SN Computer Science (2024) 5:646
https://doi.org/10.1007/s42979-024-02989-w

ORIGINAL RESEARCH

# Accelerating Graph Analytics Using Attention-Based Data Prefetcher

Pengmiao Zhang[1] · Rajgopal Kannan[2] · Anant V. Nori[3] · Viktor K. Prasanna[1]

Received: 1 November 2022 / Accepted: 22 May 2024

## Abstract

Graph analytics shows promise for solving challenging problems on relational data. However, memory constraints arise from the large size of graphs and the high complexity of algorithms. Data prefetching is a crucial technique to hide memory access latency by predicting and fetching data into the memory cache beforehand. Traditional prefetchers struggle with fixed rules in adapting to complex memory access patterns in graph analytics. Machine learning (ML) algorithms, particularly long short-term memory (LSTM) models, excel in memory access prediction. However, they encounter challenges such as difficulty in learning interleaved access patterns and high storage costs when predicting in large memory address space. In addition, there remains a gap between designing a high-performance ML-based memory access predictor and developing an effective ML-based prefetcher for an existing memory system. In this work, we propose a novel Attention-based prefetching framework to accelerate graph analytics applications. To achieve high-performance memory access prediction, we propose A2P, a novel Attention-based memory Access Predictor for graph analytics. We use the multi-head self-attention mechanism to extract features from memory traces. We design a novel *bitmap labeling* method to collect future deltas within a spatial range, making interleaved patterns easier to learn. We introduce a novel *super page* concept, allowing the model to surpass physical page constraints. To integrate A2P into a memory system, we design a three-module prefetching framework composed of an existing memory hierarchy, a prefetch controller, and the predictor A2P. In addition, we propose a hybrid design to combine A2P and existing hardware prefetchers for higher prefetching performance. We evaluate A2P and the prefetching framework using the widely used GAP benchmark. Prediction experiments show that for the top three predictions, A2P outperforms the widely used state-of-the-art LSTM-based model by 23.1% w.r.t. Precision, 21.2% w.r.t. Recall, and 10.4% w.r.t. Coverage. Prefetching experiments show that A2P provides 18.4% IPC Improvement on average, outperforming state-of-the-art prefetchers BO by 17.2%, ISB by 15.0%, and Delta-LSTM by 10.9%. The hybrid prefetcher combining A2P and ISB achieves 21.7% IPC Improvement, outperforming the hybrid of BO and ISB by 16.3%.

**Keywords** Attention · Memory access prediction · Prefetching · Graph analytics

## Introduction

Graph analytics involves studying and interpreting data presented in graph structures, where nodes represent entities, and edges denote relationships. It aims to extract insights and patterns from these graphical representations, making it valuable across various domains such as social networks and biological systems [1]. Especially in the era of big data, graph analytics holds significant promise for exploring entity relationships, surpassing traditional relational databases due to its explicit representation of relations [2].

Applications of graph analytics often face memory constraints [3]. Many frameworks, including Pregel, Giraph, GraphLab, and ComBLAS [4–7], store graphs in a compressed sparse format (CSR or CSC) for efficient

✉ Pengmiao Zhang
pengmiao@usc.edu

Rajgopal Kannan
rajgopal.kannan.civ@army.mil

Anant V. Nori
anant.v.nori@intel.com

Viktor K. Prasanna
prasanna@usc.edu

[1] University of Southern California, Los Angeles, USA
[2] DEVCOM Army Research Lab, Los Angeles, USA
[3] Intel Labs, Bangalore, India

Published online: 13 June 2024

sequential edge access [8]. Nevertheless, obtaining values of scattered neighboring vertices involves fine-grained random accesses. Especially for large graphs, these accesses elevate cache misses and bottlenecks the processing speed of graph analytics.

Data prefetching is technique that hides memory access latency by decoupling and overlapping data transfers and computation [9]. In order to reduce CPU stalls on a cache miss, a data prefetcher predicts future data accesses, initiates a data fetch, and brings the data closer to the processor before it is requested. The most essential step for prefetching is accurate *memory access prediction*. The goal of memory access prediction is to exploit the correlation between history memory accesses to predict future one or more memory access addresses.

Existing rule-based hardware data prefetchers rely on explicit spatial or temporal locality of references [10] to predict future accesses. However, they are not powerful enough to adapt to the increasingly complex memory access patterns from graph analytics algorithms. For prefetchers based on spatial locality [11–13], the prediction range is typically within a page, which limits the diversity of prediction and shows low prediction accuracy. For prefetchers based on temporal locality [14, 15], record and replay mechanism is widely used but it shows low generalizability for unseen patterns.

Machine learning (ML) algorithms have demonstrated success in the domain of sequence prediction [16], offering valuable insights into memory access prediction. The memory access stream can be modeled as a time-series sequence. Powerful sequence models such as long short-term memory (LSTM) [17] have been studied to predict memory accesses. Due to the sparsity of memory addresses for an application [18], prior works [18–21] take the memory access deltas (a "delta" is defined as the difference between consecutive access addresses) as input and predict the next delta through classification. LSTM-based delta prediction has shown higher prediction performance than traditional prefetchers [19, 22] due to its high accuracy and generalizability.

However, existing LSTM-based modeling methods show low performance when working on complex memory patterns in graph analytics. First, due to the large number of parameters and the recurrent structure, training an LSTM-based model is hard and its performance is not stable [23]. Second, existing methods predict only one next delta while deltas between interleaved patterns hinder the model training. Third, existing ML-based methods discard the locality of references [10] used in traditional prefetchers. The model output delta is in the entire address space, which causes an extremely large output dimension and high model storage cost for predicting diverse memory access patterns in graph analytics.

In addition, there still exists a gap between the design of a high-performance ML-based memory access predictor and the development of an effective ML-based prefetcher for a memory system. First, the integration of a predictor into a hardware system necessitates interfaces for extracting memory accesses and returning predictions. Second, a prefetch controller is essential to process the history memory accesses for predictor input and manage the predicted results for issuing prefetch. Third, a seamless integration of an ML-based predictor and existing prefetchers is crucial for enhancing implementation flexibility and improving prefetching performance.

To address the above limitations, we introduce a novel ML-based data prefetching framework for accelerating graph analytics. This framework comprises a novel attention-based predictor seamlessly integrated into the last-level cache memory system.

To develop a high-performance ML-based memory access predictor, we propose A2P, a novel Attention-based memory Access Predictor for graph analytics. Attention mechanism [24] has achieved huge success for sequence modeling tasks in many fields compared to LSTM. We propose three novel optimizations to the model. First, through tokenization [25], we map the memory access deltas to tokens, which are numerical values that can be processed directly by a neural network. Second, we propose a novel *bitmap labeling* method to collect deltas within a page to the current address from future accesses. In this way, we model memory access prediction as a multi-label classification problem. We develop an attention-based model to fit the mapping between the delta tokens and the bitmap labels to achieve a high prediction performance. Third, we introduce a novel concept *super page*, which relaxes the spatial range from a page size to larger ranges, aiming to detect delta patterns beyond pages.

To seamlessly integrate A2P into an existing memory system, we have devised a three-module prefetching framework comprising the memory system, a prefetch controller, and the memory access predictor A2P. First, we design interfaces for the last level cache (LLC) data prefetching in a memory system. We monitor the memory requests from the L2 cache to LLC and issue prefetch to LLC when a prediction is generated. Second, we develop a prefetch controller as an intermediate agent between the memory system and the predictor. The controller buffers history memory accesses and passes a sequence of memory addresses to the predictor. The controller also manages the post-processing and controls the prefetch degree using the predictions given by A2P. In addition, we introduce a hybrid prefetching design that combines A2P with a temporal prefetcher, which further improves prefetching performance and increases the implementation flexibility for multiple prefetchers working collaboratively.

Our contribution can be summarized as follows:

- We develop A2P, a novel attention-based memory access prediction model for graph analytics. We use delta token sequences for model input and use an attention-based network for feature extraction.
- We propose a novel *bitmap labeling* method to collect multiple future deltas within a spatial range as labels. Based on bitmap labeling, the memory access prediction is reduced to a multi-label classification problem, which enables multiple memory access predictions in each inference.
- We introduce a novel concept *super page* to relax the range of spatial region from the typical one-page size to several bits larger, which enables the model to be trained by patterns beyond page range while still taking advantage of spatial locality.
- We design a novel three-module prefetching framework that integrates A2P into a memory system. The framework consists of a memory hierarchy with interfaces for LLC prefetching, a prefetch controller to manage model input and predictions, and a memory access predictor using A2P.
- We present a hybrid prefetch design to combine A2P with existing temporal prefetchers for more flexible implementation and higher prefetch performance.
- We evaluate our method using widely used graph analytics benchmark *GAP* [26]. Prediction experiments show that for the top three predictions, A2P outperforms the widely used state-of-the-art LSTM-based model by 23.1% w.r.t. Precision, 21.2% w.r.t. Recall, and 10.4% w.r.t. Coverage.
- We evaluate the prefetching performance of A2P in the three-module prefetching framework through ChampSim [27] simulation. Results show that A2P provides 18.4% IPC Improvement, outperforming state-of-the-art prefetchers BO by 17.2%, ISB by 15.0%, and Delta-LSTM by 10.9%. The hybrid prefetcher combining A2P and ISB results in 21.7% IPC Improvement, outperforming the hybrid prefetcher using BO and ISB by 16.3%.

## Graph Analytics

### Background

Graph analytics problems are pervasive in various domains such as web and social networks, transportation networks, and biological systems. However, as the volume of data grows due to the advent of big data, graph analytics encounters a significant challenge: high processing latency. Efficiently processing large-scale graphs becomes crucial to address this issue. Researchers have conducted numerous studies to accelerate graph analytics methods and enhance their performance.

First, many distributed frameworks have been proposed to process very large graphs on clusters [4, 5]. However, because of the high communication overheads of distributed systems, even single threaded implementations of graph algorithms have been shown to outperform many such frameworks running on several machines [28].

Second, the growth in DDR capacity allows large graphs to fit in the main memory of a single server. Consequently, many frameworks have been developed for high performance graph analytics on multicore platforms [29–31]. However, multi-threaded graph algorithms may incur race conditions and hence, require expensive synchronization (atomics or locks) primitives that can significantly decrease performance and scalability. Furthermore, graph computations are characterized by large communication volume and irregular access patterns that make it challenging to efficiently utilize the resources even on a single machine [32].

Third, recent advances in hardware technologies offer potentially new avenues to accelerate graph analytics, in particular, new memory technologies, such as high bandwidth memory (HBM) and scratchpad caches. However, many graph analytics frameworks are based on the conventional push-pull vertex-centric processing paradigm [29, 33–35], which allows every thread to access and update data of arbitrary vertices in the graph. Without significant pre-processing, this leads to unpredictable and fine-grained random memory accesses, thereby decreasing the utility of the wide memory buses and deterministic caching features offered by these new architectures. Some frameworks and application specific programs [36–38] have adopted optimized edge-centric programming models that improve access locality and reduce synchronization overhead. However, these programming models require touching all or a large fraction of the edges of the graph in each iteration, and are not work optimal for algorithms with dynamic active vertex sets, such as BFS, seeded random walk, etc. A work inefficient implementation can significantly underperform an efficient serial algorithm if the useful work done per iteration is very small.

In this work, we propose a novel ML-based prefetching framework using a novel attention-based memory access predictor. Through accurately predicting memory accesses and effectively prefetching data before requested, our approach significantly accelerates the execution of graph analytics applications.

### Graph Analytics Applications

In this work, we evaluate our prefetching approach on five popular graph analytics applications:

**Breadth-first search (BFS)**—Used for rooted graph traversal or search. The BFS algorithm finds the parent of every reachable node in the BFS tree rooted at a given vertex. BFS is a fundamental algorithm and often used within other graph applications.

**Single source shortest path (SSSP)**—Finds the shortest distance to all the nodes in a weighted graph from a given source vertex. Using the same setting as GAP [26], we use non-negative edges in this work. For unweighted graphs, BFS can return the shortest path considering all the edges with unit weight.

**PageRank (PR)**—A node ranking algorithm that determines the "popularity" of nodes in a graph, originally used to sort web search results [39]. PR is also an important benchmark for the performance of sparse matrix–vector (SpMV) multiplication, which is widely used in many scientific and engineering applications [40–42].

**Connected components (CC)**—Labels connected components in a graph. A connected component means a subgraph that all of its nodes are connected to each other. Two nodes are connected if there is a path between the two nodes. A connected component is maximal, which means any nodes connected to the component is part of the component.

**Betweenness centrality (BC)**—Approximates the betweenness centrality score for all the nodes in the graph by only computing the shortest paths from a subset of the vertices. BC is a metric that attempts to measure the importance of vertices within a graph. BC can be computationally demanding as it requires computing all of the shortest paths between all pairs of vertices.

## ML for Memory Access Prediction

### Problem Formulation

The goal of memory access prediction is to exploit the correlation between history memory accesses to predict one or more future memory access addresses. Due to the sparsity of memory address space for an application, treating memory access prediction as classification instead of regression is a better option [18].

Figure 1 shows the fields in a physical memory address. Data fetch operation is in the unit of a block (cache line). Thus, memory access prediction considers only the block address space, ignoring the block offset field.

Let $X_t = \{x_1, x_2, \ldots, x_N\}$ be the sequence of $N$ history block addresses at time $t$. Let $Y_t = \{y_1, y_2, \ldots, y_k\}$ be a set of $k$ outputs associated with future $k$ block addresses. Our goal is to approximates $P(Y_t \mid X_t)$, the probability that the future addresses $Y_t$ will be accessed given the history events $X_t$.

Because memory access prediction is modeled as a classification problem, the number of classes will be extremely

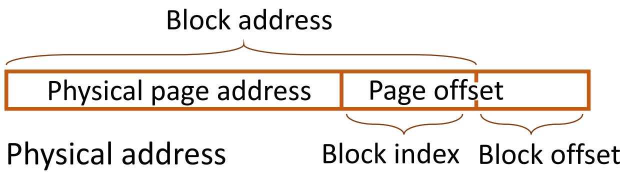


**Fig. 1** Fields in a physical address

large when considering each unique block address as a class. A commonly used technique to reduce the number of classes is to work on block deltas instead of block addresses directly [19, 22]. A *block delta* is defined as the block address difference between consecutive memory accesses. We use *delta* for short in later sections because we only work on block address space.

A ML model can be developed and trained to learn the probability $P(Y_t \mid X_t)$. The vector of history accesses $X_t$ is defined as *input feature*, the actually accessed future addresses $Y_t$ is defined as *output label*. Using samples of input features and output labels in a long memory trace, an ML model can be trained to adapt to the data and construct an approximation of the true probability.

### Recurrent Neural Networks

Recurrent neural networks (RNNs) are widely used for the task of memory access prediction [18–20, 43]. RNNs exhibit temporal dynamic behavior by storing sequential information in their internal states. By assuming the dependence between the current input and previous inputs, RNNs perform better in sequence processing than basic neural networks that consider the time steps as dimensions without time-series information.

Long short-term memory (LSTM) [17] is a variant of RNN that overcomes gradient vanishing and exploding problems of basic RNNs. An LSTM block (different from the *address block* in "Problem Formulation") 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 $\mathbf{y}^{(t)}$, as is shown in Fig. 2. The operation of each set of gates of the layer is given by Eq. 1.

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

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

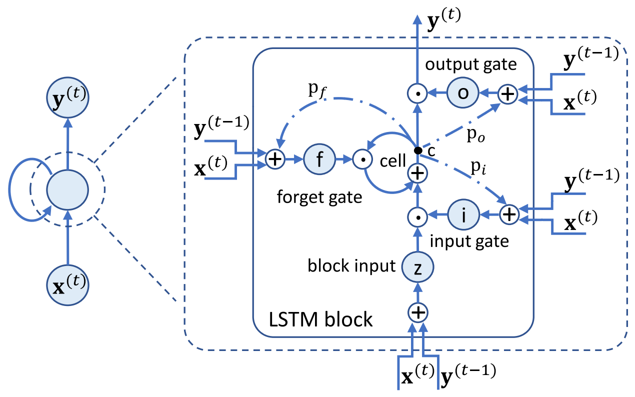


**Fig. 2** The structure of LSTM

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; $\sigma$ and tanh are sigmoid and hyperbolic tangent functions that serve as non-linear activation functions; $\odot$ is the operation of Hadamard vector multiplication.

## Attention Mechanism

Attention mechanism has shown powerful sequence modeling capability without using recurrent structures. The Transformer [24], a sequence model based on multi-head self-attention initially proposed for machine translation, has achieved huge success for sequence modeling tasks in many fields compared to traditional recurrent models.

The original Transformer uses an encoder-decoder structure with a sinusoidal position encoding. A general *Transformer layer* consists mainly of a multi-head attention and a point-wise feed-forward, as is shown in Fig. 3.

**Self-attention.** Self-attention takes the embedding of items as input, converts them to three matrices through linear projection, then feeds them into a scaled dot-product attention defined as:

$$\mathrm{Attention}(Q,K,V)=\mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2}$$

where $Q$ represents the queries, $K$ the keys, $V$ the values, $d$ the dimension of layer input.

**Multi-head self-attention.** One self-attention operation can be considered as one "head", we can apply Multi-head Self-Attention (MSA) operation as follows:

$$\begin{aligned}\mathrm{MSA}(Q,K,V) &= \mathrm{Concat}\left(\mathrm{head}_1,\ldots,\ \mathrm{head}_{\mathrm{H}}\right)W^O\\ \mathrm{head_i} &= \ \mathrm{Attention}\ \left(QW_i^Q, KW_i^K, VW_i^V\right),\end{aligned} \tag{3}$$

where the projection matrices $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d\times d}$, H is the total number of heads, $i$ is the index of heads from 1 to H.

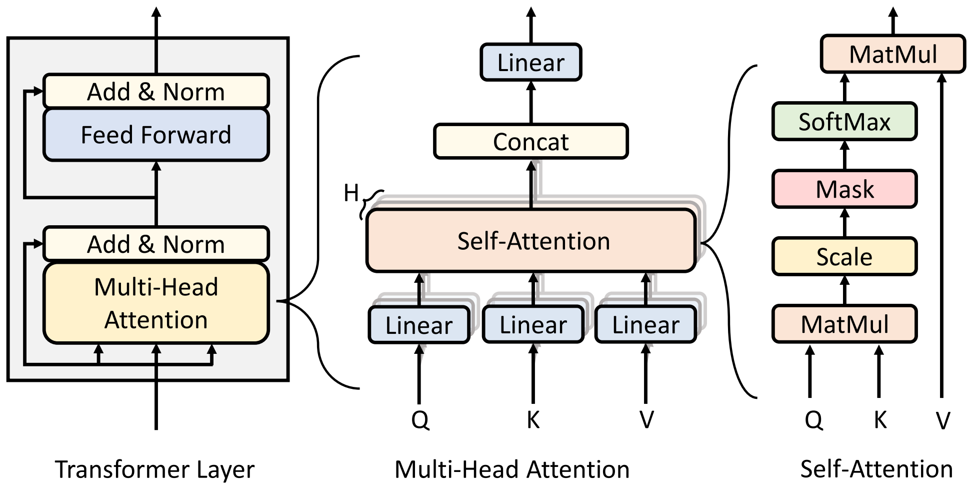
**Point-wise feed-forward.** Point-wise Feed-Forward Network (FFN) is defined as follows:

$$\mathrm{FFN}(x) = \max\left(0, xW_1 + b_1\right)W_2 + b_2 \tag{4}$$

## Attention-Based Memory Access Predictor

In this section we describe A2P, a novel Attention-based memory Access Prediction model for graph analytics. The overall model structure is shown in Fig. 4. A2P takes the deltas of block addresses as input, tokenizes the deltas, and uses the delta tokens for neural network processing (see "Delta Token Input"). Then we collect future deltas within a spatial range using bitmaps for model training labels (see "Delta Bitmap Labeling"). We formulate the memory access prediction task as a multi-label classification problem and design an attention-based neural network to fit the mapping from input delta tokens to bitmap labels (see "Attention-Based

**Fig. 3** The structure of a transformer layer



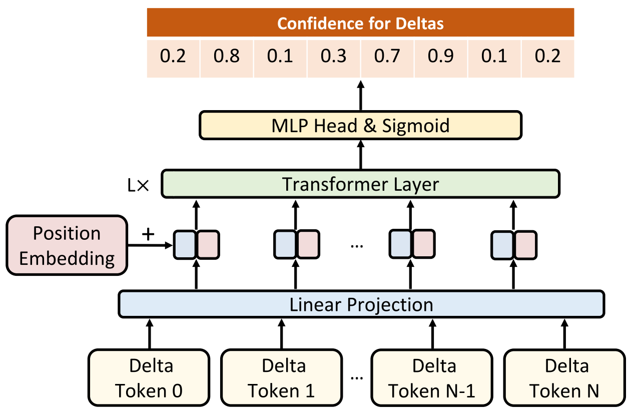


**Fig. 4** Overall structure of A2P

Network”). During inference, the model predicts the confidence (probability) of deltas in a bitmap which enables multiple delta predictions in one inference. Furthermore, we introduce the notion of *super page* that enables the model to learn patterns in a larger spatial range (see “Super Page”).

## Delta Token Input

The memory access address is vast and sparse [22], so it is common to use deltas (address difference between consecutive accesses) instead of the raw address for memory access prediction. However, the deltas are still not appropriate for model input because of the large range. By formatting the deltas as classes, we can tokenize the deltas: mapping deltas into numerical values for model processing.

Figure 5 illustrates the preprocessing steps for model input using an example access sequence. First, the raw address is shifted by a block offset (6-bit in the example). Then the deltas are computed from consecutive block addresses. The delta values are in a large range, so we map the deltas to tokens, which can be used for numerical calculation in a neural network.

## Delta Bitmap Labeling

Unlike existing methods predicting the next one consecutive delta [19, 22], we propose to predict multiple future deltas within a spatial range. A heuristic spatial range is one-page size, which is commonly used in state-of-the-art spatial prefetchers [11–13]. We design a novel *bitmap labeling* method to collect the labels for model training.

Figure 6 illustrates the delta bitmap labeling method. First, we scan a window of future memory accesses to collect multiple future deltas to the current block address. Then we define a bitmap with the size as the range of deltas, which enables positive and negative delta predictions. For example, given the spatial range as a $a$-bit page offset with a $b$-bit block offset, the delta range will be $\pm 2^{(a-b)}$, leading to the bitmap size as $2^{(a-b)+1}$. Figure 6 shows a simple example with delta range of $\pm 4$ and bitmap size at 8. By mapping both the positive and negative deltas into the bitmap index, and setting the corresponding locations as 1, we

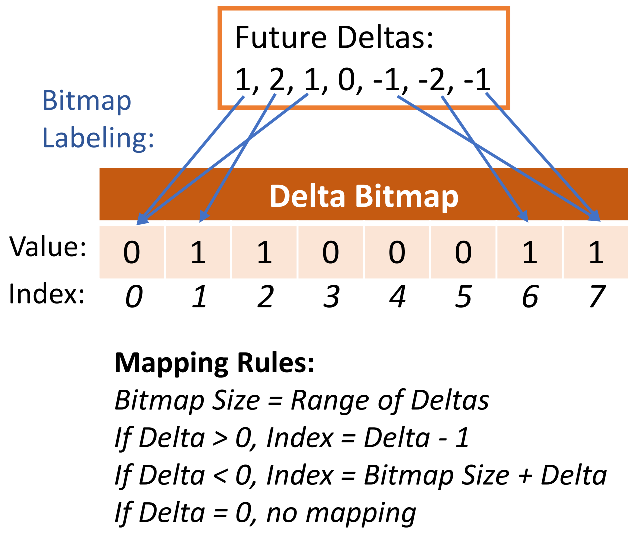
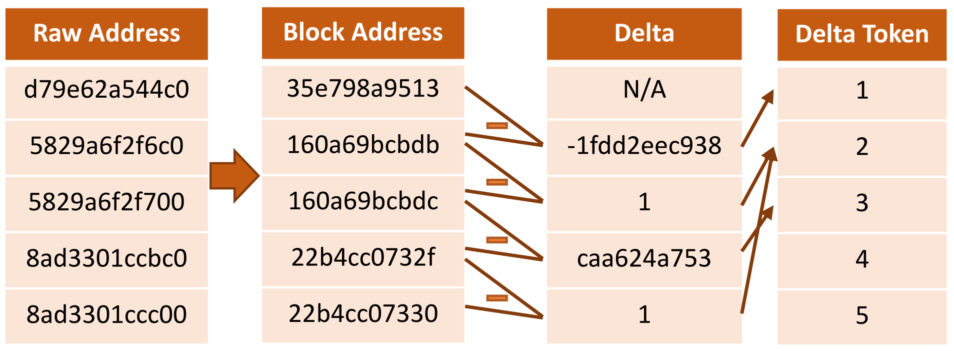


**Fig. 6** Delta bitmap labeling process and the mapping rules from delta to bitmap index

**Fig. 5** Preprocessing for delta token input



| Raw Address | Block Address | Delta | Delta Token |
|---|---|---|---|
| d79e62a544c0 | 35e798a9513 | N/A | 1 |
| 5829a6f2f6c0 | 160a69bcbdb | -1fdd2eec938 | 2 |
| 5829a6f2f700 | 160a69bcbdc | 1 | 3 |
| 8ad3301ccbc0 | 22b4cc0732f | caa624a753 | 4 |
| 8ad3301ccc00 | 22b4cc07330 | 1 | 5 |

can construct a bitmap with multiple labels for model training. Zero delta will not be labeled or predicted because it means the same address as the current request. Using bitmap labeling, the model output dimension can be dramatically reduced from large delta range at entire address space to a small page range, compared to predicting the next consecutive delta.

### Attention-Based Network

With the above well-defined input and output, we develop an attention-based network to learn the mapping, as is shown in Fig. 4. First the delta sequence is processed by a dense linear projection as model input layer. Then, learnable 1D position embeddings [44] are incorporated to insert temporal information. With processed input and position embeddings, multi-head attention-based Transformer layers (Fig. 3) are used to extract the latent features. At last, using the features extracted from the last Transformer layer, a multi-layer perceptron (MLP) is used for classification output. Through a sigmoid activation function for each bit in output bitmap, the model predicts the probability for each corresponding deltas, also referred to as delta confidence.

### Super Page

In "Delta Bitmap Labeling" we set the spatial range as a page size following existing prefetching methods. Considering the high learning capability of neural network models, we propose to relax the spatial range so that the model can learn and predict patterns beyond a page.

We define the relaxed range as *super page*, as shown in Fig. 7. With *n-bit relaxation* from page offset, we can have a super page range. Then we collect bitmap labels and predict deltas within the super page instead of a physical page range. For example, for a 12-bit physical page with 2-bit relaxation, we can collect labels in a 14-bit super page. Different graph analytics applications can benefit variously from the super page. Particularly, large graphs whose nodes are stored beyond pages which are accessed in a spatial pattern will benefit significantly from the super page.

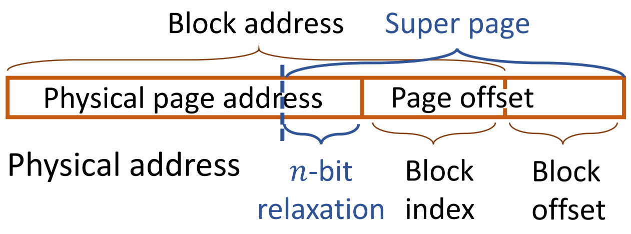


**Fig. 7** Super page with $n$-bit relaxation

## A2P for Data Prefetching

Data prefetching can be either hardware-based or software-directed or a combination of both [45]. Hardware-based prefetching, requiring some support unit connected to the cache, can dynamically handle prefetches at run-time without compiler intervention. Software-directed approaches rely on compiler technology to insert explicit prefetch instructions. Our model A2P works in the physical address space and targets a hardware prefetcher for the last level cache (LLC).

### A2P in System

Figure 8 shows integrating our memory access prediction model A2P with existing computer architecture and memory hierarchy. There are three modules in the A2P-based prefetcher: the memory system, a prefetch controller, and the memory access predictor using A2P.

#### Memory System

The hardware memory system consists of a CPU core, a first-level (L1) cache, second level (L2) cache, last level cache (LLC), and a DRAM. The CPU core requests data from a memory address and accesses the hierarchy of the memory system until a hit in a memory level. We implement our ML-based predictor for the last level cache (LLC) data prefetching with following reasons. First, the L1 cache and L2 cache are fast (approximately 4–20 cycles), typically implemented with table-based simple regular prefetchers. ML-based models require higher latency than regular prefetchers and may fail to contribute to the L1 and L2 cache prefetching. In contrast, a miss in LLC requests data from DRAM, requiring higher latency (approximately 200 cycles). This latency allows the inference of the hardware implemented A2P model. Second, the memory accesses of LLC are filtered by the cache misses of L1 and L2 caches, leading to high irregularity and difficulty in prediction. Machine learning algorithms have high potential to handle irregular streams than traditional table-based models. Third, for multi-core systems, the L1 and L2 caches are typically private while the L3 cache is shared. Therefore, one ML model can work for a multi-core system without the cost of multiple implementations for each core.

#### Prefetch Controller

To make use of the A2P predictions for data prefetching, we design a prefetch controller. The controller is responsible

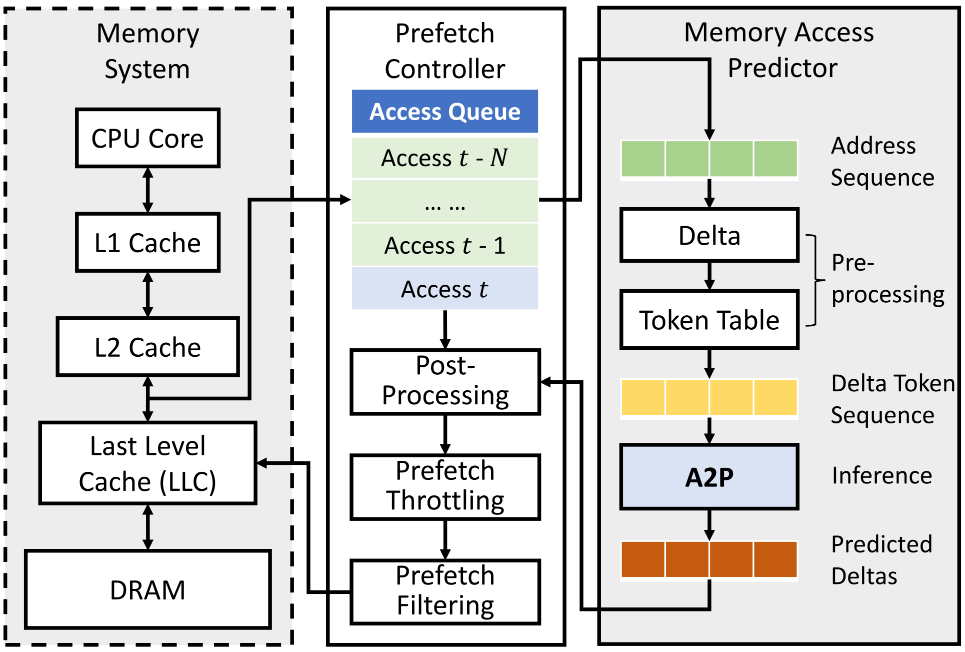


**Fig. 8** A2P-based three-module data prefetching framework

for buffering the history memory accesses, postprocessing the predictions, throttling the prefetch degree, and filtering repetitive requests.

**Buffering.** An access queue is designed to store the sequence of history memory accesses for model input. When a prediction is ready, the access queue pops the base address with time stamp $t$ for postprocessing.

**Postprocessing.** The output of A2P is in the format of a bitmap with an index mapped to a specific delta value. The prefetch controller converts the bitmap to a list of deltas. To acquire the actual predicted address, the predicted deltas at time $t$ should add to the base address at time $t$.

**Prefetch throttling.** Prefetch degree is the number of cache lines to prefetch for each trigger. A2P predicts multiple deltas with the confidence of predictions, leading to a variable degree of prefetching, up to the size of the bitmap. Therefore, the prefetch controller throttles the prefetch by selecting the predictions with top $k$ confidence for a prefetch degree at $k$.

**Prefetch filtering.** Before issuing a prefetch, the predicted accesses need to be filtered to avoid repetitive prefetch requests and save bandwidth. The prefetch controller stores the recent prefetches and accesses in a filter table, using the replacement policy of FIFO to update.

### Memory Access Predictor

We use A2P as the memory access predictor. An advantage of the attention-based network is its high parallelizability compared to LSTM-based networks. There is no recurrent structure in the attention mechanism. Considering a fully parallel implementation of the memory access prediction using A2P, the latency can be estimated using the critical path based on Figs. 4 and 8 as:

$$\begin{aligned}Latency = {} & \underbrace{T_{delta} + T_{token}}_{\text{Preprocessing}} + \underbrace{T_{mm_l} + T_{av}}_{\text{Linear projection}} + \underbrace{T_{mm_h} + T_{av}}_{\text{Output head}} \\ & \underbrace{+ L \times [\, \underbrace{4T_{mm_a} + 3T_{av}}_{\text{Multi-head attention}} + T_{mm_f} + 2(T_{add} + T_{norm})],}_{\text{Transformer layer}} \end{aligned} \quad (5)$$

where $T_{delta}$ and $T_{token}$ are the preprocessing latency for calculating the delta sequence and tokenization, $T_{add}$ is the vector addition latency, $T_{mm_l}$ is the Linear projection latency, $T_{mm_h}$ is the MLP head latency, $T_{av}$ is the latency for activation functions, mask, and scale operations, $T_{mm_a}$ is the latency of multi-head attention, $T_{mm_f}$ is the feed-forward latency, $T_{norm}$ is the normalization latency. $L$ is the number of Transformer layers.

In this work, we implemented A2P using the configuration shown in Table 1. Assuming full parallelism, $T_{delta} = T_{token} = 1$ cycle, matrix multiplication $T_{mm} = 1 + \log_2 D = 7$ for $D = 64$, and $T_{av} = 1$ for an activation function using look-up table. This leads to the overall estimated latency of A2P inference as 102 cycles according to Eq. 5. For a given memory system in Table 4 ("Simulator"), the DRAM latency is estimated as 250 cycles, the predictor can provide suggested prefetches before the requested data is fetched and will not introduce extra stalls. Techniques for more efficient implementations such as replacing matrix multiplication by lookup tables [46] and combinational logics [47] can further reduce the model inference latency.

**Table 1** A2P model configuration

| Configuration | Value | Configuration | Value |
|---|---|---|---|
| History *N* | 9 | Network dimension | 64 |
| Transformer dimension | 64 | Transformer layer *L* | 2 |
| Transformer heads | 4 | MLP head layer | 1 |

## Hybrid Design with Temporal Prefetcher

Though we propose the notion of super page to expand the prediction range, A2P is still a spatial prefetcher: learning from the patterns within a spatial range. While A2P has high generalizability and can provide predictions for unseen patterns, it cannot predict memory accesses beyond a super page. Temporal prefetchers can predict the entire address space based on a record-and-replay mechanism. For example, irregular stream buffer (ISB) [15] records PC-specific memory access streams using tables and predicts future accesses based on the detection of matched streams. The large output space of ISB leads to higher coverage. However, ISB has low generalizability due to the extremely large input and output space. ISB also fails when there is no exact match between the streams and the table records.

We enhance the prefetching performance of A2P by combining it with a temporal prefetcher. For the combined prefetcher, A2P will be triggered and perform inference under two cases. (a) When a miss happens for an LLC request, A2P predicts deltas based on the current page address. (b) When a temporal prefetcher provides a memory access prediction, A2P predicts deltas based on the page address of the predicted access given by the temporal prefetcher. In this way, the combined prefetcher performs spatial-temporal prefetching and can achieve higher prefetch performance.

## Evaluation

### Benchmark Suite

We evaluate A2P and the baselines using the application traces generated from GAP [26] through simulator ChampSim [27]. The physical memory address configuration is shown in Table 2. After skipping the first 1 M instructions for warm-up, we use the next 40 M instructions for experiments. We use the first 20 M instructions for model training, the next 20 M instructions for testing. The statistics of the benchmark suite is shown in Table 3.

**Table 2** Memory address configuration

| Raw address | Page offset | Block offset |
|---|---|---|
| 64-Bit | 12-Bit | 6-Bit |

### Evaluation of Prediction

#### Metrics

Since the models can give multiple predictions with top *k* confidence, we use Precision@*k*, Recall@*k*, and Coverage@*k* to evaluate the prediction performance. These metrics are widely used to evaluate recommender systems [48, 49] and have a good fit for our problem.

- Precision@*k*: the proportion of correct predictions in the top-*k* predictions. A correct prediction refers to the case in which the predicted address is requested in the following k accesses.
- Recall@*k*: the proportion of correct predictions in the following *k* memory accesses. Repetitive memory accesses or incorrect repetitive access predictions can lead to a difference between Recall and Precision.
- Coverage@*k*: the cardinality of the set of all predictions over the entire set of addresses in testing. It measures a model's ability in covering the entire range of memory accesses for an application.

#### Implementation

To evaluate the effectiveness of our model, we implement four models as below. We denote bitmap labeling for a page range by "-BP", bitmap labeling for a super page range by "-BSP". Specifically, "-BSP-*n*" denotes bitmap labeling for super page with *n*-bit relaxation.

- LSTM-Delta. This is a widely used state-of-the-art model for memory access prediction [19, 22]. It takes delta tokens as input and uses the next delta as label. An LSTM model is trained and outputs deltas with top-*k* confidence in prediction.

**Table 3** Statistics of the benchmark suite

| Applications | # Addresses (K) | # Deltas (K) | # Pages (K) |
|---|---|---|---|
| BC | 218 | 202 | 5.1 |
| BFS | 316 | 165 | 15.7 |
| CC | 158 | 262 | 2.5 |
| PR | 311 | 683 | 4.9 |
| SSSP | 179 | 201 | 4.1 |

- LSTM-BP. An LSTM model takes delta tokens as input and learns from delta bitmap labels within a page. The output is deltas with top-$k$ confidence in the bitmap.
- Attention-BP. An attention-based model takes delta tokens as input and learns from delta bitmap labels within a page. The output is deltas with top-$k$ confidence in the bitmap.
- Attention-BSP (A2P). An attention-based model takes delta tokens as input and learns from delta bitmap within a super page. The output is deltas with top-$k$ confidence in the bitmap. We explore the super page size with relaxation bit $n = 1, 2, 3$, and 4.

For LSTM-Delta, the output dimension will be the number of deltas in Table 3, up to 683K. By using bitmap labeling within a page, we reduce the absolute value of deltas to a page range shifted by block offset: $2^{(12-6)} = 64$, leading to the output dimension to be 128 according to the mapping rules in Fig. 6. Bitmap labeling provides 5K× compression for output dimension. For super page with $n$-bit relaxation, the output dimension will be increased by $2^n$. For $n$ =1, 2, 3, and 4, the output dimensions are still significantly smaller than LSTM-Delta model.

## Results

Figure 9 shows the Precision, Recall and Coverage at $k$ top predictions for all the implemented models. We can make several observations. First, models using bitmap for labeling generally achieve higher Precision and Recall than LSTM-Delta which learns only from the next delta. Even for $k = 1$, bitmap labeling models that learns only within a spatial range outperform LSTM-Delta. This is because though LSTM-Delta learns from the entire address space, the model is hard to be trained and the interleaved patterns even hamper the model learning on patterns within a spatial range. Second, increasing $k$, the Precision and Recall first increase and then drop when $k > 3$. This is because when $k = 1$, a correct prediction requires exact match, when $k$ increases to 3, more candidates will be considered and there is higher probability to match the prediction and future accesses. However, more predictions with $k > 3$ will involve more predictions with low confidence, which leads to more incorrect predictions. Third, the relaxation for super page range contributes to the model performance on Precision and Recall. 1-bit and 2-bit relaxation shows notable performance improvement, while larger super page shows little contribution, or even negatively impacts the model performance. For example, the Precision@7 for super page with 4-bit relaxation (Attention-BSP-4) shows lower Precision than physical page range without relaxation (Attention-BP). In addition, from the Coverage plot, we observe that the Coverage of LSTM-Delta does not outperform other models for $k < 7$ though it learns from the entire address space. Only when low-confidence predictions are involved ($k > 5$), LSTM-Delta shows higher Coverage.

Figures 10 and 11 show the Precision@3 and Recall@3, respectively, for all the applications in detail. We use the best-performing super page size for each applications as the results in Attention-BSP. Using geometric mean, LSTM-Delta, LSTM-BP, Attention-BP, and Attention-BSP achieve Precision of 0.212, 0.366, 0.390, 0.443, respectively; achieve Recall of 0.211, 0.340, 0.368, 0.423, respectively. Attention-BSP model outperforms the baseline LSTM-Delta by 23.1% w.r.t. Precision and 21.2% w.r.t. Recall. We also observe that Attention-based model achieves higher Precision and Recall than the LSTM-based model when using the same labeling method (BP). Particularly, PR benefits the most from the relaxation of page size. This is because the memory access of PR shows notable spatial patterns beyond pages. Super page successfully enables the model to detect patterns in a larger range and contributes to the prediction performance of PR.

Figure 12 shows the Coverage@3 of all the models and for all the applications. It shows that LSTM-Delta, LSTM-BP, Attention-BP, and Attention-BSP achieve average Coverage of 0.802, 0.897, 0.898, 0.907, respectively. We observe that learning within a bitmap does not significantly

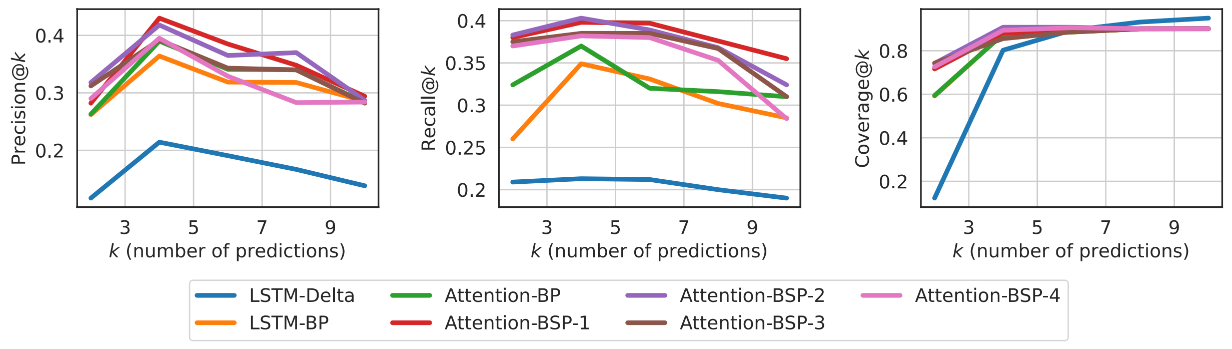


**Fig. 9** Precision, Recall, and Coverage at $k$ top predictions for A2P and baselines

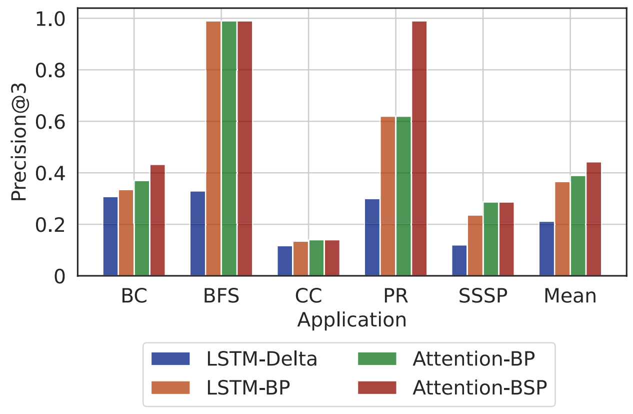


**Fig. 10** Precision@3 for A2P (attention-BSP) and baselines

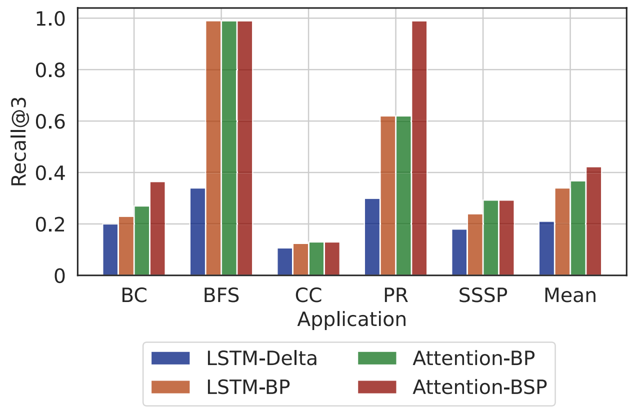


**Fig. 11** Recall@3 for A2P (attention-BSP) and baselines

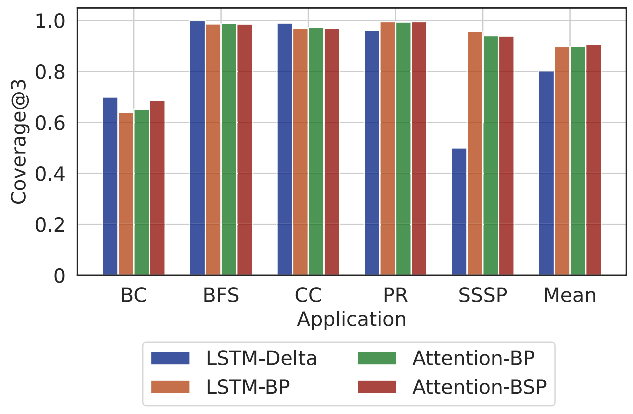


**Fig. 12** Coverage@3 for A2P (attention-BSP) and baselines

decrease the Coverage. Though for applications BC, BFS, and CC, models with bitmap labeling show slightly lower Coverage, for PR and SSSP these models show even higher Coverage than LSTM-Delta. Also super page contributes to the Coverage of BC. Overall, A2P achieves 10.4% higher Coverage than LSTM-Delta.

**Table 4** Simulation parameters of ChampSim

| Parameter | Value |
|---|---|
| CPU | 4 GHz, 4-wide OoO, 256-entry ROB, 64-entry LSQ |
| L1 I-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 | $t_{RP} = t_{RCD} = t_{CAS} = 50$, 2 channels 8 ranks, 8 banks, 32K rows, 1600MT/s |

## Evaluation of Prefetching

### Simulator

We use ChampSim [27] for the simulation experiments. ChampSim is a trace-based simulator that models modern out-of-order(OoO) core microarchitectures. The simulation parameters are shown in Table 4. The configuration is widely used in the literature [43, 50, 51]. We implement A2P and the baselines for LLC prefetching and disable prefetchers for other cache levels.

### Metrics

In measuring the performance of prefetcher, we use different metrics from memory access prediction. The performance of data prefetching is measured given a fixed number of instructions or memory accesses through simulation. There is no specific ground truth (label) to indicate if a prefetch is correct or not. When the simulation is finished, the recorded cache hits, prefetch hits, and IPC performance can be used to evaluate the prefetching performance.

**Prefetch accuracy.** If the prefetched line is a hit in the cache before being replaced, then it is termed as a prefetch hit. The prefetch accuracy refers to the percentage of prefetch hits to all prefetches:

$$\text{Prefetch qccuracy} = \frac{\text{Prefetch hits}}{\text{Prefetch hits} + \text{Prefetch misses}} \tag{6}$$

**Prefetch coverage.** Percentage of misses avoided due to the contribution of prefetching, the value can be calculated using the equation below:

$$\text{Prefetch coverage} = \frac{\text{Prefetch hits}}{\text{Prefetch hits} + \text{Cache misses}} \tag{7}$$

**IPC improvement.** The speedup can be deducted from the ratio of the improved new IPC and the base IPC. The IPC Improvement is computed as:

$$\text{IPC improvement} = \frac{\text{IPC}_{new} - \text{IPC}_{base}}{\text{IPC}_{base}} \quad (8)$$

## Baselines

We compare A2P with state-of-the-art prefetchers using table-based and ML-based prediction models. We implement the following prefetchers for this:

- **BO** (best-offset prefetcher) [11] is a spatial prefetcher. It updates scores for a pre-defined list of deltas within a page according to the records of recent requests. It predicts deltas with the highest scores when a cache miss occurs.
- **ISB** (irregular stream buffer) [15] is a temporal prefetcher. It learns temporally correlated memory accesses streams predicts using a record-and-replay mechanism.
- **VLDP** (variable length delta prefetcher) [13] is a spatial prefetcher. It uses multiple prediction tables which store predictions based on history memory accesses under different lengths, achieving variable degree prefetching.
- **Domino** [52] is a temporal prefetcher focused on enhancing prefetching effectiveness by exclusively utilizing the history of the last two miss addresses to identify a suitable match for prefetching.
- **Delta-LSTM** [19] is an ML-based prefetcher that uses a sequence of consecutive memory access deltas as input and uses LSTM as the predictor.
- **A2P** denotes our ML-based prefetcher that uses the model in this work as the predictor.
- **BO-ISB** is a hybrid prefetcher that combines BOP and ISB. The two prefetchers work in parallel.
- **A2P-ISB** is a hybrid prefetcher that combines A2P and ISB. The two prefetchers operate following the rules in "Hybrid Design with Temporal Prefetcher".

For fair comparison, we set the prefetch degree as 2 for all the implemented prefetchers, including individual prefetchers and hybrid prefetchers. For each trigger of prefetch, at most two cache lines can be prefetched.

## Results

Figure 13 shows the prefetch accuracy of the implemented prefetchers. A2P achieves prefetch accuracy at 57.9% w.r.t. geometric mean, higher than other individual prefetchers: BO at 36.3%, VLDP at 40.4%, ISB at 37.7%, Domino at 24.1% and Delta-LSTM at 46.7%. The hybrid prefetcher BO-ISB shows prefetch accuracy at 34.8%, which is slightly lower than individual BO and ISB. This is because the hybrid prefetcher compromises the prefetch accuracy with prefetch coverage by selecting different predictions from both the individual prefetchers. In contrast, the hybrid prefetcher A2P-ISB achieves 59.5% prefetch accuracy, which is the highest among all the prefetchers and slightly higher than A2P. This improvement results from our hybrid prefetch design described in "Hybrid Design with Temporal Prefetcher", which enables a further step of spatial prediction based on a temporal prediction.

Figure 14 shows the prefetch coverage of the implemented prefetchers. A2P achieves prefetch coverage at 32.1% w.r.t. geometric mean, higher than baseline individual prefetchers: BO at 6.7%, VLDP at 4.5%, ISB at 19.0%, Domino at 15.3% and Delta-LSTM at 21.4%. Hybrid prefetcher BO-ISB shows 26.5% prefetch coverage, higher than individual prefetchers BO and ISB. A2P-ISB achieves prefetch coverage at 39.2%, which is the highest among all the implemented prefetchers.

Figure 15 shows the IPC improvement using the implemented prefetchers. Compared with the individual baseline prefetchers, A2P achieves 18.4% IPC improvement, higher

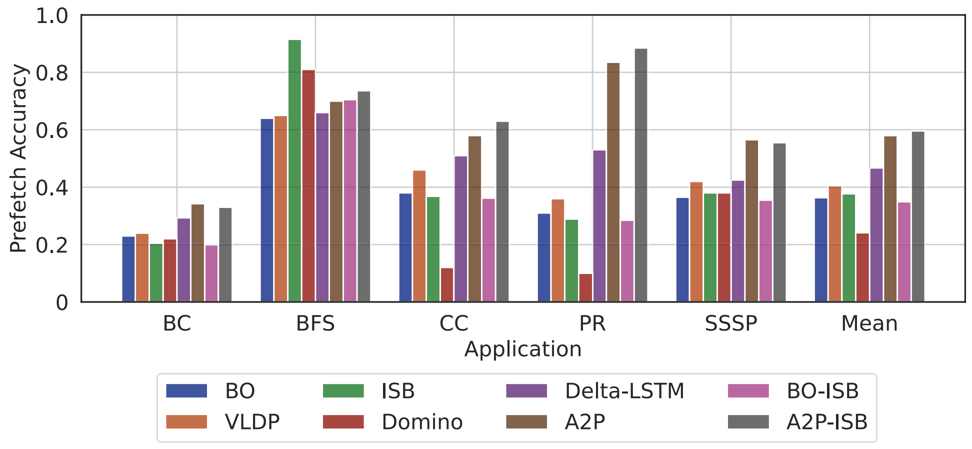


**Fig. 13** Prefetch accuracy of the implemented prefetchers

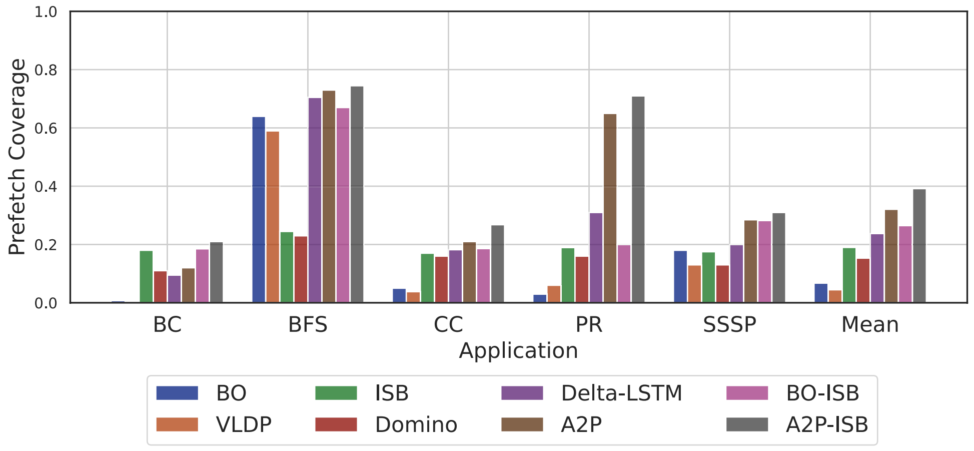


**Fig. 14** Prefetch coverage of the implemented prefetchers

than individual prefetchers BO, VLDP, ISB, Domino and Delta-LSTM which have 1.1%, 0.9%, 3.3%, 2.7% and 7.5% IPC Improvement. Hybrid prefetcher A2P-ISB shows the highest IPC improvement at 21.7% w.r.t. geometric mean, higher than the baseline hybrid prefetcher BO-ISB which shows 5.3% IPC improvement.

Overall, our A2P-based prefetching framework achieves higher prefetch accuracy and prefetch coverage compared with the baseline individual prefetchers, including rule-based prefetchers BO, ISB, and ML-based prefetcher Delta-LSTM. The higher prefetch accuracy and prefetch coverage lead to 15.0% higher IPC improvement compared with the best rule-based individual prefetcher ISB and 10.9% higher IPC Improvement compared with the ML-based prefetcher Delta-LSTM. By combining A2P and ISB and using our hybrid prefetch design, A2P-ISB gives the highest prefetch accuracy and prefetch coverage among all the prefetchers. A2P-ISB raises the IPC improvement of individual A2P by 3.3%, outperforming the baseline hybrid prefetcher BO-ISB by 16.3%.

## Discussion

### Advantages of A2P-Based Prefetcher

The proposed A2P-based data prefetching framework shows advantages in multiple aspects in accelerating graph analytics applications.

**Spatio-temporal locality**. A2P reads the consecutive history accesses and learns the temporal patterns, then predicts future accesses within a spatial region. By making use of the spatio-temporal locality, A2P shows higher adaptability when handling interleaved and complex memory access patterns, achieving higher Precision and Recall compared to the baselines.

**Parallelizability**. The multi-head attention mechanism is embarrassingly parallelizable. In contrast, LSTM, as a variant of the recurrent neural network, requires recurrent steps and hard to be paralleled. The parallelizability of A2P facilitates its hardware implementation for a practical ML-based hardware data prefetcher.

**Flexibility**. The integration of A2P into the memory system can be easily adapted for prefetching at other levels

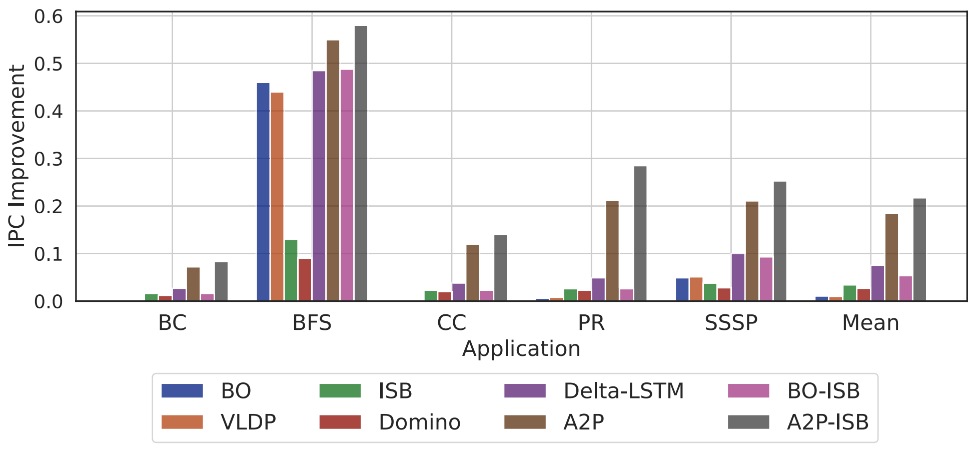


**Fig. 15** IPC improvement of the implemented prefetchers

of caches. The hybrid design enables A2P to collaborate with various temporal prefetchers, encompassing not only existing rule-based prefetchers but also potential future ML-based temporal prefetchers.

## Broader Impacts

Our approach has broader impacts in accelerating applications from more domains, prefetching on various hardware platforms, and extension to more prediction problems in systems.

**Application to more domains**. Beyond accelerating graph analytics applications, the proposed prefetching framework also offers potential to be applied to other domains. For example, computer vision applications can benefit from the proposed prefetching framework. The efficient data prefetching can significantly accelerate the training and inference phases of image processing models, leading to quicker image recognition [53–55], feature extraction [56], and video analysis [57].

**Extension to other platforms**. The A2P-based prefetching framework, initially designed for single core systems, can be further extended to various architecture. In multi-core shared-memory systems [58], individual core access patterns can be predicted, improving cache utilization and reducing contention. For GPU shared-memory systems [59], predictive models can optimize data movement between global memory and shared memory. Additionally, in heterogeneous HPC platforms [60], extended A2P that takes information from diverse memory hierarchies can accelerate applications running in distributed and parallel paradigms.

**Transferring to related problems**. Our investigation into A2P has primarily concentrated on hardware prefetchers. Notably, A2P can be transferred to related prediction processes within a system. For software data prefetching [61], A2P can analyze historical patterns of virtual address access to improve memory utilization. For instruction prefetching [62], the model can utilize its predictive capabilities to foresee upcoming instructions, ensuring a constant flow for the processor and enhancing instruction throughput. For branch prediction [63], A2P with a slightly modified binary output layer can learn from complex branch traces, foreseeing branch outcomes and reducing the performance impact of mispredictions.

## Conclusion

In this paper, we presented a novel attention-based data prefetching framework for graph analytics. We proposed A2P, which addresses the shortcomings of LSTM model in learning interleaved patterns and large output dimensions. The key ideas of our model are using an attention-based neural network for prediction, delta bitmaps for multi-label model training, and spatial range within a super page to constrain the output dimension. We propose a novel three-module framework that integrates A2P into an existing memory system. Experimental results show that A2P outperforms the state-of-the-art LSTM-based model by 23.1% w.r.t. Precision, 21.2% w.r.t. Recall, and 10.4% w.r.t. Coverage, at top 3 predictions. The prefetcher using A2P as the predictor provides 18.4% IPC improvement, outperforming state-of-the-art prefetchers BO by 17.2%, ISB by 15.0%, and Delta-LSTM by 10.9%. The hybrid prefetcher combining A2P and ISB achieves 21.7% IPC improvements, outperforming the hybrid of BO and ISB by 16.3%. In future work, we plan to explore the incorporation of more context information to improve the performance of memory access prediction and data prefetching.

**Acknowledgements** This work has been supported by the U.S. National Science Foundation under Grants CCF-1912680, PPoSS-2119816, and OAC-2209563. Distribution Statement A: Approved for public release. Distribution is unlimited.

**Funding** Open access funding provided by SCELC, Statewide California Electronic Library Consortium.

**Data availability** The dataset used in this research is from the ML-Based Data Prefetching Competition, part of the International Symposium on Computer Architecture (ISCA) 2021 MLArchSys workshop. https://utexas.app.box.com/s/2k54kp8zvrqdfaa8cdhfquvcxwh7yn85.

## Declarations

**Conflict of interest** On behalf of all authors, the corresponding author states that there is no conflict of interest.

## References

1. Lakhotia K, Kannan R, Pati S, Prasanna V. Gpop: a scalable cache-and memory-efficient framework for graph processing over parts. ACM Trans Parallel Comput (TOPC). 2020;7(1):1–24.
2. Drosou A, Kalamaras I, Papadopoulos S, Tzovaras D. An enhanced graph analytics platform (gap) providing insight in big network data. J Innov Digit Ecosyst. 2016;3(2):83–97.
3. Basak A, Li S, Hu X, Oh SM, Xie X, Zhao L, Jiang X, Xie Y. Analysis and optimization of the memory hierarchy for graph

processing workloads. In: 2019 IEEE international symposium on high performance computer architecture (HPCA). IEEE; 2019. p. 373–86.
4. Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G. Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM; 2010. p. 135–46.
5. Han M, Daudjee K. Giraph unchained: barrierless asynchronous parallel execution in pregel-like graph processing systems. Proc VLDB Endow. 2015;8(9):950–61.
6. Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein JM. Distributed graphlab: a framework for machine learning and data mining in the cloud. Proc VLDB Endow. 2012;5(8):716–27.
7. Buluç A, Gilbert JR. The combinatorial blas: design, implementation, and applications. Int J High Perform Comput Appl. 2011;25(4):496–509.
8. Siek JG, Lee L-Q, Lumsdaine A. The boost graph library: user guide and reference manual, portable documents. London: Pearson Education; 2001.
9. Byna S, Chen Y, Sun X-H. A taxonomy of data prefetching mechanisms. In: 2008 international symposium on parallel architectures, algorithms, and networks (i-span 2008). IEEE; 2008. p. 19–24.
10. Kumar S, Wilkerson C. Exploiting spatial locality in data caches using spatial footprints. In: Proceedings of the 25th annual international symposium on computer architecture (Cat. No. 98CB36235). IEEE; 1998. p. 357–68.
11. Michaud P. Best-offset hardware prefetching. In: 2016 IEEE international symposium on high performance computer architecture (HPCA). IEEE; 2016. p. 469–80.
12. Shevgoor M, Koladiya S, Balasubramonian R, Wilkerson C, Pugsley SH, Chishti Z. Efficiently prefetching complex address patterns. In: 2015 48th annual IEEE/ACM international symposium on microarchitecture (MICRO). IEEE; 2015. p. 141–52.
13. Kim J, Pugsley SH, Gratz PV, Reddy AN, Wilkerson C, Chishti Z. Path confidence based lookahead prefetching. In: 2016 49th annual IEEE/ACM international symposium on microarchitecture (MICRO). IEEE; 2016. p. 1–12.
14. Wenisch TF, Ferdman M, Ailamaki A, Falsafi B, Moshovos A. Practical off-chip meta-data for temporal memory streaming. In: 2009 IEEE 15th international symposium on high performance computer architecture. IEEE; 2009. p. 79–90.
15. Jain A, Lin C. Linearizing irregular memory accesses for improved correlated prefetching. In: Proceedings of the 46th annual IEEE/ACM international symposium on microarchitecture. 2013. p. 247–59..
16. Lim B, Zohren S. Time-series forecasting with deep learning: a survey. Philos Trans R Soc A. 2021;379(2194):20200209.
17. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J. Lstm: a search space odyssey. IEEE Trans Neural Netw Learn Syst. 2016;28(10):2222–32.
18. Hashemi M, Swersky K, Smith JA, Ayers G, Litz H, Chang J, Kozyrakis C, Ranganathan P. Learning memory access patterns. arXiv preprint arXiv:1803.02329 (2018).
19. Srivastava A, Lazaris A, Brooks B, Kannan R, Prasanna VK. Predicting memory accesses: the road to compact ml-driven prefetcher. In: Proceedings of the international symposium on memory systems. 2019. p. 461–70.
20. Zhang P, Srivastava A, Wang T-Y, De Rose CA, Kannan R, Prasanna VK. C-memmap: clustering-driven compact, adaptable, and generalizable meta-lstm models for memory access prediction. Int J Data Sci Anal 13, 3–16 (2022)
21. Srivastava A, Wang T-Y, Zhang P, De Rose CAF, Kannan R, Prasanna VK. Memmap: Compact and generalizable meta-lstm models for memory access prediction. In: Pacific-Asia conference on knowledge discovery and data mining. Springer; 2020. p. 57–68.
22. Hashemi M, Swersky K, Smith JA, Ayers G, Litz H, Chang J, Kozyrakis C, Ranganathan P. Learning memory access patterns. CoRR arXiv:1803.02329 (2018).
23. Zeyer A, Bahar P, Irie K, Schlüter R, Ney H. A comparison of transformer and lstm encoder decoder models for asr. In: 2019 IEEE automatic speech recognition and understanding workshop (ASRU). IEEE; 2019. p. 8–15.
24. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Advances in neural information processing systems. 2017. p. 5998–6008.
25. Webster JJ, Kit C. Tokenization as the initial phase in nlp. In: COLING 1992 volume 4: the 14th international conference on computational linguistics (1992).
26. Beamer S, Asanović K, Patterson D. The gap benchmark suite. arXiv preprint arXiv:1508.03619 (2015).
27. Gober N, Chacon G, Wang L, Gratz PV, Jimenez DA, Teran E, Pugsley S, Kim J. The championship simulator: Architectural simulation for education and competition. arXiv preprint arXiv:2210.14324. 2022 Oct 25.
28. McSherry F, Isard M, Murray DG. Scalability! but at what COST?. In15th Workshop on Hot Topics in Operating Systems (HotOS XV) 2015.
29. Shun J, Blelloch GE. Ligra: a lightweight graph processing framework for shared memory. ACM Sigplan Not. 2013;48:135–46.
30. Sundaram N, Satish N, Patwary MMA, Dulloor SR, Anderson MJ, Vadlamudi SG, Das D, Dubey P. Graphmat: high performance graph analytics made productive. Proc VLDB Endow. 2015;8(11):1214–25.
31. Nguyen D, Lenharth A, Pingali K. A lightweight infrastructure for graph analytics. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. ACM; 2013. p. 456–71.
32. Lumsdaine A, Gregor D, Hendrickson B, Berry J. Challenges in parallel graph processing. Parallel Process Lett. 2007;17(01):5–20.
33. Zhang K, Chen R, Chen H. Numa-aware graph-structured analytics. ACM SIGPLAN Not. 2015;50(8):183–93.
34. Grossman S, Litz H, Kozyrakis C. Making pull-based graph processing performant. In: Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming. ACM; 2018. p. 246–60.
35. Besta M, Podstawski M, Groner L, Solomonik E, Hoefler T. To push or to pull: on reducing communication and synchronization in graph computations. In: Proceedings of the 26th international symposium on high-performance parallel and distributed computing. ACM; 2017. p. 93–104.
36. Roy A, Mihailovic I, Zwaenepoel W. X-stream: edge-centric graph processing using streaming partitions. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. ACM; 2013. p. 472–88.
37. Zhu X, Han W, Chen W. Gridgraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In: 2015 USENIX annual technical conference (USENIX ATC 15). USENIX Association; 2015. p. 375–86. https://www.usenix.org/conference/atc15/technical-session/presentation/zhu
38. Zhou S, Lakhotia K, Singapura SG, Zeng H, Kannan R, Prasanna VK, Fox J, Kim E, Green O, Bader DA. Design and implementation of parallel pagerank on multicore platforms. In: High performance extreme computing conference (HPEC). IEEE; 2017. p. 1–6.
39. Page L, Brin S, Motwani R, Winograd T. The pagerank citation ranking: bringing order to the web. Technical report, Stanford InfoLab. 1999.
40. Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW, et al. The landscape of parallel computing research: a view from

berkeley. Technical report, UCB/EECS-2006-183, EECS Department, University of California, Berkeley (2006).
41. Vuduc R, Demmel JW, Yelick KA. Oski: A library of automatically tuned sparse matrix kernels. J. Phys.: Conf. Ser. 16 521.
42. Pingali K, Nguyen D, Kulkarni M, Burtscher M, Hassaan MA, Kaleem R, Lee T-H, Lenharth A, Manevich R, Méndez-Lojo M, et al. The tao of parallelism in algorithms. ACM Sigplan Not. 2011;46:12–25.
43. Zhang P, Srivastava A, Brooks B, Kannan R, Prasanna VK. Raop: recurrent neural network augmented offset prefetcher. In: The international symposium on memory systems (MEMSYS 2020). (2020)
44. Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, et al. An image is worth 16x16 words: transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020).
45. Chen T-F, Baer J-L. A performance study of software and hardware data prefetching schemes. ACM SIGARCH Comput Archit News. 1994;22(2):223–32.
46. Razlighi MS, Imani M, Koushanfar F, Rosing T. Looknn: neural network with no multiplication. In: Design, automation and test in Europe conference and exhibition (DATE), 2017. IEEE; 2017. p. 1775–80.
47. Nazemi M, Fayyazi, A, Esmaili A, Khare A, Shahsavani SN, Pedram M. Nullanet tiny: Ultra-low-latency dnn inference through fixed-function combinational logic. In: 2021 IEEE 29th annual international symposium on field-programmable custom computing machines (FCCM). IEEE; 2021. p. 266–7.
48. Chen M, Liu P. Performance evaluation of recommender systems. Int J Perform Eng. 2017;13(8):1246.
49. Silveira T, Zhang M, Lin X, Liu Y, Ma S. How good your recommender system is? A survey on evaluations in recommendation. Int J Mach Learn Cybern. 2019;10(5):813–31.
50. Bhatia E, Chacon G, Pugsley S, Teran E, Gratz PV, Jiménez DA. Perceptron-based prefetch filtering. In: 2019 ACM/IEEE 46th annual international symposium on computer architecture (ISCA). IEEE; 2019. p. 1–13.
51. Shi Z, Jain A, Swersky K, Hashemi M, Ranganathan P, Lin C. A hierarchical neural model of data prefetching. In: Proceedings of the 26th ACM international conference on architectural support for programming languages and operating systems. 2021. p. 861–73.
52. Bakhshalipour M, Lotfi-Kamran P, Sarbazi-Azad H. Domino temporal data prefetcher. In: 2018 IEEE International symposium on high performance computer architecture (HPCA). IEEE; 2018. p. 131–42.
53. Zhang J, Li C, Kosov S, Grzegorzek M, Shirahama K, Jiang T, Sun C, Li Z, Li H. Lcu-net: a novel low-cost u-net for environmental microorganism image segmentation. Pattern Recogn. 2021;115: 107885.
54. Zhang J, Li C, Yin Y, Zhang J, Grzegorzek M. Applications of artificial neural networks in microorganism image analysis: a comprehensive review from conventional multilayer perceptron to popular convolutional neural network and potential visual transformer. Artif Intell Rev. 2023;56(2):1013–70.
55. Li X, Li C, Rahaman MM, Sun H, Li X, Wu J, Yao Y, Grzegorzek M. A comprehensive review of computer-aided whole-slide image analysis: from datasets to feature extraction, segmentation, classification and detection approaches. Artif Intell Rev. 2022;55(6):4809–78.
56. Kulwa F, Li C, Zhang J, Shirahama K, Kosov S, Zhao X, Jiang T, Grzegorzek M. A new pairwise deep learning feature for environmental microorganism image analysis. Environ Sci Pollut Res. 2022;29(34):51909–26.
57. Chen A, Li C, Zou S, Rahaman MM, Yao Y, Chen H, Yang H, Zhao P, Hu W, Liu W, et al. Svia dataset: a new dataset of microscopic videos and images for computer-aided sperm analysis. Biocybern Biomed Eng. 2022;42(1):204–14.
58. Ma L, Agrawal K, Chamberlain RD. A memory access model for highly-threaded many-core architectures. Future Gener Comput Syst. 2014;30:202–15.
59. Yang Y, Xiang P, Mantor M, Rubin N, Zhou H. Shared memory multiplexing: a novel way to improve gpgpu throughput. In: Proceedings of the 21st international conference on parallel architectures and compilation techniques. 2012. p. 283–92.
60. Mittal S, Vetter JS. A survey of cpu–gpu heterogeneous computing techniques. ACM Comput Surv (CSUR). 2015;47(4):1–35.
61. Callahan D, Kennedy K, Porterfield A. Software prefetching. ACM SIGARCH Comput Archit News. 1991;19(2):40–52.
62. Falsafi B, Wenisch TF. A primer on hardware prefetching. Berlin: Springer Nature; 2022.
63. Smith JE. A study of branch prediction strategies. In: 25 years of the international symposia on computer architecture (selected papers). 1998. p. 202–15.