Noname manuscript No.
(will be inserted by the editor)

Machine Learning-Based Cache Miss Prediction

Edin Jelačić · Cristina Seceleanu · Ning Xiong · Peter Backeman · Sharifeh Yaghoobi · Tiberiu Seceleanu

the date of receipt and acceptance should be inserted later

Abstract Integrating machine learning into computer architecture simulation offers a new approach to performance analysis, moving away from traditional algorithmic methods. While existing simulators accurately replicate hardware, they often suffer from slow execution, complex documentation, and require deep CPU knowledge, limiting their usability for quick insights. This paper presents a deep learning-based approach for simulating a key CPU component, cache memory. Our model "learns" cache characteristics by observing cache miss distributions, without needing detailed manual modeling. This method accelerates simulations and adapts to different program needs, demonstrating accuracy comparable to traditional simulators. Tested on Sysbench and image processing algorithms, it shows promise for faster, scalable, and hardware-independent simulations.

Keywords Machine Learning · Cache · Simulation.

1 Introduction

As hardware costs increase, computer architecture simulation has become essential for optimizing and estimating performance, identifying bottlenecks, and providing a cost & time-effective solution without the explicit need for physical hardware. A commonly used way to perform this optimization is the analysis of large amounts of data obtained from such simulators, as illustrated by Fig. 1. Several algorithmic simulators and instrumentation tools are available for dynamic memory, security analysis, and performance monitoring, including DynamoRIO Cachesim [5], Valgrind [25], Intel PIN [23], and QEMU [4]. These tools serve specific purposes and use cases, but they share the drawback of requiring either algorithmic processing of each individual instruction, or instrumentation of execution on an existing CPU, which introduces a significant performance overhead in terms of computation time [31]. This overhead may lead to unreliable performance measurements and is therefore preferably avoidable.

Mälardalen University, Västerås, Sweden
E-mail: {edin.jelacic, cristina.seceleanu, ning.xiong, peter.backeman, sharifeh.yaghoobi, tiberiu.seceleanu}@mdu.se
www.mdu.se

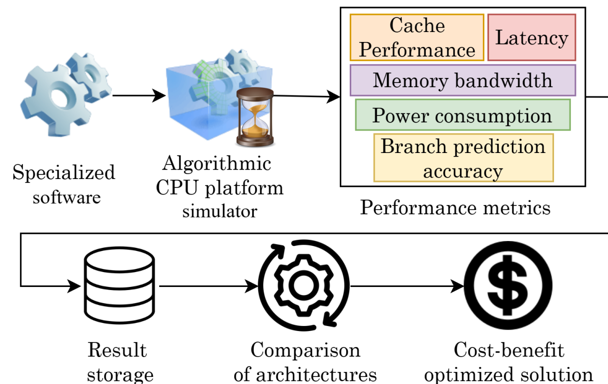

Fig. 1: Commonly utilized industrial software simulation paradigm

In this work, we focus on the prediction of cache misses based on a novel statistical machine learning design. Traditional algorithmic simulators provide detailed insight into program execution, but face limitations due to high overhead and the need for detailed architectural knowledge, especially in complex systems. Two main challenges arise in instruction-accurate simulation of x86 CISC architectures: the construction of complex, error-prone models for each architecture [24], and significant performance issues, with simulators running much slower than native execution [22]. Recent advances like Ithemal [24], SimNet [22], and the latest

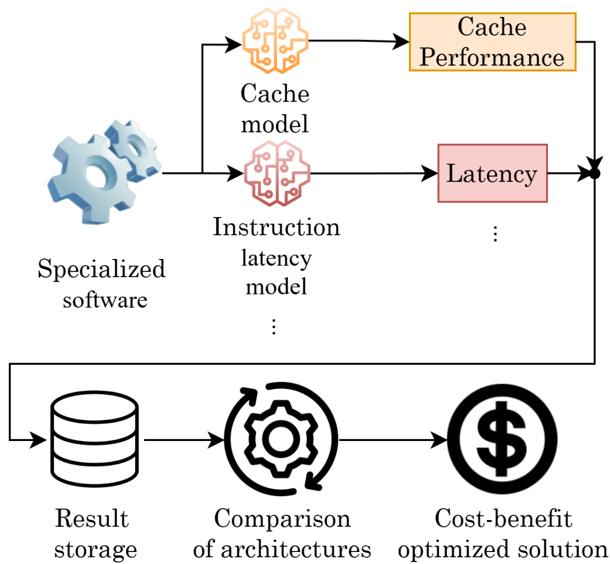

Fig. 2: Proposed industrial software simulation paradigm

TAO [28] address these challenges using deep learning to predict instruction latencies and other performance metrics in specific architectures without extensive manual system definitions.

In modern computing, optimizing the speed of memory access is critical to improving performance and energy efficiency. Cache memory plays a vital role in this by providing rapid data retrieval to the processor. Consequently, the optimization of a hardware platform concerns the organization of the cache memory utilized by the platform. Each distinct application demands a varied utilization of cache memory throughout its runtime. Dimensioning of memory resources with respect to an executing program strongly impacts cache miss occurrences that lead to potentially significant delays in processing. Therefore, system designers, particularly in high-cost industrial environments, are keen to cater to specific software needs when deciding on prospective platform investment.

However, managing cache size and structure remains a challenge due to trade-offs between cost, performance, energy consumption, and cache misses, which lead to increased memory access times and degraded system performance. While existing simulators model cache behavior, faster and more adaptable solutions are needed to better address these inefficiencies. A promising, yet underexplored approach is the use of machine learning for modeling application-specific cache behavior.

This work focuses on the design and implementation of a neural network-based model with the purpose of predicting cache performance within an existing cache simulation framework, aiming to enhance and improve it for specific use cases, without assuming a fixed CPU architecture. Complementary to some existing work, our research here aims ultimately: (i) to replace a traditional simulator for cache prediction with a holistic machine learning model; (ii) for this model to function and generate performance metrics across various program executions using a hardware-agnostic approach; (iii) for this model to focus on the comparison of as many hardware combinations as possible while targeting a limited set of programs.

We introduce a *long-short-term memory* (LSTM) [11] deep learning approach for predictive cache miss modeling on program execution traces on multiple prospective architectures and demonstrate this on benchmark traces for testing. Our choice is justified by the fact that LSTMs are variants of recurrent neural networks, capable of adequately handling long-term sequential dependencies. To our knowledge, it is the only work that addresses the issue of simulating cache performance on various architectures in this manner. Our ML-based method offers the potential for faster predictions and greater flexibility to adapt to varying program and architecture characteristics without the need for detailed full-system architectural simulation. This ultimate goal framework, along with its utility, is illustrated

In summary, the key contributions of this paper are:

- A novel method for modeling different cache levels using a recurrent multivariate regression model;
- Pioneering the use of LSTM networks in modeling cache miss distribution, independent of the architecture;
- The ability of our method to observe the impact of core/cache configurations on cache miss distribution per individual core and core count for the corresponding first-level caches and shared unified caches.

The validation of this approach comes by testing it on widely used benchmark software and an image processing algorithm across various cache sizes and core counts, and the evaluation of the efficiency of the method is performed by comparison to an algorithmic cache simulator.

The remainder of the paper is structured as follows. Section 2 provides essential background on memory, cache, neural networks, and the DynamoRio Cachesim simulator. Section 3 details our method to predict the cache failure distribution throughout the execution of the program. We follow with Section 4 where we dive into the data, the instruction features and the process of training and deploying our model. Section 5 presents

the results of our model's performance, and Section 6 discusses the successes and challenges of our approach, while Section 7 explores alternative approaches and state-of-the-art techniques, comparing them with our objectives. We conclude with final thoughts and suggestions for future work in Section 8.

2 Background

In contemporary microarchitectures, *cache memory* embodies a hardware implementation of small, yet high-speed memory situated in close proximity to the processor. In modern processors, it serves as a hardware-implemented memory layer positioned between individual registers and the main memory blocks. The latter are commonly identified today based on the main implementation technology, e.g. *DRAM* (Dynamic Random-Access Memory).

2.1 Memory hierarchy

Processor registers are compact memory locations, usually 32 or 64 bits, and serve as architectural components of the processor instruction set. The processor operates on these registers directly, through various instructions. Registers are used to store instructions, counters, memory addresses, operands, and the immediate results of instruction execution.

The main memory constitutes volatile read/write storage, where programs are loaded directly during runtime. It is notably large, and it is common for contemporary desktop platforms to be equipped with more than 32 gigabytes, due to the necessities of modern operating systems and increasingly complex software. However, accessing DRAM for data whenever necessary would be prohibitively slow, as DRAM lies architecturally and physically outside the processor. This would represent a bottleneck with the processor being starved waiting for incoming data, particularly given that a program's execution often relies on reusing data.

2.2 Cache memory

Cache memory is a specialized component integrated into the processor to reduce latency by storing recently accessed data. It is structured into multiple levels, including L1, L2, and L3, each with varying capacities and speeds. L1 is the fastest but the smallest, while L3 offers greater capacity but higher access latency. The processor utilizes the cache in a hierarchical manner to minimize reliance on slower main memory, significantly reducing program execution delays. In multicore processors, usually each core has at least one cache level (L1), with other levels usually shared between cores.

Throughout program execution, the processor employs cache memory to store frequently used data and instructions. When the processor is instructed to access a specific address by the instruction code, it initiates the process by checking the L1 cache. If the data is not found in L1, it proceeds to check L2, and so forth. If the data is not found in any of the caches, it is retrieved from the main memory and stored in one of the cache levels [16], and until data is retrieved, the execution is paused. This event is known as a *cache miss*, and various advances in modern computer design aim to minimize these occurrences and mitigate the corresponding latency penalties. Strategies include continuous increases in on-die cache memory sizes throughout history [9], cache memory restructuring, the incorporation of various optimizations [10], and the introduction of mechanisms, such as pre-fetchers, which anticipate the most likely instructions or data that the processor will need and fetch them before they are needed [20].

In addition to the above, the evolution of processors, exemplified by advancements like multithreading, can be viewed as a step towards minimizing the time-delay associated with a cache miss by implementing thread-level parallelism [17]. Predicting cache misses, implementing strategies for their avoidance or scalable mitigation techniques for compulsory cache misses is of great significance, especially in a modern computing landscape, where memory speed lags behind that of contemporary processor designs [6].

2.3 Neural Networks

Neural networks (NN) have proven to be useful as universal function approximators [15, 26]. They have been used, in various forms, for improvements or predictions in computer architectures [3]. The basic neural network is the *feed-forward* NN, called so due to its structure of a mesh of neurons with one or multiple inputs, one or multiple outputs and one or more hidden layers representing linear functions paired with nonlinear activation functions.

An NN feedforward estimates a function f, with data flowing from the input $\mathbf{x}$, processing in the hidden layer(s) defined by a parameter set $\mathbf{\Theta}$, the output being a vector of real-valued results, commonly denoted by the *prediction* $\mathbf{y}$. More precisely, NN represent statistical mappings $p(\mathbf{y}|\mathbf{x};\mathbf{\Theta})$ in which the parameters $\mathbf{\Theta}$ are tuned, by backpropagation, in such a way as to locally minimize some cost function that maps the weights to the outputs of the objective cost function. Feed-forward

NNs do not have feedback loops that map past quantities to internal memory when making predictions. This makes them only appropriate for calculating predictions in which the predictions have no temporal relationship with historical data.

In those problems where the dataset on which we are trying to calculate the predictions is sequential in nature, sequential NN or RNN (recurrent neural networks) are commonly used. By sequential nature, we mean that the sequence of inputs is a factor in predicting the desired distribution, as $p(\mathbf{y}|(\mathbf{x}, \mathbf{x}_{-1}, \mathbf{x}_{-2}, ..., \mathbf{x}_{-k}); \mathbf{\Theta})$ where k is the number of time steps in the sequence.

The most prominent sequential architecture is the LSTM NN, composed of a memory cell and multiple non-linear gates that regulate the flow of data, represent current, and selectively memorize prior states. This minimizes the vanishing / exploding gradient problem, typically present in RNN backpropagation [11,32]. Such an architecture is shown in Fig. 3. Due to its prominence in the field of sequential timeseries multivariate forecasting, we opted for this model as well.

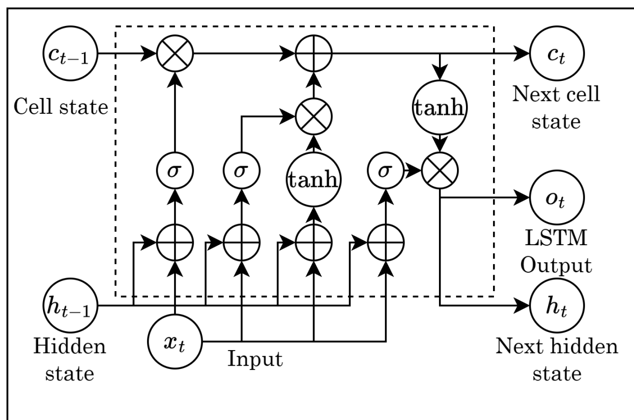

Fig. 3: LSTM Cell architecture

2.4 DynamoRIO Cachesim

DynamoRIO Cachesim is a simulator tool designed to analyze the cache performance of applications by simulating the impact of various cache configurations on memory access patterns. It functions by dynamically instrumenting running applications to collect detailed memory access data, which is then utilized to simulate cache behavior.

The simulator models L1 and L2 cache levels by default and can be extended to support more complex configurations. It supports varying sizes, cache associativities, and block sizes to provide insights into cache performance metrics. The simulator operates on an executable by injecting instrumentation code into the application's binary at runtime, capturing each memory access that the application performs. These captures are stored in a format that Cachesim and other components in the DynamoRIO suite can use to simulate desired cache configurations and compute metrics such as cache hit rates, miss rates, performance across different cache levels, register states, and latency impacts. DynamoRIO itself is extensible and well-documented, offering the ability for users to build their own client tools for performance analysis within the DynamoRIO framework or build upon existing ones.

3 The Approach

In this section, we present our approach that relies on equating predictive modeling to the statistical estimation of a cache miss distribution across a program's execution through running aggregates of cache misses. Specifically, given a program of length K and a sequence length of N ($N < K$), to estimate such a distribution we generate $\lceil K/N \rceil$ data points, with each data point at most of value N (observe Fig. 4 for an example with a sequence length of 9 instructions).

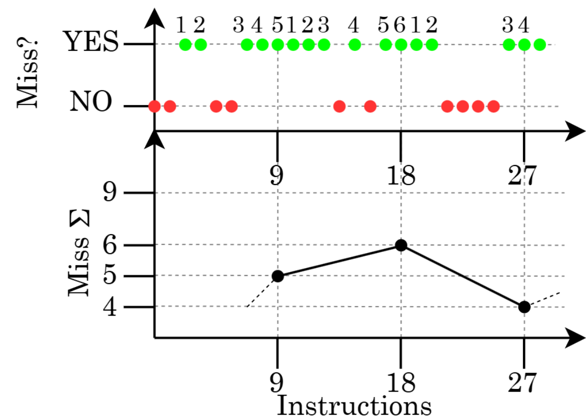

Fig. 4: Miss sub-sequence summing

Our model utilizes deep learning to take a program trace and specified cache characteristics as input, returning the cache miss distribution as output. By program traces, here we refer to x86 assembly traces of executed programs. When access to the original source code and compilation procedure is unavailable, these traces can be obtained using instrumentation tools, such

as Intel PIN [23], Valgrind [25], or other methods like debuggers and disassemblers. Hence, we turn the task of predictive modeling of cache behavior into one of multivariate sequential regression. Program traces are represented by the x86 assembly instructions (the instructions themselves, with the corresponding operands, such as constants and registers), which the execution of a program consists of, as exemplified below:

```
...
cmp    %rax $0x0000000000000022
jbe    $0x00007f9290f3e0c9
mov    %rdi -> %rsi
sub    %rax %rsi -> %rsi
cmp    %rsi $0x000000000000000f
...
```

In the above, each instruction is supplemented with additional features procured by the cache simulator that we employ for initial data gathering, that is, DynamoRIO Cachesim [5].

4 Implementation

For gathering our traces, we use Cachesim, with modifications enabling writing features of a trace to a suitable SQLite database. Namely, we implement a mechanism for writing out the results of Cachesim simulation, for each execution, to a dedicated SQLite file, in a format that could be utilized later for machine learning with reasonable overhead by the pandas [33] library for Python.

4.1 DynamoRIO Instruction Features

DynamoRIO has the ability to supplement the executed instructions with various features, of which we keep those that we deem most important for the task at hand. The features for our deep learning algorithm are shown in Table 1. The reasoning for the delta encoding is the reduction of the range of possible values by way of storing merely the relative difference between sequential addresses rather than the absolute addresses. More importantly, programs generally exhibit spatio-temporal locality [10], which the delta encoding exploits by improving overall storage efficiency and reduces the overall computational cost of the encoding as opposed to computing for 64-bit addresses.

[1] The full list may be observed as part of the trace_entry.h header file, which is part of the DynamoRIO repository at https://github.com/DynamoRIO/dynamorio.

Table 1: Table of instruction row features.

Feature Name	**Values**	**Explanation**
Disassembly String	String	The instruction itself (e.g., `sub %rax %rsi` → `%rsi`)
Instruction Number	Positive integers	The ordinal number of the instruction during the execution of the program.
Access Address Delta	Integers	The positive or negative offset in the memory of the address of requested data w.r.t. the previous instruction, initialized from zero and resetting at each thread switch.
PC Address Delta	Integers	The positive or negative offset of the Program Counter (PC) w.r.t. the previous instruction's PC, initialized from 0 and resetting at each thread switch.
Instruction Type	Categorical	The type of the recorded program event. Can be an Instruction Fetch (ifetch), Memory Store (write), Memory Load (read), etc.[1]
Byte Count	Positive integers	Number of bytes loaded or read.
Core	Positive integers	The current simulated core that the thread is running on.
Thread Switch	Boolean	True (1) if a thread switch occurred for this entry, false (0) if it has not.
Core Switch	Boolean	True (1) if a core switch occurred for this entry, false (0.0) if it has not.
L1D size	Positive integers	The size (in bytes) of the L1 Data cache, constant throughout a single execution.
L1I size	Positive integers	The size (in bytes) of the L1 Instruction cache, constant throughout a single execution.
LL size	Positive integers	The size (in bytes) of the unified Last Level cache, constant throughout a single execution.

4.2 Tokenization

The key aspect for applying deep learning in our approach is the tokenization process, a fundamental preprocessing step that involves converting raw text data into a format interpretable by machine learning models. This is the process of transforming the input text of the disassembled instruction string into a sequence of discrete elements, or tokens, which represent the building blocks of the text. In our case, these elements are the assembly instructions, their operands and operators such as brackets and assignments (an example is shown in Fig. 5).

For this procedure, we opt for the "Byte Level" tokenizer and the "Whitespace" preprocessor by Hugging-

Face [1]. The tools available from HuggingFace are widely used and highly regarded in the field of natural language processing [7], albeit we only use a small tool subset in our work.

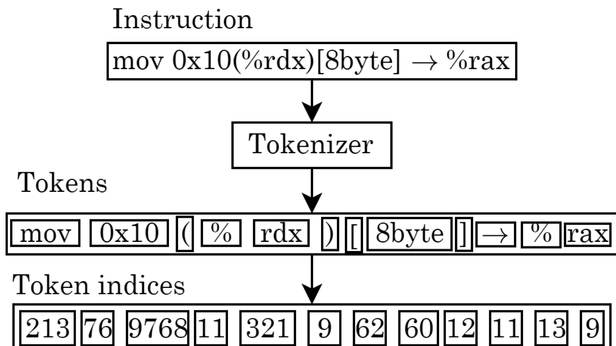

Fig. 5: x86 Instruction tokenization example (the numbers for token indices are purely exemplary)

The tokenizer is initially trained on the set of instructions present in the corresponding program traces, upon which a dictionary of tokens and corresponding token indices are generated and stored. After the tokenizer is prepared (trained, stored or loaded), each instruction is broken down into individual tokens and then the individual tokens are converted into token indices, numerical representations of tokens that a deep learning model can process. The outlined tokenization procedure is supplemented in the deep learning model with a dedicated trainable embedding layer, which contextualizes each token's ID in the form of a specialized lookup table with preset maximum embedding length, as per Fig. 6. In our case, the maximum embedding length was 15, up until which a shorter instruction string was to be padded with [PAD] tokens, and after which longer instruction strings were truncated.

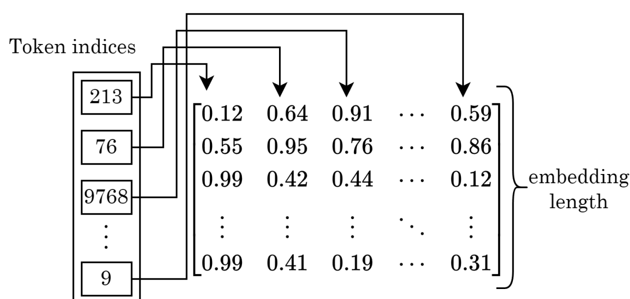

Fig. 6: Token indices lookup table, generated by the embedding layer, whereby each token is assigned a corresponding vector of a-priori fixed length (15 here)

This ensures that each instruction's components, the operands, and the location registers are given numerical values that the model can mathematically operate on. For the desired model output, we take the cache misses for the three simulated cache levels, for which traces are executed and stored: L1 data, L1 instruction and shared LL (last-level) cache, summed up across sub-sequences. In this way, we introduce profiling of the trace without sliding the prediction window one-by-one instruction-wise, but rather sub-sequence by sub-sequence, without overlapping, which enables a significant speedup both in training and inference.

4.3 Model and Training Procedure

Our model is composed of three components: (i) a **token processing embedding layer**, (ii) an **LSTM layer** for the instructions and other features, and (iii) **an affine, fully connected dimension-reduction layer** that takes the final output of the major LSTM layer as input. The model form is described in Fig. 7 and is implemented in the PyTorch framework [29].

When a certain program is selected (typically one with parameters, such as a benchmark with variable time and stress loading limits or targets), we gather the dataset database initially by running a simulation through a custom tool within the DynamoRIO framework, with various cache parameter combinations, within a certain reasonable span of these parameters. This dataset is termed the "training/validation dataset of program X", as it is the one in which all training will take place. In this work we opt for a training / validation ratio of 0.9.

For "testing dataset of program X", we generate data on the same program family but with cache parameters previously unseen by the model. By *program family*, henceforth, we will refer to a set of programs with broadly similar purposes and methods, making specific cache configuration performance modeling reasonable. The models themselves are based on various hyperparameters, such as **batch size** (the number of samples that is fed into our model at each iteration of the training process), **sub-sequence length**, **LSTM hidden dimension width**, **depth**, **learning rate** and **dropout** between neighboring LSTM layers.

We also propose hyperparameter optimization for each program family, since different program types have different cache access patterns in different parts of the execution. Hyperparameter optimization is taken care of by Optuna framework [2], chosen because of the easily modifiable *Trial* class provided. The loss function that we use is MSE and we limit training to 10 epochs and the hyperparameter optimization to 5 epochs on

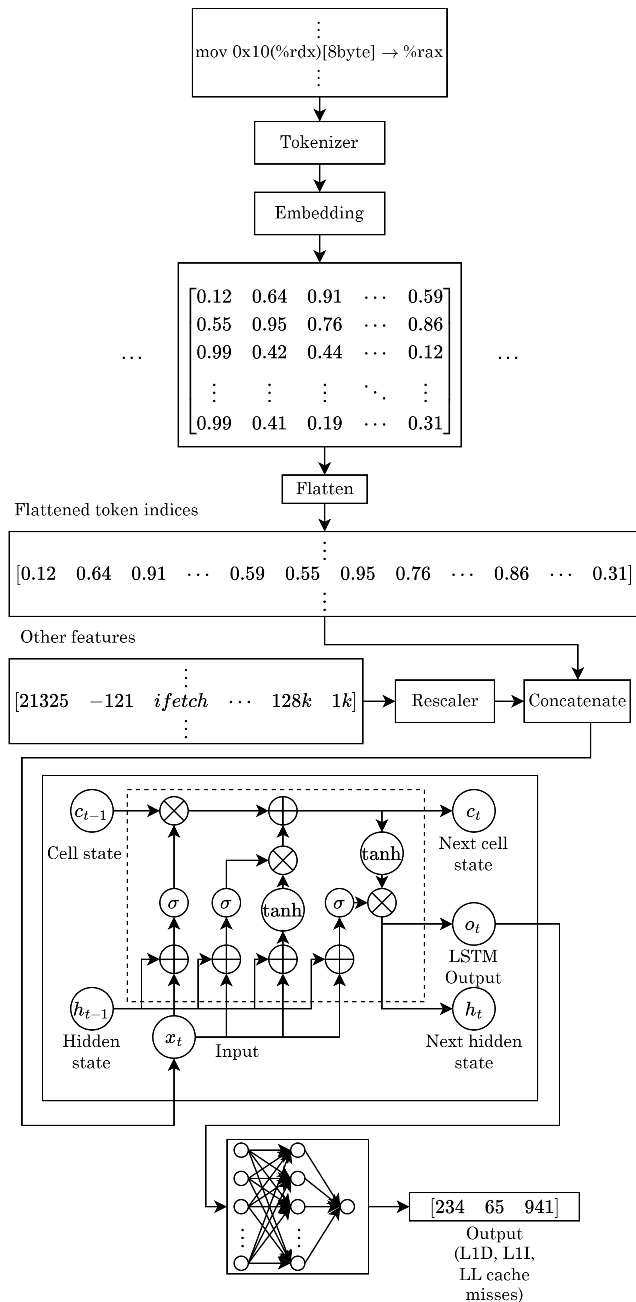

Fig. 7: Utilized LSTM model outline

a greatly reduced dataset. The reason for such limits is the relatively long training time on our hardware, reaching up to 2 hours for a single epoch on the full training dataset.

We gear the optimization towards minimizing the validation loss within the initial 5 epochs, but with an emphasis on compute time limitations. Namely, an optimization trial of 5 epochs that takes longer than a given preset amount of time is automatically considered a failure, along with its hyperparameters. This leads to a potential exclusion of otherwise satisfactory parameters due to the constraints on execution time. Hyperparameter optimization is performed on a highly reduced data set, comprising only two databases (runs). It should be noted that several runs with improper batch size / sequence length had to be discarded during the hyperparameter optimization, as they showed to retrieve too much data for our compute machine's RAM to handle. This was also taken care of automatically by the framework with some minor tweaks to the code.

In total, the final parameters utilized were rounded to 330 for the hidden layer dimension size, 2 LSTM layers, 0.05 for the dropout between the LSTM layers, 60 for the batch size and $N = 200$ for the sequence length.

5 Results

The experimental evaluation is carried out on an i7-13700HX, 32 GB RAM, Nvidia RTX A1000 (6 GB VRAM, 2048 CUDA cores) laptop platform. To attain a solid base for testing this approach out, a program family has to be selected such that it consists of a practical set of programs that stress specific platform components, with a broad array of algorithms. For this purpose, Sysbench [14] is chosen, along with three of its utility benchmarks: CPU, Memory, and FileIO. The CPU benchmark tests computation-intensive tasks, as efficient cache utilization, particularly L1D and L1I caches, is crucial for reducing latency. FileIO simulates disk read/write operations, with performance sensitive to cache sizes, especially the LL cache, as effective caching reduces disk access times. The Memory benchmark assesses the system's ability to handle memory operations, with larger caches improving data retrieval speed from main memory.

Another valuable use case that we used is a highly specific image processing algorithm termed “Good Features to Track” or GFTT, exposed as a memory-intensive application [34]. Such algorithms represent a range of workloads, providing a comprehensive analysis of how cache configurations impact system performance. We introduce a wrapper utility for DynamoRIO's cache simulator (drcachesim), to gather simulation data in the form of serverless SQlite3 databases, where one database corresponds to one program execution. The wrapper is available on GitHub (https://github.com/ptrbman/dynamorio-missing-instructions).

5.1 Performance Metrics

In our work, by *model performance* we refer to the ability of the model to capture the distribution of cache

misses from the original trace. We measure this ability by the values of the mean squared error and the model's R^2, both of which are defined in further text.

In the following, let

$$\mathbf{y_{L1D}} = [y_{L1D_0}, y_{L1D_1}, \ldots, y_{L1D_j}, \ldots, y_{L1D_K}]^\intercal \quad (1)$$

be the actual mini-sums of L1D cache misses across K sub-sequences, where $K = \textit{program length} \;/\; \textit{sub} - \textit{sequence length}$ and $y_{L1D_j}, j \in [0, K]$ represents the number of cache misses in the L1 data cache in the j-th sub-sequence. Further, let

$$\mathbf{\hat{y}_{L1D}} = [\hat{y}_{L1D_0}, \hat{y}_{L1D_1}, \ldots, \hat{y}_{L1D_K}]^\intercal \quad (2)$$

be the mini-sums of L1D cache misses across K sub-sequences as predicted outputs of model inference. Similarly, we define $\mathbf{y_{L1I}}$, $\mathbf{y_{L1I}}$, $\mathbf{\hat{y}_{L1I}}$, $\mathbf{y_{LL}}$, and $\mathbf{\hat{y}_{LL}}$.

The mean squared error E_{MSE}, for each of the cache types, respectively, is defined as per eq. (3).

$$E_{MSE} = \frac{1}{K}\sum_{i=1}^{K}(y_i - \hat{y}_i)^2 = \frac{1}{K}(\mathbf{y} - \mathbf{\hat{y}})^\intercal(\mathbf{y} - \mathbf{\hat{y}}) \quad (3)$$

We use E_{MSE} for both training the model and as a measurement of the model's performance on the training and test datasets, since for our approach the distribution of errors throughout an execution is the most significant observation and we ideally desire to penalize the model for large outliers while retaining a smooth gradient for smaller deviations of predicted values as compared to the real values [12]. The E_{MSE} and its root square counterpart, the root mean squared error $E_{RMSE} = \sqrt{E_{MSE}}$ tell us how well the model performs with cache miss predictions throughout the execution of a program, which is one aspect of the prediction that we are interested in.

Another component of the prediction that we are interested in is the model's ability to predict total cache miss numbers, which we attain by summing up the outputs throughout the model's execution on a single program. For this measurement, we define the relative error percentage metric, based on the L1 norm, relative to the actual values, that is, E_{REP}. We let the total sum of actual cache misses be $T = \sum^K \mathbf{y}$ and the total sum of predicted cache misses be $\hat{T} = \sum^K \mathbf{\hat{y}}$, then the E_{REP} is defined as per eq. (4).

$$E_{REP} = \frac{|\sum^K \mathbf{y} - \sum^K \mathbf{\hat{y}}|}{\sum^K \mathbf{y}} \cdot 100 = \frac{|T - \hat{T}|}{T} \cdot 100 \quad (4)$$

The equation eq. (4) that we utilize here is similar to the equation for the *mean absolute error percentage*, more commonly abbreviated as MAPE. However, while MAPE concerns the relative distribution of errors across an entire execution, E_{REP} concerns the relative discrepancy of the aggregate sums. This is of importance because the total number of cache misses is critical and a significant concern is how well the total predicted cache misses align with the total actual cache misses. For example, a E_{REP} of 0 is ideal, whereas an E_{REP} of 50 means that predictions overshoot or undershoot the real values by 50%.

The two focal values to be observed are the **mean squared error** (paired with the R^2 and E_{RMSE}), demonstrating how well the forecast of cache misses along a program execution "tracks" the actual cache misses on average, as well as the **total numbers of cache misses** at the end of executions, where each execution regards a specific cache size / core combination.

The *determination coefficient*, R^2, is a statistical measure that indicates how well the independent variables in a regression model explain the variability of the dependent variable. It is calculated by taking the ratio of the sum of the squared differences between the predicted values and the mean of the dependent variable to the sum of the squared differences between the actual values and the mean of the dependent variable. Mathematically, $R^2 = 1 - (SS_{res}/SS_{tot})$, where SS_{res} is the sum of squared residuals (the differences between observed and predicted values), and SS_{tot} is the total sum of squares (the differences between observed values and their mean). However, it can be negative in the case that the model is highly misleading.

5.2 Execution

Training runs on several cache size, core count and program combinations:

- L1 Data cache sizes: 512 bytes, 8 kilobytes, 32 kilobytes,
- L1 Instruction cache sizes: 512 bytes, 8 kilobytes, 32 kilobytes,
- LL unified cache sizes: 1 kilobyte, 8 kilobyte, 32 kilobytes,
- Core counts: 1, 2, 4,
- Benchmarks: Sysbench CPU, Sysbench Memory, Sysbench FileIO, GFTT.

Note that we have 3 possible sizes for each L1D, L1I, LL, 3 core counts, and four benchmarks in total, giving us $3 \times 3 \times 3 \times 3 \times 4 = 324$ traces to train on.

To limit the amount of training time, due to the sheer volume of data, a limit is set for the maximum amount of instructions recorded for each execution by the DynamoRIO wrapper to $K = 25 \times 10^6$. The training datasets consist of 90% of all datasets, while 5% of the

dataset are set aside for validation during training and 5% for any residual testing.

For the full model testing, a brand new dataset is collected in a similar manner, but with different options:

- L1 Data cache sizes: 1 kilobyte, 2 kilobytes, 64 kilobytes,
- L1 Instruction cache sizes: 1 kilobyte, 2 kilobytes, 64 kilobytes,
- LL unified cache sizes: 2 kilobytes, 4 kilobyte, 16 kilobytes,
- Core counts: 1, 2, 4,
- Benchmarks: Sysbench CPU, Sysbench Memory, Sysbench FileIO, GFTT.

Of particular interest here are the 64 kB cache sizes, as they are twice the size of the maximum cache sizes from training, which potentially may produce significant issues for the model. The general executions are all of the form observed in figures 8 and 9, for the benchmark CPU L1D errors.

Fig. 8 shows an example of our predictions on one sample of executions of L1D misses of the CPU benchmark; Fig. 9 presents the first 200 sub-sequences. The trace ends at 125000 on the x-axis, due to the fact that we have $K = 25 \times 10^6$ and $N = 200$, which then corresponds to $K/N = 125 \times 10^3$.

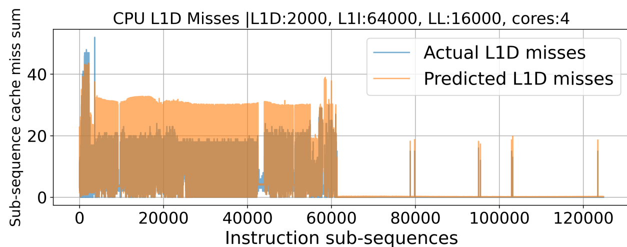

Fig. 8: Example of execution, L1D cache, Sysbench CPU Benchmark.

In this example, we can observe that the model has a tendency to exceed the number of cache misses fairly regularly. For this particular run, the E_{MSE} was 39.1 and E_{REP} was 67.4%, meaning that although the overall distribution of cache predictions is satisfactory with the E_{RMSE} around 6, the total aggregate counts leave room for improvement. The values for all E_{MSE} cache / core combinations for the tested programs, as well as E_{REP} can be observed in Figs. 10 to 13. Overall error confidence intervals with respect to core count for E_{MSE} and E_{REP} can be observed in Figs. 14 to 17. For example, for the L1I cache performance for the Memory benchmark, for the combination with L1D size 64k, 2 cores, L1I at 1k, LL at 2k, the E_{MSE} is 9.9 and the E_{REP} is 31.0, meaning that the model is 31% off the mark in terms of aggregate instructions when compared to the real value but that the overall distributed mean squared error is relatively small.

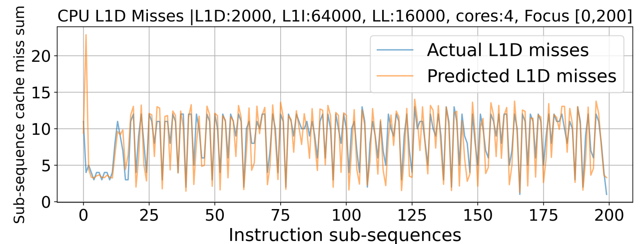

Fig. 9: Example of execution, L1D cache, Sysbench CPU Benchmark, first 200 sub-sequences

Overall, the model generally achieves satisfactory results for the distribution of cache misses, with a few notable outliers, which is to be expected given that the training procedure has been optimized to minimize MSE loss. However, when it comes to the total aggregate count of cache misses, the model consistently overestimates, particularly in the case of the GFTT program. For instance, in certain runs, especially with the 64k L1I - 16k LL cache combination, the model predicts up to 10 times more total cache misses than observed in the actual data, as shown in fig. 18. This overestimation may be due to insufficient training or the model's difficulty fully capturing the complexity of the executing algorithm, especially for edge-case cache sizes that fall well outside the range of training data. These extreme inputs could be pushing the model to output values that significantly deviate from reality.

Regarding the analysis of output values per core count, the relatively small differences between metrics across various core counts suggest that the model performs consistently, regardless of core count. While this limits our ability to infer a strong relationship between core count and model performance, it could also indicate that the model is resilient across different core configurations. Additionally, this observation may point to potential opportunities for further enhancing the underlying DynamoRIO cache simulation to better capture the complexities of multi-core environments.

The performance of the model (in terms of E_{RMSE}) when comparing the different programs chosen may be observed at Fig. 19. The E_{RMSE} drops off significantly when raising the L1D size and L1I size, which is due to the overall fewer misses, naturally due to the cache having to retrieve data from the main memory less often. The R^2 values across different cache sizes indicate

Fig. 10: CPU Benchmark prediction measurements; top in left to right order: L1D E_{MSE}, L1I E_{MSE}, LL E_{MSE}; bottom in left-to-right order: L1D E_{REP}, L1I E_{REP}, LL E_{REP}, green is better, red is worse

Fig. 11: Memory Benchmark prediction measurements; top in left to right order: L1D E_{MSE}, L1I E_{MSE}, LL E_{MSE}; bottom in left-to-right order: L1D E_{REP}, L1I E_{REP}, LL E_{REP}, green is better, red is worse

Fig. 12: FileIO Benchmark prediction measurements; top in left to right order: L1D E_{MSE}, L1I E_{MSE}, LL E_{MSE}; bottom in left-to-right order: L1D E_{REP}, L1I E_{REP}, LL E_{REP}, green is better, red is worse

how well the model explains the variability in the data, with higher R^2 values (close to 1) being desirable as they indicate better model performance. In this case,

L1D E_{MSE} (rows: Configuration (L1D size - Cores); columns: Configuration (L1I size - LL size))

	1k-16k	1k-2k	1k-4k	2k-16k	2k-2k	2k-4k	64k-16k	64k-2k	64k-4k
1k-1	30.5	38.0	36.8	31.6	39.0	38.1	30.6	25.0	27.2
1k-2	30.5	36.8	35.2	31.4	37.8	36.6	29.5	27.0	28.6
1k-4	32.4	35.0	33.7	32.6	36.0	34.7	29.9	29.6	30.6
2k-1	13.7	10.2	10.6	13.7	10.4	10.7	54.6	20.2	22.3
2k-2	13.7	10.2	10.7	13.9	10.2	10.7	58.4	20.9	24.0
2k-4	13.5	10.7	11.2	13.6	10.8	11.3	67.0	23.7	30.4
64k-1	15.9	15.1	15.2	16.0	15.1	15.3	18.4	18.0	17.9
64k-2	15.6	14.9	15.0	15.7	15.0	15.1	18.7	18.1	18.1
64k-4	15.7	14.9	15.0	15.8	14.9	15.1	19.5	18.4	18.6

L1I E_{MSE}

	1k-16k	1k-2k	1k-4k	2k-16k	2k-2k	2k-4k	64k-16k	64k-2k	64k-4k
1k-1	20.9	17.8	18.7	20.5	17.1	18.1	0.1	0.1	0.1
1k-2	20.0	16.8	17.7	18.8	15.5	16.3	0.1	0.1	0.1
1k-4	17.8	14.7	15.5	15.1	12.5	13.1	0.1	0.1	0.1
2k-1	21.0	18.0	18.8	20.7	17.3	18.3	0.1	0.1	0.1
2k-2	20.1	17.0	17.9	19.0	15.6	16.5	0.1	0.1	0.1
2k-4	17.9	14.8	15.7	15.3	12.7	13.3	0.1	0.1	0.1
64k-1	26.2	26.8	26.7	27.5	28.0	27.9	0.1	0.1	0.1
64k-2	25.9	26.4	26.3	26.9	27.3	27.3	0.1	0.1	0.1
64k-4	25.2	25.6	25.6	25.8	26.2	26.2	0.1	0.1	0.1

LL E_{MSE}

	1k-16k	1k-2k	1k-4k	2k-16k	2k-2k	2k-4k	64k-16k	64k-2k	64k-4k
1k-1	18.6	27.9	21.9	16.7	30.7	23.6	13.5	9.6	10.9
1k-2	17.1	28.3	22.4	15.6	30.6	23.7	12.6	8.8	9.3
1k-4	15.2	31.5	24.9	14.3	32.0	24.9	11.7	7.3	7.0
2k-1	18.7	24.7	23.0	16.9	27.1	24.9	13.0	8.1	8.2
2k-2	17.4	25.1	23.7	15.9	27.0	25.2	12.4	7.7	7.5
2k-4	15.8	27.4	26.4	14.8	27.9	26.7	11.9	7.1	6.7
64k-1	24.2	29.7	31.6	23.4	29.1	30.1	16.2	15.2	15.1
64k-2	23.3	29.3	30.5	22.4	28.4	28.8	16.5	15.3	15.4
64k-4	22.3	28.2	28.8	21.3	26.9	27.2	17.2	15.7	15.9

L1D E_{REP}

	1k-16k	1k-2k	1k-4k	2k-16k	2k-2k	2k-4k	64k-16k	64k-2k	64k-4k
1k-1	13.6	20.6	19.5	13.7	21.1	20.2	1.1	9.3	9.5
1k-2	13.2	19.2	17.7	13.2	19.9	18.6	3.6	9.8	9.1
1k-4	13.9	16.8	15.7	13.6	17.6	16.4	6.0	9.2	6.8
2k-1	7.8	2.7	3.2	7.3	2.4	2.8	28.1	15.6	15.1
2k-2	8.5	3.9	4.6	8.2	3.4	4.0	31.5	14.4	14.7
2k-4	8.8	6.0	6.7	8.6	5.2	5.9	37.4	13.3	16.1
64k-1	0.8	6.6	5.9	1.7	7.3	6.6	2.2	2.6	2.1
64k-2	1.2	7.3	6.6	2.2	7.9	7.3	4.5	0.4	0.1
64k-4	1.1	8.3	7.6	1.9	9.1	8.4	9.9	4.2	4.7

L1I E_{REP}

	1k-16k	1k-2k	1k-4k	2k-16k	2k-2k	2k-4k	64k-16k	64k-2k	64k-4k
1k-1	84.9	70.2	74.1	82.8	56.0	63.0	357.6	425.2	393.6
1k-2	84.3	69.2	73.4	78.8	52.0	58.9	390.0	446.4	405.9
1k-4	80.9	64.9	69.2	68.9	43.1	49.3	496.6	463.2	433.4
2k-1	85.5	70.8	74.6	84.1	57.7	64.3	517.0	421.8	418.8
2k-2	85.0	69.8	73.8	80.0	53.7	60.3	515.0	456.6	452.6
2k-4	81.7	65.9	70.3	70.3	44.5	50.8	562.2	535.3	546.8
64k-1	116.2	118.0	117.9	131.0	131.7	131.6	630.8	672.8	646.9
64k-2	116.8	118.0	118.0	130.4	130.4	130.5	726.8	699.1	677.5
64k-4	115.7	116.1	116.3	126.4	126.2	126.5	1092.8	793.5	799.8

LL E_{REP}

	1k-16k	1k-2k	1k-4k	2k-16k	2k-2k	2k-4k	64k-16k	64k-2k	64k-4k
1k-1	4.4	3.4	0.6	3.1	5.9	1.9	0.4	10.5	14.8
1k-2	2.8	5.3	1.4	1.5	7.8	3.9	3.6	7.1	10.5
1k-4	0.3	9.4	5.6	1.0	11.5	7.7	8.5	2.0	0.5
2k-1	4.1	1.7	2.1	2.9	4.2	4.5	1.2	6.8	7.3
2k-2	2.6	3.4	3.8	1.5	6.0	6.2	3.9	3.9	3.8
2k-4	0.3	7.2	7.6	1.0	9.5	9.7	9.1	3.2	3.7
64k-1	34.0	29.8	43.3	33.6	29.9	42.9	1.3	6.6	5.5
64k-2	33.1	28.7	42.2	32.6	28.6	41.5	0.6	4.6	3.5
64k-4	30.9	26.5	39.9	30.0	26.4	39.2	5.0	0.8	0.2

Fig. 13: GFTT prediction measurements; top in left to right order: L1D E_{MSE}, L1I E_{MSE}, LL E_{MSE}; bottom in left-to-right order: L1D E_{REP}, L1I E_{REP}, LL E_{REP}, green is better, red is worse

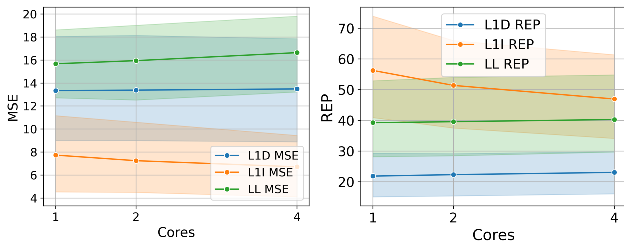

Fig. 14: Sysbench CPU Benchmark Error plot per core count

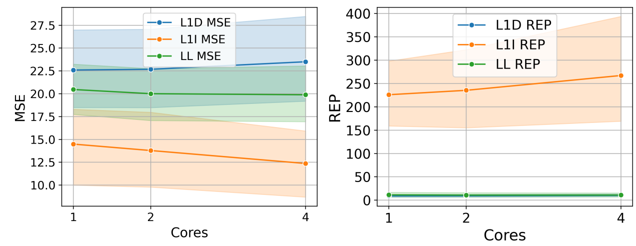

Fig. 17: GFTT Error plot per core count

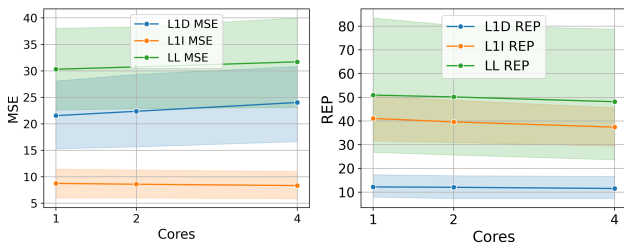

Fig. 15: Sysbench Memory Benchmark Error plot per core count

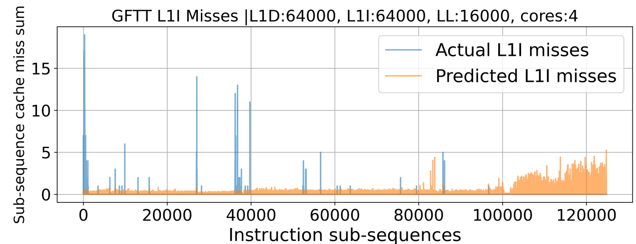

Fig. 18: GFTT, worst execution, L1I cache

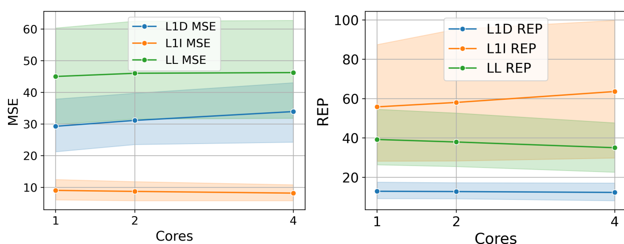

Fig. 16: Sysbench FileIO Benchmark Error plot per core count

CPU and FileIO show mostly positive R^2 values, indicating reasonable model performance.

The GFTT L1I shows a negative R^2 at cache sizes of 2k and 64k, suggesting that the model's performance in these specific cases may require further refinement - a topic for future work. Importantly, the CPU benchmark's cache behavior remains the most reliably well-modeled among our tests. Although FileIO and Memory benchmarks exhibit significant deviations at 2k for the L1D cache size and at 4k for the LL cache size, these findings highlight areas where the model could be improved - some improvements are suggested in section 8. This may indicate that our current model, in its present form, does not fully capture the cache dynam-

ics for these particular types of programs. Alternatively, there may be unique characteristics in these workloads sensitive to parameters beyond those explored in our study (Table 1) that the model has yet to account for.

All of the final numerical results of the executions may be observed for review purposes in the appendix, Tables 2 to 5.

6 Discussion

Our machine learning-based cache simulation model demonstrated consistent performance across the CPU, Memory, FileIO and GFTT benchmarks, with each completing 25 million instructions in approximately 45 seconds, whereas the duration was approximately 22 seconds on the i7-13700HX platform using DynamoRio. However, our model provides the premises for a critical advantage: the ability to execute in parallel multiple sequential subsets of a program trace. This capability allows for distributed processing across multiple platforms, with the potential of significantly reducing the overall simulation time compared to DynamoRIO.

Our assumption above is based on the following. Each execution trace was divided into eight parts, and the model was run and timed on each part, with the goal of analyzing the predictive power of the model on subsections of the program trace running on multiple GPUs. On average, the execution time of each section was 17.35 seconds with $\sigma = 2.14$ seconds, producing, on average, a **21% time reduction** to our model when compared to running the simulation through Cachesim.

While our model excels in scenarios with predictable data access patterns, particularly in CPU workloads, it encounters challenges with more complex I/O operations, as reflected in the negative R^2 values observed in the FileIO benchmark at smaller L1D cache sizes. This may be due to several contributing factors, including synchronous disk operations, which are known to increase memory latency and cause cache inefficiencies, leading to more frequent cache misses. Background I/O processes, such as fsync [2], can further exacerbate cache pressure, resulting in additional misses. Frequent and diverse file accesses can lead to sub-optimal data access patterns, increasing cache misses and making the modeling process more complex [19].

Despite these challenges, the flexibility of the model and the potential for parallelization represent significant advantages. Unlike DynamoRIO, which requires running the entire program for effective simulation, our approach allows for faster, more scalable simulations. This work represents a proof-of-concept and a first step towards a more ambitious model. Future work should focus on enhancing the model's accuracy across varied and complex workloads, particularly by improving its handling of edge cases and its generalization capabilities. These improvements could make the model a more robust tool for simulated cache performance across a wider range of applications.

[2] fsync is a system call that forces a file's in-memory data to be written to disk, ensuring data integrity in case of a system crash. However, this operation can introduce significant performance overhead, especially in I/O-intensive workloads, due to the additional time required for disk synchronization [18].

7 Related Work

The predictive capabilities of LSTM-based models for cache miss occurrences on instrumentation traces have previously been investigated, as demonstrated by R. Jha et al. [13], validating the feasibility of employing LSTMs to anticipate cache miss patterns derived from program behaviors. This investigation underpins our own predictive modeling endeavors, yet our approach uniquely emphasizes the variability of cache sizes alongside the programs being analyzed. Various successful integrations of LSTM architectures into prefetchers aimed at mitigating cache misses - rather than simply modeling their distribution - have been documented, exemplified by J. Rogers [30] and Y. Zeng et al. [36]. For simulation of program performance using sequential deep learning, the Ithemal tool [24] epitomizes a sophisticated LSTM-driven technique to forecast instruction block latency; it operates, though, under the presumption of negligible cache misses within the program trace, contrasting with the focus of our method.

An alternative approach to latency prediction, predicated on parallelization rather than sequential modeling, is detailed by Li et al. [22] with the SimNet temporal convolutional network framework. Further work by Li et al. [21] showcases a higher-level simulation framework called PerfVec that is demonstrated in an instruction latency reduction objective against different combinations of cache sizes.

The work of S. Pandey et al. [27] suggests optimizations of SimNet and Ithemal to reduce the costs of moving datasets from the CPU to the GPU for overhead reduction purposes, and the most very recent work by S. Pandey et al. [28] showcases an effort in the direction of a microarchitecture simulator and dataset gathering approach that can be transferred between architectures differing by certain cache size and CPU pipeline characteristics. However, this tool focuses on the forecasting of metrics including cache misses on the granularity level of a single instruction, whereas our tool treats instruction sub-sequences in a statistical manner, which gives

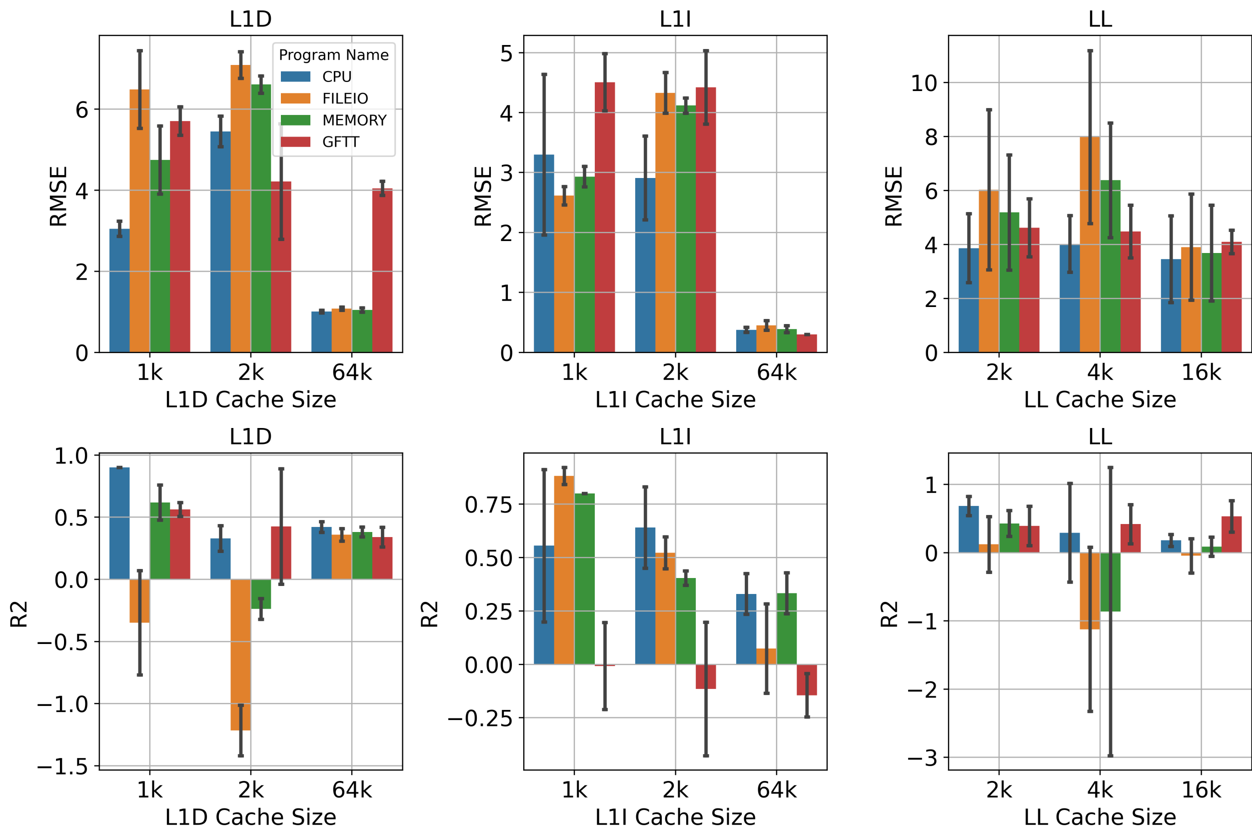

Fig. 19: E_{RMSE}/R^2 analysis of the programs used in the model testing, cache E_{RMSE}/R^2 against respective cache sizes (an R^2 of 1.0 is ideal), standard deviation for the confidence intervals

the user the choice between more granularity (fewer instructions per sub-sequence, but slower) or more speed (more instructions per sub-sequence, but less focus on individual instructions).

Additionally, our approach gives the user flexibility in terms of providing estimates for hypothetical but possible cache and multicore/single-core architectures, whereas the availability of combinations in the referenced work is constrained by the microarchitecture loaded in gem5. Moreover, our work also implements the ability of introducing higher core counts and the corresponding L1D/L1I caches, as well as the corresponding analysis that can be attained from running the model.

The robust and frequently updated software instrumentation tool, DynamoRIO, is a preferred solution among contemporary computer science researchers, as evidenced by [8,35]. We utilize DynamoRIO not solely for its prominence and ongoing advancements, but also for its open-source status, user-friendliness, and seamless integration with our tool for capturing program execution traces into its infrastructure.

8 Conclusion and Future work

This work introduces a new method for modeling cache performance tailored to specific applications using a distributable deep learning model.

We have shown that our approach not only mimics traditional simulators but also replicates results across multiple architectures. Our model handles complex Sysbench benchmarks and GFTT algorithm implementations with remarkable accuracy, even when faced with unfamiliar cache sizes and core counts. Although it tends to predict higher cache miss rates, it is excellent at pinpointing high-activity sections of program execution.

Though DynamoRIO outpaces our model on a single machine, our model has the potential to outperform it through parallelization: just three to four concurrent GPU iterations would suffice.

While relying on tools that provide hardware-specific traces, our approach itself is hardware-agnostic.

We have also provided the initial steps towards extending our approach on individual cores of multicore architectures.

Future work. We are set on fine-tuning our model on the present architecture, such as employing positional encoding for the tokens. We will also focus on minimizing cache-miss discrepancies during training by honing in on specific application benchmarks rather than broad benchmark sets. In addition, we plan to enhance the model architecture with batch normalization and regularization techniques to push performance further.

Extracting more features from Cachesim and breaking down instructions more granularly are on the agenda, as is expanding our approach using advanced simulators like gem5 and ZSim for richer data, for example extracting used registers, individual instruction latencies, branch predictor types and branch predictions and mispredictions. Richer trace data may prove to be a key component in tackling intense workloads that depend heavily on additional information about the executed instructions and the underlying access patterns, such as the FileIO benchmark where our approach faced a significant challenge, as was shown in section 6. In addition to this, we will explore features of independent cores with regard to cache modeling.

We are also exploring different recurrent architectures, such as temporal convolutional networks and ga ted recurrent units, to boost predictive accuracy and efficiency. Crucially, our model is non-fixed; by feeding architectures as features into the model, we can effectively compare various CPU characteristics. Our ultimate goal? Develop a flexible ML simulation framework that holistically models hardware for specialized software, allowing direct comparisons across diverse CPU configurations. The challenges are significant, but so are the opportunities, and we are committed to making substantial contributions now and in the future.

Acknowledgement. The author's work was partially supported by the Swedish Knowledge Foundation via the project PerFlex - *Performant and Flexible digital Systems through Verifiable Artificial Intelligence*, grant nr. 20220033.

References

1. Summary of the tokenizers. https://huggingface.co/docs/transformers/en/tokenizer_summary. (Accessed on 03/04/2024)
2. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)
3. Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M.A., Al-Amidie, M., Farhan, L.: Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. Journal of Big Data 2021 8:1 **8**, 1–74 (2021). DOI 10.1186/S40537-021-00444-8. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8
4. Bellard, F.: Qemu, a fast and portable dynamic translator. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05, p. 41. USENIX Association, USA (2005)
5. Bruening, D., Zhao, Q., Amarasinghe, S.: Transparent dynamic instrumentation (2012). DOI 10.1145/2151024.2151043
6. Carvalho, C.: The gap between processor and memory speeds. In: Proc. of IEEE International Conference on Control and Automation, vol. 5000, p. 15000 (2002)
7. Castaño, J., Martínez-Fernández, S., Franch, X., Bogner, J.: Analyzing the evolution and maintenance of ml models on hugging face. In: Proceedings of the 21st International Conference on Mining Software Repositories, MSR '24, p. 607–618. Association for Computing Machinery, New York, NY, USA (2024). DOI 10.1145/3643991.3644898
8. Darashkevich, E., Rusyaev, M.R., Russia, H., Korostinskiy, M.R., Bugayenko, M.Y.: A minority of c++ objects account for the majority of allocation cpu time URL https://github.com/EugeneDar/dynamotool
9. Etiemble, D.: 45-year CPU evolution: one law and two equations. In: Second Workshop on Pioneering Processor Paradigms. Vienne, Austria (2018). URL https://hal.science/hal-01719766
10. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach, 5 edn. Morgan Kaufmann, Amsterdam (2012)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**, 1735–1780 (1997). DOI 10.1162/NECO.1997.9.8.1735
12. Jadon, A., Patil, A., Jadon, S.: A comprehensive survey of regression based loss functions for time series forecasting (2022). URL https://arxiv.org/abs/2211.02989
13. Jha, R., Karuvally, A., Tiwari, S., Moss, J.E.B.: Cache miss rate predictability via neural networks
14. Kopytov, A.: Sysbench manual. MySQL AB pp. 2–3 (2012)
15. Kratsios, A.: The universal approximation property. Annals of Mathematics and Artificial Intelligence **89**, 435–469 (2019). DOI 10.1007/s10472-020-09723-1. URL http://arxiv.org/abs/1910.03344http://dx.doi.org/10.1007/s10472-020-09723-1
16. Kumar, S., Singh, P.K.: An overview of modern cache memory and performance analysis of replacement policies. Proceedings of 2nd IEEE International Conference on Engineering and Technology, ICETECH 2016 **17**, 210–214 (2016). DOI 10.1109/ICETECH.2016.7569243
17. Kwak, H., Lee, B., Hurson, A., Suk-Han Yoon, Woo-Jong Hahn: Effects of multithreading on cache performance. IEEE Transactions on Computers **48**(2), 176–184 (1999). DOI 10.1109/12.752659. URL http://ieeexplore.ieee.org/document/752659/
18. Lee, C.G., Byun, H., Noh, S., Kang, H., Kim, Y.: Write optimization of log-structured flash file system for parallel i/o on manycore servers. In: Proceedings of the 12th ACM International Conference on Systems and Storage, SYSTOR '19, p. 21–32. Association for Computing Machinery, New York, NY, USA (2019). DOI 10.1145/3319647.3325828
19. Lee, J., Bahn, H.: File access characteristics of deep learning workloads and cache-friendly data management. In: 2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI),

pp. 328–331 (2023). DOI 10.1109/EECSI59885.2023.10295817
20. Lee, J., Kim, H., Vuduc, R.: When prefetching works, when it doesn't, and why. ACM Trans. Archit. Code Optim. **9**(1) (2012). DOI 10.1145/2133382.2133384
21. Li, L., Flynn, T., Hoisie, A.: Learning generalizable program and architecture representations for performance modeling (2024)
22. Li, L., Pandey, S., Flynn, T., Liu, H., Wheeler, N., Hoisie, A.: Simnet: Accurate and high-performance computer architecture simulation using deep learning. Proceedings of the ACM on Measurement and Analysis of Computing Systems **6**(2), 1–24 (2022)
23. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. Acm sigplan notices **40**(6), 190–200 (2005)
24. Mendis, C., Renda, A., Amarasinghe, S., Carbin, M.: Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In: International Conference on machine learning, pp. 4505–4515. PMLR (2019)
25. Nethercote, N., Seward, J.: Valgrind: a framework for heavyweight dynamic binary instrumentation. In: Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '07, p. 89–100. Association for Computing Machinery (2007). DOI 10.1145/1250734.1250746
26. Nishijima, T.: Universal approximation theorem for neural networks (2021). URL https://arxiv.org/abs/2102.10993v1
27. Pandey, S., Li, L., Flynn, T., Hoisie, A., Liu, H.: Scalable deep learning-based microarchitecture simulation on gpus. In: SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15 (2022). DOI 10.1109/SC41404.2022.00084
28. Pandey, S., Yazdanbakhsh, A., Liu, H.: Tao: Re-thinking dl-based microarchitecture simulation. Proceedings of the ACM on Measurement and Analysis of Computing Systems **8**, 1–25 (2024). DOI 10.1145/3656012
29. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
30. Rogers, J.: Effects of an lstm composite prefetcher (2019)
31. Sandberg, A., Nikoleris, N., Carlson, T.E., Hagersten, E., Kaxiras, S., Black-Schaffer, D.: Full speed ahead: Detailed architectural simulation at near-native speed. In: 2015 IEEE International Symposium on Workload Characterization, pp. 183–192. IEEE (2015)
32. Sherstinsky, A.: Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. Physica D: Nonlinear Phenomena **404**, 132306 (2020)
33. pandas development team, T.: pandas-dev/pandas: Pandas (2020). DOI 10.5281/zenodo.3509134
34. Yaghoobi, S.: Leveraging machine learning for fast performance prediction for industrial systems : Data-driven cache simulator. Master's thesis, Mälardalen University, School of Innovation, Design and Engineering (2024)
35. Yang, Y., Gao, C., Li, Z., Wang, Y., Wang, R.: Binary level concolic execution on windows with rich instrumentation based taint analysis. In: International Symposium on Dependable Software Engineering: Theories, Tools, and Applications, pp. 351–367. Springer (2023)
36. Zeng, Y., Guo, X.: Long short term memory based hardware prefetcher: a case study. In: Proceedings of the International Symposium on Memory Systems, MEMSYS '17, p. 305–311. Association for Computing Machinery, New York, NY, USA (2017). DOI 10.1145/3132402.3132405

Appendix - PyTorch model code

```
class CombinedLSTMModel(nn.Module):
    def __init__(
        self,
        token_vocab_size, # characteristic of the tokenizer
        token_embedding_dim, # number of tokens per instruction
        access_feature_size, # number of trace features
        hidden_dim, # matrix size of the hidden dimension of the LSTM layer
        output_dim, # number of outputs of our model
        num_layers, # number of consecutive LSTM layers
        dropout, # dropout rate between the LSTM layers
    ):
        super(CombinedLSTMModel, self).__init__()
        self.token_embedding = nn.Embedding(token_vocab_size, token_embedding_dim)
        # Input size must be the sum of flattened token embedding size
        # and access feature size
        self.lstm = nn.LSTM(
            input_size=token_embedding_dim**2 + access_feature_size,
            hidden_size=hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout,
        )
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, token_features, access_features):
        # Embed the token features
        embedded_tokens = self.token_embedding(token_features)
        # Assume access_features is already of the appropriate size and
        # requires no embedding
        # LSTM can handle variable sequence lengths if necessary, but here we
        # focus on batch size flexibility
        elongated_embeddings = embedded_tokens.view(
            token_features.size(0), token_features.size(1), -1
        )
        combined_features = torch.cat(
            (elongated_embeddings, access_features), dim=2
        )  # Concatenate along the feature dimension
        lstm_out, _ = self.lstm(combined_features)
        # Using the last time step's output
        output = self.fc(lstm_out[:, -1, :])
        return output
```

Appendix - Execution Results

Table 2: Table of results for Sysbench CPU Benchmark Model executions

				L1D		L1I		LL		L1D				L1I				LL				Execution time per block							
L1D Size (bytes)	L1I Size (bytes)	LL Size (bytes)	Cores	Total Actual	Total Predicted	Total Actual	Total Predicted	Total Actual	Total Predicted	MSE	RMSE	R2	Abs. Err. Percent	MSE	RMSE	R2	Abs. Err. Percent	MSE	RMSE	R2	Abs. Err. Percent	t_1 s	t_2 s	t_3 s	t_4 s	t_5 s	t_6 s	t_7 s	t_8 s
1k	1k	2k	1	929,857	970,716	343,013	428,490	1,056,247	970,716	10.5	3.2	0.9	4.4	5.6	2.4	0.8	24.9	25.4	5.0	0.8	20.0	17	17	15	14	16	17	19	16
1k	1k	2k	2	929,842	981,834	343,007	426,989	1,056,225	981,834	10.1	3.2	0.9	5.6	5.5	2.4	0.8	24.5	26.1	5.1	0.8	21.3	18	16	16	18	17	16	17	18
1k	1k	4k	1	929,857	945,486	343,013	429,432	952,906	945,486	12.0	3.5	0.9	1.7	5.6	2.4	0.8	25.2	18.9	4.3	0.8	4.4	18	16	14	19	18	16	18	16
1k	1k	4k	2	929,842	955,566	343,007	427,865	952,871	955,566	11.6	3.4	0.9	2.8	5.6	2.4	0.8	24.7	19.7	4.4	0.8	8.1	17	16	16	18	18	17	17	18
1k	1k	2k	4	929,794	999,360	343,014	424,442	1,056,170	999,360	9.5	3.1	0.9	7.5	5.4	2.3	0.8	23.7	27.0	5.2	0.8	23.1	18	14	15	17	18	17	18	16
1k	1k	4k	4	929,794	974,292	343,014	424,945	952,791	974,292	11.1	3.3	0.9	4.8	5.4	2.3	0.8	23.9	22.8	4.8	0.8	17.1	17	13	13	16	16	14	18	18
1k	1k	16k	1	929,857	1,058,499	343,013	435,221	386,988	1,058,499	8.0	2.8	0.9	13.8	5.9	2.4	0.8	26.9	25.3	5.0	0.1	55.7	17	16	14	16	16	17	17	17
1k	1k	16k	2	929,842	1,061,302	343,007	433,769	386,985	1,061,302	8.1	2.8	0.9	14.1	5.9	2.4	0.8	26.5	25.5	5.0	0.1	55.0	17	14	14	16	17	16	16	18
1k	1k	16k	4	929,794	1,063,067	343,014	431,174	386,969	1,063,067	8.4	2.9	0.9	14.3	5.8	2.4	0.8	25.7	25.7	5.1	0.1	52.7	18	14	17	16	19	18	18	16
1k	2k	2k	1	929,857	1,006,300	314,724	402,809	1,027,114	1,006,300	8.9	3.0	0.9	8.2	5.6	2.4	0.8	28.0	24.4	4.9	0.8	23.7	17	16	18	18	14	17	18	16
1k	2k	2k	2	929,842	1,013,684	314,737	399,673	1,027,116	1,013,684	8.7	3.0	0.9	9.0	5.6	2.4	0.8	27.0	24.6	5.0	0.8	24.5	16	13	16	16	17	18	18	16
1k	2k	4k	1	929,857	980,388	314,724	402,579	932,916	980,388	10.7	3.3	0.9	5.4	5.8	2.4	0.8	27.9	17.0	4.1	0.8	6.9	16	17	17	15	16	14	18	17
1k	2k	4k	2	929,842	987,532	314,737	399,427	932,909	987,532	10.5	3.2	0.9	6.2	5.7	2.4	0.8	26.9	18.1	4.3	0.8	10.3	16	17	16	14	17	19	18	18
1k	2k	16k	1	929,857	1,072,163	314,724	409,211	382,912	1,072,163	8.0	2.8	0.9	15.3	6.3	2.5	0.7	30.0	24.5	4.9	0.1	54.4	16	15	17	17	16	14	18	20
1k	2k	16k	2	929,842	1,073,120	314,737	406,698	382,911	1,073,120	8.1	2.8	0.9	15.4	6.2	2.5	0.7	29.2	24.4	4.9	0.1	53.1	16	16	17	17	18	19	18	18
1k	2k	2k	4	929,794	1,024,456	314,774	390,779	1,027,090	1,024,456	8.5	2.9	0.9	10.2	5.6	2.4	0.8	24.1	25.0	5.0	0.8	25.1	16	17	13	17	17	17	15	17
1k	2k	4k	4	929,794	1,001,137	314,774	391,069	932,855	1,001,137	10.2	3.2	0.9	7.7	5.7	2.4	0.8	24.2	21.7	4.7	0.8	18.2	16	14	16	19	17	18	18	18
1k	2k	16k	4	929,794	1,072,981	314,774	400,051	382,891	1,072,981	8.3	2.9	0.9	15.4	6.2	2.5	0.7	27.1	24.1	4.9	0.1	49.4	18	15	16	16	14	18	16	18
1k	64k	2k	1	929,857	1,033,081	2,911	4,557	666,006	1,033,081	8.9	3.0	0.9	11.1	0.1	0.3	0.4	56.5	16.4	4.0	0.6	31.0	16	14	15	16	19	16	17	19
1k	64k	2k	2	929,842	1,029,859	3,167	4,700	666,269	1,029,859	9.0	3.0	0.9	10.8	0.1	0.4	0.4	48.4	16.5	4.1	0.6	31.1	16	13	14	14	16	17	18	19
1k	64k	4k	1	929,857	1,050,885	2,911	5,009	572,400	1,050,885	8.9	3.0	0.9	13.0	0.1	0.3	0.4	72.1	19.6	4.4	0.5	49.2	18	16	14	17	19	19	19	19
1k	64k	4k	2	929,842	1,047,054	3,167	5,089	572,675	1,047,054	9.0	3.0	0.9	12.6	0.1	0.4	0.4	60.7	20.0	4.5	0.5	49.9	16	15	13	17	15	17	19	17
1k	64k	16k	1	929,857	1,129,502	2,911	5,718	230,767	1,129,502	8.9	3.0	0.9	21.5	0.1	0.3	0.4	96.4	11.7	3.4	0.2	60.8	18	14	14	18	20	17	15	19
1k	64k	16k	2	929,842	1,128,150	3,167	5,889	231,036	1,128,150	8.9	3.0	0.9	21.3	0.1	0.4	0.3	85.9	11.8	3.4	0.2	57.8	17	13	14	15	18	16	18	19
1k	64k	2k	4	929,794	1,021,430	3,534	5,174	666,598	1,021,430	9.4	3.1	0.9	9.9	0.2	0.4	0.4	46.4	16.7	4.1	0.6	30.8	18	16	13	18	16	17	16	17
1k	64k	4k	4	929,794	1,037,912	3,534	5,502	573,007	1,037,912	9.3	3.0	0.9	11.6	0.2	0.4	0.4	55.7	20.3	4.5	0.5	50.4	16	15	15	17	16	18	16	18
1k	64k	16k	4	929,794	1,125,346	3,534	6,401	231,279	1,125,346	9.1	3.0	0.9	21.0	0.2	0.4	0.4	81.1	12.0	3.5	0.2	51.4	17	16	14	19	16	18	17	17
2k	1k	2k	1	653,093	859,704	343,013	427,986	989,786	859,704	25.8	5.1	0.4	31.6	5.6	2.4	0.8	24.8	22.3	4.7	0.8	18.7	18	16	15	16	17	18	18	18
2k	1k	2k	2	653,059	874,546	343,007	426,834	989,742	874,546	26.4	5.1	0.4	33.9	5.5	2.4	0.8	24.4	23.4	4.8	0.8	20.4	19	19	15	16	16	17	14	15
2k	1k	4k	1	653,093	825,456	343,013	428,950	936,508	825,456	24.2	4.9	0.5	26.4	5.6	2.4	0.8	25.1	19.0	4.4	0.8	0.4	19	13	17	16	17	18	15	18
2k	1k	4k	2	653,059	838,541	343,007	427,779	936,463	838,541	24.6	5.0	0.5	28.4	5.6	2.4	0.8	24.7	19.2	4.4	0.8	4.3	17	19	13	17	18	16	18	18
2k	1k	16k	1	653,093	978,540	343,013	436,372	393,565	978,540	31.0	5.6	0.3	49.8	5.9	2.4	0.8	27.2	25.6	5.1	0.1	52.9	16	15	13	17	17	16	18	16
2k	1k	16k	2	653,059	985,513	343,007	435,292	393,554	985,513	31.1	5.6	0.3	50.9	5.9	2.4	0.8	26.9	26.0	5.1	0.1	52.9	19	15	15	16	18	16	15	18
2k	2k	2k	1	653,093	907,625	314,724	404,139	964,751	907,625	26.8	5.2	0.4	39.0	5.8	2.4	0.8	28.4	23.4	4.8	0.8	23.6	13	11	10	11	11	12	16	17
2k	2k	2k	2	653,059	918,673	314,737	401,728	964,728	918,673	27.3	5.2	0.4	40.7	5.7	2.4	0.8	27.6	24.2	4.9	0.7	24.8	12	10	10	11	11	12	12	12
2k	2k	4k	1	653,093	872,860	314,724	404,308	921,435	872,860	25.2	5.0	0.4	33.7	6.0	2.4	0.8	28.5	17.7	4.2	0.8	4.2	19	15	13	16	17	17	15	18
2k	2k	4k	2	653,059	885,091	314,737	401,539	921,412	885,091	25.9	5.1	0.4	35.5	5.9	2.4	0.8	27.6	18.3	4.3	0.8	7.8	15	16	16	16	17	15	17	15
2k	2k	16k	1	653,093	1,004,574	314,724	411,536	389,470	1,004,574	32.2	5.7	0.3	53.8	6.3	2.5	0.7	30.8	25.1	5.0	0.1	53.1	21	18	14	15	14	17	14	17
2k	2k	16k	2	653,059	1,008,908	314,737	409,720	389,464	1,008,908	32.3	5.7	0.3	54.5	6.3	2.5	0.7	30.2	25.2	5.0	0.1	52.5	20	18	16	16	14	17	17	15
2k	64k	2k	1	653,093	958,709	2,911	4,296	640,176	958,709	30.2	5.5	0.3	46.8	0.1	0.3	0.4	47.6	15.3	3.9	0.7	27.6	17	15	13	16	17	17	18	17
2k	64k	2k	2	653,059	953,539	3,167	4,416	640,406	953,539	29.9	5.5	0.3	46.0	0.1	0.4	0.4	39.4	15.3	3.9	0.7	27.3	20	13	16	16	16	15	17	18
2k	64k	4k	1	653,093	984,560	2,911	4,816	593,580	984,560	31.9	5.7	0.3	50.8	0.1	0.3	0.4	65.4	16.8	4.1	0.6	36.2	16	14	15	16	17	17	18	18
2k	64k	4k	2	653,059	978,740	3,167	4,893	593,832	978,740	31.6	5.6	0.3	49.9	0.1	0.4	0.4	54.5	16.8	4.1	0.6	36.4	16	14	16	17	17	17	15	18
2k	64k	16k	1	653,093	1,098,346	2,911	5,778	234,682	1,098,346	38.9	6.2	0.1	68.2	0.1	0.3	0.4	98.5	11.7	3.4	0.2	55.8	16	14	16	17	17	17	17	17
2k	64k	16k	2	653,059	1,096,890	3,167	5,934	234,953	1,096,890	39.0	6.2	0.1	68.0	0.1	0.4	0.4	87.4	11.8	3.4	0.2	52.8	16	16	16	17	18	19	18	17
2k	1k	2k	4	653,011	900,456	343,014	425,028	989,682	900,456	27.3	5.2	0.4	37.9	5.5	2.3	0.8	23.9	25.3	5.0	0.7	23.1	18	18	16	16	14	16	17	16
2k	1k	4k	4	653,011	868,687	343,014	425,663	936,333	868,687	25.9	5.1	0.4	33.0	5.5	2.3	0.8	24.1	21.6	4.6	0.8	13.6	17	16	13	17	19	16	18	18
2k	1k	16k	4	653,011	994,444	343,014	433,108	393,524	994,444	31.4	5.6	0.3	52.3	5.8	2.4	0.8	26.3	26.4	5.1	0.1	51.7	18	18	18	17	16	14	14	18
2k	2k	2k	4	653,011	936,983	314,774	394,140	964,697	936,983	28.0	5.3	0.4	43.5	5.8	2.4	0.8	25.2	25.4	5.0	0.7	26.1	12	11	11	11	12	12	12	12
2k	2k	4k	4	653,011	909,891	314,774	394,228	921,307	909,891	27.0	5.2	0.4	39.3	5.9	2.4	0.8	25.2	21.2	4.6	0.8	15.9	17	15	14	19	14	17	18	18
2k	2k	16k	4	653,011	1,011,652	314,774	404,442	389,432	1,011,652	32.4	5.7	0.3	54.9	6.3	2.5	0.7	28.5	25.1	5.0	0.1	49.7	18	16	15	16	18	17	15	16
2k	64k	2k	4	653,011	940,205	3,534	4,920	640,711	940,205	29.3	5.4	0.3	44.0	0.1	0.4	0.4	39.2	15.3	3.9	0.7	26.3	18	18	15	16	17	18	18	19
2k	64k	4k	4	653,011	962,839	3,534	5,285	594,115	962,839	30.7	5.5	0.3	47.4	0.2	0.4	0.4	49.5	16.7	4.1	0.6	35.7	17	16	16	17	14	15	17	18
2k	64k	16k	4	653,011	1,093,036	3,534	6,432	235,184	1,093,036	39.1	6.3	0.1	67.4	0.2	0.4	0.4	82.0	12.0	3.5	0.2	45.8	17	13	17	16	14	16	18	19
64k	1k	2k	1	75,338	72,780	343,013	864,998	399,109	72,780	1.0	1.0	0.4	3.4	30.3	5.5	-0.1	152.2	10.6	3.3	0.6	53.0	19	15	15	19	15	18	18	19
64k	1k	2k	2	75,448	72,227	343,007	825,070	399,217	72,227	1.0	1.0	0.4	4.3	27.9	5.3	0.0	140.5	10.5	3.2	0.6	50.8	16	14	16	15	16	16	16	18
64k	1k	4k	1	75,338	73,601	343,013	842,304	191,035	73,601	1.0	1.0	0.5	2.3	28.9	5.4	0.0	145.6	19.2	4.4	-1.0	145.3	17	13	14	16	16	15	17	20
64k	1k	4k	2	75,448	73,258	343,007	800,436	191,161	73,258	1.0	1.0	0.4	2.9	26.3	5.1	0.1	133.4	19.6	4.4	-1.1	149.0	18	14	13	16	16	16	19	18
64k	1k	16k	1	75,338	71,651	343,013	841,378	90,556	71,651	0.9	1.0	0.5	4.9	28.6	5.4	0.0	145.3	2.5	1.6	0.2	26.1	17	14	14	16	18	15	18	17
64k	1k	16k	2	75,448	71,401	343,007	797,472	90,703	71,401	1.0	1.0	0.5	5.4	25.8	5.1	0.1	132.5	2.5	1.6	0.2	24.9	20	13	14	15	15	18	16	19
64k	2k	2k	1	75,338	74,126	314,724	646,284	385,200	74,126	1.0	1.0	0.4	1.6	17.5	4.2	0.3	105.3	6.6	2.6	0.8	32.3	19	13	16	16	19	15	16	19
64k	2k	2k	2	75,448	74,230	314,737	609,430	385,357	74,230	1.0	1.0	0.4	1.6	15.7	4.0	0.4	93.6	6.6	2.6	0.8	30.6	16	13	13	14	16	17	18	19
64k	2k	4k	1	75,338	74,811	314,724	624,193	190,097	74,811	1.0	1.0	0.4	0.7	16.4	4.0	0.3	98.3	18.1	4.3	-0.9	131.9	19	14	16	17	16	15	16	19
64k	2k	4k	2	75,448	75,087	314,737	590,308	190,228	75,087	1.0	1.0	0.4	0.5	14.7	3.8	0.4	87.6	18.1	4.3	-0.9	131.7	17	13	14	17	15	18	17	18
64k	2k	16k	1	75,338	73,604	314,724	618,119	90,644	73,604	1.0	1.0	0.5	2.3	16.2	4.0	0.3	96.4	2.3	1.5	0.3	22.3	16	13	16	17	14	17	18	16
64k	2k	16k	2	75,448	73,635	314,737	589,930	90,788	73,635	1.0	1.0	0.5	2.4	14.7	3.8	0.4	87.4	2.3	1.5	0.3	21.1	17	13	13	16	16	16	17	19
64k	64k	2k	1	75,338	57,844	2,911	2,921	77,144	57,844	1.1	1.0	0.4	23.2	0.1	0.4	0.2	0.4	1.3	1.1	0.4	21.7	17	17	14	16	16	16	20	20
64k	64k	2k	2	75,448	58,246	3,167	3,130	77,474	58,246	1.1	1.0	0.4	22.8	0.2	0.4	0.2	1.2	1.4	1.2	0.4	21.3	17	14	16	16	18	15	16	20
64k	64k	4k	1	75,338	56,796	2,911	2,870	74,068	56,796	1.1	1.0	0.4	24.6	0.1	0.4	0.2	1.4	1.3	1.1	0.4	20.7	17	14	16	16	15	18	15	16
64k	64k	4k	2	75,448	57,059	3,167	3,071	74,395	57,059	1.1	1.0	0.4	24.4	0.2	0.4	0.2	3.0	1.4	1.2	0.3	20.5	17	14	16	20	17	17	16	19
64k	64k	16k	1	75,338	51,295	2,911	2,625	69,030	51,295	1.1	1.0	0.4	31.9	0.1	0.4	0.2	9.8	1.3	1.1	0.4	27.3	17	16	14	15	17	16	17	19
64k	64k	16k	2	75,448	51,105	3,167	2,804	69,319	51,105	1.1	1.0	0.4	32.3	0.2	0.4	0.2	11.5	1.3	1.1	0.3	27.8	17	16	14	18	16	16	19	17
64k	1k	2k	4	75,561	71,770	343,014	767,735	399,338	71,770	1.0	1.0	0.4	5.0	24.6	5.0	0.1	123.8	10.5	3.2	0.6	47.5	17	14	17	16	15	15	18	16
64k	1k	4k	4	75,561	72,657	343,014	744,153	191,332	72,657	1.0	1.0	0.4	3.8	22.9	4.8	0.2	116.9	20.7	4.6	-1.2	159.4	19	13	15	17	15	19	19	17
64k	1k	16k	4	75,561	70,834	343,014	742,269	90,830	70,834	1.0	1.0	0.5	6.3	22.5	4.7	0.2	116.4	2.5	1.6	0.2	22.7	18	14	14	15	15	16	16	18
64k	2k	2k	4	75,561	74,496	314,774	580,450	385,492	74,496	1.0	1.0	0.4	1.4	14.2	3.8	0.4	84.4	6.7	2.6	0.8	29.6	16	16	14	17	14	19	17	16
64k	2k	4k	4	75,561	75,086	314,774	564,912	190,382	75,086	1.0	1.0	0.4	0.6	13.3	3.6	0.5	79.5	18.3	4.3	-0.9	133.7	17	13	14	16	15	17	19	18
64k	2k	16k	4	75,561	73,031	314,774	566,406	90,913	73,031	1.0	1.0	0.4	3.3	13.2	3.6	0.5	79.9	2.3	1.5	0.3	18.5	18	15	14	15	16	15	17	21
64k	64k	2k	4	75,561	58,703	3,534	3,645	77,942	58,703	1.1	1.1	0.4	22.3	0.2	0.4	0.2	3.1	1.4	1.2	0.3	20.7	17	14	14	17	15	16	18	19
64k	64k	4k	4	75,561	57,393	3,534	3,585	74,739	57,393	1.1	1.1	0.4	24.0	0.2	0.4	0.2	1.4	1.4	1.2	0.3	20.1	17	14	15	18	18	15	18	16
64k	64k	16k	4	75,561	50,418	3,534	3,316	69,582	50,418	1.1	1.1	0.4	33.3	0.2	0.4	0.2	6.2	1.3	1.2	0.3	28.9	17	16	15	17	16	18	18	19

Table 3: Table of results for Sysbench FileIO Benchmark Model executions

				L1D		L1I		LL		L1D				L1I				LL				Execution time per block							
L1D Size (bytes)	L1I Size (bytes)	LL Size (bytes)	Cores	Total Actual	Total Predicted	Total Actual	Total Predicted	Total Actual	Total Predicted	MSE	RMSE	R2	Abs. Err. Percent	MSE	RMSE	R2	Abs. Err. Percent	MSE	RMSE	R2	Abs. Err. Percent	t_1 s	t_2 s	t_3 s	t_4 s	t_5 s	t_6 s	t_7 s	t_8 s
1k	1k	2k	1	1,654,670	1,399,186	1,351,265	1,418,858	2,886,788	1,399,186	53.70	7.30	-0.60	15.40	6.30	2.50	0.90	5.00	71.00	8.40	0.20	9.10	19	16	18	17	17	18	18	19
1k	1k	2k	2	1,649,213	1,366,599	1,351,290	1,417,640	2,880,669	1,366,599	57.20	7.60	-0.80	17.10	6.30	2.50	0.90	4.90	76.00	8.70	0.20	10.00	19	16	18	17	17	17	18	17
1k	1k	4k	1	1,654,670	1,441,105	1,351,265	1,427,518	2,517,207	1,441,105	52.40	7.20	-0.60	12.90	6.40	2.50	0.90	5.60	115.50	10.70	-0.60	18.60	19	21	20	21	17	18	17	17
1k	1k	4k	2	1,649,213	1,399,176	1,351,290	1,424,298	2,509,735	1,399,176	57.30	7.60	-0.80	15.20	6.30	2.50	0.90	5.40	121.00	11.00	-0.70	17.50	20	16	18	20	17	17	18	19
1k	1k	2k	4	1,624,166	1,306,244	1,351,163	1,413,247	2,854,762	1,306,244	62.80	7.90	-1.10	19.60	6.20	2.50	0.90	4.60	86.20	9.30	0.00	11.60	19	17	17	18	17	18	18	19
1k	1k	4k	4	1,624,166	1,323,401	1,351,163	1,416,690	2,471,721	1,323,401	64.20	8.00	-1.10	18.50	6.40	2.50	0.90	4.80	124.00	11.10	-0.80	14.90	19	17	18	17	17	18	18	19
1k	1k	16k	1	1,654,670	1,833,876	1,351,265	1,431,797	465,345	1,833,876	29.70	5.40	0.10	10.80	6.60	2.60	0.90	6.00	38.20	6.20	-0.40	95.50	19	17	18	16	17	18	17	17
1k	1k	16k	2	1,649,213	1,778,815	1,351,290	1,432,972	466,087	1,778,815	33.90	5.80	0.00	7.90	6.50	2.50	0.90	6.00	37.20	6.10	-0.40	89.20	19	16	17	16	17	17	17	19
1k	1k	16k	4	1,624,166	1,648,160	1,351,163	1,430,321	467,819	1,648,160	44.20	6.70	-0.50	1.50	6.30	2.50	0.90	5.90	35.00	5.90	-0.30	75.20	19	17	17	17	16	16	17	17
1k	2k	2k	1	1,654,670	1,504,889	1,098,937	1,210,159	2,618,505	1,504,889	48.40	7.00	-0.50	9.10	23.40	4.80	0.40	10.10	87.50	9.40	-0.10	4.50	19	14	16	16	16	17	17	17
1k	2k	2k	2	1,649,213	1,477,612	1,099,776	1,218,090	2,613,402	1,477,612	51.60	7.20	-0.60	10.40	22.60	4.80	0.40	10.80	90.70	9.50	-0.20	5.20	18	15	17	17	17	16	17	18
1k	2k	4k	1	1,654,670	1,524,201	1,098,937	1,254,832	2,356,975	1,524,201	47.80	6.90	-0.50	7.90	21.40	4.60	0.50	14.20	104.20	10.20	-0.60	14.20	19	18	16	16	16	17	18	17
1k	2k	4k	2	1,649,213	1,485,032	1,099,776	1,257,981	2,347,743	1,485,032	52.60	7.30	-0.60	10.00	20.50	4.50	0.50	14.40	110.40	10.50	-0.70	13.00	19	17	16	17	18	17	17	18
1k	2k	16k	1	1,654,670	1,878,589	1,098,937	1,246,866	460,524	1,878,589	28.00	5.30	0.10	13.50	20.40	4.50	0.50	13.50	34.70	5.90	-0.40	84.40	20	14	17	17	18	17	17	22
1k	2k	16k	2	1,649,213	1,843,033	1,099,776	1,278,425	461,020	1,843,033	30.20	5.50	0.10	11.80	18.90	4.40	0.50	16.20	34.30	5.90	-0.30	81.50	19	16	18	17	17	18	18	17
1k	2k	2k	4	1,624,166	1,408,820	1,100,469	1,227,855	2,588,553	1,408,820	57.20	7.60	-0.90	13.30	20.80	4.60	0.50	11.60	95.90	9.80	-0.30	7.10	19	18	17	16	16	17	18	18
1k	2k	4k	4	1,624,166	1,412,142	1,100,469	1,252,833	2,305,122	1,412,142	59.00	7.70	-0.90	13.10	19.20	4.40	0.50	13.80	116.90	10.80	-1.00	10.70	19	15	18	17	16	18	18	18
1k	2k	16k	4	1,624,166	1,738,869	1,100,469	1,324,401	462,690	1,738,869	37.60	6.10	-0.20	7.10	16.00	4.00	0.60	20.30	32.70	5.70	-0.30	71.50	20	14	17	17	19	17	17	19
1k	64k	2k	1	1,654,670	1,794,144	3,199	9,616	1,410,778	1,794,144	31.00	5.60	0.10	8.40	0.10	0.40	0.30	200.60	22.00	4.70	0.10	11.00	19	17	18	17	17	19	17	17
1k	64k	2k	2	1,649,213	1,749,370	3,430	10,987	1,406,335	1,749,370	33.80	5.80	0.00	6.10	0.20	0.40	0.10	220.30	24.50	5.00	0.00	8.90	19	17	18	17	17	17	18	19
1k	64k	4k	1	1,654,670	1,851,791	3,199	10,814	1,012,708	1,851,791	29.60	5.40	0.10	11.90	0.10	0.40	0.30	238.00	40.00	6.30	-0.50	54.90	19	17	18	18	16	17	17	18
1k	64k	4k	2	1,649,213	1,805,836	3,430	12,091	1,006,750	1,805,836	32.30	5.70	0.00	9.50	0.20	0.40	0.20	252.50	40.20	6.30	-0.50	52.90	19	16	17	17	17	17	18	17
1k	64k	16k	1	1,654,670	2,104,130	3,199	8,026	257,728	2,104,130	27.70	5.30	0.20	27.20	0.20	0.40	0.20	150.90	12.60	3.50	0.10	70.30	19	16	18	17	17	17	17	18
1k	64k	16k	2	1,649,213	2,094,499	3,430	8,619	258,188	2,094,499	27.70	5.30	0.10	27.00	0.20	0.40	0.20	151.30	12.70	3.60	0.10	66.30	19	16	18	18	18	17	18	17
1k	64k	2k	4	1,624,166	1,634,794	3,882	14,119	1,385,018	1,634,794	40.60	6.40	-0.30	0.70	0.40	0.60	-0.40	263.70	31.70	5.60	-0.30	3.90	19	16	18	17	17	16	17	17
1k	64k	4k	4	1,624,166	1,693,946	3,882	14,937	969,195	1,693,946	38.50	6.20	-0.30	4.30	0.30	0.60	-0.30	284.80	41.60	6.40	-0.60	50.60	19	16	18	17	16	17	18	18
1k	64k	16k	4	1,624,166	2,073,697	3,882	10,320	262,969	2,073,697	27.90	5.30	0.10	27.70	0.30	0.60	-0.10	165.80	12.70	3.60	0.10	55.70	20	17	17	17	17	18	17	17
2k	1k	2k	1	1,322,828	1,139,818	1,351,265	1,421,099	2,661,216	1,139,818	51.50	7.20	-1.30	13.80	6.30	2.50	0.90	5.20	63.40	8.00	0.20	10.50	18	16	19	17	16	16	17	17
2k	1k	2k	2	1,310,469	1,124,203	1,351,290	1,420,521	2,647,139	1,124,203	54.20	7.40	-1.40	14.20	6.20	2.50	0.90	5.10	66.20	8.10	0.10	10.60	18	16	17	18	16	16	17	17
2k	1k	4k	1	1,322,828	1,151,576	1,351,265	1,428,865	2,414,616	1,151,576	51.50	7.20	-1.30	12.90	6.40	2.50	0.90	5.70	132.60	11.50	-0.90	25.30	18	16	18	17	16	16	16	17
2k	1k	4k	2	1,310,469	1,123,641	1,351,290	1,426,319	2,388,914	1,123,641	54.60	7.40	-1.40	14.30	6.30	2.50	0.90	5.60	131.90	11.50	-0.90	23.60	18	16	17	17	17	16	18	18
2k	1k	16k	1	1,322,828	1,570,214	1,351,265	1,434,600	468,409	1,570,214	46.10	6.80	-1.00	18.70	6.60	2.60	0.90	6.20	35.30	5.90	-0.30	78.50	20	16	18	17	17	17	18	17
2k	1k	16k	2	1,310,469	1,504,578	1,351,290	1,434,870	467,197	1,504,578	49.70	7.10	-1.20	14.80	6.50	2.50	0.90	6.20	34.70	5.90	-0.30	73.50	18	15	17	17	17	17	17	17
2k	2k	2k	1	1,322,828	1,251,750	1,098,937	1,202,260	2,416,573	1,251,750	51.20	7.20	-1.30	5.40	23.90	4.90	0.40	9.40	88.60	9.40	-0.40	7.20	18	16	17	19	18	18	18	19
2k	2k	2k	2	1,310,469	1,231,764	1,099,776	1,216,327	2,403,720	1,231,764	53.50	7.30	-1.40	6.00	22.70	4.80	0.40	10.60	88.40	9.40	-0.40	7.10	18	17	16	16	19	17	19	18
2k	2k	4k	1	1,322,828	1,251,153	1,098,937	1,241,884	2,260,930	1,251,153	51.00	7.10	-1.30	5.40	22.00	4.70	0.40	13.00	120.80	11.00	-1.00	20.70	18	15	17	16	16	17	18	17
2k	2k	4k	2	1,310,469	1,225,761	1,099,776	1,249,885	2,243,335	1,225,761	53.60	7.30	-1.40	6.50	20.90	4.60	0.50	13.60	123.30	11.10	-1.00	19.10	19	14	16	16	16	17	16	18
2k	2k	16k	1	1,322,828	1,664,623	1,098,937	1,248,036	463,809	1,664,623	45.30	6.70	-1.00	25.80	20.30	4.50	0.50	13.60	33.80	5.80	-0.30	75.90	19	15	17	17	16	16	17	18
2k	2k	16k	2	1,310,469	1,616,386	1,099,776	1,280,080	462,495	1,616,386	47.90	6.90	-1.10	23.30	18.70	4.30	0.50	16.40	33.50	5.80	-0.30	73.30	18	16	18	16	16	16	17	17
2k	64k	2k	1	1,322,828	1,577,630	3,199	10,052	1,244,121	1,577,630	41.30	6.40	-0.80	19.30	0.10	0.40	0.30	214.20	29.60	5.40	-0.20	11.70	19	17	17	16	16	16	17	17
2k	64k	2k	2	1,310,469	1,516,702	3,430	11,155	1,225,755	1,516,702	43.90	6.60	-0.90	15.70	0.20	0.40	0.10	225.20	32.30	5.70	-0.30	9.30	19	14	17	17	16	17	17	17
2k	64k	4k	1	1,322,828	1,642,835	3,199	11,108	988,309	1,642,835	42.00	6.50	-0.90	24.20	0.10	0.40	0.30	247.20	34.00	5.80	-0.30	41.80	19	16	18	18	18	16	17	18
2k	64k	4k	2	1,310,469	1,582,519	3,430	12,142	976,708	1,582,519	43.90	6.60	-0.90	20.80	0.20	0.40	0.10	254.00	34.30	5.90	-0.30	39.00	19	16	17	16	16	17	17	17
2k	64k	16k	1	1,322,828	2,046,465	3,199	8,084	256,443	2,046,465	52.50	7.20	-1.30	54.70	0.20	0.40	0.30	152.70	12.60	3.50	0.10	67.90	19	16	16	17	17	17	18	19
2k	64k	16k	2	1,310,469	2,033,553	3,430	8,691	256,855	2,033,553	53.00	7.30	-1.30	55.20	0.20	0.40	0.20	153.40	12.70	3.60	0.10	63.60	18	17	17	16	17	17	18	16
2k	1k	2k	4	1,248,824	1,093,122	1,351,163	1,416,780	2,581,011	1,093,122	54.90	7.40	-1.40	12.50	6.20	2.50	0.90	4.90	68.00	8.20	0.00	9.90	19	16	18	16	16	17	18	17
2k	1k	4k	4	1,248,824	1,080,042	1,351,163	1,419,892	2,274,658	1,080,042	54.70	7.40	-1.40	13.50	6.40	2.50	0.90	5.10	113.50	10.70	-0.80	17.40	18	15	18	17	17	17	17	19
2k	1k	16k	4	1,248,824	1,373,943	1,351,163	1,431,353	459,020	1,373,943	54.60	7.40	-1.40	10.00	6.30	2.50	0.90	5.90	33.00	5.70	-0.20	65.40	18	16	16	18	16	18	17	17
2k	2k	2k	4	1,248,824	1,180,458	1,100,469	1,233,963	2,341,336	1,180,458	54.60	7.40	-1.40	5.50	20.50	4.50	0.50	12.10	86.00	9.30	-0.40	6.80	19	17	18	17	18	19	18	19
2k	2k	4k	4	1,248,824	1,174,920	1,100,469	1,252,067	2,156,179	1,174,920	54.10	7.40	-1.30	5.90	19.20	4.40	0.50	13.80	119.10	10.90	-1.00	15.00	18	16	15	16	16	16	17	17
2k	2k	16k	4	1,248,824	1,505,025	1,100,469	1,327,444	454,454	1,505,025	51.10	7.10	-1.20	20.50	15.60	3.90	0.60	20.60	32.10	5.70	-0.20	67.90	18	15	15	16	16	17	17	17
2k	64k	2k	4	1,248,824	1,383,731	3,882	13,689	1,136,180	1,383,731	46.90	6.80	-1.00	10.80	0.40	0.60	-0.40	252.60	35.30	5.90	-0.30	7.90	19	17	19	16	16	18	17	18
2k	64k	4k	4	1,248,824	1,442,470	3,882	14,420	923,174	1,442,470	46.30	6.80	-1.00	15.50	0.40	0.60	-0.30	271.50	35.20	5.90	-0.20	35.10	18	15	17	17	17	17	18	17
2k	64k	16k	4	1,248,824	1,997,835	3,882	10,376	263,977	1,997,835	56.80	7.50	-1.50	60.00	0.30	0.50	-0.10	167.30	12.70	3.60	0.10	51.70	18	16	17	17	17	18	16	17
64k	1k	2k	1	86,625	85,525	1,351,265	1,468,141	1,219,563	85,525	1.10	1.10	0.40	1.30	8.20	2.90	0.80	8.60	9.00	3.00	0.80	9.30	20	16	19	17	16	17	18	17
64k	1k	2k	2	86,691	85,354	1,351,290	1,462,752	1,218,428	85,354	1.10	1.10	0.40	1.50	8.00	2.80	0.80	8.20	9.00	3.00	0.80	9.10	19	16	17	17	17	17	17	17
64k	1k	4k	1	86,625	86,377	1,351,265	1,467,031	580,371	86,377	1.10	1.00	0.40	0.30	8.20	2.90	0.80	8.60	49.90	7.10	-2.40	118.70	20	17	18	17	16	17	17	18
64k	1k	4k	2	86,691	85,961	1,351,290	1,461,447	579,852	85,961	1.10	1.10	0.40	0.80	8.00	2.80	0.80	8.20	49.60	7.00	-2.40	118.40	20	17	18	18	18	18	17	18
64k	1k	16k	1	86,625	83,948	1,351,265	1,466,795	103,209	83,948	1.10	1.00	0.40	3.10	8.10	2.80	0.80	8.50	2.90	1.70	0.10	20.90	19	16	19	18	17	17	18	18
64k	1k	16k	2	86,691	83,774	1,351,290	1,461,825	103,091	83,774	1.10	1.00	0.40	3.40	7.90	2.80	0.90	8.20	2.80	1.70	0.10	20.60	19	16	19	18	16	17	17	17
64k	2k	2k	1	86,625	89,585	1,098,937	1,435,525	1,125,835	89,585	1.20	1.10	0.30	3.40	16.10	4.00	0.60	30.60	11.80	3.40	0.70	16.10	19	16	18	18	16	17	18	19
64k	2k	2k	2	86,691	89,100	1,099,776	1,430,127	1,124,576	89,100	1.20	1.10	0.30	2.80	15.80	4.00	0.60	30.00	11.70	3.40	0.70	15.80	18	16	19	17	18	18	18	18
64k	2k	4k	1	86,625	90,148	1,098,937	1,434,461	505,939	90,148	1.10	1.10	0.40	4.10	16.10	4.00	0.60	30.50	58.60	7.70	-3.90	144.80	19	17	18	18	17	17	17	18
64k	2k	4k	2	86,691	89,457	1,099,776	1,428,972	503,894	89,457	1.20	1.10	0.40	3.20	15.80	4.00	0.60	29.90	58.50	7.70	-3.90	145.10	18	16	18	17	17	18	18	17
64k	2k	16k	1	86,625	86,168	1,098,937	1,435,211	103,367	86,168	1.10	1.00	0.40	0.50	16.30	4.00	0.60	30.60	2.70	1.60	0.20	18.30	20	16	18	18	17	17	17	18
64k	2k	16k	2	86,691	85,955	1,099,776	1,430,509	103,241	85,955	1.10	1.10	0.40	0.80	15.90	4.00	0.60	30.10	2.60	1.60	0.20	18.10	19	17	19	17	16	18	18	17
64k	64k	2k	1	86,625	78,465	3,199	2,406	88,587	78,465	1.20	1.10	0.30	9.40	0.20	0.40	0.10	24.80	1.50	1.20	0.30	7.40	20	16	18	19	17	17	18	18
64k	64k	2k	2	86,691	76,578	3,430	2,550	88,750	76,578	1.20	1.10	0.30	11.70	0.20	0.40	0.10	25.70	1.50	1.20	0.30	8.80	19	18	17	18	17	17	16	18
64k	64k	4k	1	86,625	77,396	3,199	2,387	85,404	77,396	1.20	1.10	0.30	10.70	0.20	0.40	0.10	25.40	1.50	1.20	0.30	6.50	20	16	18	18	17	18	20	19
64k	64k	4k	2	86,691	75,393	3,430	2,530	85,536	75,393	1.20	1.10	0.30	13.00	0.20	0.40	0.10	26.30	1.50	1.20	0.30	8.10	20	17	19	18	17	17	17	19
64k	64k	16k	1	86,625	70,652	3,199	2,299	80,072	70,652	1.10	1.10	0.40	18.40	0.20	0.40	0.10	28.10	1.40	1.20	0.30	15.00	20	17	18	18	17	18	18	18
64k	64k	16k	2	86,691	68,295	3,430	2,431	80,123	68,295	1.10	1.10	0.40	21.20	0.20	0.40	0.10	29.10	1.50	1.20	0.30	17.10	20	17	19	18	17	17	17	19
64k	1k	2k	4	86,868	84,828	1,351,163	1,454,100	1,217,782	84,828	1.10	1.10	0.40	2.30	7.80	2.80	0.90	7.60	9.00	3.00	0.80	8.70	19	16	18	18	18	17	18	18
64k	1k	4k	4	86,868	85,143	1,351,163	1,452,716	579,493	85,143	1.10	1.10	0.40	2.00	7.70	2.80	0.90	7.50	49.00	7.00	-2.30	117.20	19	17	20	18	17	17	17	19
64k	1k	16k	4	86,868	83,124	1,351,163	1,454,987	103,032	83,124	1.10	1.00	0.40	4.30	7.60	2.80	0.90	7.70	2.80	1.70	0.20	20.10	20	16	19	17	17	17	18	18
64k	2k	2k	4	86,868	87,781	1,100,469	1,421,823	1,123,772	87,781	1.20	1.10	0.30	1.10	15.40	3.90	0.60	29.20	11.40	3.40	0.70	15.20	19	17	18	18	17	17	18	18
64k	2k	4k	4	86,868	87,890	1,100,469	1,420,883	502,203	87,890	1.20	1.10	0.30	1.20	15.30	3.90	0.60	29.10	57.80	7.60	-3.80	144.10	19	16	19	17	17	17	17	18
64k	2k	16k	4	86,868	85,190	1,100,469	1,423,897	103,176	85,190	1.10	1.10	0.40	1.90	15.50	3.90	0.60	29.40	2.60	1.60	0.20	17.80	19	16	18	18	17	18	17	17
64k	64k	2k	4	86,868	73,015	3,882	2,942	89,194	73,015	1.20	1.10	0.30	15.90	0.20	0.50	0.20	24.20	1.60	1.30	0.30	12.10	18	16	18	18	17	17	18	19
64k	64k	4k	4	86,868	71,639	3,882	2,904	85,846	71,639	1.20	1.10	0.30	17.50	0.20	0.50	0.10	25.20	1.60	1.30	0.20	11.50	20	17	20	17	17	17	18	19
64k	64k	16k	4	86,868	64,205	3,882	2,780	80,214	64,205	1.20	1.10	0.30	26.10	0.20	0.50	0.10	28.40	1.50	1.20	0.30	21.00	20	17	18	19	17	17	17	18

Table 4: Table of results for Sysbench Memory Benchmark Model executions

				L1D		L1I		LL		L1D				L1I				LL				Execution time per block							
L1D Size (bytes)	L1I Size (bytes)	LL Size (bytes)	Cores	Total Actual	Total Predicted	Total Actual	Total Predicted	Total Actual	Total Predicted	MSE	RMSE	R2	Abs. Err. Percent	MSE	RMSE	R2	Abs. Err. Percent	MSE	RMSE	R2	Abs. Err. Percent	t_1 s	t_2 s	t_3 s	t_4 s	t_5 s	t_6 s	t_7 s	t_8 s
1k	1k	2k	1	1,375,523	1,157,767	726,633	872,551	1,879,632	1,157,767	31.9	5.6	0.5	15.8	7.9	2.8	0.8	20.1	42.9	6.5	0.6	0.3	17	14	14	16	17	16	19	18
1k	1k	2k	2	1,375,355	1,145,833	726,948	872,259	1,879,087	1,145,833	34.2	5.8	0.4	16.7	7.7	2.8	0.8	20.0	45.6	6.8	0.6	0.9	17	14	14	15	16	17	17	17
1k	1k	4k	1	1,375,523	1,186,618	726,633	882,332	1,628,029	1,186,618	29.3	5.4	0.5	13.7	8.0	2.8	0.8	21.4	53.6	7.3	0.4	5.7	17	14	14	16	15	18	17	18
1k	1k	4k	2	1,375,355	1,178,483	726,948	881,037	1,627,343	1,178,483	31.0	5.6	0.5	14.3	7.9	2.8	0.8	21.2	54.5	7.4	0.4	3.6	17	14	15	15	16	16	16	17
1k	1k	2k	4	1,374,963	1,130,585	727,131	868,038	1,878,337	1,130,585	37.8	6.1	0.4	17.8	7.5	2.7	0.8	19.4	50.9	7.1	0.6	2.0	17	14	15	16	16	16	17	17
1k	1k	4k	4	1,374,963	1,160,957	727,131	874,718	1,626,862	1,160,957	34.7	5.9	0.4	15.6	7.7	2.8	0.8	20.3	57.5	7.6	0.4	0.2	17	14	14	16	16	16	17	17
1k	1k	16k	1	1,375,523	1,420,896	726,633	872,110	393,838	1,420,896	16.5	4.1	0.7	3.3	8.4	2.9	0.8	20.0	31.9	5.6	-0.1	86.2	18	14	14	16	16	16	17	17
1k	1k	16k	2	1,375,355	1,406,617	726,948	875,580	393,871	1,406,617	18.4	4.3	0.7	2.3	8.3	2.9	0.8	20.4	31.8	5.6	-0.1	84.9	18	14	14	16	16	17	17	17
1k	1k	16k	4	1,374,963	1,373,452	727,131	875,837	393,932	1,373,452	22.3	4.7	0.6	0.1	8.2	2.9	0.8	20.5	31.4	5.6	-0.1	80.6	18	14	14	16	15	17	17	17
1k	2k	2k	1	1,375,523	1,228,521	576,357	700,589	1,702,896	1,228,521	28.2	5.3	0.5	10.7	16.7	4.1	0.4	21.6	55.8	7.5	0.4	4.6	17	14	15	16	15	16	16	17
1k	2k	2k	2	1,375,355	1,220,455	577,390	698,105	1,703,300	1,220,455	29.8	5.5	0.5	11.3	16.5	4.1	0.4	20.9	57.4	7.6	0.4	4.0	17	14	14	16	15	16	18	17
1k	2k	4k	1	1,375,523	1,229,360	576,357	717,414	1,508,862	1,229,360	27.8	5.3	0.5	10.6	17.1	4.1	0.4	24.5	54.7	7.4	0.3	1.5	17	14	14	16	16	16	16	16
1k	2k	4k	2	1,375,355	1,229,114	577,390	717,382	1,508,841	1,229,114	28.8	5.4	0.5	10.6	16.8	4.1	0.4	24.2	56.7	7.5	0.3	0.6	17	14	14	17	15	16	16	17
1k	2k	16k	1	1,375,523	1,418,073	576,357	671,881	389,519	1,418,073	17.7	4.2	0.7	3.1	16.4	4.1	0.4	16.6	28.3	5.3	0.0	74.6	17	14	14	15	17	16	16	17
1k	2k	16k	2	1,375,355	1,413,127	577,390	693,758	389,549	1,413,127	18.6	4.3	0.7	2.7	16.1	4.0	0.4	20.2	28.3	5.3	0.0	74.5	18	14	14	16	16	16	17	17
1k	2k	2k	4	1,374,963	1,201,551	580,038	685,609	1,705,315	1,201,551	33.3	5.8	0.5	12.6	16.1	4.0	0.4	18.2	60.8	7.8	0.3	1.9	17	13	14	16	15	16	17	17
1k	2k	4k	4	1,374,963	1,220,017	580,038	703,048	1,510,533	1,220,017	31.6	5.6	0.5	11.3	16.1	4.0	0.4	21.2	62.0	7.9	0.2	4.1	17	14	14	16	15	16	16	16
1k	2k	16k	4	1,374,963	1,394,899	580,038	724,734	389,556	1,394,899	21.1	4.6	0.7	1.4	15.5	3.9	0.5	24.9	28.4	5.3	0.0	73.0	17	14	16	16	16	16	16	17
1k	64k	2k	1	1,375,523	1,450,931	2,998	5,307	1,066,028	1,450,931	14.6	3.8	0.8	5.5	0.1	0.3	0.4	77.0	20.9	4.6	0.4	18.0	18	14	15	16	16	16	17	17
1k	64k	2k	2	1,375,355	1,440,978	3,204	5,499	1,065,844	1,440,978	15.7	4.0	0.7	4.8	0.1	0.4	0.4	71.6	21.9	4.7	0.4	17.4	18	14	14	16	15	16	16	17
1k	64k	4k	1	1,375,523	1,478,505	2,998	5,826	821,613	1,478,505	14.0	3.7	0.8	7.5	0.1	0.3	0.4	94.3	30.0	5.5	0.0	51.0	17	14	14	16	16	19	17	17
1k	64k	4k	2	1,375,355	1,466,891	3,204	5,971	821,739	1,466,891	15.0	3.9	0.8	6.7	0.1	0.4	0.4	86.4	30.2	5.5	0.0	50.2	18	14	15	16	17	16	17	17
1k	64k	16k	1	1,375,523	1,614,702	2,998	5,635	238,609	1,614,702	13.0	3.6	0.8	17.4	0.1	0.3	0.4	88.0	12.0	3.5	0.1	67.1	18	14	17	16	16	16	17	18
1k	64k	16k	2	1,375,355	1,614,995	3,204	5,795	238,806	1,614,995	13.1	3.6	0.8	17.4	0.1	0.4	0.4	80.9	12.1	3.5	0.1	64.2	18	15	15	17	16	18	17	17
1k	64k	2k	4	1,374,963	1,413,231	3,611	5,954	1,065,797	1,413,231	18.8	4.3	0.7	2.8	0.2	0.4	0.4	64.9	24.6	5.0	0.3	15.2	17	14	14	16	16	16	19	17
1k	64k	4k	4	1,374,963	1,435,853	3,611	6,470	822,149	1,435,853	17.9	4.2	0.7	4.4	0.2	0.4	0.4	79.2	30.7	5.5	0.0	47.5	18	14	14	16	16	16	17	18
1k	64k	16k	4	1,374,963	1,613,818	3,611	6,259	239,128	1,613,818	13.3	3.6	0.8	17.4	0.2	0.4	0.4	73.3	12.0	3.5	0.1	57.9	18	14	14	16	16	16	17	17
2k	1k	2k	1	1,012,999	985,394	726,633	871,607	1,717,481	985,394	44.2	6.6	-0.2	2.7	7.8	2.8	0.8	20.0	40.3	6.3	0.6	0.6	17	14	14	17	15	17	16	20
2k	1k	2k	2	1,012,850	988,731	726,948	872,160	1,716,601	988,731	46.0	6.8	-0.3	2.4	7.6	2.8	0.8	20.0	41.9	6.5	0.6	0.5	17	14	14	16	14	16	17	16
2k	1k	4k	1	1,012,999	1,006,178	726,633	880,628	1,611,174	1,006,178	41.6	6.4	-0.2	0.7	8.0	2.8	0.8	21.2	63.4	8.0	0.3	13.2	17	14	14	15	18	16	19	17
2k	1k	4k	2	1,012,850	1,000,976	726,948	879,346	1,609,861	1,000,976	43.4	6.6	-0.2	1.2	7.8	2.8	0.8	21.0	62.4	7.9	0.3	11.4	19	14	14	16	15	15	17	17
2k	1k	16k	1	1,012,999	1,258,112	726,633	870,533	401,147	1,258,112	40.4	6.4	-0.1	24.2	8.4	2.9	0.8	19.8	30.7	5.5	-0.1	76.5	17	14	15	16	15	17	16	17
2k	1k	16k	2	1,012,850	1,245,659	726,948	874,618	401,192	1,245,659	41.7	6.5	-0.2	23.0	8.2	2.9	0.8	20.3	30.8	5.5	-0.1	75.4	18	15	14	16	18	15	17	20
2k	2k	2k	1	1,012,999	1,059,390	576,357	685,884	1,570,333	1,059,390	44.5	6.7	-0.3	4.6	16.7	4.1	0.4	19.0	56.5	7.5	0.3	2.2	19	13	14	15	16	17	19	16
2k	2k	2k	2	1,012,850	1,060,374	577,390	687,318	1,570,379	1,060,374	46.0	6.8	-0.3	4.7	16.5	4.1	0.4	19.0	57.1	7.6	0.3	2.2	19	14	16	16	15	17	19	19
2k	2k	4k	1	1,012,999	1,071,401	576,357	701,928	1,514,993	1,071,401	42.4	6.5	-0.2	5.8	17.4	4.2	0.4	21.8	63.4	8.0	0.2	9.8	16	14	16	17	18	16	19	19
2k	2k	4k	2	1,012,850	1,073,501	577,390	702,792	1,513,926	1,073,501	43.6	6.6	-0.2	6.0	17.0	4.1	0.4	21.7	63.6	8.0	0.2	7.7	17	14	17	17	15	18	16	18
2k	2k	16k	1	1,012,999	1,285,568	576,357	676,109	396,893	1,285,568	41.6	6.5	-0.2	26.9	16.1	4.0	0.4	17.3	28.5	5.3	0.0	70.8	18	14	14	18	15	17	16	19
2k	2k	16k	2	1,012,850	1,281,327	577,390	695,793	396,939	1,281,327	42.6	6.5	-0.2	26.5	15.8	4.0	0.4	20.5	28.6	5.3	0.0	70.6	17	14	15	15	15	15	16	16
2k	64k	2k	1	1,012,999	1,314,829	2,998	5,163	982,521	1,314,829	38.8	6.2	-0.1	29.8	0.1	0.3	0.4	72.2	23.3	4.8	0.4	16.5	17	14	14	15	16	15	16	20
2k	64k	2k	2	1,012,850	1,294,089	3,204	5,304	981,914	1,294,089	39.6	6.3	-0.1	27.8	0.1	0.4	0.4	65.5	24.4	4.9	0.3	14.9	17	13	16	16	20	16	18	17
2k	64k	4k	1	1,012,999	1,357,122	2,998	5,690	887,243	1,357,122	40.0	6.3	-0.1	34.0	0.1	0.3	0.4	89.8	23.6	4.9	0.3	28.9	17	14	14	16	15	19	16	18
2k	64k	4k	2	1,012,850	1,334,332	3,204	5,844	886,744	1,334,332	40.8	6.4	-0.2	31.7	0.1	0.4	0.4	82.4	24.3	4.9	0.2	27.2	17	14	14	16	15	17	17	19
2k	64k	16k	1	1,012,999	1,563,732	2,998	5,630	242,397	1,563,732	45.6	6.8	-0.3	54.4	0.1	0.3	0.4	87.8	12.0	3.5	0.2	62.7	17	14	15	16	15	16	16	16
2k	64k	16k	2	1,012,850	1,562,214	3,204	5,802	242,591	1,562,214	45.6	6.8	-0.3	54.2	0.1	0.4	0.4	81.1	12.0	3.5	0.2	59.8	17	14	14	16	16	16	18	18
2k	1k	2k	4	1,012,240	998,042	727,131	869,211	1,715,345	998,042	48.6	7.0	-0.4	1.4	7.4	2.7	0.8	19.5	45.4	6.7	0.6	0.2	16	14	14	16	16	16	17	17
2k	1k	4k	4	1,012,240	997,827	727,131	873,941	1,608,532	997,827	46.1	6.8	-0.3	1.4	7.6	2.8	0.8	20.2	60.4	7.8	0.3	7.2	17	14	14	17	15	16	16	17
2k	1k	16k	4	1,012,240	1,216,142	727,131	875,802	401,277	1,216,142	44.7	6.7	-0.3	20.1	8.1	2.9	0.8	20.4	30.5	5.5	-0.1	72.2	17	14	14	16	15	17	16	20
2k	2k	2k	4	1,012,240	1,058,619	580,038	681,070	1,571,729	1,058,619	48.3	7.0	-0.4	4.6	16.0	4.0	0.4	17.4	58.9	7.7	0.3	1.2	17	14	15	17	14	17	18	21
2k	2k	4k	4	1,012,240	1,066,920	580,038	691,316	1,514,548	1,066,920	46.1	6.8	-0.3	5.4	16.1	4.0	0.4	19.2	65.0	8.1	0.2	4.6	23	14	16	18	15	18	18	18
2k	2k	16k	4	1,012,240	1,265,878	580,038	725,787	397,046	1,265,878	44.5	6.7	-0.3	25.1	15.2	3.9	0.5	25.1	28.7	5.4	0.0	69.2	17	14	14	15	16	16	16	17
2k	64k	2k	4	1,012,240	1,244,733	3,611	5,782	981,018	1,244,733	41.3	6.4	-0.2	23.0	0.2	0.4	0.4	60.1	26.6	5.2	0.3	10.7	17	14	14	15	16	16	18	17
2k	64k	4k	4	1,012,240	1,281,504	3,611	6,274	886,714	1,281,504	42.4	6.5	-0.2	26.6	0.2	0.4	0.4	73.7	25.5	5.1	0.2	22.8	17	14	14	16	17	17	17	16
2k	64k	16k	4	1,012,240	1,555,893	3,611	6,248	242,938	1,555,893	45.7	6.8	-0.3	53.7	0.2	0.4	0.4	73.0	12.0	3.5	0.2	53.1	17	14	14	16	16	18	17	16
64k	1k	2k	1	83,523	83,118	726,633	956,454	728,023	83,118	1.1	1.0	0.4	0.5	10.1	3.2	0.8	31.6	6.6	2.6	0.8	19.0	18	15	16	18	17	17	17	18
64k	1k	2k	2	83,631	82,971	726,948	952,510	727,479	82,971	1.1	1.0	0.4	0.8	9.9	3.2	0.8	31.0	6.6	2.6	0.8	18.7	18	14	16	16	16	17	17	17
64k	1k	4k	1	83,523	83,626	726,633	954,093	200,532	83,626	1.0	1.0	0.4	0.1	10.0	3.2	0.8	31.3	55.7	7.5	-4.9	314.8	19	14	14	16	15	16	16	17
64k	1k	4k	2	83,631	83,448	726,948	950,014	200,644	83,448	1.1	1.0	0.4	0.2	9.9	3.1	0.8	30.7	55.5	7.4	-4.8	313.9	19	16	15	17	16	17	17	17
64k	1k	16k	1	83,523	82,514	726,633	958,158	98,516	82,514	1.0	1.0	0.4	1.2	10.3	3.2	0.8	31.9	2.6	1.6	0.2	26.0	21	14	17	17	17	17	18	17
64k	1k	16k	2	83,631	82,198	726,948	954,959	98,509	82,198	1.0	1.0	0.4	1.7	10.1	3.2	0.8	31.4	2.6	1.6	0.2	25.4	18	15	17	18	18	17	18	18
64k	2k	2k	1	83,523	86,365	576,357	921,405	523,851	86,365	1.1	1.1	0.4	3.4	18.5	4.3	0.4	59.9	21.2	4.6	0.2	61.0	18	14	16	17	16	17	17	17
64k	2k	2k	2	83,631	86,050	577,390	917,704	522,607	86,050	1.1	1.1	0.4	2.9	18.2	4.3	0.4	58.9	21.3	4.6	0.2	60.9	18	15	14	16	16	17	18	18
64k	2k	4k	1	83,523	86,735	576,357	918,659	199,589	86,735	1.1	1.0	0.4	3.8	18.4	4.3	0.4	59.4	53.9	7.3	-4.7	305.0	18	14	15	16	15	16	16	17
64k	2k	4k	2	83,631	86,402	577,390	914,935	199,674	86,402	1.1	1.1	0.4	3.3	18.1	4.3	0.4	58.5	53.7	7.3	-4.7	304.2	18	14	15	17	16	17	17	18
64k	2k	16k	1	83,523	84,192	576,357	923,641	98,603	84,192	1.1	1.0	0.4	0.8	18.8	4.3	0.3	60.3	2.5	1.6	0.2	23.7	18	16	15	17	16	17	17	18
64k	2k	16k	2	83,631	83,888	577,390	920,215	98,597	83,888	1.1	1.0	0.4	0.3	18.4	4.3	0.4	59.4	2.5	1.6	0.2	23.2	18	14	16	18	17	17	18	17
64k	64k	2k	1	83,523	72,831	2,998	2,231	84,904	72,831	1.1	1.1	0.4	12.8	0.2	0.4	0.2	25.6	1.4	1.2	0.3	9.5	18	16	16	19	17	19	17	19
64k	64k	2k	2	83,631	72,139	3,204	2,384	85,122	72,139	1.2	1.1	0.3	13.7	0.2	0.4	0.2	25.6	1.4	1.2	0.3	10.2	20	15	17	18	18	18	18	18
64k	64k	4k	1	83,523	71,724	2,998	2,207	81,822	71,724	1.1	1.1	0.4	14.1	0.2	0.4	0.2	26.4	1.4	1.2	0.3	8.7	19	16	15	18	18	18	19	18
64k	64k	4k	2	83,631	71,011	3,204	2,357	82,032	71,011	1.2	1.1	0.3	15.1	0.2	0.4	0.2	26.4	1.5	1.2	0.3	9.4	18	14	15	16	16	17	17	17
64k	64k	16k	1	83,523	65,442	2,998	2,103	76,695	65,442	1.1	1.0	0.4	21.6	0.2	0.4	0.2	29.8	1.3	1.2	0.3	17.0	18	15	17	17	18	16	20	19
64k	64k	16k	2	83,631	64,557	3,204	2,226	76,811	64,557	1.1	1.1	0.4	22.8	0.2	0.4	0.2	30.5	1.4	1.2	0.3	17.9	19	15	15	16	16	18	17	18
64k	1k	2k	4	83,824	82,454	727,131	945,433	727,585	82,454	1.1	1.0	0.4	1.6	9.7	3.1	0.8	30.0	6.6	2.6	0.8	17.9	18	14	16	16	16	17	17	17
64k	1k	4k	4	83,824	82,806	727,131	942,816	200,943	82,806	1.1	1.0	0.4	1.2	9.6	3.1	0.8	29.7	54.8	7.4	-4.7	310.2	18	15	15	18	16	19	18	17
64k	1k	16k	4	83,824	81,633	727,131	950,155	98,644	81,633	1.1	1.0	0.4	2.6	9.8	3.1	0.8	30.7	2.6	1.6	0.2	24.2	19	14	16	16	17	18	17	19
64k	2k	2k	4	83,824	85,023	580,038	911,297	523,691	85,023	1.1	1.1	0.4	1.4	17.6	4.2	0.4	57.1	20.9	4.6	0.2	59.2	18	14	14	16	18	16	17	17
64k	2k	4k	4	83,824	85,251	580,038	908,672	199,913	85,251	1.1	1.1	0.4	1.7	17.5	4.2	0.4	56.7	53.0	7.3	-4.6	300.1	18	14	15	16	16	16	16	17
64k	2k	16k	4	83,824	83,173	580,038	914,607	98,723	83,173	1.1	1.0	0.4	0.8	17.8	4.2	0.4	57.7	2.4	1.6	0.2	22.2	18	14	14	17	16	16	17	17
64k	64k	2k	4	83,824	70,903	3,611	2,772	85,640	70,903	1.2	1.1	0.3	15.4	0.2	0.5	0.2	23.2	1.5	1.2	0.3	11.6	20	15	16	17	17	18	17	19
64k	64k	4k	4	83,824	69,693	3,611	2,733	82,536	69,693	1.2	1.1	0.3	16.9	0.2	0.5	0.2	24.3	1.6	1.3	0.3	10.8	19	15	14	16	16	17	17	17
64k	64k	16k	4	83,824	62,909	3,611	2,536	77,073	62,909	1.2	1.1	0.3	25.0	0.2	0.5	0.2	29.8	1.4	1.2	0.3	19.7	18	16	14	16	17	18	16	17

Table 5: Table of results for the GFTT Model executions

				L1D		L1I		LL		L1D				L1I				LL				Execution time per block							
L1D Size (bytes)	L1I Size (bytes)	LL Size (bytes)	Cores	Total Actual	Total Predicted	Total Actual	Total Predicted	Total Actual	Total Predicted	MSE	RMSE	R2	Abs. Err. Percent	MSE	RMSE	R2	Abs. Err. Percent	MSE	RMSE	R2	Abs. Err. Percent	t_1 s	t_2 s	t_3 s	t_4 s	t_5 s	t_6 s	t_7 s	t_8 s
1k	1k	2k	1	2,023,632	1,606,586	314,146	534,562	1,819,176	1,606,586	38.0	6.2	0.5	20.6	17.8	4.2	0.1	70.2	27.9	5.3	0.5	3.4	15	14	14	15	15	20	20	20
1k	1k	2k	2	2,023,632	1,634,488	314,146	531,671	1,819,176	1,634,488	36.8	6.1	0.5	19.2	16.8	4.1	0.2	69.2	28.3	5.3	0.5	5.3	16	14	14	14	14	15	15	15
1k	1k	4k	1	2,023,632	1,629,368	314,146	547,014	1,691,447	1,629,368	36.8	6.1	0.5	19.5	18.7	4.3	0.1	74.1	21.9	4.7	0.5	0.6	21	21	20	21	18	19	21	21
1k	1k	4k	2	2,023,632	1,664,880	314,146	544,691	1,691,447	1,664,880	35.2	5.9	0.5	17.7	17.7	4.2	0.1	73.4	22.4	4.7	0.5	1.4	21	23	18	19	17	20	20	20
1k	1k	2k	4	2,023,632	1,683,170	314,146	518,085	1,819,176	1,683,170	35.0	5.9	0.5	16.8	14.7	3.8	0.3	64.9	31.5	5.6	0.4	9.4	11	13	13	13	13	14	15	15
1k	1k	4k	4	2,023,632	1,706,171	314,146	531,413	1,691,447	1,706,171	33.7	5.8	0.6	15.7	15.5	3.9	0.2	69.2	24.9	5.0	0.5	5.6	20	21	18	19	19	19	20	19
1k	1k	16k	1	2,023,632	1,748,195	314,146	580,855	1,317,132	1,748,195	30.5	5.5	0.6	13.6	20.9	4.6	0.0	84.9	18.6	4.3	0.6	4.4	25	25	18	19	17	20	19	21
1k	1k	16k	2	2,023,632	1,756,234	314,146	578,944	1,317,132	1,756,234	30.5	5.5	0.6	13.2	20.0	4.5	0.0	84.3	17.1	4.1	0.7	2.8	32	20	17	19	19	20	20	20
1k	1k	16k	4	2,023,632	1,742,038	314,146	568,377	1,317,132	1,742,038	32.4	5.7	0.6	13.9	17.8	4.2	0.1	80.9	15.2	3.9	0.7	0.3	20	24	19	19	17	19	21	19
1k	2k	2k	1	2,023,632	1,595,868	273,047	425,870	1,779,048	1,595,868	39.0	6.2	0.5	21.1	17.1	4.1	0.0	56.0	30.7	5.5	0.4	5.9	24	15	16	16	16	17	18	18
1k	2k	2k	2	2,023,632	1,621,258	273,047	415,058	1,779,048	1,621,258	37.8	6.2	0.5	19.9	15.5	3.9	0.1	52.0	30.6	5.5	0.4	7.8	21	22	19	19	17	19	18	20
1k	2k	4k	1	2,023,632	1,615,371	273,047	444,989	1,662,461	1,615,371	38.1	6.2	0.5	20.2	18.1	4.3	0.0	63.0	23.6	4.9	0.5	1.9	21	16	14	15	20	18	20	20
1k	2k	4k	2	2,023,632	1,648,167	273,047	433,853	1,662,461	1,648,167	36.6	6.0	0.5	18.6	16.3	4.0	0.1	58.9	23.7	4.9	0.5	3.9	24	18	16	17	17	16	17	16
1k	2k	16k	1	2,023,632	1,745,551	273,047	499,265	1,307,282	1,745,551	31.6	5.6	0.6	13.7	20.5	4.5	-0.1	82.8	16.7	4.1	0.7	3.1	23	18	18	17	19	20	21	21
1k	2k	16k	2	2,023,632	1,757,274	273,047	488,108	1,307,282	1,757,274	31.4	5.6	0.6	13.2	18.8	4.3	0.0	78.8	15.6	3.9	0.7	1.5	20	19	21	19	18	20	21	19
1k	2k	2k	4	2,023,632	1,668,115	273,047	390,707	1,779,048	1,668,115	36.0	6.0	0.5	17.6	12.5	3.5	0.3	43.1	32.0	5.7	0.4	11.5	22	19	18	18	17	20	20	20
1k	2k	4k	4	2,023,632	1,691,980	273,047	407,725	1,662,461	1,691,980	34.7	5.9	0.5	16.4	13.1	3.6	0.3	49.3	24.9	5.0	0.4	7.7	21	19	16	14	18	21	18	20
1k	2k	16k	4	2,023,632	1,749,309	273,047	461,196	1,307,282	1,749,309	32.6	5.7	0.6	13.6	15.1	3.9	0.2	68.9	14.3	3.8	0.7	1.0	19	25	20	21	19	20	18	20
1k	64k	2k	1	2,023,632	1,836,278	1,056	5,546	1,468,709	1,836,278	25.0	5.0	0.7	9.3	0.1	0.3	-0.1	425.2	9.6	3.1	0.7	10.5	20	22	20	22	20	20	17	19
1k	64k	2k	2	2,023,632	1,824,667	1,056	5,771	1,468,709	1,824,667	27.0	5.2	0.6	9.8	0.1	0.3	-0.1	446.4	8.8	3.0	0.7	7.1	20	20	21	19	17	20	24	20
1k	64k	4k	1	2,023,632	1,831,490	1,056	5,212	1,355,125	1,831,490	27.2	5.2	0.6	9.5	0.1	0.3	-0.1	393.6	10.9	3.3	0.7	14.8	20	20	18	19	18	20	21	20
1k	64k	4k	2	2,023,632	1,838,692	1,056	5,343	1,355,125	1,838,692	28.6	5.3	0.6	9.1	0.1	0.3	-0.1	405.9	9.3	3.1	0.7	10.5	20	17	17	19	19	20	21	20
1k	64k	16k	1	2,023,632	2,045,826	1,056	4,832	1,162,057	2,045,826	30.6	5.5	0.6	1.1	0.1	0.3	0.0	357.6	13.5	3.7	0.7	0.4	20	19	18	19	23	16	18	20
1k	64k	16k	2	2,023,632	2,096,027	1,056	5,175	1,162,057	2,096,027	29.5	5.4	0.6	3.6	0.1	0.3	0.0	390.0	12.6	3.5	0.7	3.6	19	18	22	21	19	20	21	21
1k	64k	2k	4	2,023,632	1,836,476	1,056	5,948	1,468,709	1,836,476	29.6	5.4	0.6	9.2	0.1	0.3	-0.1	463.2	7.3	2.7	0.8	2.0	23	21	23	24	23	19	22	25
1k	64k	4k	4	2,023,632	1,886,448	1,056	5,633	1,355,125	1,886,448	30.6	5.5	0.6	6.8	0.1	0.3	-0.1	433.4	7.0	2.7	0.8	0.5	20	19	18	19	19	20	21	19
1k	64k	16k	4	2,023,632	2,145,800	1,056	6,300	1,162,057	2,145,800	29.9	5.5	0.6	6.0	0.1	0.3	-0.1	496.6	11.7	3.4	0.7	8.5	21	25	18	19	20	20	22	20
2k	1k	2k	1	1,462,803	1,501,582	314,146	536,595	1,765,780	1,501,582	10.2	3.2	0.7	2.7	18.0	4.2	0.1	70.8	24.7	5.0	0.5	1.7	20	19	18	17	16	17	15	17
2k	1k	2k	2	1,462,803	1,519,541	314,146	533,491	1,765,780	1,519,541	10.2	3.2	0.7	3.9	17.0	4.1	0.2	69.8	25.1	5.0	0.5	3.4	19	18	17	18	25	21	22	21
2k	1k	4k	1	1,462,803	1,510,327	314,146	548,358	1,721,743	1,510,327	10.6	3.3	0.7	3.2	18.8	4.3	0.1	74.6	23.0	4.8	0.5	2.1	20	18	17	18	20	19	20	21
2k	1k	4k	2	1,462,803	1,530,552	314,146	546,055	1,721,743	1,530,552	10.7	3.3	0.7	4.6	17.9	4.2	0.1	73.8	23.7	4.9	0.5	3.8	20	19	21	19	19	20	27	21
2k	1k	16k	1	1,462,803	1,576,213	314,146	582,882	1,325,555	1,576,213	13.7	3.7	0.6	7.8	21.0	4.6	0.0	85.5	18.7	4.3	0.6	4.1	20	18	18	26	18	20	21	21
2k	1k	16k	2	1,462,803	1,587,632	314,146	581,073	1,325,555	1,587,632	13.7	3.7	0.6	8.5	20.1	4.5	0.0	85.0	17.4	4.2	0.7	2.6	20	17	19	21	20	22	20	23
2k	2k	2k	1	1,462,803	1,497,188	273,047	430,568	1,730,394	1,497,188	10.4	3.2	0.7	2.4	17.3	4.2	0.0	57.7	27.1	5.2	0.4	4.2	22	17	20	19	18	20	19	21
2k	2k	2k	2	1,462,803	1,511,886	273,047	419,732	1,730,394	1,511,886	10.2	3.2	0.7	3.4	15.6	4.0	0.1	53.7	27.0	5.2	0.4	6.0	22	18	19	18	18	21	21	22
2k	2k	4k	1	1,462,803	1,503,892	273,047	448,695	1,693,860	1,503,892	10.7	3.3	0.7	2.8	18.3	4.3	0.0	64.3	24.9	5.0	0.5	4.5	19	17	19	18	19	20	23	20
2k	2k	4k	2	1,462,803	1,521,894	273,047	437,777	1,693,860	1,521,894	10.7	3.3	0.7	4.0	16.5	4.1	0.1	60.3	25.2	5.0	0.5	6.2	21	20	19	19	19	20	22	19
2k	2k	16k	1	1,462,803	1,569,702	273,047	502,588	1,315,805	1,569,702	13.7	3.7	0.6	7.3	20.7	4.5	-0.2	84.1	16.9	4.1	0.7	2.9	23	17	18	19	19	21	22	19
2k	2k	16k	2	1,462,803	1,582,855	273,047	491,592	1,315,805	1,582,855	13.9	3.7	0.6	8.2	19.0	4.4	-0.1	80.0	15.9	4.0	0.7	1.5	20	20	20	19	18	21	22	21
2k	64k	2k	1	1,462,803	1,690,298	1,056	5,510	1,449,749	1,690,298	20.2	4.5	0.4	15.6	0.1	0.3	-0.1	421.8	8.1	2.8	0.8	6.8	23	17	19	18	20	22	20	21
2k	64k	2k	2	1,462,803	1,673,469	1,056	5,877	1,449,749	1,673,469	20.9	4.6	0.4	14.4	0.1	0.3	-0.1	456.6	7.7	2.8	0.8	3.9	20	19	17	19	19	22	22	22
2k	64k	4k	1	1,462,803	1,683,601	1,056	5,478	1,392,395	1,683,601	22.3	4.7	0.3	15.1	0.1	0.3	-0.1	418.8	8.2	2.9	0.8	7.3	21	18	18	18	20	20	20	23
2k	64k	4k	2	1,462,803	1,677,347	1,056	5,836	1,392,395	1,677,347	24.0	4.9	0.3	14.7	0.1	0.3	-0.1	452.6	7.5	2.7	0.8	3.8	21	19	18	18	20	22	19	19
2k	64k	16k	1	1,462,803	1,873,174	1,056	6,516	1,170,425	1,873,174	54.6	7.4	-0.6	28.1	0.1	0.3	-0.1	517.0	13.0	3.6	0.7	1.2	21	20	20	18	20	21	21	21
2k	64k	16k	2	1,462,803	1,924,095	1,056	6,494	1,170,425	1,924,095	58.4	7.6	-0.7	31.5	0.1	0.3	-0.1	515.0	12.4	3.5	0.7	3.9	20	19	18	18	20	21	21	23
2k	1k	2k	4	1,462,803	1,550,036	314,146	521,220	1,765,780	1,550,036	10.7	3.3	0.7	6.0	14.8	3.8	0.3	65.9	27.4	5.2	0.4	7.2	20	18	17	18	17	22	22	21
2k	1k	4k	4	1,462,803	1,560,273	314,146	534,841	1,721,743	1,560,273	11.2	3.4	0.7	6.7	15.7	4.0	0.2	70.3	26.4	5.1	0.5	7.6	16	14	15	18	16	17	18	21
2k	1k	16k	4	1,462,803	1,591,357	314,146	570,694	1,325,555	1,591,357	13.5	3.7	0.6	8.8	17.9	4.2	0.1	81.7	15.8	4.0	0.7	0.3	21	19	19	24	18	20	19	21
2k	2k	2k	4	1,462,803	1,539,410	273,047	394,649	1,730,394	1,539,410	10.8	3.3	0.7	5.2	12.7	3.6	0.3	44.5	27.9	5.3	0.4	9.5	25	18	17	18	21	22	19	21
2k	2k	4k	4	1,462,803	1,549,745	273,047	411,889	1,693,860	1,549,745	11.3	3.4	0.7	5.9	13.3	3.6	0.3	50.8	26.7	5.2	0.4	9.7	21	17	17	18	18	21	21	23
2k	2k	16k	4	1,462,803	1,588,866	273,047	465,042	1,315,805	1,588,866	13.6	3.7	0.6	8.6	15.3	3.9	0.1	70.3	14.8	3.9	0.7	1.0	21	18	18	19	18	22	23	21
2k	64k	2k	4	1,462,803	1,656,982	1,056	6,708	1,449,749	1,656,982	23.7	4.9	0.3	13.3	0.1	0.3	-0.1	535.3	7.1	2.7	0.8	3.2	26	18	17	19	20	21	22	21
2k	64k	4k	4	1,462,803	1,698,397	1,056	6,830	1,392,395	1,698,397	30.4	5.5	0.1	16.1	0.1	0.3	-0.1	546.8	6.7	2.6	0.8	3.7	20	18	20	18	19	20	19	24
2k	64k	16k	4	1,462,803	2,009,624	1,056	6,993	1,170,425	2,009,624	67.0	8.2	-1.0	37.4	0.1	0.3	-0.1	562.2	11.9	3.5	0.7	9.1	20	19	19	19	18	23	22	21
64k	1k	2k	1	739,204	787,815	314,146	684,905	1,033,108	787,815	15.1	3.9	0.4	6.6	26.8	5.2	-0.3	118.0	29.7	5.5	-0.1	29.8	20	18	18	18	19	24	23	21
64k	1k	2k	2	739,204	792,815	314,146	684,835	1,033,108	792,815	14.9	3.9	0.4	7.3	26.4	5.1	-0.3	118.0	29.3	5.4	-0.1	28.7	21	19	18	18	18	20	21	19
64k	1k	4k	1	739,204	782,650	314,146	684,375	913,889	782,650	15.2	3.9	0.4	5.9	26.7	5.2	-0.3	117.9	31.6	5.6	-0.1	43.3	22	18	17	19	19	19	20	21
64k	1k	4k	2	739,204	787,743	314,146	684,817	913,889	787,743	15.0	3.9	0.4	6.6	26.3	5.1	-0.3	118.0	30.5	5.5	-0.1	42.2	22	18	18	18	18	20	20	24
64k	1k	16k	1	739,204	744,781	314,146	679,309	835,901	744,781	15.9	4.0	0.3	0.8	26.2	5.1	-0.3	116.2	24.2	4.9	0.1	34.0	21	17	18	19	19	20	23	22
64k	1k	16k	2	739,204	747,844	314,146	680,935	835,901	747,844	15.6	4.0	0.4	1.2	25.9	5.1	-0.3	116.8	23.3	4.8	0.2	33.1	22	17	18	19	20	21	20	20
64k	2k	2k	1	739,204	793,012	273,047	632,685	1,011,131	793,012	15.1	3.9	0.4	7.3	28.0	5.3	-0.6	131.7	29.1	5.4	-0.1	29.9	20	17	17	19	21	21	24	23
64k	2k	2k	2	739,204	797,793	273,047	629,005	1,011,131	797,793	15.0	3.9	0.4	7.9	27.3	5.2	-0.5	130.4	28.4	5.3	-0.1	28.6	22	18	18	17	19	18	20	21
64k	2k	4k	1	739,204	788,192	273,047	632,507	899,551	788,192	15.3	3.9	0.4	6.6	27.9	5.3	-0.6	131.6	30.1	5.5	-0.1	42.9	21	17	19	19	24	19	21	21
64k	2k	4k	2	739,204	793,077	273,047	629,302	899,551	793,077	15.1	3.9	0.4	7.3	27.3	5.2	-0.5	130.5	28.8	5.4	0.0	41.5	20	19	18	20	21	22	20	21
64k	2k	16k	1	739,204	751,842	273,047	630,605	829,418	751,842	16.0	4.0	0.3	1.7	27.5	5.2	-0.5	131.0	23.4	4.8	0.2	33.6	22	17	18	18	20	22	20	20
64k	2k	16k	2	739,204	755,298	273,047	628,974	829,418	755,298	15.7	4.0	0.4	2.2	26.9	5.2	-0.5	130.4	22.4	4.7	0.2	32.6	22	20	21	19	24	21	21	21
64k	64k	2k	1	739,204	758,090	1,056	8,161	740,252	758,090	18.0	4.2	0.3	2.6	0.1	0.3	-0.2	672.8	15.2	3.9	0.4	6.6	21	18	17	19	18	19	20	22
64k	64k	2k	2	739,204	741,935	1,056	8,438	740,252	741,935	18.1	4.3	0.3	0.4	0.1	0.3	-0.2	699.1	15.3	3.9	0.4	4.6	17	16	14	18	19	20	21	21
64k	64k	4k	1	739,204	754,878	1,056	7,887	740,213	754,878	17.9	4.2	0.3	2.1	0.1	0.3	-0.2	646.9	15.1	3.9	0.4	5.5	20	19	19	18	18	20	20	24
64k	64k	4k	2	739,204	738,388	1,056	8,210	740,213	738,388	18.1	4.2	0.3	0.1	0.1	0.3	-0.2	677.5	15.4	3.9	0.4	3.5	20	19	18	18	18	20	19	20
64k	64k	16k	1	739,204	722,813	1,056	7,718	717,581	722,813	18.4	4.3	0.2	2.2	0.1	0.3	-0.2	630.8	16.2	4.0	0.3	1.3	17	19	18	22	18	20	21	21
64k	64k	16k	2	739,204	706,138	1,056	8,731	717,581	706,138	18.7	4.3	0.2	4.5	0.1	0.3	-0.2	726.8	16.5	4.1	0.3	0.6	21	17	18	18	18	16	23	17
64k	1k	2k	4	739,204	800,714	314,146	678,952	1,033,108	800,714	14.9	3.9	0.4	8.3	25.6	5.1	-0.2	116.1	28.2	5.3	0.0	26.5	17	18	18	18	19	21	23	21
64k	1k	4k	4	739,204	795,675	314,146	679,616	913,889	795,675	15.0	3.9	0.4	7.6	25.6	5.1	-0.2	116.3	28.8	5.4	0.0	39.9	21	19	19	20	19	21	19	19
64k	1k	16k	4	739,204	747,317	314,146	677,526	835,901	747,317	15.7	4.0	0.4	1.1	25.2	5.0	-0.2	115.7	22.3	4.7	0.2	30.9	20	19	17	19	20	18	19	22
64k	2k	2k	4	739,204	806,470	273,047	617,510	1,011,131	806,470	14.9	3.9	0.4	9.1	26.2	5.1	-0.5	126.2	26.9	5.2	0.0	26.4	21	17	21	18	19	21	19	23
64k	2k	4k	4	739,204	801,390	273,047	618,340	899,551	801,390	15.1	3.9	0.4	8.4	26.2	5.1	-0.5	126.5	27.2	5.2	0.0	39.2	20	18	18	18	21	20	20	20
64k	2k	16k	4	739,204	753,537	273,047	618,131	829,418	753,537	15.8	4.0	0.4	1.9	25.8	5.1	-0.4	126.4	21.3	4.6	0.2	30.0	20	18	17	19	18	21	19	21
64k	64k	2k	4	739,204	708,299	1,056	9,436	740,252	708,299	18.4	4.3	0.2	4.2	0.1	0.3	-0.3	793.5	15.7	4.0	0.4	0.8	22	18	16	15	17	18	19	17
64k	64k	4k	4	739,204	704,701	1,056	9,502	740,213	704,701	18.6	4.3	0.2	4.7	0.1	0.3	-0.3	799.8	15.9	4.0	0.4	0.2	21	18	17	18	20	19	19	20
64k	64k	16k	4	739,204	666,141	1,056	12,596	717,581	666,141	19.5	4.4	0.2	9.9	0.1	0.3	-0.5	1092.8	17.2	4.1	0.3	5.0	21	19	19	18	18	20	20	22