International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I7P123

Volume 74 Issue 7, 372-388, July 2026

Original Article

Machine Learning-Driven Optimization of Cache Memory Prefetching Processes

Remegius Praveen Sahayaraj L[1], Anitha E[2], Aswini E[3]

[1]*Computer Science and Engineering, Loyola-ICAM College of Engineering and Technology, Tamil Nadu, India.*
[2]*School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Vellore, Tamil Nadu, India.*
[3]*Artificial Intelligence and Data Science, St. Joseph's College of Engineering, Tamil Nadu, India.*

[1]*Corresponding Author : pravinsahayaraj@gmail.com*

Received: 04 July 2025 Revised: 18 June 2026 Accepted: 24 June 2026 Published: 28 July 2026

Abstract - *Despite the fact that computer memory has a hierarchical structure that is specifically designed to mitigate the speed disparity between the memory and the processor, it is undeniable that a bottleneck still persists. A prefetcher effectively addresses and alleviates the aforementioned issue by proactively and pre-emptively fetching pertinent memory blocks into the memory levels that are in closest proximity to the processor, even before an explicit request is made by the conventional MMU components. Traditional prefetchers, on the other hand, rely on table-based techniques that are restricted by the proportional increase in memory demands or are incapable of forecasting intricate memory access patterns. The proposed model enhances cache prefetching by implementing an LSTM-based prefetcher that learns from dynamic program traces, thereby eliminating the linear relationship between fetch count and space while enhancing the capability to identify and forecast intricate access patterns.*

Keywords - *Accuracy of Binary Values, LSTM, Memory, Model Compression, Prefetchers.*

1. Introduction

The Memory Wall In 1965, Gordon Moore forecasted that the number of transistors on a single silicon chip would double every year. The time period for doubling was later revised to 2 years and soon came to be known as Moore's law, as it transcended from being regarded as a mere forecast to almost an inevitable truth.

The actual development in transistor technology has largely followed this trajectory for over 50 years after this prediction. While processors saw rapid improvements in speed, improvements in memory technology were largely focused around economically increasing capacity. This has caused a divergence in microprocessor and memory speeds over the last few decades.

Article [8] demonstrates this divergence through the comparison of memory requests per second on average by a single processor system against the latency of DRAM accesses per second on a logarithmic scale [4].

The number of CPU cycles per memory access is steadily growing, and this diverging gap between CPU and memory performance threatens to hit a memory wall, where an increase in CPU speeds no longer results in better execution times because of the bottleneck created by relatively slower memory technology [18]. Present-day computer memory systems, as shown in Figure 1, are not designed as a single storage pool, but rather as a hierarchy of levels that vary by three main factors - size, speed and cost. This kind of structuring primarily exploits the following observations:

- Processes in execution often access only a small subset of memory at any given time
- The distribution of memory accesses frequencies is nonuniform, i.e., some addresses or contents of memory are accessed more often than others.

By strategically placing frequently accessed data in levels closer to the processor, the average latency of memory accesses can be reduced, resulting in an overall improvement in execution time.

However, the number of levels that can be introduced into the hierarchy are limited by cost as well as consumption of power through static and dynamic discharge. Prefetching is the process by which a memory block is obtained from a lower level in the memory hierarchy before a direct solicitation is made by the processor, thereby emphasizing the proactive nature of the memory retrieval process. This proactive approach ensures that the necessary data is readily available for the processor's use, optimizing the overall efficiency and

performance of the system. The purpose of prefetching is to mitigate the effects of memory latency. The idea that lies within the core of prefetching is the prediction of future memory accesses ahead of time, thereby serving as an effective means to mitigating the effects of the memory wall. The objective of this research is to redesign and prototype a system that can learn complex memory access patterns (without imposing a physical limitation on the pattern learnability) to predict and prefetch the relevant blocks that may be required prior to a clear request from conventional MMU components. This approach is a practical method to trade computation for a reduction in memory latency, as a solution to the approaching memory wall.

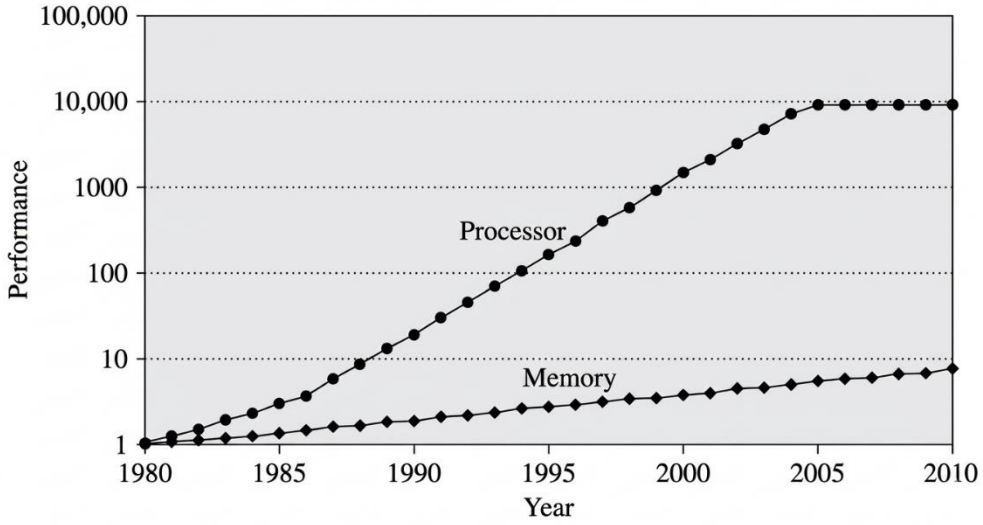


Fig. 1 Gap in performance, measured as the difference in the time between processor memory requests and the latency of a DRAM access (Hennessy, J. L. and Patterson, D. A. 2011, p. 73)

2. Related Works

Learning Memory Access Patterns by [7] demonstrates that LSTM-based prefetchers predict cache misses with better accuracy than typical CPU prefetchers, but found that its memory footprint was too large to be considered for hardware implementation. They had used an embedding-based and a clustering-based approach to LSTM, which was trained in an offline scenario using data obtained from a cache prefetcher. The paper showed the usage of address deltas instead of absolute addresses to overcome the side effects of Address Space Layout Randomization (ASLR). One of the major differences between this paper and the proposed work is that it trains it model on cache misses alone.

Effects of an LSTM Composite Prefetcher by [13] augments the LSTM prefetcher with a simple stream prefetcher. This was shown to improve results as the neural network could now focus on learning a smaller subset of access patterns that are not picked up by the stream prefetcher. The paper, treating the prediction of cache misses as a classification problem, implements clustering techniques while pre-processing. While this was shown to increase accuracy, the inclusion of clustering in the data pre-processing pipeline created limitations for extension into live systems, which have been addressed in the proposed work. Similar to the above paper, the Long Short-Term Memory-Based Hardware Prefetcher by [21] augmented the LSTM with a stream prefetcher. Here, some LSTM components were implemented in hardware (similar to look-up tables for activation functions) while storing local deltas in a dedicated page table.

This enabled a reduction in the LSTM model size but had the limitation that page tables can only be finite. The paper also shines a light on how hardware implementations for mathematical functions are crucial to the realisation of a hardware implementation of a neural network-based prefetcher.

Predicting Memory Accesses: The Road to Compact ML-driven Prefetchers by [15] compared the benefits of vanilla LSTM models against compressed LSTMs. The paper went on to show that using compression techniques like vocabulary binarization drastically reduces model size at the cost of a mild loss in accuracy. Additionally, they noticed that not all of the situations they looked at had acceptable prediction accuracy using conventional models like Naïve Bayes and Logistic Regression. Furthermore, the memory access prediction sequence is more accurately captured by deep learning models.

MemMAP: Compact and Generalizable Meta-LSTM Models for Memory Access Prediction by [23] proposes and experimentally demonstrates the achievement of equal levels of accuracy as the specialized approach with a much smaller memory footprint using a clustered meta-learning-based

approach. The proposed MemMAP approach has shown to adapt quickly to different applications and yield high accuracy, thus making it generalisable to applications that were not seen during training. A Neural Network Prefetcher for Arbitrary Memory Access Patterns by [12] provided a context-based Neural Network (NN) prefetcher that links machine and programme contextual information with memory access patterns and dynamically adjusts to arbitrary memory access patterns. Online training is used for this correlation in order to recognise and dynamically adjust to the code's unique access patterns.

The work [16] presents a comprehensive analysis of the performance issues in deduplication-based storage caches and proposes a content-driven cache management approach called CDAC. The model evaluates existing deduplication-aware caching algorithms and identifies their limitations. It discusses the challenges of implementing deduplication in SSD caches, such as deduplication overhead and low cache space utilization. The proposed CDAC approach focuses on exploiting content redundancy and intensity of content sharing in cache management strategies. CDAC is implemented based on LRU and ARC algorithms and is shown to outperform state-of-the-art deduplication-aware caching algorithms in terms of read cache hit ratio and IOPS under real-world workloads.

The model also provides background information on data deduplication and discusses factors affecting the efficiency of deduplication-aware caching algorithms, such as deduplication ratio and misidentified hot blocks. Overall, the authors present a comprehensive evaluation of CDAC and its performance compared to baseline approaches, providing evidence for the effectiveness of CDAC in improving the performance of deduplication-based SSD caches. The model [9] presents a proposal for defending against cache side-channel attacks, specifically the Flush+Reload attack, by exploiting the L4 DRAM cache. The authors introduce an approach called Bypassing Cache Architecture (ByCA) that bypasses the shared L3 cache when accessing cache blocks that have been flushed by the attacker using the clflush instruction. ByCA eliminates the timing differences that the attacker relies on to infer the victim's cache access patterns, thus nullifying the Flush+Reload attack. They provide a comprehensive literature survey on cache side-channel attacks, such as Flush+Reload, Prime+Probe, and Evict+Reload, which can steal confidential information by inferring the execution flow and accessing patterns of a victim process. These attacks exploit timing differences in cache accesses to determine whether certain cache blocks have been accessed by the victim.

The contribution [26] discusses the challenges and solutions for caching in Content-Centric Mobile Ad Hoc Networks (CCMAN). The authors propose a Cache Space Efficient Caching (CSEC) scheme based on the Generalized Dominating Set (GDS) method to improve cache space utilization in CCMAN. The authors elucidated the notion of Content-Centric Networking (CCN) and its capacity to mitigate network traffic and content latency in the delivery of multimedia content. They underscored the significance of in-network caching in CCMAN and the obstacles presented by the restricted cache size and the simultaneous use of all wireless nodes as users and routers. The theoretical examination of CCMAN's caching performance is included in the model. It derives the cache hit probability function from the link between cache frequency and cache probability and defines cache utility as the ratio of cache hit ratio to occupied cache space. Using the LRU caching replacement technique, the model evaluates the cache hit ratio and cache usefulness in terms of cache probability. Based on the theoretical analysis, the authors propose the CSEC scheme. This scheme uses the GDS method to select caching locations that are meant for the replicas of the content, ensuring that the cached contents can be fully utilized. By offering a trade-off between cache hit ratio performance and occupied cache space, the authors contend that their proposed CSEC enhances both caching performance and cache space efficiency. Through simulation, the suggested CSEC scheme's performance is assessed. The findings demonstrate that, in comparison to the two current DS-based caching systems, GDSC and CoopCache, the suggested scheme achieves optimal performance on cache utility, ARTH gain, and ARTT gain. Additionally, the authors examine how node mobility affects caching performance and show that, in mobility circumstances, the suggested CSEC performs better than both GDSC and CoopCache.

The contributors of work [17] provide a critical literature survey by discussing various existing approaches to cache analysis and profiling. It reviews traditional hardware debuggers, simulation tools, performance counter sampling, and measurement-based reverse engineering. The work then introduces CacheFlow as a novel technique for cache analysis, highlighting its advantages in terms of simplicity, versatility, and behavioral insights. The authors validate CacheFlow's capabilities through experiments on synthetic and real benchmarks, analyzing cache behavior, cache pollution, timing, system-level properties, interference between applications, and cache replacement policy. The findings suggest that CacheFlow can provide valuable insights into cache behavior and aid system designers in understanding the interplay between applications, system components, and cache hierarchy.

The work [11] presents a scheme for managing flash-based Solid-State Cache (SSC) to enhance the HDD-based storage system's input/output performance in home cloud contexts. A Hybrid Unit 2Level-LRU (HyU 2Level-LRU) cache management policy is put forth by the authors, which handles write request data groups and read request data groups independently. The plan is to minimise disc head movement during destaging, decrease disc access, and boost cache hit

ratio. The data management plan, cache logical structure, and system architecture are all described in the model. In order to reduce disc head movement during destaging, the HyU 2Level-LRU scheme handles dirty pages in groups while keeping frequently referred clean pages on a separate LRU list. Two crucial criteria that must be optimised in accordance with workload characteristics are the size of the dirty page group and the L1 list.

The authors analyze the impact of these cache parameters on various workloads and propose equations to determine the optimal group size and L1 list size. They also evaluate the performance of the HyU 2Level-LRU scheme through trace-based simulation experiments. The results show that the proposed scheme reduces execution time compared to the Page Level LRU scheme and improves cache hit ratio in different workloads.

The work [10] discusses the concept of coded caching and its application in systems with multiple antennas and heterogeneous cache sizes. It emphasises the advantages of using coded caching in conjunction with multiple transmit antennas to achieve full multiplexing and caching gains. The authors introduce a novel approach that provides full multiplexing and programmed caching advantages to both cache-aided and cache-less users by utilising perfect matchings on a bipartite graph. They demonstrate that, in a single-stream centralised programmed caching setup, an additional L antenna can support up to around L/Γ cache-less users without introducing additional delay costs. The difficulties caused by cache-size variability in coded caching systems are covered in the paper [14]. It investigates several methods, such as layering caches, placing them decentralised, and optimising subfile sizes, to lessen the detrimental consequences of cache-size unevenness. The primary findings of the authors' research are presented, together with the delivery times that can be achieved in systems with both cache-aided and cache-less users. They demonstrate that treating cache-aided and cache-less users differently in the presence of cache-less users is the best course of action, albeit at the expense of a longer delivery time. In order to account for the varying storage limitations of various device kinds, they also take into account scenarios with varying cache sizes.

The work [3] provides a comprehensive analysis of list-based cache replacement policies under nonuniform access patterns. The authors suggest using cutting-edge techniques to examine how well list-based caches work when they have random or first-in, first-out replacement rules, nonuniform access to items and lists, and tree structure. Their novel methods' accuracy is compared to simulations, and they discover that as the number of items and the cache capacity grow in a fixed ratio, their approaches swiftly converge to the equilibrium distribution. The model extends first-in first-out (CLIMB) policies, which are used to analyse linear list-based caches with random replacement, to list-based caches with nonuniform access. The writers create both precise and approximative techniques for performance analysis, such as a revised mean-field approximation and a unique perturbation method.

In order to decrease the filtering impact and data redundancy and enhance the performance of a network of caches, the article proposes a caching system called Frozen Cache [1]. The authors contend that because of issues including data dependencies, cache size limitations, and subpar replacement rules, current cache management strategies created for a single cache are insufficient for a network of caches. They propose Frozen Cache as a solution to mitigate the filtering effect and improve cache performance. The model introduces variations of the Frozen Cache policy, such as FC1 and FC2, which further enhance the performance of the caches. FC1 caches the first C distinct requested chunks, while FC2 filters out one-time requests by caching only content that has received at least one hit.

These variations aim to improve cache hit rate and reduce data redundancy. In addition to the Frozen Cache policy, the method [22] proposes a coordinated cache scheme that integrates with Frozen Cache to further boost cache hit rate. The coordinated scheme aims to reduce redundant copies of data chunks in the network by ensuring that only one router in a path caches a particular data chunk. The coordination is achieved through message exchange between routers, indicating their willingness to cache specific content. The content is then cached by the router with the highest closeness to the consumer. The performance evaluation results show that Frozen Cache outperforms existing caching schemes in terms of cache hit rate, average content download time, and traffic reduction ratio. The coordinated scheme further improves cache hit rate and reduces data redundancy. The evaluation results demonstrate the effectiveness of these schemes in reducing the filtering effect, improving cache hit rate, and reducing data redundancy.

Microprocessor emulation, activity extraction, and performance-reliability metric evaluation are all included in the framework [25]. Using an ARM-v8 microprocessor as an example, the authors apply the framework to analyse the metric degradation of an I-Cache. It is discovered that while hold 0 power improves, read 0 access time, Qcrit, and Static Noise Margin (SNM) deteriorate with increasing stress time. In the L1 I-Cache, ageing results in a broader Qcrit distribution. The 8T structure is more reliable than the 6T structure, while requiring more space, according to a comparison of the two caches. Accurate metric evaluation requires the incorporation of gate length (Lg) fluctuation and Random Telegraph Noise (RTN). The effect of cache settings on the hit rate and time-dependent Soft Error Rate (SER) is also examined by the writers. It is discovered that although the hit rate stays the same, increasing the cache line size from 8 to 32 B results in a 3.5-fold increase in SER.

The methodology's accuracy [2, 5] and the significance of different elements in cache parameter selection are discussed by the authors. They point out that accelerated testing can be used to objectively verify the methodology's accuracy, which is contingent upon the accuracy of the underlying models. The choice of a cache's parameters is determined by the designer's specifications on aspects like area, complexity of hardware implementation, ageing, performance, and reliability. The article [6] provides a critical literature review of the optimization of caching strategies in device-to-device (D2D) communication networks. It highlights the importance of caching at the network edge to reduce backhaul transmission load and improve network latency. The authors propose a joint optimization algorithm that considers both caching and recommendation decisions to maximize the cache hit ratio and improve caching efficiency.

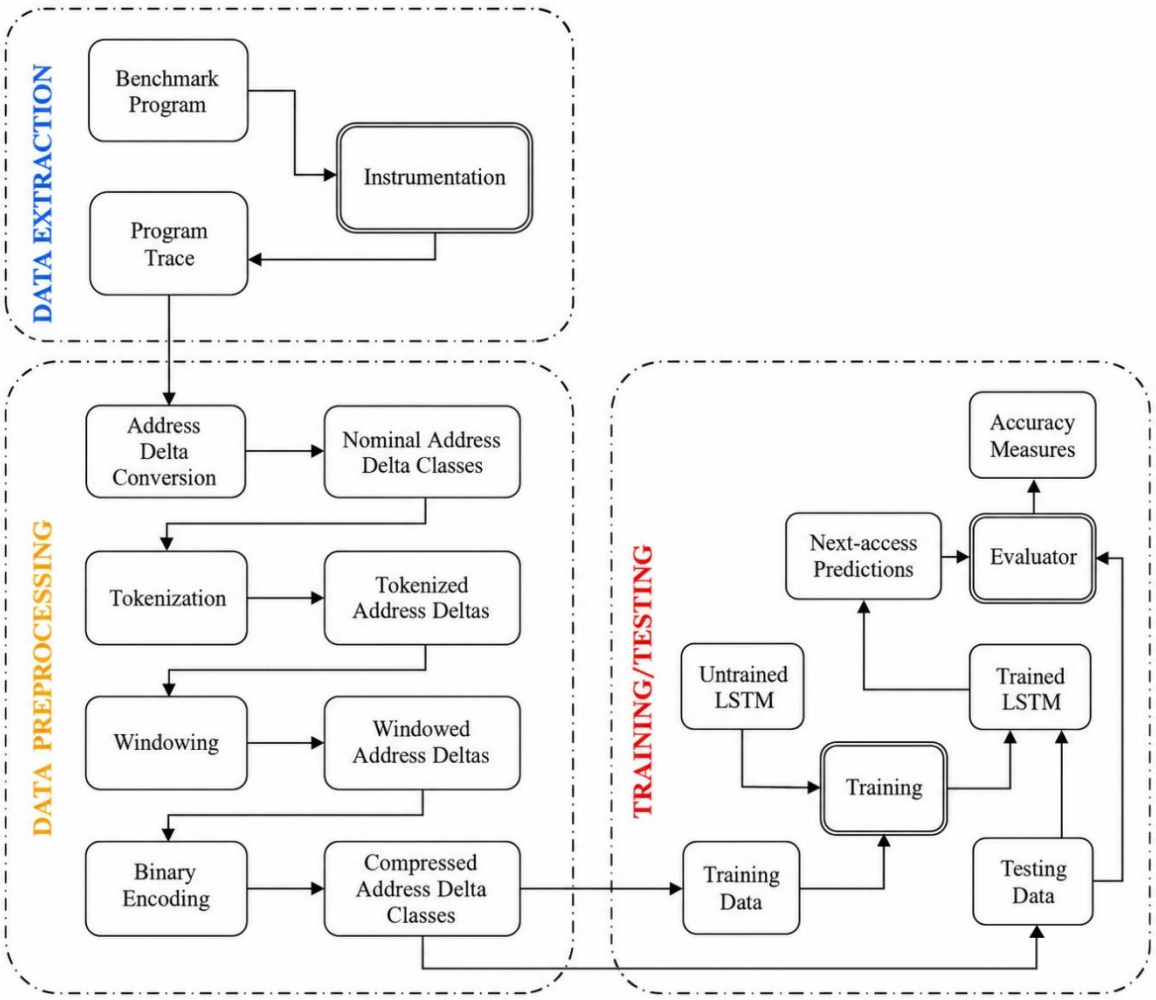


Fig. 2 System design

The model presents a comprehensive review of related works in the field, discusses the system model of the D2D-assisted wireless content caching network, formulates the caching efficiency maximization problem, and proposes algorithms for caching and recommendation decision optimization. The model also evaluates the proposed algorithm through numerical simulations and comparisons with baselines, and analyses the impact of various parameters on cache efficiency. Overall, it provides valuable insights into the optimization of caching and recommendation strategies in D2D communication networks. A transformer-oriented method named TransforMAP [24] was introduced for predicting memory access by utilizing the Transformer framework. In contrast to conventional LSTM-based methods that mainly emphasize sequential access prediction, TransforMAP learns to recognize both temporal and spatial memory access patterns while being able to predict several future cache lines at the same time. The research delivered enhancements in memory access prediction performance relative to traditional hardware prefetchers. Anyhow, the computational demands and resource needs tied to Transformer architectures can pose difficulties for implementation in hardware environments with limited resources.

Recent research has investigated reinforcement learning for managing adaptive caching and prefetching. The RL-CoPref [19] framework adaptively modifies prefetch activation and the degree of prefetching based on the runtime behavior of the program and the conditions of memory bandwidth. Reinforcement learning methods can adapt to evolving workloads more efficiently than static rule-based approaches by consistently learning from system feedback. However, online learning systems frequently add extra decision-making burdens and might need longer training durations to reach consistent performance. Prefetch-Adaptive Intelligent Cache Replacement (PAIC) utilizes machine learning models [20] to manage cache replacement and prefetching strategies, enhancing cache performance across varying workloads. Although these methods enhance adaptability, they frequently necessitate several prediction modules and added complexity in implementation, potentially affecting hardware viability.

Table 1. Literature comparison

Approach	Technique	Strength	Limitation
Stream Prefetchers	Rule-Based	Low overhead	Poor handling of complex patterns
LSTM Composite Prefetcher	LSTM + Stream	Improved accuracy	Large memory footprint
MemMAP	Meta-LSTM	Better generalization	Additional preprocessing complexity
TransforMAP	Transformer	Captures long-range dependencies	High computational cost
RL-CoPref	Reinforcement Learning	Adaptive behaviour	Training and runtime overhead
Proposed Model	Binary-Encoded LSTM	Compact, accurate, deployable	Requires offline training

Although good progress has been made in cache prefetching and memory access prediction as shown in Table 1, several limitations remains in existing approaches. Traditional hardware prefetchers are generally lightweight and easy to implement, but often struggle to capture complex and irregular memory access patterns.

Deep learning-based approaches have demonstrated improved prediction capability; however, many of them suffer from large model sizes, increased computational overhead, and limited hardware deployability. More recent Transformer-based and reinforcement learning-based techniques provide enhanced adaptability and predictive performance, but they typically require greater computational resources and introduce additional implementation complexity. Furthermore, several existing approaches are designed and evaluated for specific workloads, thereby limiting their generalizability across diverse domains.

These limitations create the need for a compact and efficient prediction framework capable of balancing prediction accuracy, memory footprint, adaptability, and practical deployment feasibility. The study also provides a critical analysis of cache parameters and their impact on performance. The study mentions several existing defense mechanisms against cache side-channel attacks, such as cache partitioning, modified clflush instructions, and cache replacement policies. However, these mechanisms either degrade overall cache utilization or restrict the use of clflush instructions, limiting their effectiveness. Hence, systems which can reflect upon the disadvantages that were discussed above are required and should work towards facilitating better systems with improved efficiency. The proposed binary-encoded LSTM-based prefetching model is designed to address these challenges by combining sequential learning capability with reduced model complexity and memory requirements.

3. System Design

The purpose of the system is to predict future memory accesses that will be requested by the processor ahead of time. These predictions can serve as early interleaved requests to the memory management unit and, therefore, enable ahead-of-time caching and, in turn, an overall improvement in memory access speeds. The memory access prediction system is modularised as depicted in Figure 2.

3.1. Data Extraction

The purpose of the data extraction module is to obtain and record the data that will be used by the prediction system. This data comprises of memory accesses made by a program during execution. The benchmark programs are obtained from the PARSEC benchmark suite, a free and open-source suite of programs collated and provided by Princeton University in collaboration with Intel.

The benchmark programs selected for use from within the suite are given as follows.

- Blackscholes - Option pricing with Black-Scholes Partial Differential Equation (PDE)

- streamcluster - Online clustering of an input stream
- freqmine - Frequent itemset mining
- raytrace - Real-time raytracing
- swaptions - Pricing of a portfolio of swaptions

Instrumentation is the process by which a program is profiled dynamically so as to obtain information such as traces, logs and performance metrics while execution. The PIN tool developed by Intel is used to perform instrumentation at runtime on compiled binary files, which in the proposed case are the PARSEC benchmark programs. The tool records information about the program's execution along its execution path to obtain a log called the program trace, which contains information pertaining to the addresses of instructions executed along with corresponding memory addresses of data referenced during the program's execution. Therefore, memory accesses made during a program's execution are obtained.

3.2. Data Pre-Processing

The memory trace obtained from the data extraction module comprises a sequence of memory addresses accessed by the program during execution. This memory address sequence is passed through a data pre-processing pipeline in which it undergoes a series of conversions.

3.2.1. Address Delta Conversion

The program trace comprises of absolute memory addresses accessed by the program during the execution of its sequence of instructions. However, since the program is loaded into memory as a process for execution. Each load will occur at a different memory location. ASLR also introduces randomization within the layout of the address space of a process. Hence, the first submodule of the data pre-processing module fetches the absolute memory addresses from the program trace, converts them into address deltas and stores the delta access sequence into a CSV file.

3.2.2. Tokenization

The data to be predicted is sequential and, in some ways, similar to predicting words in a sentence in human languages. In order to predict(classify) the word that is to appear next, a vocabulary of words has to be known. The tokenization submodule generates the vocabulary of deltas and subsequently encodes the sequence of deltas. The tokenizer identifies the vocabulary of deltas and the top n deltas by frequency and encodes the sequence of deltas that appear in a specific sequence. The tokenizer object generated by this submodule is serialized and stored in a .pickle file, enabling the object to be loaded and reused whenever required.

3.2.3. Windowing

A lookback period defines the number of immediately preceding timesteps that a model has access to while making a prediction. The lookback window at each timestep provides the data within the lookback period from that timestep. Thus, the windowing submodule converts the data into a dataset of lookback windows matched with their correct predictions. For example, a sequence [1, 2, 3, 4, 5], if converted using a lookback period of 2, would transform into [[[1, 2],[3]], [[2, 3], [4]], [[3, 4], [5]]].

3.2.4. Binary Encoding

Traditionally, the prediction of word sequences has been treated as a classification problem. Large vocabularies can result in large input and output layers and subsequently in large models. This presents a computational bottleneck to the prediction system.

One method to mitigate the model size problem due to a large vocabulary is to predict the binary value of the delta class encoding rather than assigning an output unit to individual deltas. The representation can be altered such that each input (and output) unit represents one bit in the binary encoding of the delta token. The result of this transformation has two effects:

- The model will have to predict all classes and not just one class accurately, since the actual delta can be obtained only by decoding all the binary bits from the resulting predictions of every output unit. This could result in a decrease in accuracy.
- The number of units in the input and output layers of the model reduces by a factor of log2n.

The reduction in accuracy due to this compression technique has been shown to be negligible in comparison to the reduction in model size by [15].

The binary encoding submodule takes in windowed delta sequences and produces as output windowed binary encoded sequences. The operation is performed as a list comprehension over the window, in which each delta class is individually encoded by a user-defined function. The module additionally logs the dimensions of the input and output for verification.

3.3. Training/Testing

The pre-processed data obtained from the data pre-processing module comprises a windowed sequence of encoded memory deltas accessed by the program during execution, followed by the next-access address delta that the LSTM would have to predict. This pre-processed data set is split into training and testing data over which an LSTM model is trained and evaluated on next-access prediction of address deltas.

3.3.1. Embedding Layer

The embedding layer is the first layer present in the model. The purpose of an embedding layer is to circumvent the need to deal with a large input feature set by learning the

relationship between the words by finding the optimal mapping of each of the unique words from a vocabulary to a vector of a predefined dimension referred to as the output dimension. The model takes in the pre-processed data as input, feeds it into the embedding layer and represents words as an array of continuous vectors. The model's embedding layer has an input dimension of 24 units, an output dimension of 8 units, which take in a vocabulary of a maximum length of 24.

3.3.2. LSTM

LSTMs, or Long Short-Term Memory networks, are commonly used to learn sequential data due to the ability to learn long- and short-range dependencies in temporal sequences. The LSTMs, unlike standard feedforward networks, utilize a feedback connection built into their architecture, making them capable of learning order dependencies in sequential prediction problems. This also enables them to tackle problems like vanishing gradients more effectively.

A typical LSTM cell comprises a forget gate with an activation vector ft, an input gate with an activation vector it, and an output gate with an activation vector ot. These gates control the cell state ct, the hidden state ht and the input for the particular cell xt. It can be expressed mathematically as follows.

$$ft = \sigma (Wf .[ht-1, xt] + bf) \quad (1)$$

$$it = \sigma (Wi.[ht-1, xt] + bi) \quad (2)$$

$$ot = \sigma (Wo[ht-1, xt] + bo) \quad (3)$$

$$ct = \tanh (Wc[ht-1, xt] + bc) \quad (4)$$

$$ct = ft.ct-1 + it.ct \quad (5)$$

$$ht = ot.\tanh(ct) \quad (6)$$

Depending on the window size used, the size of the input passed into the LSTM would be the number of bits in a single memory access delta multiplied by the window size.

In the proposed model, the embedding layer feds in an input of size 24, which is the window containing three consecutive memory access deltas, each 8-bit long. The model uses the Keras framework for the LSTM layer used in this model.

3.3.3. Dropout Layer

The dropout layer is a regularization layer where inputs, as well as recurrent connections, are probabilistically excluded from activation and weight update by randomly setting a certain fraction of the input units to 0, thereby 'dropping out' some useful data. This method of introducing noise has shown to be effective in improving the performance of the model as it drastically reduces overfitting. The inputs which are not set to 0 are scaled up by a factor of $1/(1 - \text{rate})$ to ensure that the overall sum of inputs is unmodified. In the proposed model, the dropout rate is set to 0.1.

3.3.4. Dense Layer

The dense layer is the most basic form of a neural network that, unlike convolutional layers, does not rely on local spatial coherence to learn features but learns features from the combinations of the features of the previous layer. These are layers of neurons that are fully connected to all the neurons of the previous layer. The proposed model operates with a dense layer that has 8 dense units and uses a sigmoid activation.

3.3.5. Activation Function

Activation functions are a function that maps a specific input with output. The role of an activation function in a neural network is to essentially activate or deactivate a node by calculating the result, given the information fed into that node. The proposed model adopts the sigmoid activation function as it works well in approximating a classifier function.

3.3.6. Loss Function

A loss function is a function that takes in input from a certain range and outputs a number representing a loss associated with that input. The purpose of the loss function is to measure how close or further away a given output it receives is from the expected output. The model uses binary cross-entropy as its loss function.

3.3.7. Optimizer Adam

The goal of an optimiser is to minimize the cost function. The cost function in a neural network is defined over the entire training set in a similar way that the loss function is defined over a single training example. Optimisers are used to optimise attributes such as weights and learning rate of a neural network in order to reduce the loss and provide the most accurate results possible. The model uses an Adam optimiser for training.

3.3.8. Evaluation and Accuracy

The pre-processed dataset is divided into three sets for training (64%), validation (16%) and testing (20%). During training, the weights that provide the highest binary accuracy on the validation set are retained and finally evaluated against the test set. The following set of metrics is used to evaluate the model at different stages.

- Binary accuracy is the percentage of thresholded binary values predicted correctly. It is calculated as the ratio of accurate predictions(y_{true}) to total predictions made(y_{pred}) for a binary classification, given as:

  $$\text{Binary accuracy} = y_{true}/y_{pred} \quad (7)$$

- Binary cross-entropy defines the distance of the true probability distribution to predicted distributions as a way to measure the error for a given binary classification. From the probability distribution, given a probability p for the binary indicator y (which, depending on the class, takes either a 0 or a 1), the binary cross entropy is given as:

  binary cross entropy = −(y log(p)+(1 − y)log(1−p)) (8)

- Mean Absolute Error can be defined as the average of the absolute variations on the actual and the predicted values obtained. For a given n values, the mean absolute error over the set of actual values x and predicted values y is given as:

$$\text{MAE} = \left(\frac{1}{n}\right)\sum_{i=1}^{n}\left|y_i - x_i\right| \quad (9)$$

- Mean Squared Error can be defined as the average of the squared prediction errors over all instances in the given set. For a given n values, the mean squared error over the set of actual values x and predicted values y is given as:

$$\text{MSE} = \left(\frac{1}{n}\right)\sum_{i=1}^{n}\left(y_i - x_i\right)^2 \quad (10)$$

3.4. Experimental Setup

The experiments were conducted on a workstation equipped with an Intel Core i7 processor, 32 GB RAM, and an NVIDIA graphics card with 4 GB dedicated memory. The implementation was processed using Python and the Keras deep learning framework with TensorFlow at backend. The experimental environment, as shown in Table 2, was configured to support the training and evaluation of Long Short-Term Memory (LSTM) models on memory access traces extracted from benchmark applications. The operating system used for experimentation was a 64-bit Windows environment. The hardware configuration provided sufficient computational resources for training the proposed model and evaluating its performance across multiple benchmark workloads. The benchmarks were obtained from the PARSEC benchmark suite and included Blackscholes, Freqmine, Raytrace, Streamcluster, and Swaptions. Memory traces were extracted using Intel PIN instrumentation and subsequently processed through the proposed preprocessing pipeline consisting of address delta conversion, tokenization, windowing, and binary encoding. The datasets were divided into training, validation, and testing sets in the ratio of 64%, 16%, and 20%, respectively. Each model was trained for 100 epochs using the Adam optimizer, while the best-performing weights were retained based on validation accuracy.

Table 2. Experimental configuration

Parameter	Value
Processor	Intel Core i7 Processor
RAM	32 GB
GPU	NVIDIA Graphics Card (4 GB VRAM)
Operating System	Windows 64-bit Environment
Programming Language	Python
Deep Learning Framework	TensorFlow/Keras
Dataset Source	PARSEC Benchmark Suite
Benchmarks Used	Blackscholes, Freqmine, Raytrace, Streamcluster, Swaptions
Trace Collection Tool	Intel PIN
Training Split	64%
Validation Split	16%
Testing Split	20%
Epochs	100
Optimizer	Adam
Embedding Input Dimension	24
Embedding Output Dimension	8
Lookback Window	3 Memory Access Deltas
Dropout Rate	0.1
Loss Function	Binary Cross Entropy
Evaluation Metrics	Binary Accuracy, Loss, MAE, MSE

The embedding layer employed an input dimension of 24 and an output dimension of 8, corresponding to the binary-encoded representation of memory access deltas. A dropout rate of 0.1 was used to reduce overfitting and improve model generalization.

The selected configuration was designed to balance prediction performance and model compactness. The use of binary encoding highly reduced the dimensionality of the input and output representations, while the compact embedding layer and LSTM architecture supported in efficient learning of memory access patterns. The experimental setup was intentionally configured to evaluate the feasibility of deploying machine learning-based prefetching models without requiring specialized computing infrastructure. The consistent results obtained across multiple benchmark workloads demonstrate the robustness and practical applicability of the proposed method.

3.5. Algorithm - LSTM-Based Cache Prefetch Decision Process

Input: Program execution trace T
Output: Predicted memory block for prefetching
Step 1: Begin
Step 2: Collect memory access trace T using Intel PIN instrumentation.
Step 3: Extract absolute memory addresses from T.
Step 4: Convert absolute memory addresses into the address delta sequence D.
Step 5: Generate a vocabulary of address deltas.
Step 6: Tokenize and encode sequence D into token sequence E.
Step 7: Apply binary encoding to transform E into binary representation B.
Step 8: Generate lookback windows W from B using a predefined window size.
Step 9: Feed W into the trained LSTM prediction model.
Step 10: Predict the binary representation of the next memory access delta.
Step 11: Decode the predicted binary output into address delta Δ.
Step 12: Reconstruct the predicted memory address using Δ.
Step 13: Generate a prefetch request for the predicted memory block.
Step 14: Load the predicted block into the cache hierarchy.
Step 15: Repeat Steps 9–14 for subsequent memory accesses.
End

The algorithm explains the overall flow of the proposed cache prefetching process. Memory access traces collected during program execution are first transformed into address delta sequences to clear the effects of address space randomization and also reduces input complexity. The processed data is then tokenized, binary encoded, and converted into lookback windows that serve as input to the LSTM model. Based on previously obtained access patterns, the trained model predicts the most probable future memory access delta. The predicted delta is then decoded and used to generate a prefetch request, thereby enabling the required memory block to be loaded into the cache before an explicit processor request is issued. This proactive prediction mechanism contributes to reducing memory access latency and it improves overall performance.

4. Performance Analysis

4.1. Graphical Analysis

The graphical analysis of the performance of the model against each of the benchmark workloads was carried out by means of learning curves, which plot learning against experience on the basis of various metrics against training and validation data. For each of the workloads as shown in Tables 3 to Table 7, clusters of evaluation, the performance learning curve used is the binary accuracy plot against training and validation sets.

Table 3. Benchmark performance evaluation for blackscholes

Benchmark	Blackscholes		
	Round A	Round B	Round C
Binary Accuracy	0.760743022	0.760443111	0.760334205
Loss	0.700559676	0.700540071	0.700495776
MAE	0.285610497	0.285614079	0.285501947
MSE	0.180766508	0.180732206	0.180511218

The Blackscholes benchmark got a binary accuracy of 76% across all three experimental rounds with very limited variations. The loss, MAE, and MSE values also remained relatively stable, showing consistent learning behaviour of the proposed model. Compared to the other workloads, the accuracy obtained for Blackscholes was comparatively lower, which may be due to the irregular and computation-intensive memory access patterns as seen with financial modelling applications. However, the model was able to maintain stable convergence and reliable prediction performance.

Table 4. Benchmark performance evaluation for freqmine

Benchmark	Freqmine		
	Round A	Round B	Round C
Binary Accuracy	0.903242409	0.903324541	0.903242410
Loss	0.352791935	0.352666397	0.352791922

MAE	0.12301176	0.12311053	0.12301645
MSE	0.077263571	0.077152908	0.077253346

The Freqmine workload shown in Table 4 has shown good prediction performance, achieving a binary accuracy of approximately 90% across all experimental rounds.

The low loss and error show that the model was able to effectively capture the sequential memory access behaviour.

The consistency seen across multiple rounds shows the stability and repeatability of the proposed architecture. The high accuracy achieved for Freqmine is because of the repetitive and pattern-oriented nature of frequent itemset mining operations, which are well suited for sequential learning using LSTM networks.

Table 5. Benchmark performance evaluation for raytrace

Benchmark	Raytrace		
	Round A	Round B	Round C
Binary Accuracy	0.976680815	0.976635018	0.976080222
Loss	0.061656073	0.061653045	0.061156370
MAE	0.039756637	0.039713365	0.039253346
MSE	0.017257117	0.017218451	0.017357536

In comparison to all the benchmark workloads, Raytrace, shown in Table 5, has shown the highest prediction accuracy, achieving nearly 97% binary accuracy consistently across all three rounds. This very low loss, MAE, and MSE values indicate efficient learning and strong prediction capability of the proposed model.

The highly repetitive and spatially correlated memory access behaviour, which is observed in rendering applications, should have contributed to the improved performance. The results stand witness to the fact that the proposed binary-encoded LSTM model is effective in learning structured and predictable access patterns from large sequential datasets.

Table 6. Benchmark performance evaluation for streamcluster

Benchmark	Streamcluster		
	Round A	Round B	Round C
Binary Accuracy	0.850029171	0.850278816	0.850829281
Loss	0.425889045	0.425369036	0.425789524
MAE	0.219201297	0.219451145	0.219901977
MSE	0.121796206	0.121646398	0.121496602

The Streamcluster benchmark, as seen in Table 6, has obtained a binary accuracy of approximately 85% in the experimental rounds, with only a few fluctuations in the performance. The model maintained stable loss and error values throughout the experiments, that demonstrates reliable convergence behaviour.

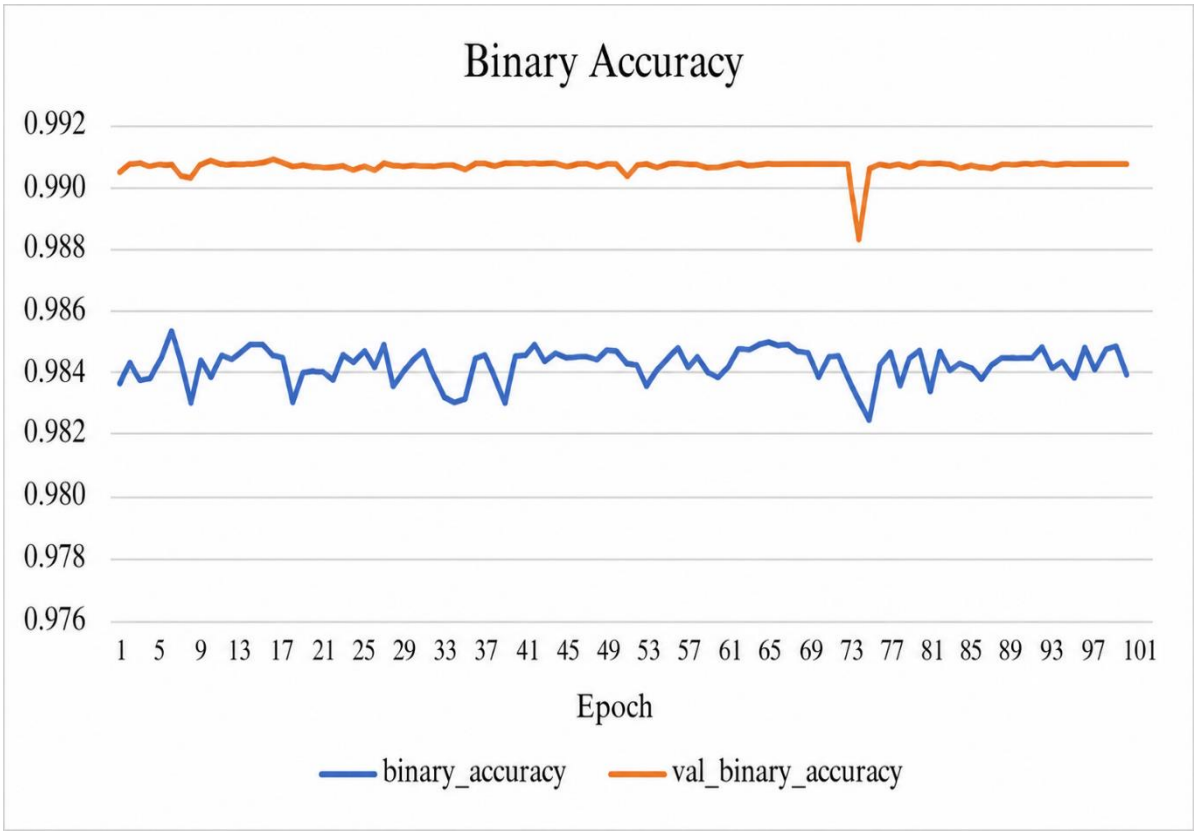
The lower accuracy compared to workloads such as Raytrace and Swaptions is because to the dynamic and less predictable memory access associated with online clustering applications. However, the proposed model has learnt meaningful sequential dependencies and maintained consistent prediction performance.

Table 7. Benchmark performance evaluation for swaptions

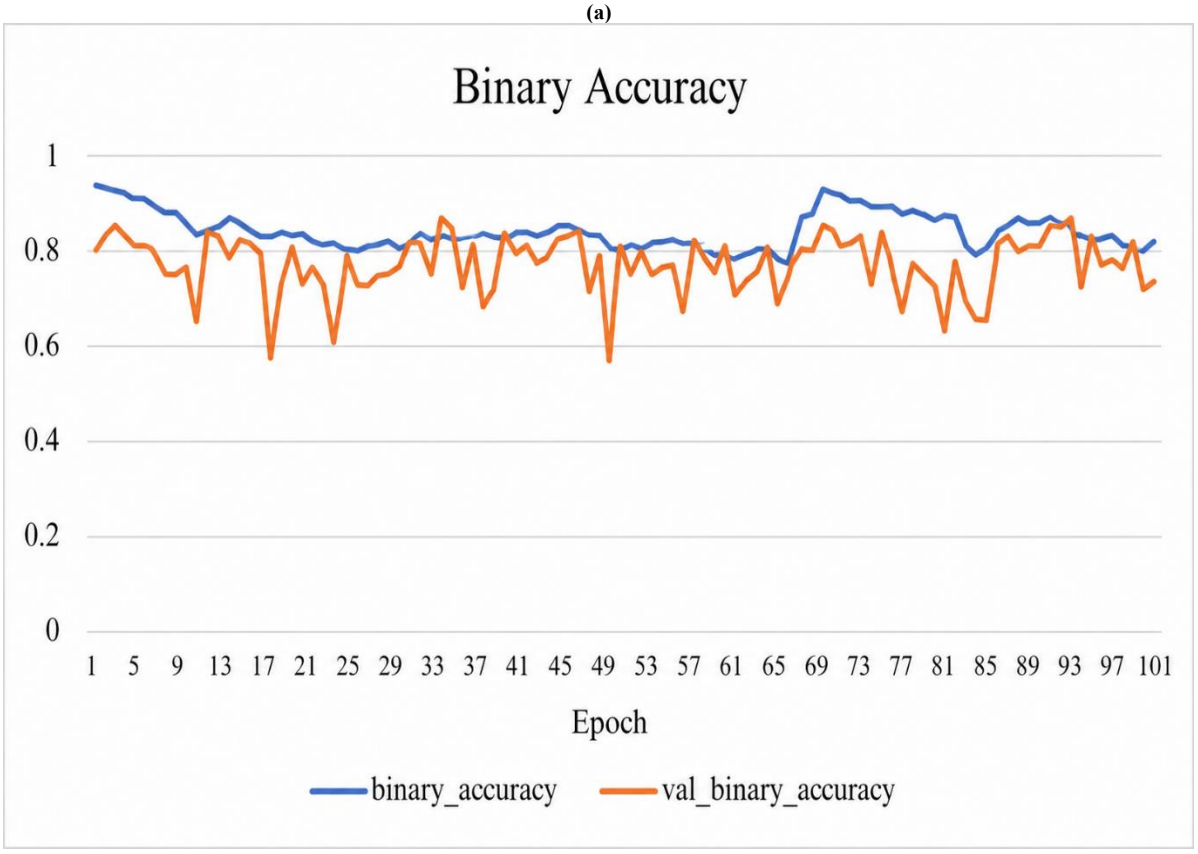
Benchmark	Swaptions		
	Round A	Round B	Round C
Binary Accuracy	0.953160703	0.953970502	0.953260202
Loss	0.172209486	0.172109658	0.172306482
MAE	0.072957084	0.072980365	0.072451081
MSE	0.041574638	0.041542151	0.041684637

The Swaptions workload, as shown in Table 7, has got a binary accuracy of 95% across all the experiments. The low loss, MAE, and MSE values indicate effective prediction capability and stable model convergence. The results show that the sequential memory access patterns present in the workload were efficiently captured by the LSTM architecture.

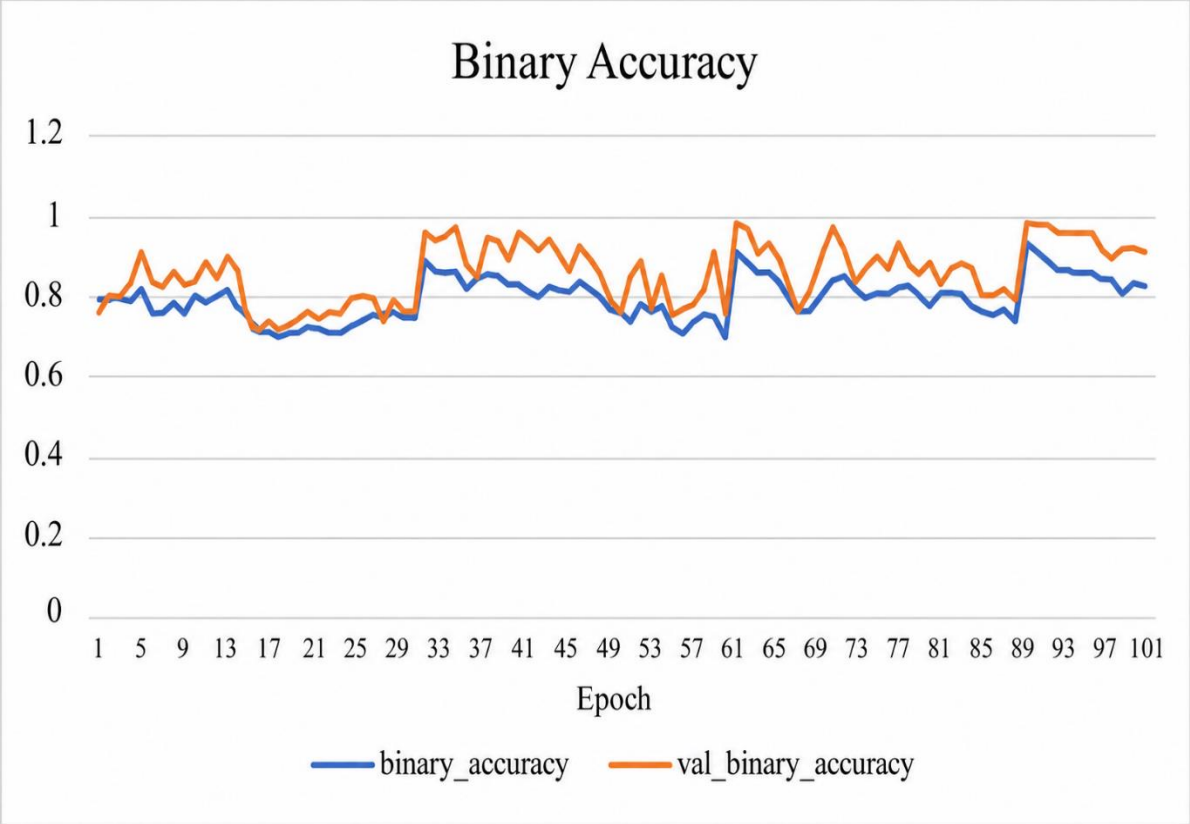
Additionally, the minimal variation across repeated rounds exhibits the robustness and repeatability of the proposed approach, thereby making it suitable for practical cache prefetching applications involving structured computational work.


Binary Accuracy
0.992
0.990
0.988
0.986
0.984
0.982
0.980
0.978
0.976
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101
Epoch
binary_accuracy
val_binary_accuracy

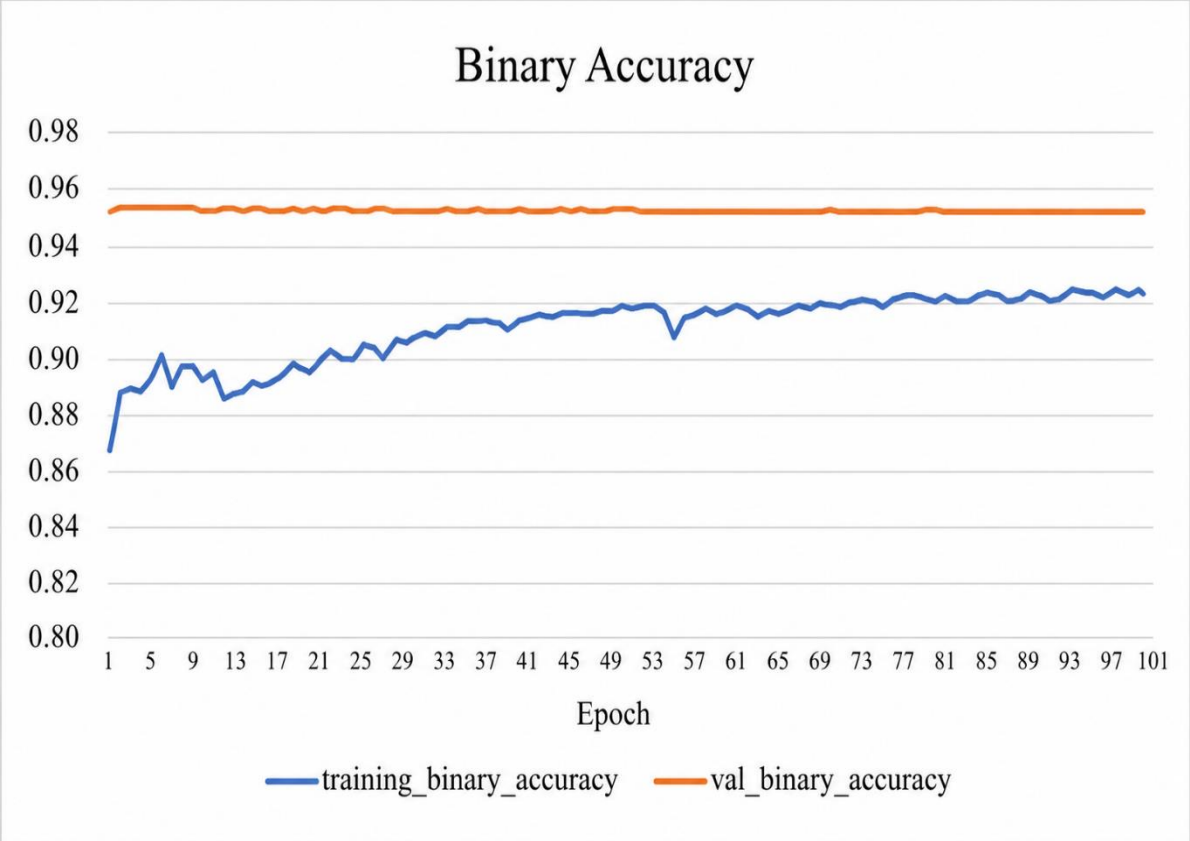
(a)


Binary Accuracy
1
0.8
0.6
0.4
0.2
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101
Epoch
binary_accuracy
val_binary_accuracy

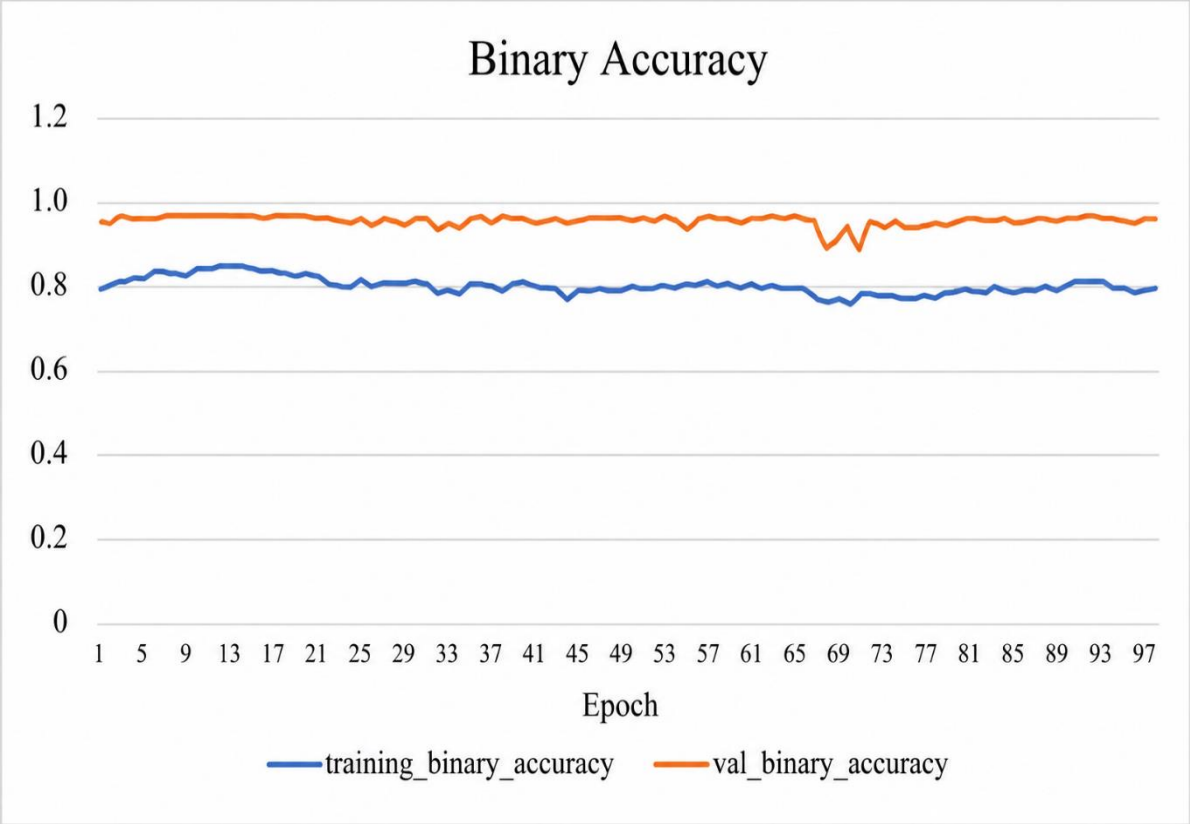
(b)


Binary Accuracy
1.2
1
0.8
0.6
0.4
0.2
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101
Epoch
binary_accuracy
val_binary_accuracy

(c)


Binary Accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101
Epoch
training_binary_accuracy
val_binary_accuracy

(d)



(e)

Fig. 3 (a)-(e) Training and validation binary accuracy curves for the Blackscholes workload across 100 training epochs

4.2. Analytical Inference

The models are trained on each workload for 100 epochs for the purpose of standardised experimentation and inference. However, it is observed from Figures 3 (a) to 3(e) that the models perform over 95% of their learning within the first epoch for most workloads.

In fact, upon examining the training logs, it is seen that for 4 out of 5 benchmark workloads, the best weights are obtained within the first 15 epochs; and for 2 of these benchmarks, the best weights are obtained within the first 3 epochs.

For some workloads, models almost flatline after the first epoch, achieving binary accuracies in the range of 95-99% against the validation set within the first epoch. This is true for freqmine, swaptions and raytrace, which achieved validation binary accuracies of 99.03%, 95.37% and 95.33% within the first epoch. This effect is noted to be fairly independent of trace size since raytrace trained on 56.2 MB while freqmine trained on 1.86 GB of trace data. Additionally, this generalised architecture model takes up only 850 KB of space in memory- a rather negligible amount of space in present-day computing systems. Since the model has been tested and found to be effective on a variety of benchmark workloads in domains ranging from financial analysis to rendering, it can be concluded that it is fairly generalisable. The ability to reach optimal weights in few epochs, coupled with a small memory footprint, makes the general model suitable for deployment as a prefetcher in live systems. The model can be pre-trained before initial installation and retrained for a few epochs at spaced-out intervals whenever a predefined minimum accuracy threshold is crossed.

4.3. Comparative Analysis with Existing Prefetchers

Table 8 provides a comprehensive view on the comparison of the proposed model with respect to the existing players in the domain. The idea of claiming the model to be effectively better than the other counterparts is based on its lightweight, converging and deployable capability.

Table 8. Benchmark performance evaluation for swaptions

Prefetching Method	Technique Used	Accuracy	Model Size	Hardware Feasibility	Adapt ability
Traditional Stream Prefetcher [7, 21, 12]	Rule-based	Moderate	Very Low	High	Low

LSTM Composite Prefetcher [13]	LSTM + Stream	High	Large	Moderate	Moderate
MemMAP [23]	Meta-LSTM	High	Compact	Moderate	High
RAOP [22]	RNN-Augmented	High	Moderate	Moderate	High
Proposed Method	Binary-Encoded LSTM	High (up to 99%)	Very Compact (850 KB)	High	High

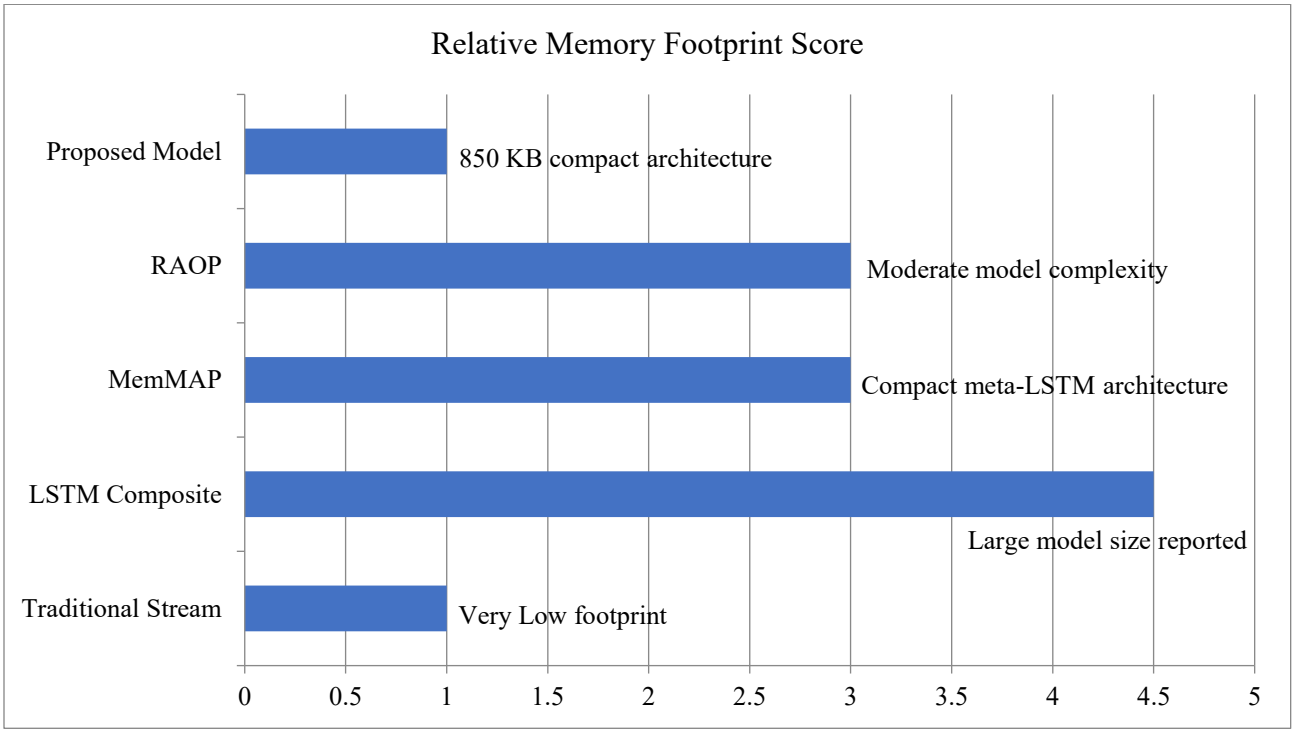


Fig. 4 Memory footprint comparison

The proposed model is light weighted due to the use of binary encoding and dimensionality reduction techniques. Usually, separate output units are assigned to each individual memory class, but the proposed model represents the address delta classes in binary form, and hence it reduces the number of input and output neurons. This reduction (log2(n)) makes the model to maintain the predictive capability as well as reduce the memory consumption and computational overhead. The commonly observed parameter growth in other classification-based neural prefetchers is also avoided in the proposed model. Combining the binary encoding and optimized embeddings supports in achieving a memory footprint of approximately 850 KB, thereby making the proposed model completely suitable for deployment in computing environments where memory is a huge constraint. The fast convergence seen in the proposed model is because of the sequential learning capability of the LSTM model, and so they are highly effective for memory access patterns. The highlight in this scenario is that the LSTM is able to efficiently learn recurring access relationship within a shorter span of time and epochs. The proposed model uses address data representation instead of an absolute memory address, which simplifies the learning phase by reducing the randomness. Hence, combined with the binary encoding and reduced feature dimensionality, the overall feature space becomes significantly more structured and computationally manageable, thereby accelerating convergence and enabling the model to attain high prediction accuracy within only a few training epochs. Furthermore, the model shows fast convergence characteristics, with optimal weights frequently obtained within a limited number of epochs for multiple benchmark workloads. This property reduces retraining overhead and makes the periodic model updates very practical in dynamic runtime environments. Hence, proves the deployable capability of the propose model is very efficient. On the other hand, the proposed model achieves competitive predictive performance while occupying only a small fraction of system memory. This reduced footprint minimizes the hardware overhead and improves feasibility for integration into practical cache memory and embedded systems. The comparative graph, as shown in Figure 4, is based on a qualitative evaluation of different prefetching approaches. Since many of the studies do not report uniform numerical values for entities such as memory footprint, implementation complexity, and hardware feasibility, a relative scoring approach was adopted for comparison. The values assigned in

the chart were derived by carefully analysing the characteristics mentioned in the respective works, including model size, architectural complexity, computational requirements, and ease of deployment. Lower values represent models with smaller memory requirements and simpler architectures, while higher values show comparatively larger and more resource-intensive implementations. The proposed model was assigned a lower footprint score because of its compact binary encoding approach, reduced dimensionality through $\log_2(n)$ compression, and lightweight embedding structure, which contribute to a significantly smaller memory footprint of approximately 850 KB. This comparison gives a practical way to highlight the relative efficiency and deployability of the proposed approach against existing prefetching techniques.

4.4. Limitations and Scalability

The proposed binary-encoded LSTM prefetching model showed encouraging prediction precision and a small memory usage across several benchmark tasks. The model's capacity to understand temporal memory access patterns, along with dimensionality reduction through binary encoding, greatly enhanced its performance. Moreover, the model showed swift convergence throughout training and upheld steady performance over multiple experimental iterations, demonstrating its ability to effectively grasp linear dependencies in memory access patterns. In spite of these promising outcomes, various limitations need to be recognized. Initially, the experimental assessment was performed solely with workloads from the PARSEC benchmark suite. While these benchmarks cover various application domains, they might not completely reflect the behavior of every real-world workload found in contemporary computing settings. Further assessment with enterprise, cloud, database, and operating system workloads would enhance the evaluation of the model's generalizability. Secondly, the existing method depends on offline training, in which the model is trained before deployment, utilizing memory traces that were collected beforehand. Although this method simplifies experimentation, dynamic runtime environments might need online or incremental learning abilities to consistently adjust to evolving access patterns.

Additionally, the proposed work emphasizes prediction accuracy and model efficiency instead of direct hardware execution. While the suggested architecture greatly lowers memory needs in comparison to numerous current machine learning-based prefetchers, real-world implementation would necessitate collaboration with cache controllers and memory management systems. This integration brings forth extra factors to consider, such as inference latency, memory bandwidth impact, and hardware resource usage. The effects of these factors were not assessed in the context of this work. Likewise, power usage and energy efficiency were not specifically assessed, even though these factors are crucial for implementation in embedded and extensive computing systems. From a scalability standpoint, the suggested architecture presents multiple benefits. Applying binary encoding lessens the dimensionality of the prediction issue, thus constraining the increase in model complexity as the size of the data expands.

The small memory usage and fairly quick convergence properties also render the model appropriate for regular retraining and adjustment to changing workloads. Future studies may investigate hybrid methods that integrate the suggested LSTM-based predictor with hardware-assisted prefetching systems, reinforcement learning strategies, or efficient Transformer designs to enhance adaptability and prediction effectiveness. Assessing more extensive and varied workload collections, together with prototype hardware implementations, would offer further understanding of the scalability and real-world applicability of the suggested method.

5. Conclusion and Future Work

The purpose of this research was to prototype a system that is capable of learning complex memory access patterns with a minimal memory footprint. This goal was achieved by means of an LSTM-based prefetcher that achieved binary accuracies as high as 99% under certain benchmark workloads while occupying only 850 KB in memory. On the average present-day computer, this amounts to about 0.085% of main memory, thereby efficiently trading a fairly negligible amount of computation and memory for reduced memory latency. The results obtained prove that LSTM-based prefetchers are now a viable practical solution to the memory wall problem and can be incorporated into live systems to experience significant gains. This work is highly extensible and can be incorporated into live systems through modified memory management units in custom OS distributions, or even through specialised architectures on real hardware systems in the future.

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Ibrahim Abdullahi, Suki Arif, and Suhaidi Hassan, “Survey on Caching Approaches in Information Centric Networking,” *Journal of Network and Computer Applications*, vol. 56, pp. 48-59, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[2] Grant Ayers et al., “Classifying Memory Access Patterns for Prefetching,” *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Association for Computing Machinery, New York, United States, pp. 513-526, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[3] Giuliano Casale, and Nicolas Gast, "Performance Analysis Methods for List-based Caches with Non-Uniform Access," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 651-664, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[4] Christian Bienia, "*Benchmarking Modern Multiprocessors*," Princeton University, 2010. [Google Scholar] [Publisher Link]

[5] Vladyslav Fedchenko, Giovanni Neglia, and Bruno Ribeiro, "Feedforward Neural Networks for Caching: N Enough or Too Much?," *ACM Sigmetrics Performance Evaluation Review*, vol. 46, no. 3, pp. 139-142, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[6] Yaru Fu et al., "Joint Content Caching, Recommendation, and Transmission Optimization for Next Generation Multiple Access Networks," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 5, pp. 1600-1614, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[7] Milad Hashemi et al., "Learning Memory Access Patterns," *Proceedings of the 35th International Conference on Machine Learning*, PMLR, pp. 1-10, 2018. [Google Scholar] [Publisher Link]

[8] John L. Hennessy, and David A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed., Elsevier, 2011. [Google Scholar] [Publisher Link]

[9] Minwoo Jang et al., "Defending against Flush+Reload Attack with DRAM Cache by Bypassing Shared SRAM Cache," *IEEE Access*, vol. 8, pp. 179837-179844, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[10] Eleftherios Lampiris, and Petros Elia, "Full Coded Caching Gains for Cache-Less users," *IEEE Transactions on Information Theory*, vol. 66, no. 12, pp. 7635-7651, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[11] Jung Kyu Park, Yunjung Seo, and Jaeho Kim, "A Flash-based SSD Cache Management Scheme for High Performance Home Cloud Storage," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 3, pp. 418-425, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[12] Leeor Peled, Uri Weiser, and Yoav Etsion, "A Neural Network Prefetcher for Arbitrary Memory Access Patterns," *ACM Transactions on Architecture and Code Optimization*, vol. 16, no. 4, pp. 1-27, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[13] Joseph Rogers, "Effects of an LSTM Composite Prefetcher," 2019. [Google Scholar]

[14] Zhan Shi et al., "Applying Deep Learning to the Cache Replacement Problem," *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, Association for Computing Machinery, New York, United States, pp. 413-425, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[15] Ajitesh Srivastava et al., "Predicting Memory Accesses: The Road to Compact ML-Driven Prefetcher," *Proceedings of the International Symposium on Memory Systems*, Association for Computing Machinery, New York, United States, pp. 461-470, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[16] Yujuan Tan et al., "Improving the Performance of Deduplication-based Storage Cache via Content-Driven Cache Management Methods," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 214-228, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[17] Dharmesh Tarapore et al., "Observing the Invisible: Live Cache Inspection for High-Performance Embedded Systems," *IEEE Transactions on Computers*, vol. 71, no. 3, pp. 559-572, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] Wm. A. Wulf, and Sally A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, pp. 20-24, 1995. [CrossRef] [Google Scholar] [Publisher Link]

[19] Huijing Yang et al., "RL-CoPref: A Reinforcement Learning-based Coordinated Prefetching Controller for Multiple Prefetchers," *The Journal of Supercomputing*, vol. 80, no. 9, pp. 13001-13026, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[20] Hui-Jing Yang et al., "A Prefetch-Adaptive Intelligent Cache Replacement Policy based on Machine Learning," *Journal of Computer Science and Technology*, vol. 38, no. 2, pp. 391-404, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[21] Yuan Zeng, and Xiaochen Guo, "Long Short Term Memory based Hardware Prefetcher: A Case Study," *Proceedings of the International Symposium on Memory Systems*,Association for Computing Machinery, New York, United States, pp. 305-311, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[22] Pengmiao Zhang et al., "RAOP: Recurrent Neural Network Augmented Offset Prefetcher," *Proceedings of the International Symposium on Memory Systems*, Association for Computing Machinery, New York, United States, pp. 352-362, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[23] Pengmiao Zhang et al., "C-MemMAP: Clustering-Driven Compact, Adaptable, and Generalizable Meta-LSTM Models for Memory Access Prediction," *International Journal of Data Science and Analytics*, vol. 13, no. 1, pp. 3-16, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[24] Pengmiao Zhang et al., "TransforMAP: Transformer for Memory Access Prediction," *arXiv preprint*, pp. 1-5, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[25] Rui Zhang et al., "A Comprehensive Framework for Analysis of Time-Dependent Performance-Reliability Degradation of SRAM Cache Memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 5, pp. 857-870, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[26] Tiankui Zhang et al., "Cache Space Efficient Caching Scheme for Content-Centric Mobile Ad Hoc Networks," *IEEE Systems Journal*, vol. 13, no. 1, pp. 530-541, 2019. [CrossRef] [Google Scholar] [Publisher Link]