RAOP: Recurrent Neural Network Augmented Offset Prefetcher

Pengmiao Zhang
University of Southern California
Los Angeles, California, USA
pengmiao@usc.edu

Ajitesh Srivastava
University of Southern California
Los Angeles, California, USA
ajiteshs@usc.edu

Benjamin Brooks
University of Southern California
Los Angeles, California, USA
bjbrooks@usc.edu

Rajgopal Kannan
US Army Research Lab-West
USA
rajgopal.kannan.civ@mail.mil

Viktor K. Prasanna
University of Southern California
Los Angeles, California, USA
prasanna@usc.edu

ABSTRACT

The rapid development of Big Data coupled with slowing down of Moore's law has made the memory performance a bottleneck in the von Neumann architecture. Machine learning has the potential to provide opportunities to address the memory performance issues, specifically through data access prediction. While recent works focusing on the prediction of memory accesses have used recurrent neural networks (RNN), there is a lack of a framework utilizing such prediction in a prefetcher. This paper introduces the RNN Augmented Offset Prefetcher (RAOP) framework, which consists of two parts: an RNN-based predictor and an offset prefetching module. By leveraging the RNN predicted access as a temporal reference, RAOP improves prefetching performance by executing offset prefetching for both the current address and the RNN predicted address. We implement the RNN in the predictor with a compressed long short-term memory (LSTM) model and demonstrate the effect of augmenting an RNN predictor to a simple next-line prefetcher in the prefetching module results in 3.22x, 4.2x, and 15.6% improvement in prefetch accuracy, coverage, and speedup. We further implement a best-offset prefetcher (BOP) in RAOP and compare it to several state-of-the-art prefetchers. Results show that RAOP achieves a mean 4.05% speedup by prefetching in last level cache, outperforming state-of-the-art prefetchers. By augmenting an RNN predictor to BOP, RAOP results in 6.5x, 9.2x, and 12.8% improvement in prefetch accuracy, coverage, and speedup.

CCS CONCEPTS

• **Computer systems organization** → **Processors and memory architectures**; **Neural networks**; • **Information systems** → Data mining.

KEYWORDS

machine learning, RNN, offset prefetcher, augmentation effect

MEMSYS 2020, September 28-October 1, 2020, Washington, DC, USA

ACM ISBN 978-1-4503-8899-3/20/09…$15.00
https://doi.org/10.1145/3422575.3422807

ACM Reference Format:
Pengmiao Zhang, Ajitesh Srivastava, Benjamin Brooks, Rajgopal Kannan, and Viktor K. Prasanna. 2020. RAOP: Recurrent Neural Network Augmented Offset Prefetcher. In *The International Symposium on Memory Systems (MEMSYS 2020), September 28-October 1, 2020, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3422575.3422807

1 INTRODUCTION

Memory performance is becoming the bottleneck of computation due to the increasing gap between CPU and memory speed [2], especially with the rapid development of AI accelerators such as GPUs and TPUs. This problem is partly addressed by the development of larger and deeper cache hierarchies in the past decades. New memory technologies like Intelligent RAM, 3D-stacking, and Non-volatile memory are in active research, but they focus on the memory hardware with a compromise of high cost and power consumption.

Prefetching techniques are widely used to hide memory latency and improve instructions per cycle (IPC). It looks at a pattern of memory accesses and uses the past information to forecast the near future access pattern so as to start fetching the data before the miss occurs [24]. A prefetching process is a form of speculation that aims to predict the next one or several instructions or data addresses. Simple hardware prefetchers exploit obvious memory access patterns, such as the adjacent spatial locality and constant stride. Offset prefetching has been proposed recently as a dynamic stride approach. Sandbox method is introduced by Pugsley et al. in [17] as the first offset prefetching method, which aims to find offsets that improve prefetching accuracy. Offset prefetching is an evolution of both next-line prefetching and stride prefetching. When a request happens for a cache block X, offset prefetcher prefetches the cache line $X + k$, where k is a dynamic offset.

Recently, machine learning-based prefetching methods have gained increasing attention [22]. One way of using machine learning (ML) to address the prefetching problem is using ML to help an existing prefetcher, such as deciding the configuration of a prefetcher or a combination of several different prefetchers [11, 18]. Another branch of research is using machine learning directly to learn and predict the memory access pattern [7, 22]. In this case, the prefetching problem can be reduced to the sequence prediction problem. Due to the complexity of the memory access pattern, this type of method requires more powerful learning models.

RNN (recurrent neural network) has emerged as a powerful tool for learning from time-series data. In particular, LSTM (Long Short-Term Memory) [6],a popular RNN variant, has shown extraordinary performance in sequence problems, such as natural language processing [13] and speech recognition [8]. Inspired by the capacity on sequence processing of RNN, some efforts have applied RNNs to memory access prediction [22, 23]. While these research focus mainly on the accuracy of memory access prediction, little effort has been made in constructing an appropriate framework to implement such a model-based prefetcher.

In this paper, we propose an RNN Augmented Offset prefetcher (RAOP) that makes use of both the RNN predicting accuracy and the principle of spatial locality. By performing offset prefetching for both the current and the predicted memory addresses, the prefetching performance is significantly improved. We implemented this prefetcher through the ChampSim [3] simulator and SPEC CUP 2017 traces and conducted experiments to evaluate our proposed approach compared to popular state-of-art prefetchers.

To summarize, our contributions are as follows:

- We propose a framework of implementing the RNN model that can predict next memory access in the microarchitecture, which serves as a part of the predictor for RAOP.
- We propose a spatio-temporal prefetching approach that makes use of both the prediction of the RNN model and the spatial offset pattern in memory.
- We present the RNN augmentation effect through an ablation study. A simple next-line prefetcher is implemented and the performance is evaluated under different configurations.
- We present experiments on the proposed prefetcher and compare it with several state-of-the-art prefetchers. RAOP achieves a mean 4.05% speedup by prefetching in last level cache, outperforming state-of-the-art prefetchers. Over BOP, RAOP results in 6.5x, 9.2x, and 12.8% improvement in prefetch accuracy, coverage, and speedup.

2 RELATED WORK

Offset prefetchers have been studied recently in the literature [12, 19]. Best-Offset prefetching approach proposed in [12] not only provides prefetch offset but also takes into account prefetch timeliness. [19] presents a multi-lookahead offset prefetcher that considers both miss coverage and timeliness when issuing prefetch requests through assigning scores to the offsets and selects the highest scoring offset for issuing prefetch requests. While also based on the offset prefetcher, this paper focuses on the improvement of memory performance due to the assistance of machine learning models.

The application of machine learning algorithms on memory prefetching has been studied in several prior works. Liao et at. [11] apply machine learning algorithms such as nearest neighbor, support vector machine, and decision tree on the optimization of prefetching configurations for a data center. The authors in [18] presents an ML-based classification method to optimize the subset of prefetchers for a set of applications and aims to maximize the effectiveness of hardware prefetchers. In these works, machine learning approaches serve as assistants of the existing prefetcher, which is different from our prefetching approach that directly provides the memory address from a machine learning model.

In [1], the authors illustrate the excellent performance of the LSTM neural network to predict given memory access patterns. They trained an LSTM model with previously designed micro-benchmarks that are interleaving of multiple patterns and it shows high accuracy of up to 95%. The work in [7] analyzed the pattern of memory access and applied k-means clustering and LSTM to the prediction of accesses deltas (jumps in memory access). They demonstrate that LSTM-based model achieves high precision compared to traditional prefetchers. Some other works [14, 15, 26] also demonstrate the effectiveness of LSTM in memory access prediction. However, these work focus mainly on the access prediction accuracy rather than the complete prefetcher performance. In this paper, we show how the last level cache (LLC) performs when an RNN-based prefetcher is utilized, and propose a spatio-temporal prefetching approach making use of the RNN access prediction.

Work [22] contributes to the implementation of RNN as a prefetcher by reducing the size of an LSTM model, which makes the RNN-based approach more practical. The authors propose a compressed LSTM approach for accurate memory access prediction. The LSTM model in this work is trained from a sequence of accesses deltas of PARSEC benchmark and predict the next access delta. According to the observation of delta distribution, the authors encode both the input and output vocabulary into binary values. In this way, they achieved a $n/\log n$ ratio of compression with negligible accuracy compromise. While this work illustrates a promising approach of applying LSTM as a prefetcher, they have not tested their approach on a full microarchitecture simulator. In this paper We apply the same compression approach in [22] while using the SPEC CPU 2017 benchmark. More importantly, we analyze the prefetching performance through the simulation of ChampSim, a trace-based simulator that models a modern an out-of-order core and comparing the performance of an RNN-based prefetcher, the proposed prefetcher, and several state-of-the-art prefetchers.

3 APPROACH

We use an RNN model as the predictor of a prefetcher. Through learning from the sequence of access deltas, the jump of adjacent addresses, the model can infer the next delta based on the previous and current accesses information. We apply a compression technique that can largely reduce the model size by encoding the vocabulary of deltas. With the predictor, we can execute offset prefetching on both the current address and the predicted address. While a simple offset prefetcher can work only when the request happens, the RNN augmented offset prefetcher can work for a second time before the next memory access is requested.

3.1 Framework

Our prefetcher is a combination of two parts: a predictor and a prefetching module. Besides, an extra delta buffer is inserted between the CPU Core and the memory management unit (MMU). The framework is shown in figure 1. The predictor part is an encapsulation of RNN models and some necessary helper functions. Considering the advancement of tensor-based algorithms, e.g. machine learning model training, translated physical addresses can

lose the logic connection inside a tensor, which makes the prediction a harder task. Therefore, we place the predictor in the virtual address (VA) space. The work in [25] has shown the feasibility of a prefetcher working on virtual space with address translation support.

The RNN models are in the structure of compressed LSTM. One model is trained for each of the applications in the SPEC 2017 benchmark that we use. While working, the RNN model for the running application receives the deltas sequence from the delta buffer and predicts the next delta. The predictor will sum the delta and current address to calculate the predicted next virtual address. The number of models can be reduced by clustering the traces and train one model for several applications. However, we use the golden line models to study the prefetching performance in this paper to avoid the prefetching compromise caused by model variation.

After the computation of prediction, the predicted virtual address is sent to the MMU for translation. This address will first try to be translated through lookaside buffer (TLB). If fail, the page table base register (PTBR) will point to the page table entries that the VA should look for and be translated there. After translation, the mapping predicted physical address will be available.

The prefetching module works on the physical address (PA) space. Our prefetcher works when the last level cache (LLC) misses and aims to prefetch the near future cache lines from the main memory to the LLC. Therefore, when the LLC miss happens, the prefetching module will receive the RNN predicted next physical address. This address will help the offset prefetcher in the prefetching module to acquire spatial nearby cache lines for both the current address and the predicted address. By fetching the cache lines before the data is needed, the prefetcher can help avoid cache miss.

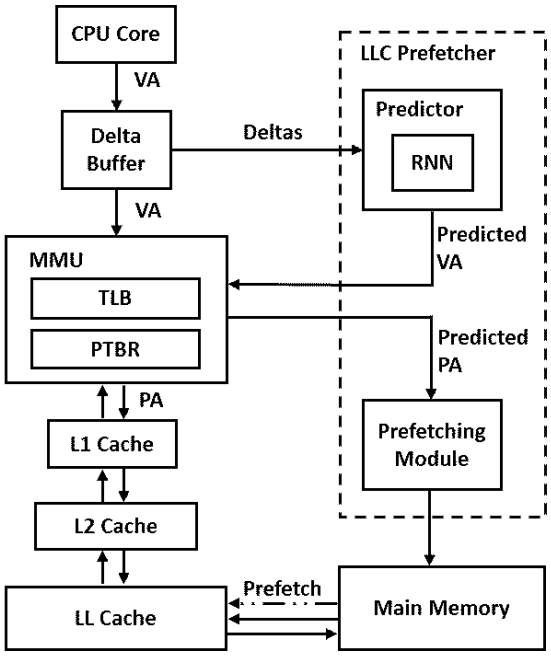

Figure 1: Framework for RNN augmented offset prefetcher

3.2 Recurrent Neural Network

Recurrent neural networks exhibit temporal dynamic behavior by storing sequential information in their internal states. By performing the same function recurrently for inputs of several time steps, RNN learns history information without the necessity of expansion of network cell numbers. By assuming the dependence between the current input and previous inputs, RNN performs better in sequence processing than basic neural networks that consider the time steps as dimensions without time-series information. However, for long sequence processing problems, basic RNN suffers from gradient vanishing and exploding problems. To overcome the shortcomings of basic RNN, some modified version has emerged, such as LSTM (Long Short-Term Memory) and GRU(Gated Recurrent Unit) [4].

In this paper, we use the LSTM network as the main structure of our prefetching predictor. An LSTM cell is composed of an input gate $\mathbf{i}^{(t)}$, a block input gate $\mathbf{z}^{(t)}$, a forget gate $\mathbf{f}^{(t)}$, an output gate $\mathbf{o}^{(t)}$, an memory cell $\mathbf{c}^{(t)}$ and an output $\mathbf{y}^{(t)}$. The operation of each set of gates of the layer is given by the following set of equations:

$$
\begin{aligned}
\mathbf{i}^{(t)} &= \sigma\left(\mathbf{W_i x^{(t)}} + \mathbf{R_i y^{(t-1)}} + \mathbf{p_i} \odot \mathbf{c^{(t-1)}} + \mathbf{b_i}\right) \\
\mathbf{z}^{(t)} &= \tanh\left(\mathbf{W_z x^{(t)}} + \mathbf{R_z y^{(t-1)}} + \mathbf{b_z}\right) \\
\mathbf{f}^{(t)} &= \sigma\left(\mathbf{W_f x^{(t)}} + \mathbf{R_f y^{(t-1)}} + \mathbf{p_f} \odot \mathbf{c^{(t-1)}} + \mathbf{b_f}\right) \\
\mathbf{o}^{(t)} &= \sigma\left(\mathbf{W_o x^{(t)}} + \mathbf{R_o y^{(t-1)}} + \mathbf{p_o} \odot \mathbf{c^{(t)}} + \mathbf{b_o}\right) \\
\mathbf{c}^{(t)} &= \mathbf{i^{(t)}} \odot \mathbf{z^{(t)}} + \mathbf{f^{(t)}} \odot \mathbf{c^{(t-1)}} \\
\mathbf{y}^{(t)} &= \mathbf{o^{(t)}} \odot \tanh\left(\mathbf{c^{(t)}}\right)
\end{aligned}
\tag{1}
$$

where $\mathbf{x^{(t)}}$ is the input vector at time step $\mathbf{t}$; $\mathbf{y^{(t-1)}}$ is the output of the previous time step; $\mathbf{c^{(t-1)}}$ is the memory state of the previous time step; $\mathbf{W_i}$, $\mathbf{W_z}$, $\mathbf{W_f}$, $\mathbf{W_o}$ are input weights for the input gate, block input gate, forget gate and output gate, respectively; $\mathbf{b_i}$, $\mathbf{b_z}$, $\mathbf{b_f}$, $\mathbf{b_o}$ are input bias for the four gates respectively; $\mathbf{R_i}$, $\mathbf{R_z}$, $\mathbf{R_f}$, $\mathbf{R_o}$ are recurrent bias for the four gates respectively; $\mathbf{p_i}$, $\mathbf{p_z}$, $\mathbf{p_f}$, $\mathbf{p_o}$ are peepholes that connects directly from the memory cell to the gates;σ and tanh are sigmoid and hyperbolic tangent functions that serve as nonlinear activation functions. $\odot$ is the operation of Hadamard vector multiplication.

3.3 Compressed LSTM for Access Prediction

We train the LSTM model with the deltas of memory access traces and try to predict the following delta given a piece of previous deltas sequence. The delta prediction problem can be formed as a classification problem rather than a regression problem because the delta values are not consecutive. This problem has similarities with natural language processing that the vocabulary (deltas in our problem) is large but the high frequently appeared words are in a relatively small range. Considering this distribution, we can compress both the output and input dimension by encoding the vocabulary as binary so that the model can achieve a nearly $n/\log n$ ratio of compression. In our model, vocabulary is encoded into a 16-bit binary value, which means the dimension is only 16 for input and output. A compact LSTM model makes not only the size smaller but the inference computation faster.

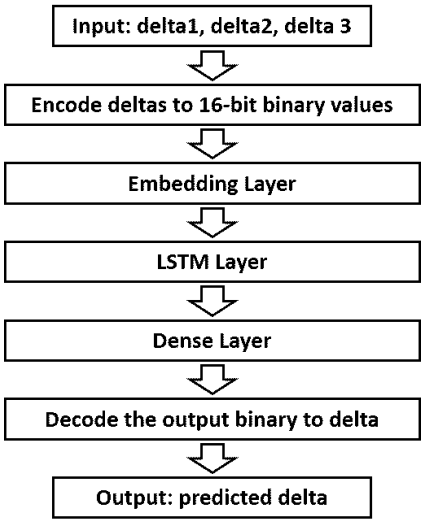

Figure 2: Compressed LSTM for access prediction

3.4 Offset Prefetching Module

Offset prefetchers prefetch the address $X+D$ if X and $X+D$ lie in the same memory page, where X is the base address and D is the offset. In this paper, we implemented a best-offset (BO) prefetcher in the prefetching module for LLC prefetching. BO algorithm [12] tries to find the optimal prefetching offset by testing several different offsets. A recent request (RR) table is constructed in a BO prefetcher to store the recent base address. An offset list and score table are constructed to learn and predict the best offset D.

The workflow of BO prefetching is shown in figure 3. The RR table updates when LLC reads data effectively from the main memory, that is when cache miss happens or prefetch hits. 63 different offsets are stored as a list d_i in the offset list and are initialized as 0. When LLC cache miss happens, the address $X - d_i$ will be looked up in the RR table and check whether this address was recently issued or not (hit/miss). The result will be fed back to the offset list table to update the scores for d_i. If there is a hit in the RR table, the score for d_i will be incremented by one. The process of testing all the d_i is called a round. A learning phase is a number of rounds and a phase completes when either the value of score ford_i or the number of rounds reaches their limits respectively. After a phase of learning, the d_i with the highest score is considered as the best offset D and the new learning phase starts with all scores are reset.

3.5 RNN Augmented Offset Prefetching

Considering only the predicted one cache line is not sufficient for a strong prefetcher. First, there is a chance that the prediction is wrong since the prediction accuracy cannot be 100%, then the prefetcher will fetch a useless cache line. Second, even though the prediction is correct, RNN is still a probability model that addresses with high appearing frequency will be predicted with higher possibility, which at the same time indicates the address should already in the cache. In this case, the prefetcher loses an opportunity to bring useful data into the cache. Considering the weakness of prefetching only the predicted cache line, we propose an RNN Augmented

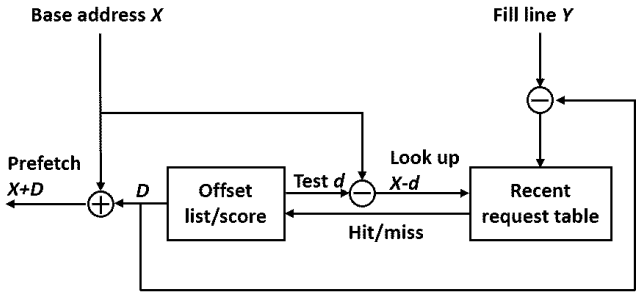

Figure 3: Best-offset prefetching workflow

Offset Prefetching approach (RAOP), which makes use of both temporal locality and spatial locality to improve the prefetching performance.

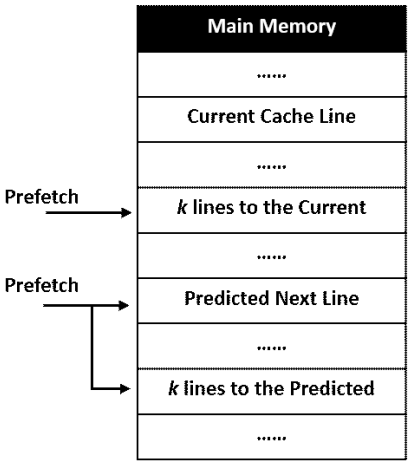

Figure 4: RNN augmented offset prefetching

Besides the temporal address reference provided by the RNN-based predictor, we propose to also prefetch M cache lines at a spatial distance of k cache lines to the current line and N cache lines at a distance of k cache lines to the RNN predicted line. Here k is the offset predicted by the offset prefetching module in RAOP framework, which serves as a spatial reference. While an increasing number of prefetching depth typically leads to higher coverage, naively increasing the depths does not always improve the overall system performance [9]. This is because the predicted stream and the actual stream will eventually diverge, which will cause a decrease of accuracy and a number of redundant prefetches.In this paper, we select the coefficient as $N = 1$ and $M = 1$, the prefetching location is shown in figure 4. The calculation of prefetching cache line addresses is shown in equation 2:

$$\begin{aligned} Addr_{Curr\_line} &= (Addr_{Curr} \gg SHIFT) \ll SHIFT \\ Addr_{k\_to\_Curr} &= ((Addr_{Curr} \gg SHIFT) + k) \ll SHIFT \\ Addr_{Pred\_Line} &= (Addr_{Pred} \gg SHIFT) \ll SHIFT \\ Addr_{k\_to\_Pred} &= ((Addr_{Pred} \gg SHIFT) + k) \ll SHIFT \\ SHIFT &= \log_2 BLOCK\_SIZE \end{aligned} \tag{2}$$

Table 1: Simulation parameters

Parameter	Value
CPU	4 GHz, 4 cores,4-wide OoO, 256-entry ROB, 64-entry LSQ
L1 L-cache	64 KB, 8-way, 8-entry MSHR, 4-cycle
L1 D-cache	64 KB, 12-way, 16-entry MSHR, 5-cycle
L2 Cache	1 MB, 8-way, 32-entry MSHR, 10-cycle
LL Cache	8 MB, 16-way, 64-entry MSHR, 20-cycle
DRAM	512 MB, 8 banks, t_{RP} = 12.5 ns, t_{RCD} = 12.5 ns, t_{CAS} = 12.5 ns

where $Addr_{Curr}$ is the current miss address, $Addr_{Curr\_line}$ is the current cache line address, k is the offset, $Addr_{k\_to\_Curr}$ is the address of the cache line k lines distance to the current miss address, $Addr_{Pred}$ is the predicted next address, $Addr_{Pred\_Line}$ is the cache line address of the predicted next cache line, $Addr_{k\_to\_Pred}$ is address of the cache line k lines distance to to the predicted cache line, *BLOCK_SIZE* is the size of one cache line.

4 EXPERIMENTS

4.1 Simulation Environment

Simulations in the paper are conducted using ChampSim [3], the simulator used in the 3rd Data Prefetching Championship (DPC-3). The parameter is shown in table 1. There is no prefetching for cache levels other than the last level cache (LLC).

4.2 Dataset

We use the application traces for DPC-3 that is generated from *SPEC CPU 2017* benchmark [5]. This benchmark is widely used for the evaluation of compute-intensive performance and memory subsystem performance. The application traces are instruction traces. When ChampSim is running, the simulator reads from an application trace file directly, which simulates the process of the application running on a computer. Besides, The application trace pieces are selected by SimPoint [16]. It implies that the trace pieces are containing some form of patterns, which is appropriate for an ML model to learn.

For the aim of our RNN model training, we need the virtual memory access sequence as the learning dataset. Therefore, we extract the virtual memory access trace from the application trace. We make use of the simulating process and print the memory address when load instructions are processing after the decoding of instructions. After the collection of raw memory access, we generate the sequence of deltas for each application that could be used directly for RNN model training. In this paper, the model input is three consecutive deltas and the training label is the next delta.

4.3 Metrics

To evaluate the performance of the prefetchers, we use the following metrics:

(1) **Cache Hit Rate**: The percentage of accesses that result in cache hit, specifically in LLC for this paper.
(2) **Prefetch Accuracy**: If the prefetched line is hit in the cache before being replaced, there is a prefetch hit. The prefetching accuracy refers to the percentage of prefetch hits to all prefetches.
(3) **Prefetch Coverage**: Percentage of misses avoided due to the contribution of prefetching, the value can be calculated using the equation 3:

$$Prefetch\ Coverage = \frac{(Prefetch\ Hits)}{(Prefetch\ Hits + Cache\ Misses)} \tag{3}$$

(4) **Useful Rate**: While a prefetcher tries to make the data available in the cache before its request, the process is not without cost. For example, it requires the processor to carry out the prefetch, the data source bandwidth to read the data, memory bandwidth to transfer the data and cache space to keep the prefetched data. As a result, the prefetching is not always useful even though there is a prefetch hit. ChampSim analyzes the useful prefetches that are actually helpful for the improvement of computer performance. We calculate the useful rate by dividing the number of useful prefetches to the total number of prefetches.
(5) **Speedup**: The speedup can be deducted from the ratio of two IPCs. The percentage improvement can be represented as:

$$Percentage\ Improvement = 100\left(1 - \frac{1}{Speedup}\right) \tag{4}$$

4.4 Model Training

We use the virtual memory access sequence without program counter to train our model. We split each trace into 70% training set and 30% testing set. To encode the vocabulary of deltas, we fit a tokenizer using the whole trace of each application to acquire the statistics of the deltas. Tokenization is a technique used widely in the area of natural language process as a tool of lexical analysis. It is the process of splitting up a larger body of text into tokens: smaller lines or words. The trained tokenizer contains information about delta distribution. Since we represent the deltas with 16-bit binary numbers, we can encode 2^{16} deltas of the highest frequency. The tokenizers are saved for the decoding of the output delta.

After the encoding of input deltas, each binary bit of the encoded delta is embedded into a 10 dimension vector. Embedding weights are trained together with the LSTM layer during the process of backpropagation. For a process of learning text sentences, word embedding can fit a vector to represent one word that is originally represented as an integer or on-hot encoded vector. The embedding layer learns its correlation with other single words then represents the word with a new vector. In our application, the embedding layer can dig more latent information in the trace, especially the correlation among different deltas.

The LSTM model is trained using the optimizer ADAM [10]. To avoid overfitting, we exploited a 0.1 dropout rate between the LSTM layer and the output dense layer. The dense layer has 16 output dimensions. For each output dimension, it is a binary classification

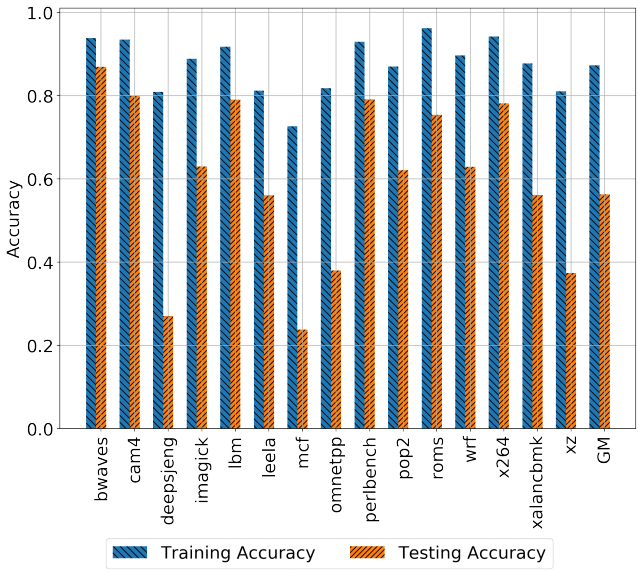

Figure 5: RNN model training and testing accuracy

problem and the result can be either 0 or 1. Therefore, we use binary cross-entropy as the loss function to train the model.

Figure 5 shows the training accuracy and testing accuracy of the model training result. The training accuracy is much higher in general since the data has been seen by the model and the geometric mean reaches 0.87. The testing accuracy notably shows the predictability of the piece of trace for each application. While some applications show more clear pattern to be predicted, such as *bwaves, cam4, lbm, perbench ,roms* and *x264*, some are hard to predict and show low testing accuracy, such as *deepsjeng* and *mcf*. The overall testing accuracy is 0.56. Though this is not as high as the training accuracy, the value is acceptable since it shows a similar performance compared to some recent literature about memory access prediction [22, 23] that use the PARSEC benchmark and [7] that use the SPEC CPU 2006 benchmark.

4.5 RNN Augmentation Effect

We define the RNN augmentation effect as the performance improvement acquired from RAOP framework compared to the performance of only the offset prefetcher implemented inside RAOP prefetching module. To understand the RNN augmentation effect on an offset prefetcher, we implement a next-line prefetcher, the simplest offset prefetcher, for the last level cache. Next-line prefetcher simply prefetches the next cache line of the current request cache line on the same page. Since next-line prefetcher keeps the same offset as one statically for all prefetching requests, it avoids the dynamic performance of other advanced offset prefetchers, which can distract the study on the effect of RNN augmentation.

We conducted experiments on exploiting LLC prefetching module with prefetchers with following configurations:

(1) **No Prefetcher**: No speculation process or prefetching action exist in the prefetching module. This is the baseline of the cache hit rate for us to demonstrate the performance improvement by prefetching.
(2) **Next-line Prefetcher**: The simplest offset prefetcher is implemented. The prefetcher prefetches the next spatial cache line in the main memory to the last level cache. This is a baseline prefetcher for us to evaluate the augmentation effect of the RNN model.
(3) **Predicted Only**: Only the RNN predicted next physical address is prefetched when LLC cache miss happens. This prefetcher can show how the RNN prediction works as a prefetcher itself without any other form of speculation.
(4) **Predicted & Adjacent**: The adjacent means the next line of the RNN predicted physical cache line. In this configuration, though the prefetcher relies solely on the RNN prediction, it makes use of the spatial locality and prefetches also the nearby cache line of the predicted address.
(5) **Next-line and Predicted**: In this prefetcher, next-line speculation is working for the current request address but not for the RNN predicted address. This is a simple combination of the next-line prefetcher and RNN prediction, which separates the reference of spatial speculation from temporal speculation.
(6) **RNN Augmented (RA) Next-line**: This prefetcher uses the proposed framework of RNN Augmented Offset Prefetcher and set the offset as 1, which performs the next-line prefetching. This prefetcher takes advantage of both the spatial speculation from next-line prefetching and the temporal speculation from RNN prediction. By prefetching the relevant next line of both the current request address and the RNN predicted address, this approach combines the spatial and temporal speculation.

Figure 6 illustrates the performance of the discussed prefetcher configurations for all applications we are testing. It shows the metrics including cache hit rate (CHR), prefetch Accuracy (ACC), prefetch coverage (COV) and useful rate (UR). The last chart shows the geometric mean (GM) of all applications. For the cache hit rate, RA Next-line performs the best on 9 out of 14 applications. Those who don't perform (*deepsjeng, leela, mcf, omnetpp and xz*) will also show low model testing accuracy on figure 5. While we use a simple structure for the model with just one LSTM layer and one dense layer, a more complex model that can learn better on the pattern will provide better augmentation effect. This inference can also be drawn from the perspective of prefetch accuracy. The prefetcher with RNN predicting configurations (Predicted Only, Predicted & Adjacent and RA Next-line) shows the highest prefetching accuracy for 10 out of 14 applications. While the RNN augmentation framework improves the performance from Next-line or Next-line & Predict, this method also makes a trade-off by a reduction of useful rate. From the GM chart, we can see the overall RNN augmentation effect. The RA Next-line achieves the highest overall cache hit rate at 0.506 that is 9.4% higher than the Next-line prefetcher at 0.462. RA Next-line achieves 0.42 overall prefetch accuracy that is 3.22 times higher than the Next-line at 0.13, but it shows a reduction of accuracy compared to the Predicted Only and Predicted & Adjacent configurations. RA Next-line also achieves the highest prefetch coverage at 0.217 that is 4.2 times higher than Next-line only. The

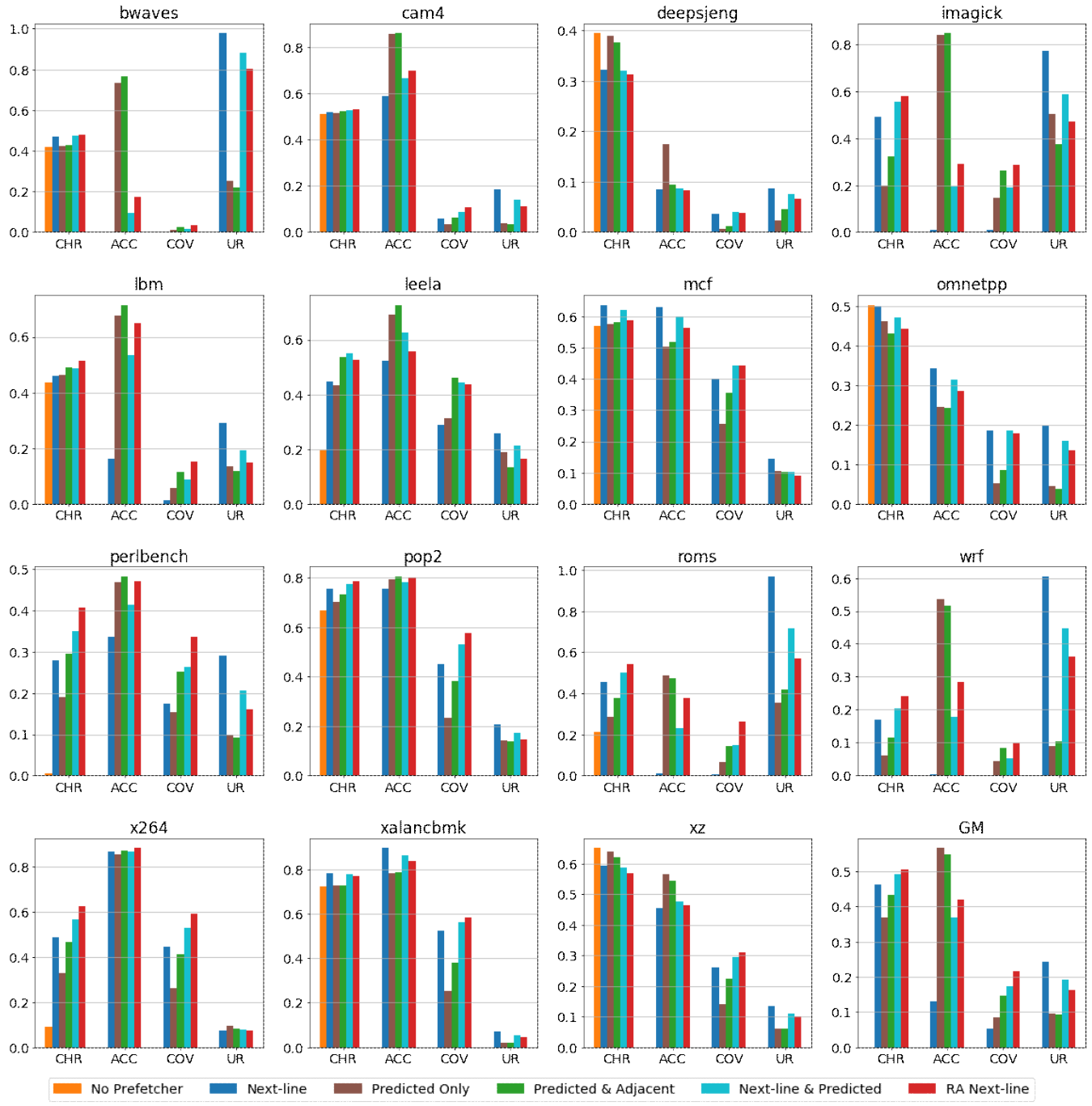

Figure 6: Metrics that show RNN augmentation effect on next-line prefetcher

compromise for the good performance on hit rate and coverage is that more prefetching accesses caused a decrease of useful rate from 0.24 for Next-line to 0.16 for RA Next-line. But the augmentation framework keeps the useful rate higher than the configurations depend solely on the prediction information.

The speedup of the configurations are shown in figure 7. The prefetcher works only on the last level cache so the speedup reveals exactly how the only prefetching process helps. Comparing the No Prefetcher performance, the Next-line prefetcher achieves 2.3% speedup while the RA Next-line reaches 2.67% speedup. With the

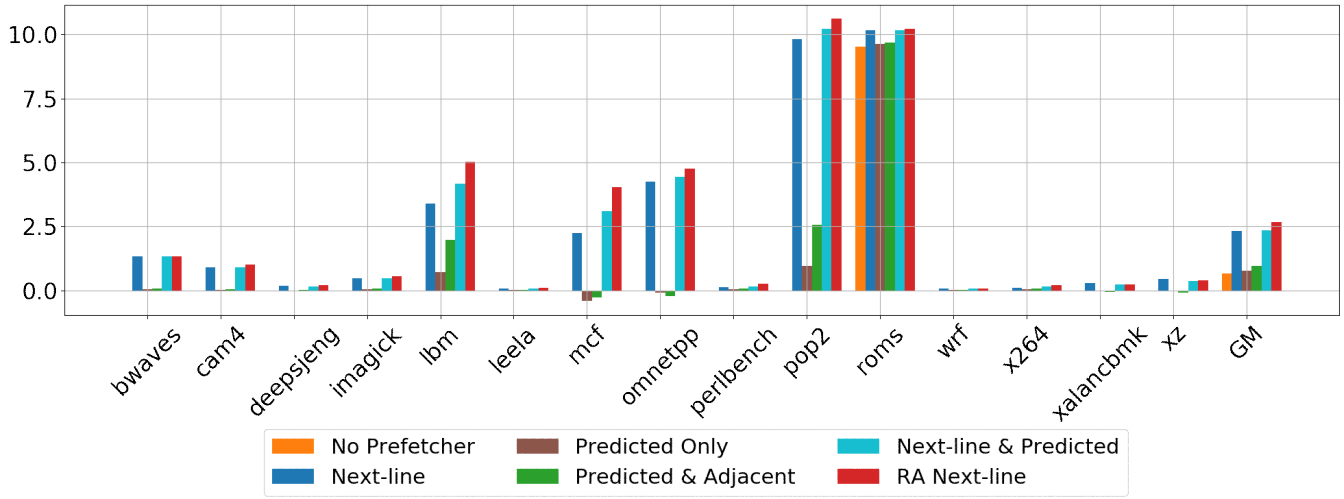

Figure 7: Speedup Percentage improvement that shows RNN augmentation effect on next-line prefetcher

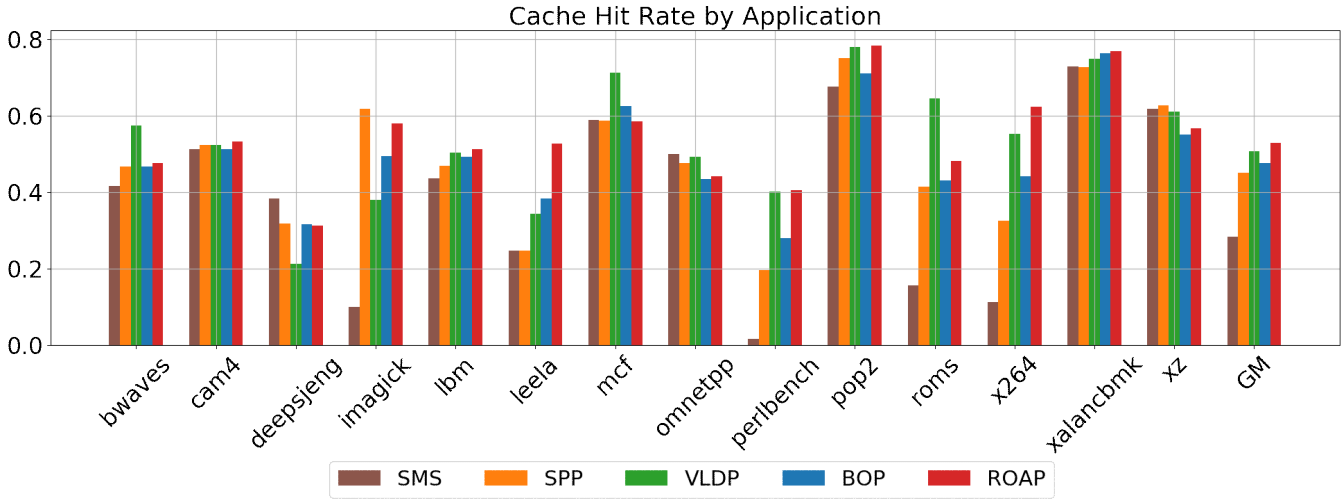

Figure 8: Cache hit rate of RAOP and state-of-the-art prefetchers

augmentation of the RNN model, the Next-line prefetcher acquired a speedup improvement by 15.6%.

4.6 Comparing to State-of-the-Art Prefetchers

The proposed framework can be used to augment the performance of any offset prefetcher. To achieve higher performance and make comparisons to state-of-the-art prefetchers, we implemented a best-offset (BO) prefetcher in the prefetching module as our final RAOP framework in this paper. We compare the performance of RAOP to following prefetchers:

(1) **The Best-Offset Prefetcher (BOP)** [12]: BOP is the base of our proposal in this section. The workflow of BOP has been introduced in Section 3.4. By comparing the performance of BOP and RAOP, we can evaluate the augmentation effect of the RNN model on the best-offset prefetcher.

(2) **Spatial Memory Streaming Prefetcher (SMS)** [21] : SMS prefetcher identifies code-correlated spatial patterns and streams at run time and predicts future accesses using these patterns. It detects purely spatial patterns by exploring the correlation between the request instruction address and the access patterns spatial near the current requested address.SMS streams the predicted cache blocks into the higher-level cache to increase the memory level parallelism and hide memory access latency.

(3) **The Signature Path Prefetcher (SPP)** [9]: SPP exploits a signature mechanism that stores up to four compressed memory access deltas. It learns the memory access pattern to correlate the signature with future deltas probabilities.

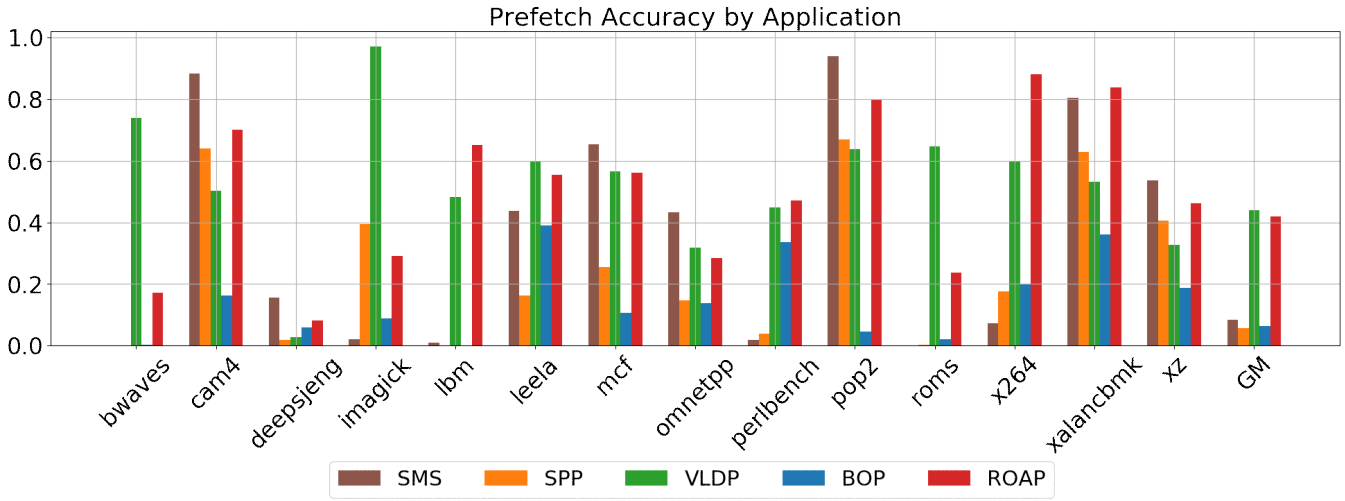

Figure 9: Prefetch accuracy rate of RAOP and state-of-the-art prefetchers

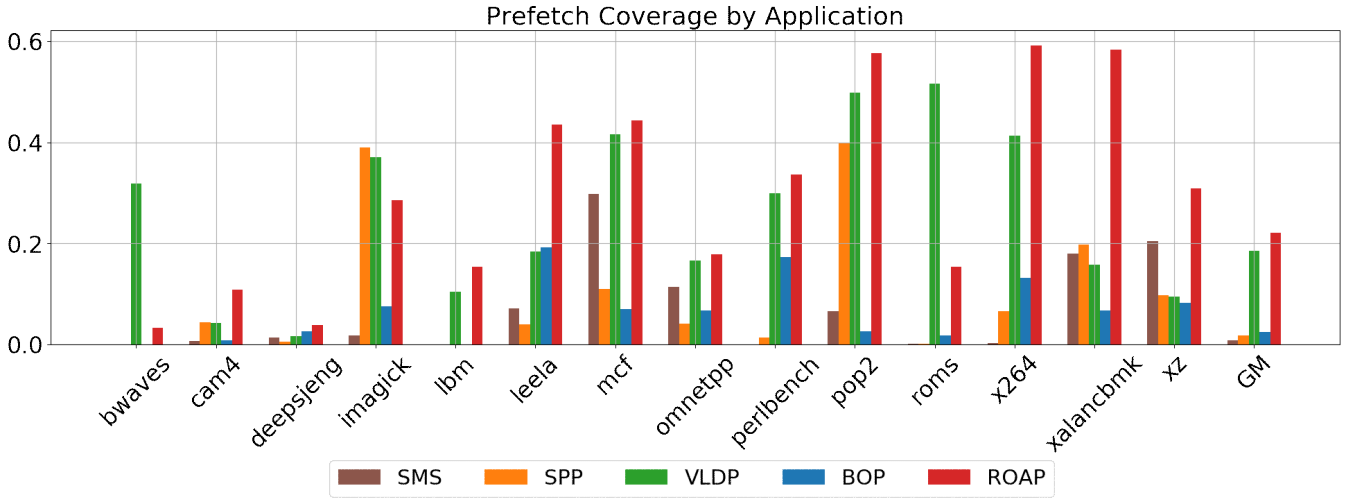

Figure 10: Prefetch coverage rate of RAOP and state-of-the-art prefetchers

SPP adapts its prefetching degree according to the probability of the speculative future deltas, which saves the DRAM bandwidth and lowers the cost of the prefetching process.

(4) **The Variable Length Delta Prefetcher (VLDP)** [20]: It uses an Offset Prediction Table (OPT) and multiple cascaded Delta Prediction Tables (DPT) to achieve table-based variable-length input speculation. The OPT can predict the offset when the first request happens on a page. Each successive DPT corresponds to a longer history length. With longer history is experienced and learned, the prefetcher can make highly accurate prefetches.

Figure 8 illustrates the performance of the proposed RAOP on the cache hit rate compared to the state-of-the-art prefetchers. RAOP achieves the highest cache hit in 7 out of the 14 applications and it holds the highest number among all the prefetcher as VLDP holds 3, SMS holds 1 and SPP holds 3. While in none of these applications BOP achieves the highest cache hit rate, it performs at the second-highest rank overall. Especially, for *cam4, lbm* and *x264*, the RNN augmentation effect covered the gap between BOP and the best performing prefetchers, SPP for *cam4*, and VLDP for *lbm* and *x264*. The last part of bars shows the geometric mean of all the applications, which clearly illustrates that the RNN augmented best-offset prefetcher achieves the highest cache hit even though BOP shows a lower cache hit than VLDP. RAOP acquires the overall hit rate at 0.53 that raises the cache hit rate of BOP at 0.476 by 11.3%.

Figure 9 shows the prefetch accuracy of the prefetchers. VLDP and RAOP are performing notably higher than the other three prefetchers overall. While SMS achieve the highest accuracy for several applications such as *cam4, mcf, pop2* and *xz*, the overall value is low due to some extremely poor performance for some other

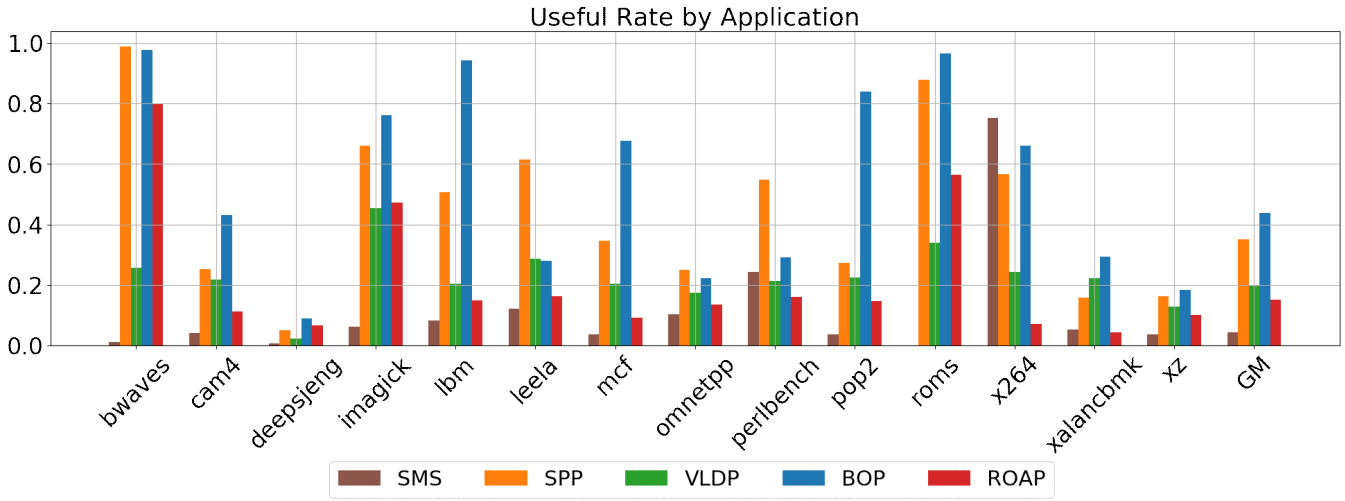

Figure 11: Prefetch useful rate of RAOP and state-of-the-art prefetchers

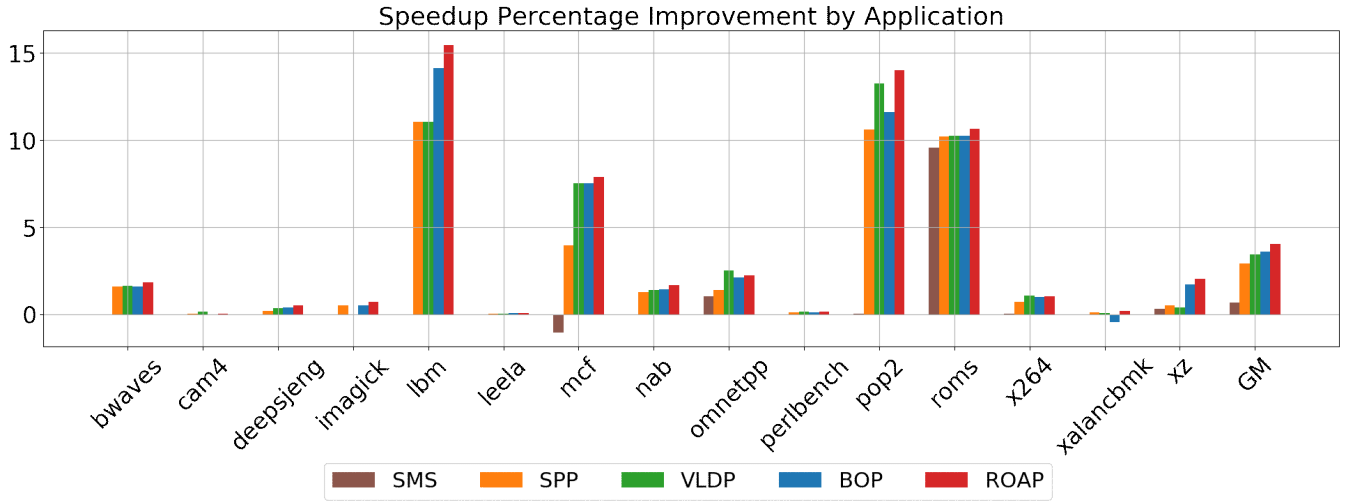

Figure 12: Speedup percentage improvement of RAOP and state-of-the-art prefetchers

applications including *bwaves* and *rooms*. RAOP does not win the highest accuracy and is a bit lower than VLDP. However, the RNN augmentation effect raised the BOP accuracy from 0.064 to 0.418, which is a 6.5 times promotion. Figure 10 shows the prefetch coverage performance which has a similar performance to the prefetch accuracy. A significant augmentation effect can be observed from this chart. RAOP has the highest coverage and boosts the coverage of BOP at 0.024 to 0.221, which is a 9.2 times improvement.

From figure 11 we can observe the cost for the extra action from the RNN augmentation process. BOP achieves the highest performance since the prefetching degree is always 1, implies only the most confident offset cache line is prefetched. SPP exploits an adaptive degree of prefetching, which also avoids waste of useless prefetching access. In contrast, VLDP is a multi-level degree prefetching approach and RAOP prefetches extra accesses in addition to the BOP. This shows the cost of our proposed approach and is a trade-off for a higher degree of prefetching.

A comprehensive indicator for the previous metrics is the speedup performance as is shown in figure 12. RAOP achieves the highest speedup percentage improvement for 10 out of 14 applications. Especially, for application *lbm* RAOP achieves the highest speedup at 15.4%, and for *pop2* it achieves the highest augmentation from BOP by 26.8%. RAOP achieves an overall 4.05% speedup as we can observe from the geometric mean (GM) and it shows an augmentation effect on BOP at 12.8%.

5 CONCLUSION

We have proposed an RNN augmented offset prefetching approach that uses the predictive ability of RNN to produce a temporal reference address for an offset prefetcher so that the offset prefetcher can work spatially on both the current address and the predicted next address. RAOP framework is separated into two parts: the RNN encapsulated in the predictor part learns from the virtual access traces of applications and predicts the next virtual address from the input of a sequence of deltas. The virtual address needs to be translated and then transmitted to the prefetching module, where an offset prefetcher is encapsulated. We present the RNN augmentation effect by implementing a simple next-line prefetcher in the prefetching module, which shows a 15.6% speedup compared to only a next-line prefetcher. We have also compared our approach with other state-of-the-art prefetchers by implementing a best-offset prefetcher in our prefetching module, the result shows the highest speedup compared to SMS, SPP, VLDP, and BOP. RAOP shows 6.5 times accuracy augmentation, 9.2 times coverage augmentation, and 12.8% speedup augmentation effect compared to raw best-offset prefetcher. The current implementation is for single-thread and we will extend it to multiple threads in the future. General model, dynamic prefetching scheme, and latency of prefetcher will also be explored in future work.

ACKNOWLEDGMENTS

This work is supported by Google Faculty Research Award, Air Force Research Laboratory grant number FA8750-18-S-7001, and National Science Foundation award number 1912680.

REFERENCES

[1] Peter Braun and Heiner Litz. 2019. Understanding Memory Access Patterns for Prefetching. In *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA*.
[2] Carlos Carvalho. 2002. The gap between processor and memory speeds. In *Proc. of IEEE International Conference on Control and Automation*.
[3] "ChampSim". 2017. https://github.com/ChampSim/ChampSim.
[4] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
[5] "SPEC CPU2017". 2017. The Standard Performance Evaluation Corporation. https://www.spec.org/cpu2017/.
[6] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).
[7] Milad Hashemi, Kevin Swersky, Jamie A Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning memory access patterns. *arXiv preprint arXiv:1803.02329* (2018).
[8] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine* 29, 6 (2012), 82–97.
[9] Jinchun Kim, Seth H Pugsley, Paul V Gratz, AL Narasimha Reddy, Chris Wilkerson, and Zeshan Chishti. 2016. Path confidence based lookahead prefetching. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.
[10] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[11] Shih-wei Liao, Tzu-Han Hung, Donald Nguyen, Chinyen Chou, Chiaheng Tu, and Hucheng Zhou. 2009. Machine learning-based prefetch optimization for data center applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 1–10.
[12] Pierre Michaud. 2016. Best-offset hardware prefetching. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 469–480.
[13] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.
[14] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. 2018. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*. 48–53.
[15] Leeor Peled, Uri Weiser, and Yoav Etsion. 2018. A neural network memory prefetcher using semantic locality. *arXiv preprint arXiv:1804.00478* (2018).
[16] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder. 2003. Using SimPoint for accurate and efficient simulation. *ACM SIGMETRICS Performance Evaluation Review* 31, 1 (2003), 318–319.
[17] Seth H Pugsley, Zeshan Chishti, Chris Wilkerson, Peng-fei Chuang, Robert L Scott, Aamer Jaleel, Shih-Lien Lu, Kingsum Chow, and Rajeev Balasubramonian. 2014. Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 626–637.
[18] S Rahman, M Burtscher, Z Zong, and A Qasem. 2015. Maximizing Hardware Prefetch Effectiveness with Machine Learning. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. 383–389.
[19] Mehran Shakerinava, Mohammad Bakhshalipour, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. 2019. Multi-Lookahead Offset Prefetching. *The Third Data Prefetching Championship* (2019).
[20] Manjunath Shevgoor, Sahil Koladiya, Rajeev Balasubramonian, Chris Wilkerson, Seth H Pugsley, and Zeshan Chishti. 2015. Efficiently prefetching complex address patterns. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 141–152.
[21] Stephen Somogyi, Thomas F Wenisch, Anastassia Ailamaki, Babak Falsafi, and Andreas Moshovos. 2006. Spatial memory streaming. *ACM SIGARCH Computer Architecture News* 34, 2 (2006), 252–263.
[22] Ajitesh Srivastava, Angelos Lazaris, Benjamin Brooks, Rajgopal Kannan, and Viktor K Prasanna. 2019. Predicting memory accesses: the road to compact ML-driven prefetcher. In *Proceedings of the International Symposium on Memory Systems*. 461–470.
[23] Ajitesh Srivastava, Ta-Yang Wang, Pengmiao Zhang, Cesar Augusto F De Rose, Rajgopal Kannan, and Viktor K Prasanna. 2020. MemMAP: Compact and Generalizable Meta-LSTM Models for Memory Access Prediction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 57–68.
[24] Steven P Vander Wiel and David J Lilja. 1997. When caches aren't enough: Data prefetching techniques. *Computer* 30, 7 (1997), 23–30.
[25] Xiangyao Yu, Christopher J Hughes, Nadathur Satish, and Srinivas Devadas. 2015. IMP: Indirect memory prefetcher. In *Proceedings of the 48th International Symposium on Microarchitecture*. 178–190.
[26] Yuan Zeng and Xiaochen Guo. 2017. Long short term memory based hardware prefetcher: a case study. In *Proceedings of the International Symposium on Memory Systems*. 305–311.