείται από ένα ίχνος, το οποίο έχει μόνο 17κ προσπελάσεις στην LLC, πράγμα που σημαίνει ότι το δίκτυο διαβάζει ένα παράθυρο αποστάσεων επαναχρησιμοποίησης με 17κ προσπελάσεις και προσπαθεί να προβλέψει το μισό ρατίο του. Η πλειονότητα των παραθύρων επαναχρησιμοποιούμενων αποστάσεων στα οποία εκπαιδεύεται το δίκτυο μας αποτελείται από 512κ προσπελάσεις. Έτσι, το δίκτυο βλέπει τον χαμηλό αριθμό αποστάσεων επαναχρησιμοποίησης και υποθέτει πολλές προσπελάσεις που συμβαίνουν μόνο μία φορά και επομένως δεν έχουν αποστάσεις επαναχρησιμοποίησης. Αυτά έχουν ως αποτέλεσμα αστοχίες (sold misses) και επομένως το δίκτυο υπολογίζει υψηλότερο ποσοστό αστοχίας. Μπορούμε με ασφάλεια να αγνοήσουμε αυτή την έξοδο, δεδομένου ότι, με μια προσομοίωση 30 φορές μεγαλύτερης διάρκειας, η έξοδος πιθανότατα θα πλησίαζε περισσότερο την πραγματική τιμή.

Όσον αφορά τις μεγαλύτερες LLC, οι προβλέψεις χειροτερεύουν σταδιακά. Η ακρίβεια του δικτύου για αυτές τις κρυφές μνήμες είναι σημαντικά χειρότερη από εκείνη του δικτύου LSTM, όπως υποδηλώνουν οι γεωμετρικοί μέσοι όροι τους 4.2. Οι γεωμετρικοί μέσοι των σφαλμάτων πρόβλεψης είναι κοντά σε αυτά που προβλέπει το LSTM για μικρότερα μεγέθη κρυφής μνήμης και χειροτερεύουν δραστικά για μεγαλύτερα μεγέθη κρυφής μνήμης.

Train-set cache sizes [MB]	Cache size [KB]							
	768	1024	1536	2048	3072	4096	6144	8192
1, 2, 8	3.0%	3.1%	3.8%	5.0%	5.8%	6.2%	6.5%	8.5%
1, 2, 4, 8	3.8%	3.6%	3.1%	4.0%	5.1%	6.0%	6.9%	8.1%
0.75, 1, 2, 4, 6, 8	2.5%	3.4%	4.1%	3.5%	4.2%	5.3%	7.2%	9.7%
LSTM Network	-	3.6%	-	3.9%	-	5.3%	-	6.4%

Πίνακας 4.2: Γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης του δικτύου DNN, εκπαιδευμένο σε διαφορετικά μεγέθη κρυφής μνήμης. Μαζί με τον γεωμετρικό μέσο όρο των απόλυτων σφαλμάτων πρόβλεψης του δικτύου LSTM, για σύγκριση.

Στη συνέχεια, προσθέσαμε τα δεδομένα της κρυφής μνήμης μεγέθους 4MB στο σύνολο εκπαίδευσης. Τώρα έχουμε τα δεδομένα για τις LLC μεγέθους 1, 2, 4 και 8 MB στο σύνολο εκπαίδευσης. Αυτό σημαίνει ότι το δίκτυο εκπαιδεύεται στα ίδια ακριβώς δεδομένα με τα προηγούμενα δίκτυα που παρουσιάσαμε.

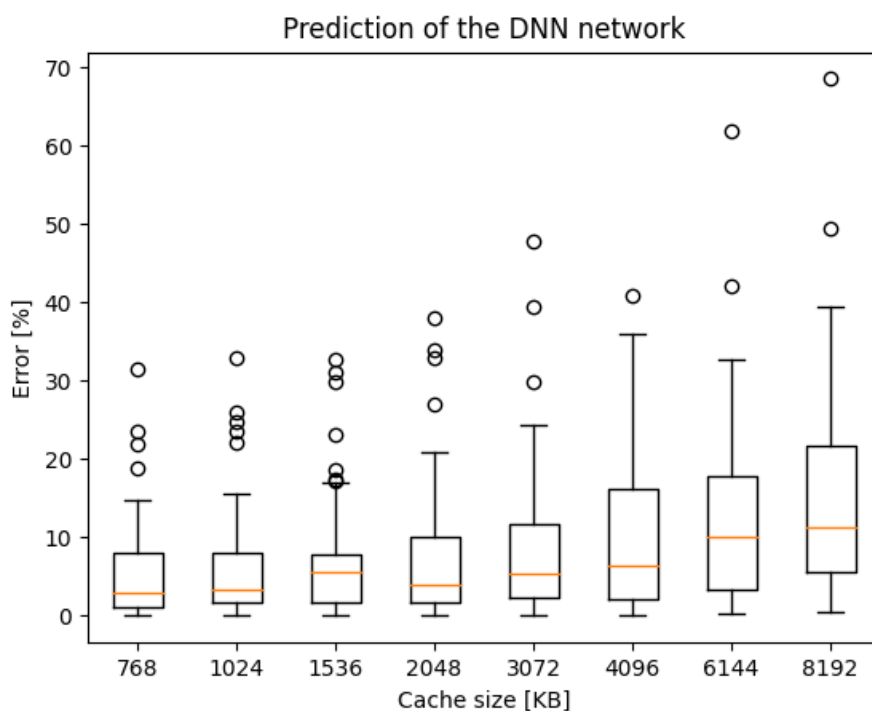


Σχήμα 4.4: Απόλυτα σφάλματα πρόβλεψης του δικτύου DNN, εκπαιδευμένα σε μεγέθη κρυφής μνήμης 1MB, 2MB, 4MB και 8MB.

Είναι ενδιαφέρον πώς αυτή η προσθήκη επηρέασε πολύ λίγο την πρόβλεψη της κρυφής μνήμης μεγέθους 4MB, στο σχήμα 4.4 ή στον πίνακα 4.2. Προφανώς, το δίκτυο μπορούσε ήδη να προβλέψει τα ποσοστά αστοχίας για το μέγεθος της κρυφής μνήμης 4MB αρκετά καλά όταν εκπαιδεύτηκε σε τρία μεγέθη κρυφής μνήμης, επομένως, η προσθήκη του το άλλαξε πολύ λίγο.

Αυτή η προσθήκη βοήθησε το δίκτυό μας να κατανοήσει καλύτερα ορισμένες από τις μικρότερες κρυφές μνήμες. Όταν εκπαιδεύσαμε για πρώτη φορά ένα δίκτυο, το εκπαιδεύσαμε μόνο για το μέγεθος της κρυφής μνήμης 2MB. Η γνώση μόνο της κρυφής μνήμης 2 MB μας έδωσε μια πρόβλεψη με γεωμετρικό μέσο όρο απόλυτων σφαλμάτων πρόβλεψης 4,7% για τη στήλη 2MB. Είναι ενδιαφέρον πώς ο γεωμετρικός μέσος όρος της κρυφής μνήμης μεγέθους 2MB (γραμμή 2 του πίνακα 4.2) είναι χαμηλότερος από αυτόν που προβλέφθηκε με μόνο αυτή την κρυφή μνήμη στο σύνολο εκπαίδευσης. Αυτό σημαίνει ότι το δίκτυο απέκτησε κάποια εικόνα για τα ποσοστά αστοχίας της κρυφής μνήμης μεγέθους 2MB από την προσθήκη της κρυφής μνήμης μεγέθους 4MB στο σύνολο εκπαίδευσης.

Τέλος, εκπαιδεύσαμε το δίκτυο σε έξι μεγέθη κρυφής μνήμης: 0,75, 1, 2, 4, 6 και 8 MB μεγέθους LLC. Αυτό γίνεται απλώς για να δούμε πώς θα συμπεριφερθεί το δίκτυο με μια τέτοια προσθήκη μεγεθών κρυφής μνήμης.



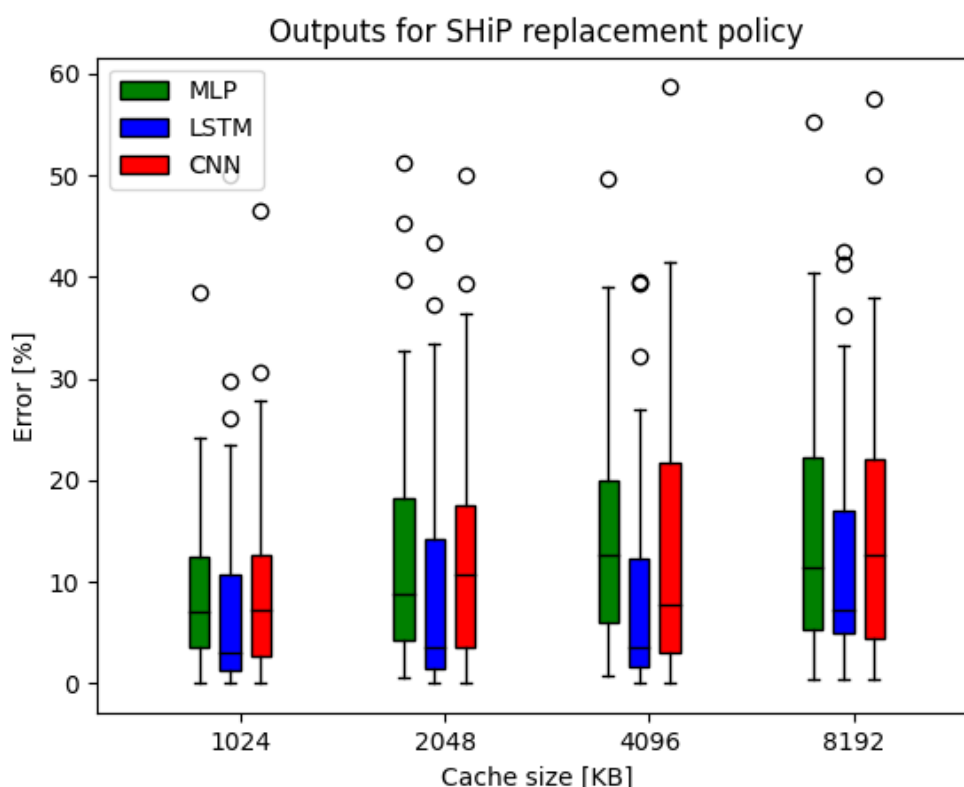
Σχήμα 4.5: Απόλυτα σφάλματα πρόβλεψης του δικτύου DNN, εκπαιδευμένα σε μεγέθη κρυφής μνήμης 0.75, 1, 2, 4, 6 και 8 MB.

Οι προβλέψεις αυτού του συνόλου εκπαίδευσης στο σχήμα 4.5 φαίνονται πολύ παρόμοιες με τα σφάλματα των προηγούμενων προβλέψεων. Τα μεγέθη και τα σχήματα των κουτιών στο θηκόγραμμα είναι αρκετά παρόμοια με εκείνο που εκπαιδεύτηκε με 4 μεγέθη κρυφής μνήμης. Η μόνη διαφορά είναι ότι οι προβλέψεις του 416.gamess έγιναν λίγο καλύτερες για μικρότερα μεγέθη κρυφής μνήμης, αλλά όχι αρκετά.

Στην τρίτη γραμμή του πίνακα 4.2 βλέπουμε ότι η πρόβλεψη της κρυφής μνήμης 768 KB έγινε ακόμη καλύτερη, καθώς και η πρόβλεψη για την κρυφή μνήμη μεγέθους 2MB. Τώρα, ο γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης για την κρυφή μνήμη μεγέθους 2MB είναι χαμηλότερος από την πρόβλεψη του δικτύου LSTM που αναφέραμε στην αρχή. Αυτό σημαίνει ότι πιθανότατα για οποιοδήποτε μέγεθος κρυφής μνήμης μικρότερο από 4MB αυτό το δίκτυο DNN θα έχει παρόμοια αποτελέσματα με το δίκτυο LSTM. Είναι επίσης ενδιαφέρον ότι ο γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης για τις κρυφές μνήμες μεγέθους 6 και 8 MB έχει αυξηθεί. Το δίκτυο εστιάζει περισσότερο στην ακριβή πρόβλεψη των μικρών μεγεθών κρυφής μνήμης και προφανώς δεν μπορεί να κατανοήσει με την ίδια ακρίβεια τα μεγαλύτερα.

4.1.3 Πρόβλεψη πολιτικής αντικατάστασης SHiP

Για τις άλλες τρεις πολιτικές αντικατάστασης, έχουμε μόνο τα σφάλματα πρόβλεψης από τα δίκτυα MLP, CNN, και LSTM. Η πρόβλεψη του StatCache έχει σχεδιαστεί για να περιγράφει τη συμπεριφορά των κρυφών μνήμης LRU και, επομένως, δεν μπορεί να εφαρμοστεί σε άλλες πολιτικές αντικατάστασης κρυφής μνήμης. Το δίκτυο DNN έχει εκπαιδευτεί και δοκιμαστεί μόνο για την πολιτική αντικατάστασης LRU- για άλλες πολιτικές αντικατάστασης, δεν έχουμε αρκετά δεδομένα για αρκετά μεγέθη κρυφής μνήμης ώστε να αξιολογήσουμε με ακρίβεια την απόδοσή του σε διάφορα μεγέθη κρυφής μνήμης.



Σχήμα 4.6: Σφάλματα πρόβλεψης των δικτύων MLP, LSTM, CNN για LLC με πολιτική αντικατάστασης SHiP.

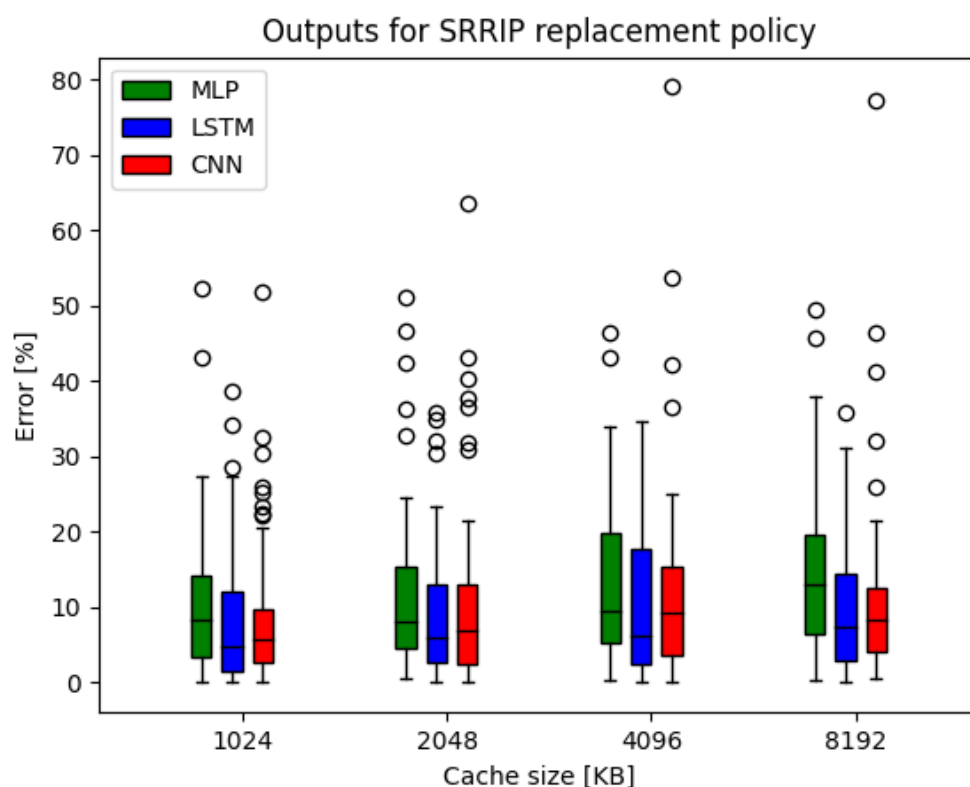
Network	Cache size [MB]			
	1	2	4	8
MLP	5.6%	7.7%	9.8%	8.6%
LSTM	2.7%	4.1%	3.7%	7.6%
CNN	5.3%	7.3%	5.8%	9.4%

Πίνακας 4.3: Γεωμετρικός μέσος όρος των σφαλμάτων πρόβλεψης για τις κρυφές μνήμες με πολιτική αντικατάστασης SHiP.

Η καλύτερη πρόβλεψη όταν εξετάζουμε τα σφάλματα πρόβλεψης, σχήμα 4.6, ή τους γεωμετρικούς μέσους όρους τους, πίνακας 4.3, είναι σαφώς η πρόβλεψη LSTM. Όλοι οι

μέσοι όροι είναι πιο κοντά στο μηδέν από οποιονδήποτε άλλο και τα θηκογράμματα φαίνονται καλύτερα για όλα τα μεγέθη της κρυφής μνήμης. Οι γεωμετρικοί μέσοι του δικτύου είναι τουλάχιστον 2% καλύτεροι από τους γεωμετρικούς μέσους του μικρότερου από τα άλλα δύο δίκτυα. Επίσης, δεν αποκλίνουν περισσότερο από 1% από τους γεωμετρικούς μέσους των σφαλμάτων πρόβλεψης του LSTM για την πολιτική αντικατάστασης LRU. Αυτό σημαίνει ότι έχουν το ίδιο εύρος με αυτά που προβλέπονται για LRU και δείχνουν τη σταθερότητα αυτού του δικτύου παρά την αλλαγή της πολιτικής αντικατάστασης. Τα σφάλματα πρόβλεψης από το δίκτυο CNN για αυτή την πολιτική αντικατάστασης είναι πολύ παρόμοια με τα σφάλματα πρόβλεψης του δικτύου MLP, γεγονός που δείχνει ότι το δίκτυο CNN παρουσιάζει κάποια αδυναμία για αυτή την πολιτική αντικατάστασης. Αυτό εξακολουθεί να εγείρει το ερώτημα, τι κάνει το δίκτυο LSTM να προβλέπει τόσο πολύ καλύτερα από τα άλλα δύο σε αυτή την πολιτική αντικατάστασης.

4.1.4 Πρόβλεψη πολιτικής αντικατάστασης SRRIP



Σχήμα 4.7: Σφάλματα πρόβλεψης των δικτύων MLP, LSTM, CNN για LLC με πολιτική αντικατάστασης SRRIP.

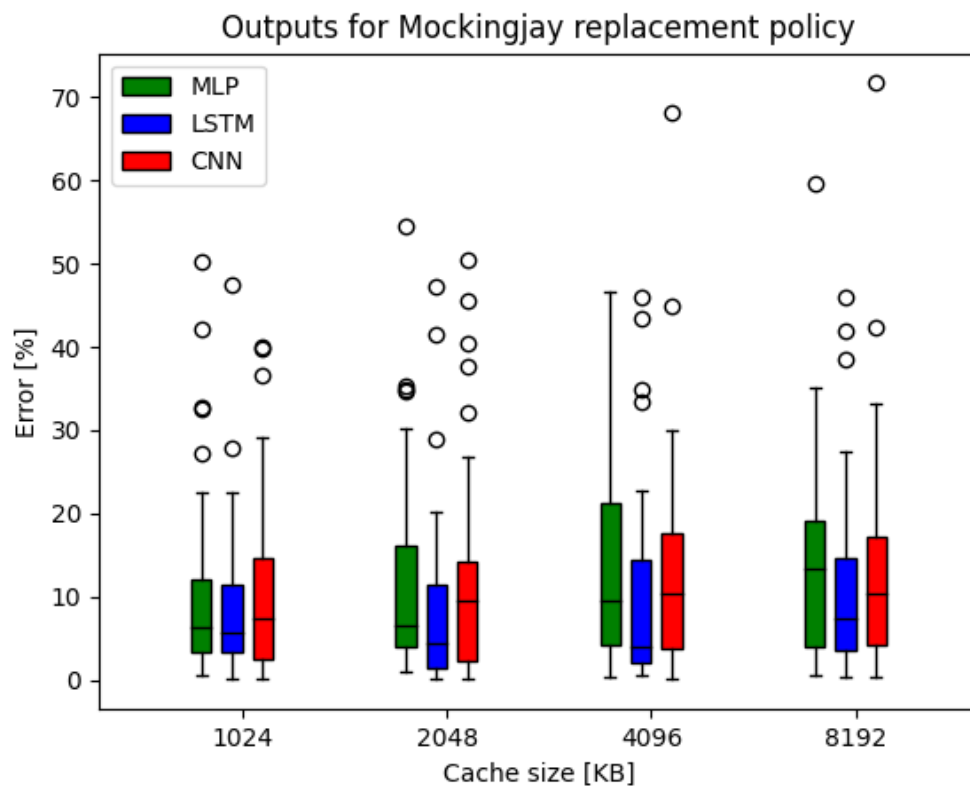
Και πάλι, οι προβλέψεις του δικτύου LSTM φαίνεται να είναι οι ισχυρότερες. Τα σφάλματα πρόβλεψης από το δίκτυο CNN είναι καλύτερα από αυτά που είχαμε για το SHiP και μοιάζουν με τα σφάλματα πρόβλεψης του δικτύου LSTM. Έχουν μικρότερα πλαίσια στο θηκογράμματα, που σημαίνει ότι το σφάλμα πρόβλεψης είναι αρκετά σταθερό. Παρόλα αυτά, τα σφάλματα πρόβλεψης του δικτύου LSTM είναι καλύτερα όσον αφορά τους μέσους και

Network	Cache size [MB]			
	1	2	4	8
MLP	5.7%	8.0%	8.6%	9.3%
LSTM	3.1%	4.6%	5.5%	5.7%
CNN	4.7%	5.2%	6.4%	7.4%

Πίνακας 4.4: Γεωμετρικός μέσος όρος των σφαλμάτων πρόβλεψης για τις κρυφές μνήμες με πολιτική αντικατάστασης SRRIP.

τους γεωμετρικούς μέσους. Αυτό που αξίζει επίσης να αναφερθεί είναι ότι πρόκειται για τον χαμηλότερο γεωμετρικό μέσο των σφαλμάτων πρόβλεψης που είχαμε για την κρυφή μνήμη μεγέθους 8MB.

4.1.5 Πρόβλεψη πολιτικής αντικατάστασης Mockingjay



Σχήμα 4.8: Σφάλματα πρόβλεψης των δικτύων MLP, LSTM, CNN για LLC με πολιτική αντικατάστασης Mockingjay.

Network	Cache size [MB]			
	1	2	4	8
MLP	6.2%	7.8%	9.0%	8.8%
LSTM	4.9%	3.6%	5.3%	6.2%
CNN	5.3%	6.2%	7.6%	8.3%

Πίνακας 4.5: Γεωμετρικός μέσος όρος των σφαλμάτων πρόβλεψης για τις κρυφές μνήμες με πολιτική αντικατάστασης Mockingjay.

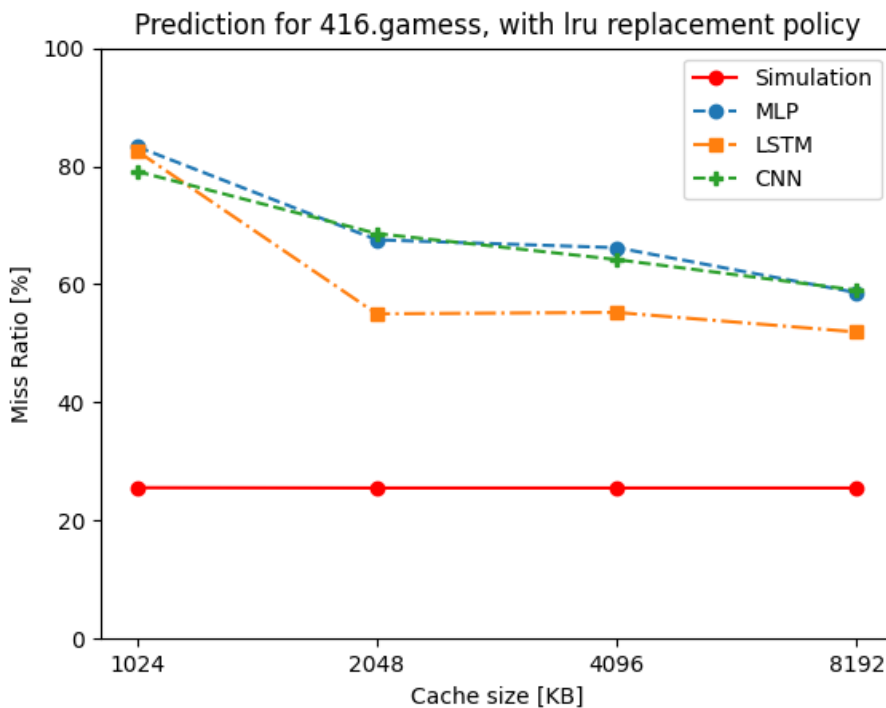
Τέλος, υπάρχει η πολιτική αντικατάστασης του Mockingjay. Όπως και προηγουμένως, το δίκτυο MLP είναι σαφώς χειρότερο από τα άλλα δύο δίκτυα στην πρόβλεψη των κρυφών μνήμης με αυτή την πολιτική αντικατάστασης. Το δίκτυο LSTM υπερέχει σημαντικά έναντι των άλλων δικτύων. Κάθε πλαίσιο στο θηκόγραμμα των σφαλμάτων πρόβλεψης από το δίκτυο LSTM είναι μικρότερο, χαμηλότερο και έχει μικρότερο μέσο όρο από τα άλλα.

Το Mockingjay είναι μια πολιτική αντικατάστασης κρυφής μνήμης που βασίζεται στις αποστάσεις επαναχρησιμοποίησης. Τις παρακολουθεί, υπολογίζει την πιθανότητα επαναχρησιμοποίησης και στη συνέχεια αποφασίζει αν ένα αντικείμενο θα παραμείνει στην κρυφή μνήμη. Αυτό θα μας οδηγήσει στο συμπέρασμα ότι μια διαδικασία όπως ένα διπλό στρώμα

LSTM βοηθά το δίκτυο να κατανοήσει καλύτερα τις αποστάσεις επαναχρησιμοποίησης και πώς θα συμπεριφερθεί η κρυφή μνήμη με αυτή την πολιτική αντικατάστασης. Αλλά το δίκτυο LSTM δεν προβλέπει το `Mockingjay` σημαντικά καλύτερα από οποιαδήποτε άλλη πολιτική αντικατάστασης. Αυτό μας οδηγεί στο συμπέρασμα ότι υπάρχουν ακόμη περιθώρια βελτίωσης όσον αφορά αυτό το δίκτυο.

4.1.6 Ανάλυση συγκεκριμένων εφαρμογών

Αυτή η ενότητα της διατριβής έχει ως στόχο να αναλύσει και να συγκρίνει τα αποτελέσματα των προαναφερθέντων δικτύων. Θα συγκρίνουμε και θα συζητήσουμε τα πλεονεκτήματα και τις αδυναμίες των δικτύων μας, δίνοντας έμφαση στα σχετικά πλεονεκτήματά τους σε πραγματικές εφαρμογές. Ας ξεκινήσουμε τη σύγκριση των αποτελεσμάτων αξιολογώντας τις προβλέψεις των δικτύων μας για ορισμένα συγκεκριμένα αρχεία αναφοράς.



Σχήμα 4.9: Προβλέψεις δικτύου για `416.gamess` με πολιτική αντικατάστασης LRU.

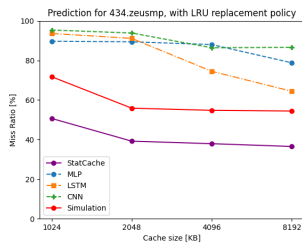
Πρώτα απ' όλα, ας ασχοληθούμε με το σημείο αναφοράς με τις περισσότερες λανθασμένες προβλέψεις στο σύνολο δεδομένων μας. Το σημείο αναφοράς `416.games` αποτελείται από ένα ίχνος με 17κ προσπελάσεις μέσα σε αυτό. Η πλειονότητα των παραθύρων στα οποία εκπαιδεύεται το δίκτυό μας αποτελείται από 512κ προσπελάσεις. Όταν έρχεται αντιμέτωπο με ένα παράθυρο αυτού του μεγέθους, το δίκτυό μας υποθέτει ότι οι περισσότερες προσπελάσεις που συμβαίνουν είναι `cold misses` και, ως εκ τούτου, δεν έχουν αποστάσεις επαναχρησιμοποίησης. Ως εκ τούτου, η πρόβλεψη είναι πολύ υψηλότερη από την πραγματική τιμή. Όπως βλέπουμε παραπάνω, στο σχήμα 4.9, τα τρία πρώτα δίκτυα προβλέπουν πολύ υψηλότερο ποσοστό αστοχίας από το πραγματικό. Αυτός ο τύπος αναφοράς μπορεί να αγνοηθεί με ασφάλεια, καθώς, με μια μεγαλύτερη προσομοίωση, το αποτέλεσμα θα είναι πιο ακριβές.

Περνάμε σε μερικά από τα χειρότερα προβλεπόμενα σημεία αναφοράς από όλα τα δίκτυά μας. Στο σχήμα 4.10 παρουσιάζουμε τρία από τα σημεία αναφοράς για τα οποία έχουμε τις μεγαλύτερες αποκλίσεις μεταξύ της προβλεπόμενης και της προσομοιωμένης τιμής. Αυτά τα τρία σημεία αναφοράς είναι τα 401.bzip2, 434.zeusmp και 456.hmmmer. Όλα τους είναι ιδιαίτερα εντατικά προγράμματα με εκατομμύρια προσπελάσεις το καθένα. Παρουσιάζουμε τις εξόδους των δικτύων μας για τις τέσσερις πολιτικές αντικατάστασης. Για την LRU, δεδομένου ότι διαχωρίσαμε τις προβλέψεις των StatCache, MLP, CNN και LSTM από τα δίκτυα DNN για να διατηρήσουμε τη σαφήνεια. Οι προβλέψεις των DNN (διαγράμματα d, e και f) επισημαίνονται ως DNN-i, όπου i είναι ο αριθμός των μεγεθών της κρυφής μνήμης εκπαίδευσης του δικτύου. Είναι σαφές ότι όλα τα δίκτυα DNN-1 είναι τυχαία και αυτό οφείλεται στο γεγονός ότι το δίκτυο δεν κατανοεί την παράμετρο μεγέθους, καθώς έχει εκπαιδευτεί μόνο σε μία παράμετρο μεγέθους.

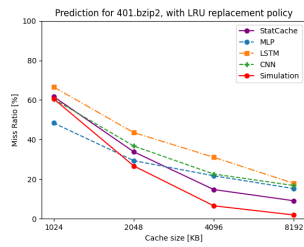
Το 434.zeusmp είναι ένα πρόγραμμα που προσομοιώνει αστροφυσικά φαινόμενα με βάση τον κώδικα υπολογιστικής ρευστοδυναμικής ZEUS-MP. Είναι σαφές ότι η πρόβλεψη από το StatCache για αυτό το σημείο αναφοράς είναι η μόνη που είναι χαμηλότερη από τις πραγματικές προσομοιωμένες αναλογίες αστοχίας. Όλες οι προβλέψεις από τα δίκτυα που προτείναμε φαίνεται να συμφωνούν ότι ο λόγος αστοχίας θα πρέπει να είναι σημαντικά υψηλότερος από τον προσομοιωμένο. Παρόλο που αυτό ισχύει για τα περισσότερα δίκτυα, το LSTM φαίνεται να έχει κάνει μια γενναία προσπάθεια να προβλέψει με ακρίβεια την αναλογία αστοχίας για την πολιτική αντικατάστασης Mockingjay (διάγραμμα m).

Το 401.bzip2 είναι ένας αλγόριθμος συμπίεσης που συμπιέζει μια σειρά αρχείων καθ' όλη τη διάρκεια της εκτέλεσής του. Η πρόβλεψη της StatCache για αυτό το αρχείο αναφοράς είναι πολύ ακριβής, ξεπερνώντας τα δίκτυά μας κατά πολύ. Η λανθασμένη πρόβλεψη των δικτύων μας για αυτό το αρχείο αναφοράς δεν είναι τόσο κακή όσο για τα άλλα δύο αρχεία αναφοράς. Η πρόβλεψη των ποσοστών αστοχίας είναι πάντα ακριβής για το μικρότερο μέγεθος κρυφής μνήμης και στη συνέχεια, καθώς τα μεγέθη κρυφής μνήμης γίνονται μεγαλύτερα, αποκλίνει.

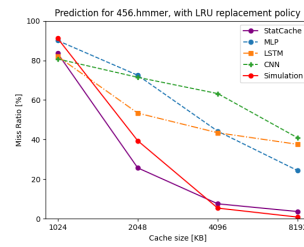
Τέλος, το αρχείο 456.hmmmer είναι ένας υπολογισμός των προφίλ κρυφών μοντέλων Μάρκοβ για να γίνει ευαίσθητη αναζήτηση σε βάσεις δεδομένων χρησιμοποιώντας στατιστικές περιγραφές της συναίνεσης μιας οικογένειας ακολουθιών. Και πάλι, η πρόβλεψη της Stat-cache είναι πολύ καλύτερη για αυτό το σημείο αναφοράς. Η απότομη πτώση της αναλογίας αστοχίας για μεγάλα μεγέθη κρυφής μνήμης φαίνεται να μπερδεύει τα δίκτυά μας, αφού προβλέπουν πάντα υψηλότερα για τα μεγαλύτερα μεγέθη κρυφής μνήμης, παρόλο που για τα μικρότερα τα βρίσκουν σωστά. Είναι ενδιαφέρον πώς το δίκτυο DNN (διάγραμμα f) δημιουργεί φαινομενικά το ίδιο σχήμα πρόβλεψης όταν εκπαιδεύεται με 1, 2 ή 3 μεγέθη κρυφής μνήμης, από το τέταρτο και πάνω. Από την άλλη πλευρά, φαίνεται να κατανοεί καλύτερα το σχήμα των συγκριτικών τιμών και να κινείται για να το ξεπεράσει. Το LSTM φαίνεται επίσης να έχει το σωστό σχήμα σε σύγκριση με τα άλλα δύο δίκτυα, αλλά δεν φαίνεται να καταλαβαίνει ακριβώς την απότομη καμπύλη.



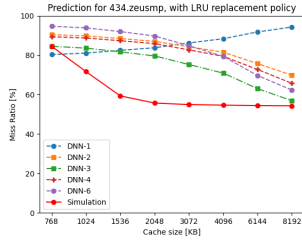
(α) 434.zeusmp, LRU



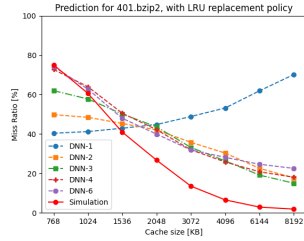
(β) 401.bzipp2, LRU



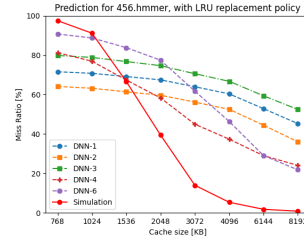
(γ) 456.hmmer, LRU



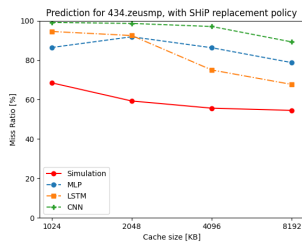
(δ) 434.zeusmp, LRU



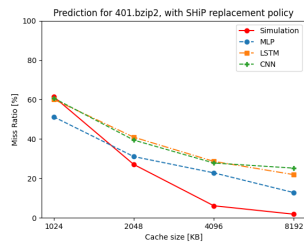
(ε) 401.bzipp2, LRU



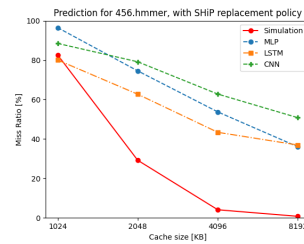
(ζ) 456.hmmer, LRU



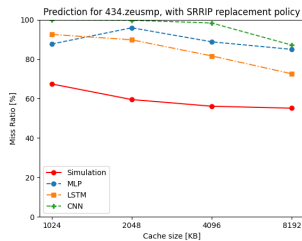
(ζ) 434.zeusmp, SHiP



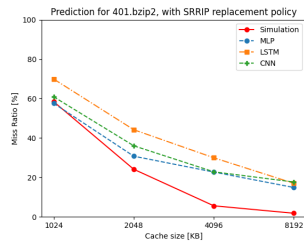
(η) 401.bzipp2, SHiP



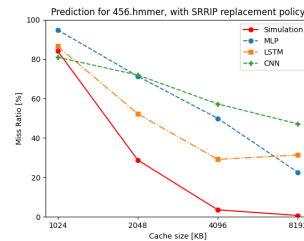
(θ) 456.hmmer, SHiP



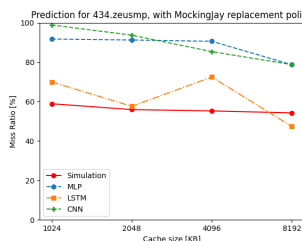
(ι) 434.zeusmp, SRRIP



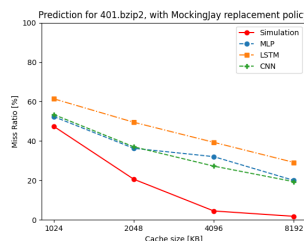
(ια) 401.bzipp2, SRRIP



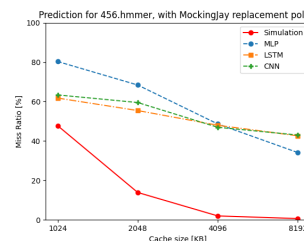
(ιβ) 456.hmmer, SRRIP



(ιγ) 434.zeusmp, Mockingjay

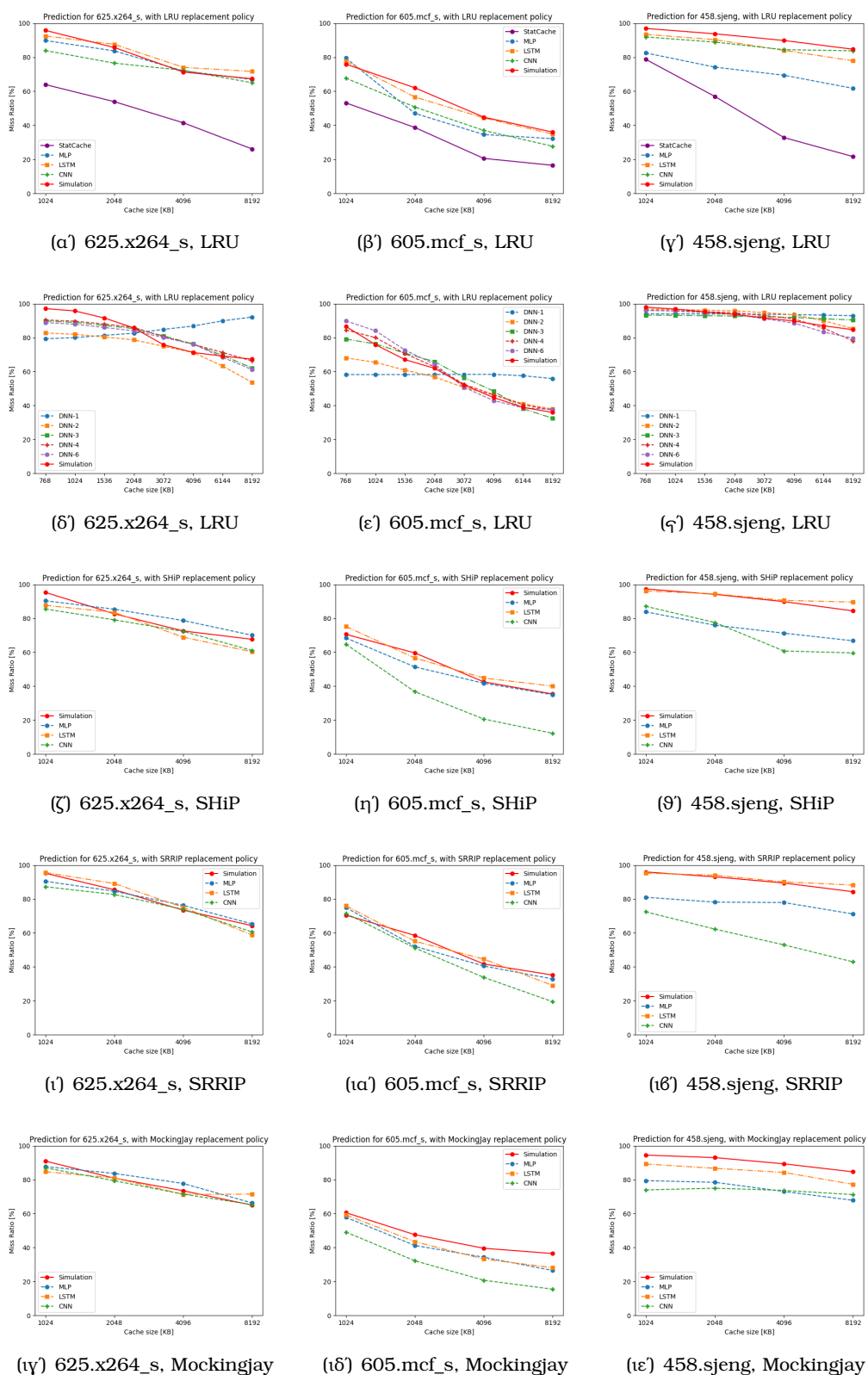


(ιδ) 401.bzipp2, Mockingjay



(ιε) 456.hmmer, Mockingjay

Σχήμα 4.10: Μερικές από τις χειρότερες προβλέψεις αρχείων αναφοράς.



Σχήμα 4.11: Μερικές από τις καλύτερες προβλέψεις αρχείου αναφοράς.

Ορισμένα από τα καλύτερα προβλεπόμενα αρχεία παρουσιάζονται στο σχήμα 4.11. Αυτή τη φορά, οι προβλέψεις από τα δίκτυα είναι πολύ κοντά στα ποσοστά αστοχίας της κρυφής μνήμης της προσομοίωσης.

Τα τρία αρχεία αναφορά που παρουσιάζουμε έχουν μερικές από τις καλύτερες προβλέψεις από τα δίκτυά μας. Προφανώς δεν είναι όλα τα αρχεία αναφοράς με καλές προβλέψεις, απλώς μερικά ενδεικτικά. Είναι ενδιαφέρον πώς, για σχεδόν όλες αυτές τις προβλέψεις, η καλύτερη πρόβλεψη είναι αυτή του δικτύου LSTM. Κατανοεί αυτά τα προβλήματα σχεδόν τέλεια, έχοντας ελάχιστες αποκλίσεις από τις πραγματικές τιμές. Δείχνει πραγματικά πώς η καλύτερη κατανόηση του προβλήματος εφαρμόζεται στην πρόβλεψη.

Το CNN και το MLP συνολικά φαίνεται να έχουν μικρές αποκλίσεις από τις τιμές της προσομοίωσης, μεγαλύτερες από αυτές του δικτύου LSTM. Το δίκτυο DNN φαίνεται να κατανοεί επίσης το πρόβλημα, και με περισσότερα δεδομένα, οι προβλέψεις γίνονται επίσης καλύτερες. Και πάλι, έχουμε το μη εκπαιδευμένο διάγραμμα DNN-1 που εμφανώς δεν καταλαβαίνει την παράμετρο του μεγέθους.

Τέλος, το StatCache υπερτερεί σημαντικά σε αυτές τις προβλέψεις. Οι προβλέψεις του StatCache δεν είναι κακές για όλα αυτά τα σημεία αναφοράς. Μπορεί να είναι κακές για το σημείο αναφοράς 458.sjeng, αλλά για τα υπόλοιπα, οι προβλέψεις των δικτύων μας είναι πολύ πιο ακριβείς.

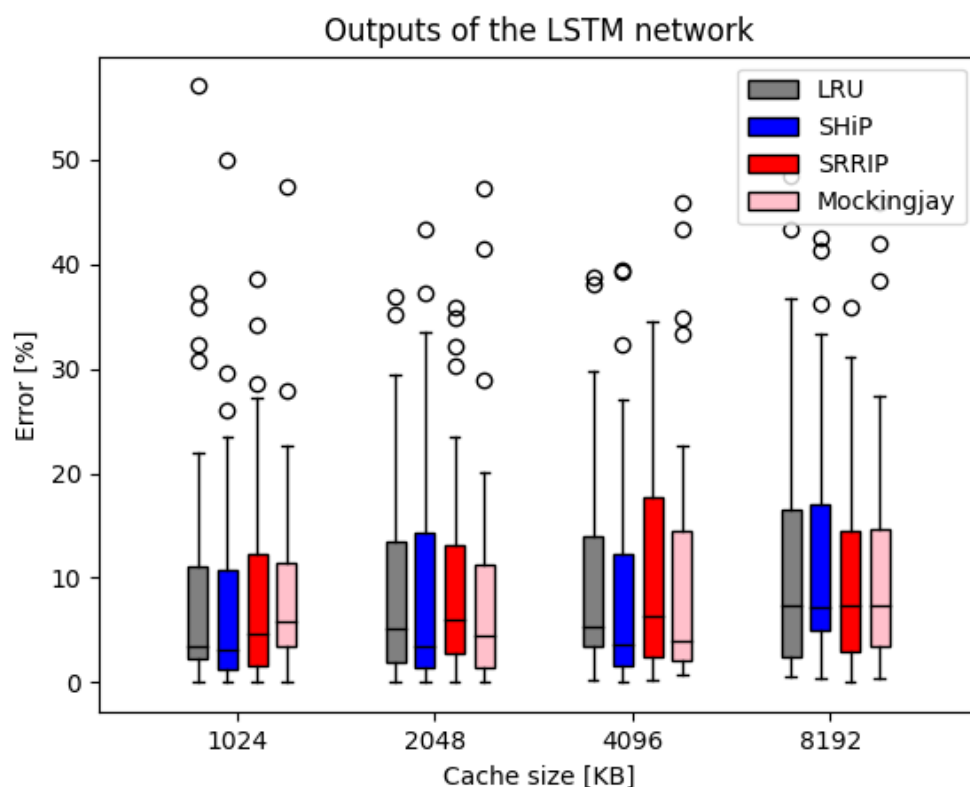
4.1.7 Συμπεράσματα

Συμπερασματικά, το δίκτυο LSTM προβλέπει σταθερά τα πιο ακριβή αποτελέσματα από τα δίκτυα που προτείναμε. Όπως βλέπουμε στο σχήμα 4.12 τα απόλυτα σφάλματα πρόβλεψής του είναι εξίσου καλά για οποιαδήποτε πολιτική αντικατάστασης. Τα αποτελέσματά του έδειξαν ότι μπορεί να προβλέψει με συνέπεια οποιοδήποτε μέγεθος κρυφής μνήμης στο οποίο εκπαιδεύεται με πολύ χαμηλούς γεωμετρικούς μέσους όρους σφαλμάτων πρόβλεψης. Έδειξε επίσης ότι είναι πολύ συνεπές στις προβλέψεις της, καθώς οι γεωμετρικοί μέσοι των σφαλμάτων πρόβλεψης στον πίνακα 4.6 δεν διαφέρουν πολύ στις διαφορετικές πολιτικές αντικατάστασης.

Πολιτική αντικατάστασης	1MB	2MB	4MB	8MB
LRU	3.6%	3.9%	5.3%	6.4%
SHiP	2.7%	4.1%	3.7%	7.6%
SRRIP	3.1%	4.6%	5.5%	5.7%
Mockingjay	4.9%	3.6%	5.3%	6.2%

Πίνακας 4.6: Γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης του δικτύου LSTM

Μια απλούστερη και λιγότερο δαπανηρή εναλλακτική λύση σε αυτό είναι το δίκτυο CNN, το οποίο έχει τη δυνατότητα να προβλέπει με μεγάλη ακρίβεια και εκπαιδεύεται πολύ πιο γρήγορα, με περίπου 20% του χρόνου εκπαίδευσης του LSTM. Αλλά είναι επίσης ασυνεπές για ορισμένες πολιτικές αντικατάστασης, με παρόμοιες προβλέψεις με το MLP, πράγμα που δεν είναι βέλτιστο. Το δίκτυο DNN παρουσιάζει τις περισσότερες δυνατότητες, καθώς έχει τις καλύτερες προβλέψεις για μικρότερα μεγέθη κρυφής μνήμης στο πλαίσιο της πολιτικής αντικατάστασης LRU, αλλά εξακολουθεί να έχει περιθώρια βελτίωσης στις μεγαλύτερες κρυφές μνήμες. Συνολικά, τα αποτελέσματα αυτά υπογραμμίζουν τις διαφορετικές περιπλοκές του προβλήματος και δείχνουν ότι είναι πράγματι δυνατή η ακριβής πρόβλεψή του.



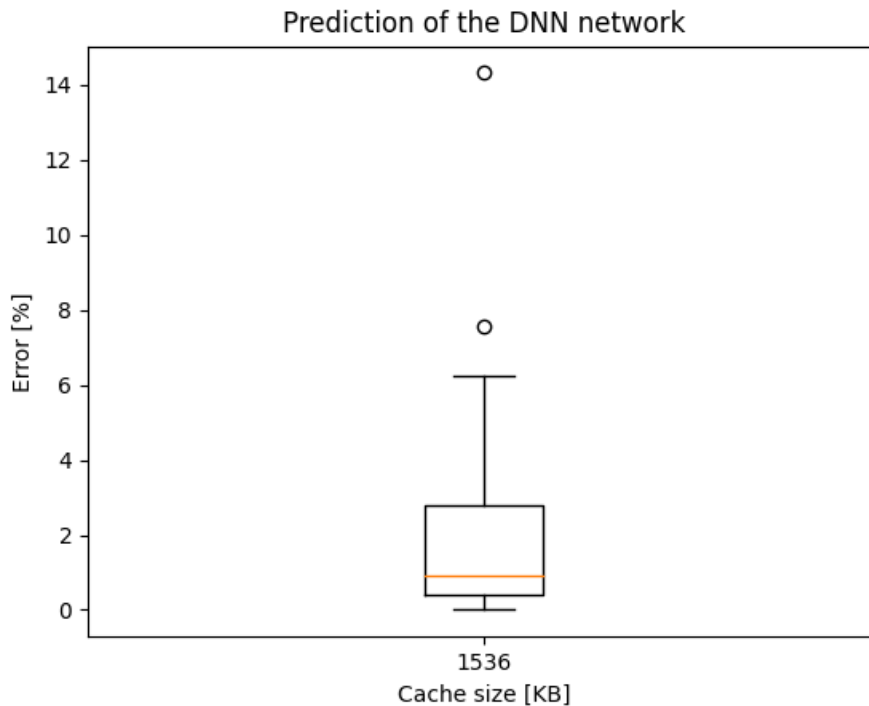
Σχήμα 4.12: Απόλυτα σφάλματα πρόβλεψης του δικτύου LSTM

4.2 Πρόβλεψη γνωστού προγράμματος

Υπάρχει και μια άλλη χρήση αυτού του δικτύου. Ας υποθέσουμε ότι έχουμε μερικά μηχανήματα που έχουν μια συγκεκριμένη αρχιτεκτονική. Θέλουμε να προβλέψουμε πώς θα συμπεριφερθεί ένα πρόγραμμα που έχει εκτελεστεί σε αυτές τις μηχανές, όταν εκτελεστεί από μηχανές με την ίδια πολιτική αντικατάστασης και άλλα μεγέθη LLC. Για να μοντελοποιήσουμε αυτή την περίπτωση χρήσης, θα χρησιμοποιήσουμε τις προσομοιώσεις της σουίτας αναφοράς για ορισμένα μεγέθη κρυφής μνήμης και θα μεταφέρουμε τα υπόλοιπα στο σύνολο δοκιμών. Με αυτόν τον τρόπο, θα είναι σαν να έχουμε μηχανές με τα μεγέθη κρυφής μνήμης LLC του συνόλου εκπαίδευσης και πολιτική αντικατάστασης LRU και να προβλέπουμε τα ποσοστά αστοχίας για τα μεγέθη κρυφής μνήμης εντός του συνόλου δοκιμής. Κάθε φορά, ολόκληρο το σύνολο των συγκριτικών δοκιμών διατηρείται στο σύνολο εκπαίδευσης και προσπαθούμε να προβλέψουμε όλες τις συγκριτικές δοκιμές ταυτόχρονα.

Αρχικά, μετακινήσαμε τα δεδομένα για ένα μέγεθος κρυφής μνήμης από το σύνολο εκπαίδευσης στο σύνολο δοκιμής για να δούμε αν και πόσο καλά το δίκτυο μπορεί να το προβλέψει με βάση τα υπόλοιπα αρχεία αναφοράς. Πιο συγκεκριμένα, το σύνολο δοκιμής αποτελείται από τη στήλη για μέγεθος κρυφής μνήμης 3MB και το σύνολο εκπαίδευσης από τα άλλα 7 μεγέθη κρυφής μνήμης [768, 1024, 1536, 2048, 4096, 6044, 8192] KB.

Οι προβλέψεις είναι πολύ καλύτερες από ό,τι θα μπορούσε να κάνει οποιοδήποτε άλλο δίκτυο. Ο γεωμετρικός μέσος όρος των παραπάνω σφαλμάτων είναι 0,9%, όπως φαίνεται



Σχήμα 4.13: Απόλυτα σφάλματα πρόβλεψης του DNN για 1 μέγεθος κρυφής μνήμης, εκπαιδευμένο με ποσοστά αστοχίας 7 μεγεθών κρυφής μνήμης.

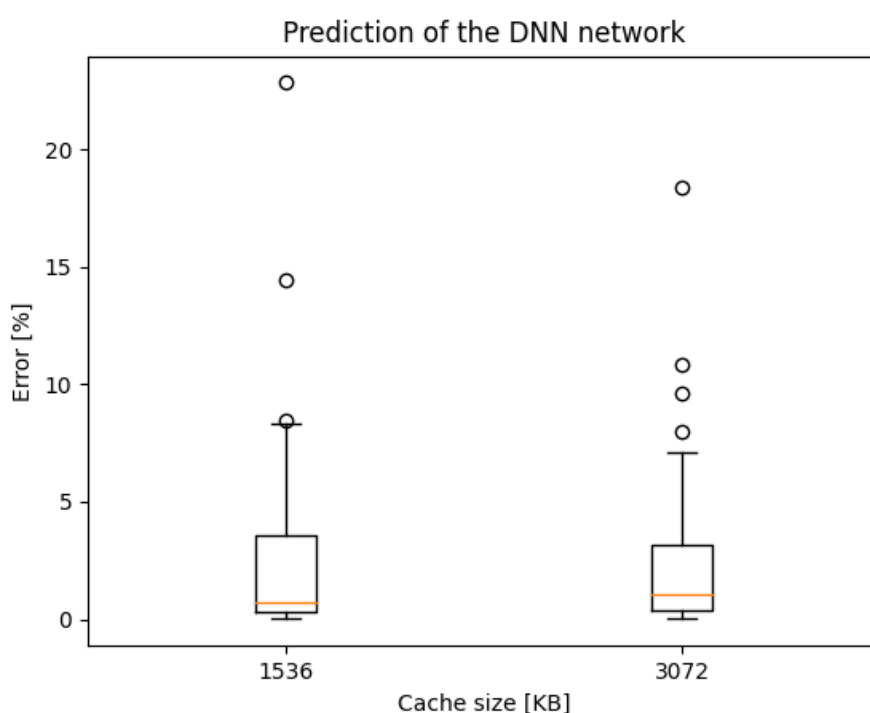
στην τελευταία γραμμή του πίνακα 4.7. Ο γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης του LSTM που είχε προβλεφθεί προηγουμένως για το LRU είναι 3,9% και 5,3% για τα δύο γειτονικά μεγέθη κρυφής μνήμης 2MB και 4MB, αντίστοιχα. Αυτό σημαίνει ότι αυτή η πρόβλεψη είναι αναμφισβήτητα πολύ καλύτερη από την καλύτερη πρόβλεψη που είχαμε μέχρι τώρα. Ήταν αναμενόμενο ότι αυτό το αποτέλεσμα θα ήταν καλό, δεδομένου ότι το δίκτυο έχει ένα πολύ μεγάλο σύνολο δεδομένων για να εκπαιδευτεί και μπορεί να κατανοήσει πολύ καλά το πρόβλημα.

Αυτό το αποτέλεσμα είναι μια καλή ένδειξη ότι το δίκτυό μας είναι κατάλληλο για την πρόβλεψη αυτού του είδους προβλήματος. Προκάλεσε επίσης το ερώτημα για το πόσα λίγα μεγέθη κρυφής μνήμης χρειάζονται στο σύνολο εκπαίδευσης για να έχουμε μια ακριβή πρόβλεψη, αφού η προσομοίωση επτά μεγεθών κρυφής μνήμης για την πρόβλεψη του όγδοου δεν είναι ένα ρεαλιστικό πρόβλημα.

Train-set cache sizes [MB]	Cache sizes [KB]							
	768	1024	1536	2048	3072	4096	6144	8192
1, 2, 4, 8	1.4%	-	0.8%	-	0.9%	-	1.1%	-
0.75, 1, 2, 4, 6, 8	-	-	0.8%	-	0.8%	-	-	-
0.75,1,1.5,2,4,6,8	-	-	-	-	0.9%	-	-	-
LSTM Network	-	3.6%	-	3.9%	-	5.3%	-	6.4%

Πίνακας 4.7: Γεωμετρικός μέσος όρος των απόλυτων σφαλμάτων πρόβλεψης του δικτύου DNN.

Το αμέσως επόμενο βήμα για να απαντηθεί αυτό το ερώτημα είναι να αφαιρεθεί άλλο ένα μέγεθος κρυφής μνήμης και να παρατηρηθεί πώς συμπεριφέρεται το σφάλμα των προβλέψεων.



Σχήμα 4.14: Απόλυτα σφάλματα πρόβλεψης του DNN για 2 μεγέθη κρυφής μνήμης, εκπαιδευμένο με ποσοστά αστοχίας 6 μεγεθών κρυφής μνήμης

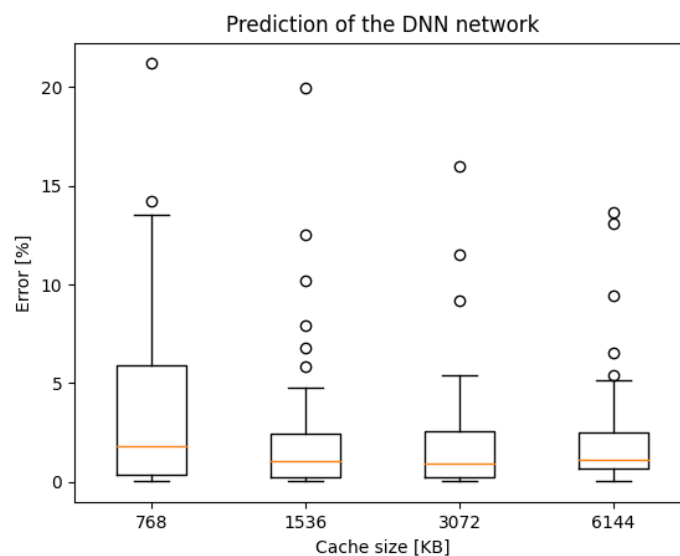
Ο γεωμετρικός μέσος όρος για τις δύο στήλες μεγέθους 1,5MB και 3MB είναι 0,8% και 0,8%, αντίστοιχα. Βλέπουμε ότι αυτοί οι γεωμετρικοί μέσοι είναι εξίσου καλοί με αυτούς που παρατηρήσαμε προηγουμένως.

Είναι σαφώς ορατό ότι, για κάθε μέγεθος κρυφής μνήμης, υπάρχει ένα μακράν χειρότερο προβλεπόμενο σημείο αναφοράς. Αυτή είναι η πρόβλεψη για το αρχείο αναφοράς 416.gamess- το ποσοστό αστοχίας του προβλέπεται πάντα υψηλότερο και θα προβλέπεται και στο μέλλον, για καλό λόγο. Αυτό το αρχείο αναφοράς αποτελείται από ένα ίχνος, το οποίο έχει μόνο 17κ προσπελάσεις στην LLC, πράγμα που σημαίνει ότι το δίκτυο διαβάζει ένα παράθυρο των αποστάσεων επαναχρησιμοποίησης με 17κ προσπελάσεις και προσπαθεί

να προβλέψει το ποσοστό αστοχίας του. Η πλειονότητα των παραθύρων αποστάσεων επαναχρησιμοποίησης στα οποία εκπαιδεύεται το δίκτυό μας αποτελείται από 512κ προσπελάσεις. Έτσι, το δίκτυο βλέπει τον χαμηλό αριθμό αποστάσεων επαναχρησιμοποίησης και υποθέτει πολλές προσπελάσεις που συμβαίνουν μόνο μία φορά και επομένως δεν έχουν αποστάσεις επαναχρησιμοποίησης. Αυτά έχουν ως αποτέλεσμα αστοχίες (cold misses) και επομένως το δίκτυο υπολογίζει υψηλότερο ποσοστό αστοχίας. Μπορούμε με ασφάλεια να αγνοήσουμε αυτή την έξοδο, δεδομένου ότι, με μια προσομοίωση 30 φορές μεγαλύτερης διάρκειας, η έξοδος πιθανότατα θα πλησίαζε περισσότερο την πραγματική τιμή.

Τέλος, κρατάμε τις τέσσερις πιο πιθανές κρυφές μνήμες LLC που θα έχει μια πραγματική αρχιτεκτονική και προσπαθούμε να προβλέψουμε τις άλλες τέσσερις τιμές. Δηλαδή, κρατάμε τις κρυφές μνήμες LLC των 1024, 2048, 4096 και 8192 KB στο σύνολο εκπαίδευσης και προσπαθούμε να προβλέψουμε τη συμπεριφορά των αρχείων αναφοράς που προσομοιώθηκαν με LLC μεγέθους 768, 1536, 3152 και 6044 KB. Με αυτή την είσοδο, υπολογίζουμε παρόμοια αποτελέσματα όπως προηγουμένως. Η κλίμακα των γεωμετρικών μέσων στην τρίτη σειρά του πίνακα 4.7 είναι παρόμοια με εκείνη για τα έξι μεγέθη κρυφής μνήμης στο σύνολο εκπαίδευσης.

Μπορούμε να παρατηρήσουμε στο 4.15 ότι τα σφάλματα των ακραίων τιμών εξακολουθούν να υφίστανται. Αξίζει επίσης να αναφέρουμε ότι η πρόβλεψη για 768 KB LLC είναι σαφώς, αλλά όχι πολύ χειρότερη από τις άλλες τρεις προβλέψεις. Αυτό είναι αποτέλεσμα του ότι δεν βρίσκεται μεταξύ των μεγεθών της κρυφής μνήμης που έχει δει το δίκτυο. Για παράδειγμα, το δίκτυο γνωρίζει την έξοδο για κρυφές μνήμες μεγέθους 1MB και 2MB και, επομένως, μπορεί να κατανοήσει καλύτερα πώς θα συμπεριφερόταν μια μηχανή για μια κρυφή μνήμη μεγέθους 1536 KB. Για τα 768 KB, είναι πιο δύσκολο να προεκτείνει τη συμπεριφορά του προγράμματος, και ως εκ τούτου αυξάνονται τα σφάλματα. Αυτό το σφάλμα εξακολουθεί να είναι πολύ χαμηλότερο από το καλύτερο σφάλμα που είχαμε προβλέποντας την κρυφή μνήμη μεγέθους 1MB προηγουμένως.



Σχήμα 4.15: Απόλυτα σφάλματα πρόβλεψης του DNN για 4 μεγέθη κρυφής μνήμης, εκπαιδευμένο με ποσοστά αστοχίας 4 μεγεθών κρυφής μνήμης

Και πάλι, η ύπαρξη μικρότερων μεγεθών κρυφής μνήμης στο σύνολο εκπαίδευσης δεν έχει νόημα, επομένως δεν θα τα παρουσιάσουμε.

Συνολικά, είναι σαφές ότι με αυτή τη χρήση, το δίκτυο προβλέπει το πρόβλημα με απόλυτη ακρίβεια. Οι προβλέψεις είναι αδιαμφισβήτητα καλύτερες από οποιοδήποτε άλλες προβλέψεις που είχαν γίνει προηγουμένως χωρίς να γνωρίζουμε το πρόβλημα. Το μειονέκτημα αυτού του δικτύου είναι ότι πρέπει να εκτελεστεί και να εκπαιδευτεί σε όλες αυτές τις διαφορετικές αρχιτεκτονικές κρυφής μνήμης. Επομένως, εξαρτάται από την εκάστοτε εφαρμογή αν αυτό αξίζει τον κόπο.

Κατακλείδα

5.1 Συμπεράσματα

Στην παρούσα διατριβή, παρουσιάζουμε έναν αριθμό δικτύων μηχανικής μάθησης που προσφέρουν λύσεις για το πρόβλημα της πρόβλεψης των ποσοστών αστοχίας σε διαφορετικές αρχιτεκτονικές κρυφής μνήμης. Η γνώση των ποσοστών αστοχίας της κρυφής μνήμης ενός προγράμματος σε διαφορετικές αρχιτεκτονικές είναι σημαντική, καθώς επιτρέπει στον χρήστη να γνωρίζει ποια είναι η καλύτερη μηχανή για να εκτελέσει το εν λόγω πρόγραμμα. Η προσομοίωση μιας διαδικασίας είναι μια δύσκολη και χρονοβόρα διαδικασία και η παράλειψή της μέσω μιας πρόβλεψης είναι μια πολύ αποτελεσματική λύση σε αυτό το πρόβλημα.

Παρουσιάζουμε τέσσερα δίκτυα μηχανικής μάθησης που κατανοούν το πρόβλημα με τον δικό τους μοναδικό τρόπο. Εμπνευσμένοι από την ήδη υπάρχουσα πιθανολογική προσέγγιση του StatCache σε αυτό το ζήτημα, αποφασίσαμε να χρησιμοποιήσουμε τα ίδια δεδομένα που χρησιμοποιεί για τις προβλέψεις του για να εκπαιδεύσουμε και να αξιολογήσουμε ένα δίκτυο μηχανικής μάθησης. Αρχικά, ένα απλό δίκτυο MLP δείχνει ότι είναι αρκετά απλό να προβλέψει τους λόγους αστοχίας με παρόμοια ακρίβεια όπως η πιθανολογική μέθοδος της StatCache. Αυτό το δίκτυο βασίζεται στις αποστάσεις επαναχρησιμοποίησης μιας εκτέλεσης του προγράμματος και εφαρμόζεται σε οποιοδήποτε μέγεθος κρυφής μνήμης και σε οποιαδήποτε πολιτική αντικατάστασης. Προβλέπει τα ποσοστά αστοχίας οποιοδήποτε προγράμματος με οποιαδήποτε πολιτική αντικατάστασης με έναν γεωμετρικό μέσο όρο παρόμοιο με αυτόν που εισάγει η StatCache.

Η πορεία προς την όσο το δυνατόν ακριβέστερη κατανόηση των δεδομένων εισόδου μας οδήγησε στην εισαγωγή δύο νέων δικτύων. Μαζί με τις NLP ενσωματώσεις των προγραμμάτων ως είσοδο, αυτά προσπαθούν να κατανοήσουν διαφορετικά τις αποστάσεις επαναχρησιμοποίησης και έτσι να υπολογίσουν ακριβέστερες προβλέψεις του προβλήματος. Το πρώτο είναι το δίκτυο LSTM, το οποίο, όπως υποδηλώνει και το όνομά του, κατανοεί τις αποστάσεις επαναχρησιμοποίησης μέσω ενός στρώματος LSTM. Αυτό το δίκτυο είναι το πιο χρονοβόρο για την εκπαίδευσή του, αλλά παράγει μακράν τα καλύτερα αποτελέσματα από όλα τα δίκτυά μας. Παράγει σταθερές, ακριβείς προβλέψεις των αναλογιών αστοχίας για οποιαδήποτε πολιτική αντικατάστασης και μέγεθος κρυφής μνήμης που εκπαιδεύεται. Το δίκτυο CNN λαμβάνει τις αποστάσεις επαναχρησιμοποίησης ως είσοδο σε ένα στρώμα CNN που έχει ως σκοπό την κατανόηση των αποστάσεων επαναχρησιμοποίησης μέσω μιας συνέλιξης των τιμών

που βρίσκονται σε κοντινή απόσταση. Η εκπαίδευση αυτού του δικτύου είναι σημαντικά λιγότερο δαπανηρή, αλλά έχει το μειονέκτημα ότι δεν είναι τόσο ακριβές όσο το δίκτυο LSTM. Οι προβλέψεις του είναι ως επί το πλείστον ελαφρώς λιγότερο ακριβείς από τις προβλέψεις του δικτύου LSTM, αλλά για ορισμένες πολιτικές αντικατάστασης, όπως η SHiP, είναι ακόμη χειρότερες, παρόμοιες με τις προβλέψεις του δικτύου MLP. Τα δύο αυτά δίκτυα αποτελούν τα βασικά βήματα που κάνει η παρούσα διατριβή για την ακριβή κατανόηση του προβλήματος.

Τέλος, παρουσιάζουμε ένα βαθύ νευρωνικό δίκτυο που λειτουργεί για οποιοδήποτε μέγεθος κρυφής μνήμης. Αυτό, όπως και τα προηγούμενα, χρησιμοποιεί τις αποστάσεις επαναχρησιμοποίησης και τις ενσωματώσεις του κώδικα για την κατανόηση του προβλήματος. Η διαφορά είναι ότι αυτό χρησιμοποιεί το μέγεθος της κρυφής μνήμης ως είσοδο και, όπως δείξαμε, είναι σε θέση να κατανοήσει το πρόβλημα για οποιοδήποτε μέγεθος κρυφής μνήμης. Αυτό το δίκτυο που προτείνουμε είναι πολύ ισχυρό για μικρότερα μεγέθη κρυφής μνήμης και έχει μια μικρή αδυναμία για μεγαλύτερα. Αυτό το δίκτυο παράγει επίσης πολύ ακριβή αποτελέσματα εάν γνωρίζει ήδη την πραγματική τιμή ενός προβλήματος για ορισμένα μεγέθη κρυφής μνήμης και τότε μπορεί να προβλέψει την αναλογία αστοχιών οποιοδήποτε μεγέθους κρυφής μνήμης με αθέατη ακρίβεια. Αυτό το δίκτυο μπορεί να αποτελέσει εφαλτήριο για περαιτέρω έρευνα προς την κατεύθυνση παρόμοιων αρχιτεκτονικών που κατανοούν καλύτερα το πρόβλημα.

5.2 Συζήτηση

Η προηγούμενη ενότητα αποτελεσμάτων, που παρουσιάστηκε στο κεφάλαιο 4, παρουσίασε μια ολοκληρωμένη ανάλυση των εμπειρικών δεδομένων που συγκεντρώθηκαν στην παρούσα μελέτη. Αυτή η ενότητα συζήτησης αποσκοπεί στη βαθύτερη κατανόηση της σημασίας αυτών των αποτελεσμάτων σε σχέση με το ευρύτερο πλαίσιο των αρχιτεκτονικών μηχανών. Με την εξέταση αυτών των ευρημάτων μέσα από διάφορους φακούς, η ενότητα αυτή επιδιώκει να διευκρινίσει το νόημα, τη σημασία και τις πιθανές επιπτώσεις τους στο πεδίο των αρχιτεκτονικών υπολογιστών. Επιπλέον, η συζήτηση αυτή θα αναλύσει κριτικά τις επιπτώσεις αυτών των αποτελεσμάτων σε σχέση με την υπάρχουσα βιβλιογραφία, με στόχο να συνεισφέρει νέες προοπτικές και δρόμους για περαιτέρω διερεύνηση στο πεδίο.

Τα μοντέλα που προτείνουμε στην παρούσα διατριβή αμφισβητούν τα αποτελέσματα προηγούμενων μηχανισμών πρόβλεψης που αποτέλεσαν τις καλύτερες εκτιμήσεις για το πρόβλημα αυτό. Αποδεικνύονται καλύτερα από τις προβλέψεις του Στατάρη με σημαντική διαφορά και προβλέπουν επίσης την έξοδο οποιασδήποτε πολιτικής αντικατάστασης. Αυτό σημαίνει ότι η πρόβλεψη των αστοχιών της κρυφής μνήμης σε μια μηχανή μπορεί πλέον να περιγραφεί ως θέμα εκπαίδευσης ενός μοντέλου μηχανικής μάθησης στις εξόδους μιας μηχανής και να το αφήσουμε να προβλέψει τα ποσοστά αστοχιών. Η εισαγωγή ενός τέτοιου δικτύου δεν θα απλοποιούσε μόνο τη διαδικασία απόφασης για το ποια είναι η καλύτερη αρχιτεκτονική για ένα πρόβλημα- θα παρέλειπε επίσης πολλές ώρες επεξεργασίας ή προσομοίωσης.

Για την περαιτέρω πλαισίωση αυτών των ευρημάτων, είναι επιτακτική ανάγκη να εξετάσουμε τους ευρύτερους παράγοντες που μπορεί να επηρεάσουν τα δίκτυά μας. Η εισαγωγή προσαρμοσμένων ιχνών για τα αρχεία αναφοράς, και μαζί με αυτό, οι ενσωματώσεις του συ-

γκεκριμένου τμήματος, θα επηρέαζαν πολύ τα αποτελέσματά μας. Η πρόβλεψη ενός δικτύου που γνωρίζει ποιο μέρος του προγράμματος εκτελείται σε κάθε ίχνος, όπως συζητείται στην ενότητα 3.2.2, μπορεί να βελτιώσει σημαντικά τις προβλέψεις από αυτά τα δίκτυα. Επιπλέον, θα μπορούσε να γίνει περαιτέρω διεύρυνση προς δεδομένα που προκύπτουν από παράλληλη επεξεργασία ή δεδομένα από πραγματικές μηχανές. Η πρόβλεψη της αναλογίας αστοχίας των διεργασιών που εκτελούνται σε πολλαπλούς πυρήνες μέσα σε μια μηχανή θα εισάγει πολύ μεγαλύτερη δυσκολία στο πρόβλημα και θα πρέπει να αποτελέσει αντικείμενο περαιτέρω διερεύνησης. Τα δεδομένα που χρησιμοποιήσαμε για την παρούσα διατριβή είναι τα δεδομένα που παράγονται από ένα πρόγραμμα προσομοίωσης. Θα ήταν ενδιαφέρον να δούμε πώς συγκρίνονται αυτά τα αποτελέσματα με τα δίκτυα που εκπαιδεύτηκαν σε ένα πραγματικό μηχανήμα, δεδομένου ότι το καθένα έχει τις δικές του δυσκολίες.

Επιπλέον, υπάρχουν δυνατότητες βελτίωσης των δικτύων μας. Όπως συζητήθηκε προηγουμένως, τα δίκτυα LSTM και CNN παρουσιάζουν δυνατότητες να έχουν ακόμη καλύτερα αποτελέσματα. Η εισαγωγή περισσότερων κριτηρίων αναφοράς στο σύνολο δεδομένων ή μια σημαντική αλλαγή στην αρχιτεκτονική τους θα μπορούσε να βελτιώσει ακόμη περισσότερο τις προβλέψεις τους. Το δίκτυο DNN είναι ένα δίκτυο το οποίο δεν έχει εξερευνηθεί από εμάς στο ίδιο βάθος με τα άλλα δίκτυα. Διαφορετικές αρχιτεκτονικές και υλοποιήσεις δικτύων θα μπορούσαν να μειώσουν σημαντικά τα σφάλματά του. Θα μπορούσε ακόμη και να επεκταθεί ώστε να έχει μια πρόσθετη είσοδο, το ποσοστό αστοχίας της κρυφής μνήμης μιας εκτέλεσης σε μια κρυφή μνήμη και στη συνέχεια να κάνει εκτιμήσεις για άλλα μεγέθη κρυφής μνήμης.

Συνοψίζοντας, η εις βάθος ανάλυση και η ερμηνεία των ευρημάτων που παρουσιάζονται σε αυτή τη συζήτηση υπογραμμίζουν την πολυπλοκότητα που ενυπάρχει στην πρόβλεψη των αστοχιών στην κρυφή μνήμη. Εν τέλει, η μελέτη αυτή αποτελεί ένα σκαλοπάτι στη συνεχή προσπάθεια να διαλευκανθούν οι πολυπλοκότητες της αρχιτεκτονικής των υπολογιστών, ενθαρρύνοντας την περαιτέρω επιστημονική έρευνα για την προώθηση της κατανόησης αυτού του τομέα.

Μέρος **III**

Επίλογος

Παραρτήματα

Βιβλιογραφία

- [1] N. Beckmann και D. Sanchez. *Modeling cache performance beyond LRU*. 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016.
- [2] X. Liu Q. Wang και M. Chabby. *Featherlight Reuse-distance Measurement*. IEEE International Symposium on High Performance Computer Architecture, 2019.
- [3] E. Berg και E. Hagersten. *StatCache: a probabilistic approach to efficient and accurate data locality analysis*. IEEE International Symposium on - ISPASS Performance Analysis of Systems and Software, 2004.
- [4] Maurice V. Wilkes. *Slave Memories and Dynamic Storage Allocation*. IEEE Trans. Electron. Comput., 14(2):270–271, 1965.
- [5] Carole Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely και Joel Emer. *SHiP: Signature-based Hit Predictor for high performance caching*. 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), σελίδες 430–441, 2011.
- [6] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely και Joel Emer. *High performance cache replacement using re-reference interval prediction (RRIP)*. ISCA '10, σελίδα 60–71, New York, NY, USA, 2010. Association for Computing Machinery.
- [7] Ishan Shah, Akanksha Jain και Calvin Lin. *Effective Mimicry of Belady's MIN Policy*. 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), σελίδες 558–572, 2022.
- [8] Alex Krizhevsky, Ilya Sutskever και Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems F. Pereira, C.J. Burges, L. Bottou και K.Q. Weinberger, επιμελητές, τόμος 25. Curran Associates, Inc., 2012.
- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun. *Deep Residual Learning for Image Recognition*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), σελίδες 770–778, 2016.
- [10] Miltiadis Allamanis, Earl T. Barr, Christian Bird και Charles Sutton. *Learning Natural Coding Conventions*. FSE 2014, New York, NY, USA, 2014. Association for Computing Machinery.

- [11] Miltiadis Allamanis και Charles Sutton. *Mining Idioms from Source Code*. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, New York, NY, USA, 2014. Association for Computing Machinery.
- [12] Amy McGovern. *Building a Basic Block Instruction Scheduler with Reinforcement Learning and Rollouts*. σελίδες 141–160, 2002.
- [13] C. Cummins, P. Petoumenos, Z. Wang και H. Leather. *End-to-End Deep Learning of Optimization Heuristics*. *26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017.
- [14] Nan Wu και Yuan Xie. *A Survey of Machine Learning for Computer Architecture and Systems*. *ACM Comput. Surv.*, 55(3), 2022.
- [15] Xiangyu Dong, Norman P. Jouppi και Yuan Xie. *A circuit-architecture co-optimization framework for exploring nonvolatile memory hierarchies*. *ACM Trans. Archit. Code Optim.*, 10(4), 2013.
- [16] Ramazan Bitirgen, Engin Ipek και Jose F. Martinez. *Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach*. *2008 41st IEEE/ACM International Symposium on Microarchitecture*, σελίδες 318–329, 2008.
- [17] Daniel Nemirovsky, Tugberk Arkose, Nikola Markovic, Mario Nemirovsky, Osman Unsal και Adrian Cristal. *A Machine Learning Approach for Performance Prediction and Scheduling on Heterogeneous CPUs*. σελίδες 121–128, 2017.
- [18] S. VenkataKeerthy, Rohit Aggarwal, Shalini Jain, Maunendra Sankar Desarkar, Ramakrishna Upadrasta και Y. N. Srikant. *IR2VEC: LLVM IR Based Scalable Program Embeddings*. *ACM Trans. Archit. Code Optim.*, 17(4), 2020.
- [19] Yulei Sui, Xiao Cheng, Guanqin Zhang και Haoyu Wang. *Flow2Vec: Value-Flow-Based Precise Code Embedding*. *Proc. ACM Program. Lang.*, 4(OOΠΣΛΑ), 2020.
- [20] John L. Henning. *SPEC CPU2006 Benchmark Descriptions*. *SIGARCH Comput. Archit. News*, 34(4):1–17, 2006.
- [21] James Bucek, Klaus Dieter Lange και Jóakimv. Kistowski. *SPEC CPU2017: Next-Generation Compute Benchmark*. *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, σελίδα 41–42, New York, NY, USA, 2018. Association for Computing Machinery.
- [22] Nathan Gober, Gino Chacon, Lei Wang, Paul V. Gratz, Daniel A. Jimenez, Elvira Teran, Seth Pugsley και Jinchun Kim. *The Championship Simulator: Architectural Simulation for Education and Competition*, 2022.
- [23] Geoffrey Webb, Claude Sammut, Claudia Perlich, Tamás Horváth, Stefan Wrobel, Kevin Korb, William Noble, Christina Leslie, Michail Lagoudakis, Novi Quadrianto,

- Wray Buntine, Lise Getoor, Galileo Namata, Jiawei Jin, Jo Anne Ting, Sethu Vijayakumar, Stefan Schaal και Luc De Raedt. *Leave-One-Out Cross-Validation*. 2010.
- [24] Diederik P. Kingma και Jimmy Ba. *Adam: A Method for Stochastic Optimization*. *CoRR*, αβσ/1412.6980, 2014.
- [25] Cem Subakan, Mirco Ravanelli, Samuele Cornell, Mirko Bronzi και Jianyuan Zhong. *Attention Is All You Need In Speech Separation*. *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, σελίδες 21–25, 2021.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

KME	Κεντρική Μονάδα Επεξεργασίας
MSE	Mean Squared Error
IR	Intermediate representation
NLP	Natural Language Processing
LRU	Least Recently Used
MLP	Multi-Layered Perceptron
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
DNN	Deep Neural Network
RL	Reinforcement Learning

Απόδοση ξενόγλωσσων όρων

Απόδοση

συστάδα
στρώμα συσσώρευσης μεγίστου
στρώμα συσσώρευσης μέσου
στρώμα συσσώρευσης
ευστοχία
πυκνό νευρωνικό δίκτυο
αποστάσεις επαναχρησιμοποίησης
αρχείο αναφοράς
ενσωμάτωση
ποσοστό αστοχίας
αρχείο ίχνους
περίπτωση χρήσης
μετασχηματιστής
βελτιστοποιητής
μέγεθος παρτίδας
γεωμετρικός μέσος όρος
χάρτης κατακερματισμού
θηκόγραμμα
μουστάκι
κλιμακωτής
κλιμάκωση
βήμα
πυρήνας
προσοχή
στρώμα εγκατάληψης
υπερβολική προσαρμογή
συνελικτικό νευρωνικό δίκτυο
ανάκτηση πληροφορίας
αντιμεταθετικότητα
απόγονος
απορρόφηση
βάση δεδομένων
γνώρισμα
διαπροσωπεία

Ξενόγλωσσος όρος

cluster
max-pooling layer
average-pooling layer
pooling layer
hit
dense neural network
reuse distances
benchmark
embedding
miss ratio
trace file
use case
transformer
optimizer
batch size
geometric mean
hashmap
boxplot
whisker
scaler
scaling
stride
kernel
attention
drop-out layer
overfitting
convolutional neural network
information retrieval
commutativity
descendant
absorption
database
attribute
interface

διαφορά	difference
δικτυακός κατάλογος	portal catalog
δικτυωτή δομή	lattice
δομικές επερωτήσεις	structural queries
δομικές σχέσεις	structural relationships
δομικό σχήμα	schema
εγκυρότητα	validity
ένωση	union
αδερφός	sibling
αμεταβλητότητα	idempotency