G-MAP: A Graph Neural Network-Based Framework for Memory Access Prediction

Abhiram Rao Gorle*, Pengmiao Zhang†, Rajgopal Kannan‡, Viktor K. Prasanna†
*Indian Institute of Technology Madras, Chennai, India
†University of Southern California, Los Angeles, California
‡DEVCOM Army Research Lab, Los Angeles, California
Contact: gabhiram@smail.iitm.ac.in, {pengmiao, prasanna}@usc.edu, rajgopal.kannan.civ@mail.mil

***Abstract*—Memory access prediction is a crucial problem in data prefetchers, as it helps us improve memory performance and reduce latency in computing systems. Existing works model the problem as a sequence prediction problem. This can be limited in its ability to capture complex patterns and dependencies in memory access behavior. In recent years, Graph Neural Networks (GNNs) have emerged as a promising technique for modeling and predicting complex relationships in graph-structured data. In this paper, we introduce G-MAP, a novel Graph Neural Network-based framework for Memory Access Prediction. First, we propose Mem2Graph, a novel approach mapping a memory access sequence to a graph representation, capturing both the spatial and temporal locality in the sequence. Second, we implement various GNNs for G-MAP, including Graph Convolutional Network (GCN), Gated Graph Sequence Neural Network (GG-NN), and Graph Attention Network (GAT). Those models take the graph generated from Mem2Graph as input and predict future memory address jumps (deltas). We evaluate the effectiveness of G-MAP using the SPEC 2006 benchmark. G-MAP using GG-NN shows the highest performance among all models, achieving 0.7526 F1-Score on the average, which is 10.77% higher than the Multi-Layer Perceptron baseline.**

***Index Terms*—memory access prediction, graph neural network**

I. Introduction

Memory latency is a major bottleneck in computer system performance, particularly due to the 'memory wall' problem [1], [2]. The memory wall problem arises as a result of the disparity between the rapid advancement of processor speeds and the enhancement of memory access speeds. As a result, the processor spends a significant amount of time waiting for data to be fetched from the memory, causing a slowdown in overall system performance.

Prefetching plays a vital role by accurately predicting future memory accesses, allowing preemptive data fetching to hide latency and enhance overall system performance [3]–[5]. The central aspect of prefetching is to be able to accurately predict future memory accesses as accurate memory access predictions provide valuable information for prefetching [6]–[8]. By accurately predicting upcoming memory accesses, prefetchers can initiate early data fetching and bring data into the cache or closer levels of the memory hierarchy before they are explicitly requested. This can hide the memory access latency and improve overall system performance.

ML-based algorithms have gained traction for the problem of memory access prediction. Rahman et al. [9] employed logistic regression and decision trees for the classification of patterns. In the realm of sequence modeling, the Long Short-Term Memory (LSTM) model has gained significant recognition due to its exceptional performance, as evidenced by a number of studies [10]–[13]. Prior investigations have also delved into the amalgamation of LSTM with existing prefetchers [6], meta-learning approaches [8], and attention mechanisms [14]. In a recent development, Zhang et al. [15] proposed TransFetch, an innovative framework that incorporates fine-grained memory address input and an attention-based model for the prediction of multi-label memory access, yielding state-of-the-art results surpassing other existing methodologies.

Recently, Graph neural networks (GNNs) [16] have become a crucial tool across multiple fields due to their ability to model complex relationships in graph structures. They have found applications in social network analysis [17], recommendation systems [18], natural language processing [19], sequence prediction [20], image classification [21], and more. In the context of memory access prediction, GNNs offer promising avenues by representing memory accesses as graphs and capturing dependencies between them to improve prediction accuracy.

Nevertheless, implementing GNNs for memory access prediction is a challenging task. First, designing effective graph representations is a significant challenge when applying GNNs to memory access prediction. Memory access patterns inherently possess complex dependencies and interactions among memory locations, making capturing such relationships in the graph representation essential. Additionally, incorporating spatial patterns within a page and temporal patterns across pages further adds to the complexity of designing an accurate graph representation. Another challenge is to ensure that GNN architectures effectively capture the dependencies between vertices in the memory access graph, enabling accurate predictions. Deciding on the appropriate GNN layers, aggregation functions, and message-passing mechanisms to capture long-range dependencies becomes an important task.

In this paper, we introduce *G-MAP*, a novel Graph Neural Network (GNN)-based framework for memory access prediction. Our approach begins by *Mem2Graph*, a novel approach that maps a memory access sequence to a graph representation. We map a window of memory access sequence to a directed graph, with each memory access as a node. To capture the temporal locality, we introduce edges from a node (representing memory access) to it's temporally subsequent memory accesses. To capture the spatial locality, we also link the nodes representing memory addresses on the same page as discussed further in Section IV-C. Our intent is to capture the inherent dependencies and relationships between memory locations by constructing the graph.

Subsequently, we implement multiple GNN models that use the generated memory access graphs as input and predict future memory access deltas (jumps of memory addresses) as output. These models include Graph Convolutional Network (GCN) [22], Gated GNN (GG-NN) [20], and Graph Attention Network (GAT) [23]. We also implement various aggregation functions including Sum, Max, and Mean for the G-MAP framework. Using SPEC CPU 2006 [24], we evaluate the models and aggregation functions and demonstrate the effectiveness G-MAP in memory access prediction.

Our main contributions can be summarized as follows:

- We propose G-MAP, a novel memory access prediction framework based on graph neural networks.
- We propose Mem2Graph, a novel graph representation of memory access sequences. By mapping each memory access as a node, connecting consecutive accesses and accesses on the same page, Mem2Graph captures both the spatial and temporal locality in memory accesses.
- We implement various GNN models (including GCN, GG-NN, and GAT) and aggregation functions (Sum, Max, and Mean) for the G-MAP framework.
- We demonstrate the effectiveness of the proposed framework using the SPEC 2006 benchmark. Results show G-MAP using GG-NN achieves the highest performance among all models, achieving F1-score at 0.7526, which is 10.77% higher than the MLP baseline.

II. Background

A. Memory Access Prediction

Memory access prediction is a vital aspect of computer architecture that aims to enhance the overall performance of memory systems. By accurately anticipating memory accesses, systems can fetch the required data from memory, mitigating the latency associated with memory operations. This technique finds applications in areas such as data prefetching, where predictions are utilized to anticipate future data demands and prefetch the required information into the cache hierarchy. Prior works have explored different prediction methods, including pattern-based predictors [25], stride predictors [26], and neural network predictors [6], [8], [27], among others.

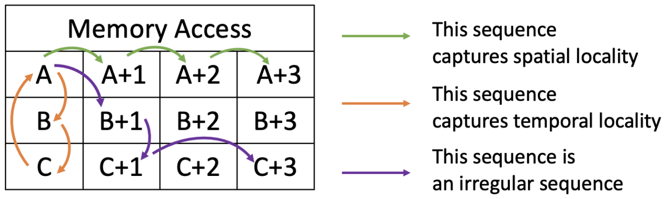
A general problem definition for memory access prediction is as follows. Let $X_t = \{x_1, x_2, ..., x_N\}$ be the sequence of N history block addresses at time t. Let $Y_t = \{y_1, y_2, ..., y_k\}$ be a set of k outputs associated with future k block addresses. A general goal is to approximate $P(Y_t|X_t)$, 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 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 [27], [28].

A *block delta* is defined as the block address difference between consecutive memory accesses. A Machine Learning (ML) model can be developed to learn this probability $P(Y_t|X_t)$. The vector of history accesses X_t is defined as the *input feature*, and the accessed *future addresses* Y_t are defined as the *output labels*. An ML model can be trained to approximate the true probability using samples of input features and output labels from a memory trace.

B. Locality of Reference

The *spatial locality* of memory reference refers to the property that access to a memory location indicates that a physically nearby location will be accessed with high probability in the near future [29]. As shown in the access sequence 1 of Figure 1, the subsequent memory access is one block away from the current block A. Spatial locality gives the insight that predicting strategies can perform well, even focusing only on a small fixed-size section.

The *temporal locality* of the memory reference means that the current memory access will likely be accessed again in the near future [29]. As is shown in the access sequence 2 of Figure 1, the recurrent accesses to block A, B, and C illustrate temporal locality, which provides clues for predicting memory accesses by recording and replaying. Sequence 3 of Figure 1 shows an example of an irregular pattern that hard to be detect using heuristic rules.



Memory Access
A
A+1
A+2
A+3
B
B+1
B+2
B+3
C
C+1
C+2
C+3
This sequence captures spatial locality
This sequence captures temporal locality
This sequence is an irregular sequence

Fig. 1: Spatial and temporal locality in memory accesses.

C. Graph Neural Network

Graph Neural Networks (GNNs) are proposed for representation learning on graphs $\mathcal{G}(\mathcal{V}, \mathcal{E})$. GNNs can learn from the structural information and vertex features and embed this information into a low-dimension vector representation, which can be used for many downstream tasks, such as node classification, link prediction, graph classification.

TABLE I: Graph Notations

Notation	Description	Notation	Description
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	input graph	v_i	i^{th} vertex
$\mathcal{V}$	set of vertices	e_{ij}	edge from v_i to v_j
$\mathcal{E}$	set of edges	L	number of layers
$\mathcal{N}(i)$	neighbors of v_i	$\boldsymbol{h}_i^{l-1}$	feature of v_i at l
$\boldsymbol{W}^l$	weight matrix of layer l	$\sigma()$	activation function

GNNs follow a message-passing paradigm (shown in Algorithm 1), which has two stages as described below.

Aggregate: Using the aggregate() function, each vertex recursively aggregates the feature vectors from the neighbours. The Aggregate function can be represented as follows:

$$\boldsymbol{a}_v^l = \text{Aggregate}\left(\boldsymbol{h}_u^{l-1} : u \in \mathcal{N}(v)\right) \tag{1}$$

where a_v^l is the aggregated vertex feature of the neighbourhood, h_u^{l-1} is the vertex feature in neighbourhood $\mathcal{N}(.)$ of vertex v.

Update: Using the update() function, each feature vector is updated to generate the updated feature vector. The Update function can be represented as follows:

$$\boldsymbol{z}_v^l = \text{Update}\left(\boldsymbol{a}_v^l, \boldsymbol{W}^l\right), \boldsymbol{h}_v^l = \sigma\left(\boldsymbol{z}_v^l\right) \tag{2}$$

where h_v^l is the vertex representation at the l^{th} layer and $\sigma(.)$ denotes the activation function.

Algorithm 1 GNN Aggregate-Update

```
Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, vertex features: $\left\{\boldsymbol{h}_1^0, \boldsymbol{h}_2^0, \ldots, \boldsymbol{h}_{|\mathcal{V}|}^0\right\}$
Output: Output vertex features $\left\{\boldsymbol{h}_1^L, \boldsymbol{h}_2^L, \ldots, \boldsymbol{h}_{|\mathcal{V}|}^L\right\}$
1: for $l = 1 \ldots L$ do
2:     for for each vertex $v \in \mathcal{V}$ do
3:         $\boldsymbol{a}_v^l = \text{Aggregate}\left(\boldsymbol{h}_u^{l-1} : u \in \mathcal{N}(v)\right)$
4:         $\boldsymbol{z}_v^l = \text{Update}\left(\boldsymbol{a}_v^l, \boldsymbol{W}^l\right)$, $\boldsymbol{h}_v^l = \sigma\left(\boldsymbol{z}_v^l\right)$
5:     end for
6: end for
```

III. Problem Formulation

Let $X_t = \{x_1, x_2, \ldots, x_N\}$ denote a sequence of N history block addresses at time t, where $x_t = \left\{b_t^1, b_t^2, \ldots, b_t^m, \ldots, b_t^{m+n}\right\}$ represents the block address with m-bit page address and n-bit block (cache line) index at time t. Let $D_t = \{d_1, d_2, \ldots, d_k\}$ be the set of k outputs associated with the unordered future k block deltas to the current block address. Our problem can be presented as: a) Map the input sequence X_t to a graph representation $\mathcal{G}(\mathcal{V}, \mathcal{E})$; b) Build a GNN-based predictive model that takes $\mathcal{G}(\mathcal{V}, \mathcal{E})$ as an input and predicts the future deltas D_t. The final predicted addresses $\widetilde{D}_t = \left\{\widetilde{d}_1, \widetilde{d}_2, \ldots, \widetilde{d}_k\right\}$ can be computed by the addition of current block address and the predicted future deltas.

IV. Approach

A. Overall Approach

The first stage involves using the *address segmentation* method proposed in TransFetch [30] to obtain the segmented memory access sequence. This is followed by two main steps. In the first step, we model this segmented memory access sequence to a graph. This *graph representation* is the input for our next step, the *GNN-based predictor*. This *GNN-based predictor* takes in the graph representation as an input, and outputs the future delta bitmaps. The overall approach is shown in Figure 2.

B. Memory Access Sequence Input

Consider a block address $a_t = \left\{b_t^1, b_t^2, \ldots, b_t^m, \ldots, b_t^{m+n}\right\}$ with an m-bit page address and n-bit block (cache line) index. This block address can be split into $K = \left\lceil \frac{m+n}{s} \right\rceil$, (where $\lceil . \rceil$ denotes the ceiling function) with at least $K-1$ segments having s bits each, and the last segment containing the remaining bits. Each of these s segments can be represented in an integer within $[0 - 2^s)$. This range can be tuned appropriately for direct model input. Each individual memory address can then can be represented as a vector in dimension K. The key intuition is that address segmentation avoids token dictionaries, helps in saving storage space, and can process unknown input classes [15].

We now look at the block address with a 7-bit page address and a 3-bit block index from Figure 2. If the block index size is used as the basis for segmentation, this block address can be split to $K = \left\lceil \frac{7+3}{3} \right\rceil = 4$ segments. Among these 4 segments, 3 have 3 bits each and the remaining segment has 1 bit. In this way, every segment can be represented with an integer ranging from 0 to 7 (both included) and each memory address would now be a vector in dimension 4. The sequence of segmented memory accesses is the input for the next stage.

C. Mem2Graph: a Graph Representation Mapping

We transform the input memory access sequence into a directed graph representation $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where a block address x_i with index i is mapped to a vertex $v_i \in \mathcal{V}$ in the graph. The attribute for a vertex v_i would be the segmented address vector of the block address x_i.

The second stage of Figure 2 illustrates the construction of the graph representation from a given sequence of memory accesses. The dashed and plain arrows indicate the temporal and spatial links, respectively. In order to capture both the temporal and spatial locality of the memory accesses, we construct the following links between the vertices:

- A vertex v_i is linked to its successive vertex v_{i+1}. This link captures the **temporal locality** in the memory access sequence.
- A vertex v_i is linked to vertex v_j, if index j is the lowest integer that is greater than index i for which v_i and v_j have the same page address. This link captures the **spatial locality** in a memory access sequence.

For instance, in Figure 2, the address with index $1, (A, 5)$ is connected to its successive address with index $2, (B, 3)$, and the next address within the same page with index $5, (A, 2)$. Similarly, the address with index $2 (B, 3)$ is connected to the addresses with index $3, (C, 1)$ and index $4, (B, 6)$. This way, we construct a directed graph representation of the sequence of segmented memory accesses, which will be the input for our Graph Neural Network (GNN).

D. GNN-based prediction

After multiple iterations of the *Aggregate-Update* paradigm, each vertex has an updated feature representation capturing its local and contextual information. Some of the state-of-the-art GNN models include Graph Convolutional Network

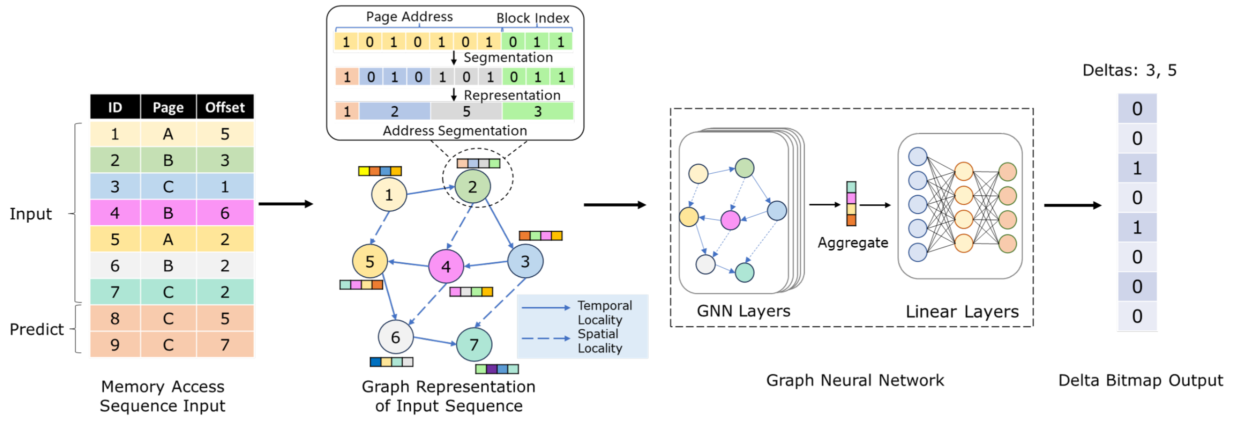


Fig. 2: The G-MAP framework: Memory accesses are mapped to a graph and processed by GNN for future access prediction.

(GCN) [31], Gated Graph Neural Network (GG-NN) [32], and Graph Attention Networks (GAT) [33].

1) Graph Convolutional Network (GCN): The GCN paper [31] analyzes graphs using its adjacency matrix. The self-connections are added to the adjacency matrix A to ensure the nodes are connected to themselves to get an updated adjacency matrix, $\widetilde{A}$. GCNs perform a localized convolutional operation on each node and the layer-wise propagation rule in can be expressed as:

$$h^{(l+1)} = \sigma\left(\tilde{A}h^{(l)}W^{(l)}\right)$$

where $W^{(l)}$ is a learnable parameter matrix and $h^{(l)}$ generalizes the node features at any arbitrary layer l.

A degree matrix $\tilde{D_{ii}}$ is defined as shown:

$$\tilde{D_{ii}} = \Sigma_j(A_{ij}) \tag{3}$$

where A_{ij} is the element of the adjacency matrix representing the connection between nodes v_i and v_j.

Using this degree matrix, an augmented adjacency matrix is formulated using $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. The layer-wise propagation rule can now be redefined as follows:

$$h^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}h^{(l)}W^{(l)}\right) \tag{4}$$

Graph Convolutional Networks (GCNs) leverage localized convolutional operations and information aggregation to learn node representations effectively.

2) Gated Graph Sequence Neural Networks (GG-NN): GG-NNs iteratively update the node representations based on information from neighboring nodes using GRUs (Gated Recurrent Units). The propagation rule for a single time step can be expressed as:

$$h_i^{(t)} = GRU\left(x_i, h_i^{(t-1)}\right) \tag{5}$$

where $h_i^{(t)}$ is the representation of node v_i at time step t, x_i is its input feature vector, and $h_i^{(t-1)}$ is the previous representation of v_i at time step $t-1$.

The GRU consists of two gating mechanisms: an update gate z and a reset gate r. These gates control the flow of information by deciding how much of the previous state is retained and how much of the new state is incorporated. Based on the reset gate, the GRU computes the new candidate state (h) as follows:

$$h_i^{(t)} = \tanh\left(Wx_i + U\left(r_i^{(t)} \odot h_i^{(t-1)}\right)\right) \tag{6}$$

where W, U are weight matrices, and $\odot$ denotes element-wise multiplication. Finally, the update gate is used to blend the previous state and the candidate state, yielding the updated node representation:

$$h_i^{(t)} = \left(1 - z_i^{(t)}\right) \odot h_i^{(t-1)} + z_i^{(t)} \odot h_i^{(t)} \tag{7}$$

By doing this over multiple iterations, GG-NNs capture the dependencies of the data.

3) Graph Attention Network (GAT): In GATs, attention coefficients represent the importance of the neighboring nodes' features in influencing a particular node's representation. The first step in the Graph Attention Layer involves applying a linear transformation using a weighted matrix W to the feature vectors of the nodes. Next, the attention coefficients a_{ij} between nodes v_i and v_j are calculated as shown below:

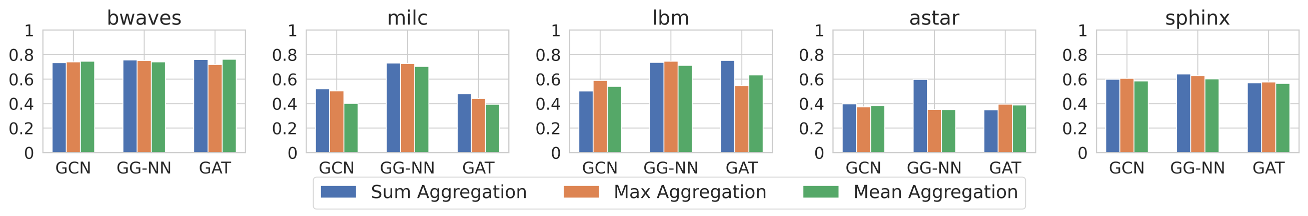
$$a_{ij} = \text{Softmax}\left(\text{LeakyReLU}\left(\mathbf{W_a}^T \cdot \left[Wh_i^l \oplus Wh_j^l\right]\right)\right) \tag{8}$$

where $\mathbf{W_a} \in \mathbb{R}^{2d'}$ and $W \subseteq \mathbb{R}^{d' \times d}$ are learned parameters, d' is the embedding dimension, and $\oplus$ is the vector concatenation operation. LeakyReLU is a leaky rectified linear unit activation function. The final step involves aggregating this information and passing it through an activation layer. The propagation rule in a single graph attention layer is as shown:

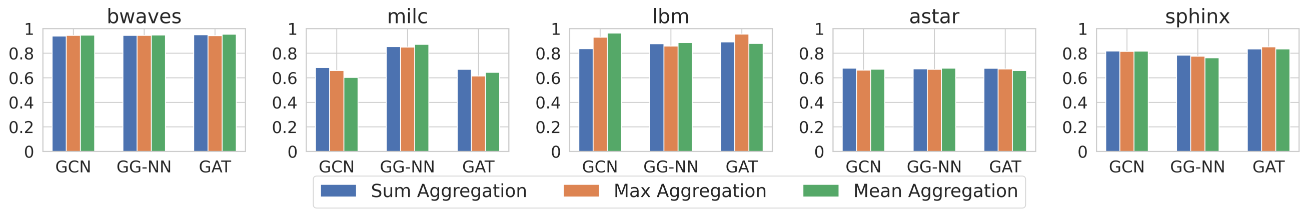
$$h_i^{(l)} = \sigma\left(\Sigma_j a_{ij} W^{(l)} h_j^{(l-1)}\right) \tag{9}$$

where $h_i{}^{(l)}$ is the representation of node v_i at the l^{th} layer, $W^{(l)}$ is the weight matrix at the l^{th} layer, and σ denotes the activation function.

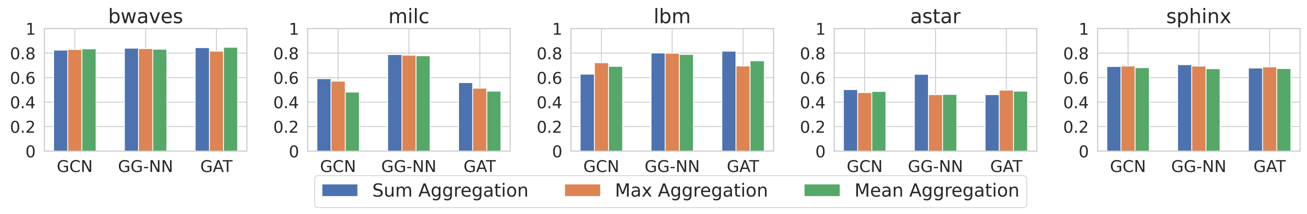
After the GNN layer, all the feature vectors are aggregated to a vector which becomes the input to the linear layers for classification, which predicts the output probabilities of deltas.



(a) Precision



(b) Recall



(c) F1-score

Fig. 3: Multi-label memory access prediction performance of various GNN models under different aggregation approaches.

E. Aggregation Methods

In various learning tasks involving GNNs, an effective aggregation of node features into a graph-level representation via aggregation functions is necessary [34]. These are usually simple functions designed such that the resulting hypothesis space is permutation invariant. We use and compare the performance of three aggregation functions: Sum, Max, and Mean.

$$\begin{aligned} \text{Sum} &= \sum_{j \in \mathcal{N}(i)} \mathbf{W}_j \cdot x_j \\ \text{Max} &= \max_{j \in \mathcal{N}(i)} \{\mathbf{W}_j \cdot x_j\} \\ \text{Mean} &= \frac{\sum_{j \in \mathcal{N}(i)} \mathbf{W}_j \cdot x_j}{|\mathcal{N}(i)|} \end{aligned} \tag{10}$$

where $\mathcal{N}(i)$ is the neighborhood of node v_i.

F. Delta Bitmap Output

The GNN layers take the memory graph generated from Mem2Graph process as input. The labels are delta bitmaps [15], which collects future memory access jumps (deltas) to the current address and convert the deltas to a bitmap. The bitmap can represent the appearance for a range of deltas (e.g. ± 128). In training, deltas appearing in a following window of accesses will set a 1 in the bitmap corresponding location, otherwise 0. In inference, the GNN models predict the probability of each bit being 1 in the bitmap.

G. Loss Function

Using Delta Bitmap as labels, the prediction models are trained for multi-label classification, each inference provides multiple memory access delta predictions. We use the binary cross-entropy loss to optimize the model, as shown below:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} y_i \log\left(p\left(y_i\right)\right) + \left(1 - y_i\right) \log\left(1 - p\left(y_i\right)\right) \tag{11}$$

where y_i is the label and $p\left(y_i\right)$ is the probability that we predict for sample i being True.

V. Experiments

A. Experimental Setting

We evaluate our approach using the application traces generated from benchmarks SPEC CPU 2006 [24]. After skipping 1M instructions for warmup, we use the first 5M instructions for training the model and the next 5M instructions for evaluation. The benchmark statistics are shown in Table II, including the unique number of block addresses, block address deltas, and page addresses.

B. GNN Models Implemented

We implement the GNN models using PyTorch Geometric [35]. We train all the modes using Adam optimizer [36] for 20 epochs, the batch size is 256. We set the memory access sequence length as 10, including 9 history memory access addresses and 1 current memory address. We collect future deltas

TABLE II: SPEC 2006 Benchmark Memory Trace Statistics

Trace	# Addr	# Delta	# Page
bwaves	236.2K	14.3K	3.7K
milc	169.9K	15.7K	19.8k
lbm	80.7K	0.5K	1.3K
astar	17.4K	11.2K	3.9K
sphinx	44.0K	9.7K	2.0K

within a page in a window of 128 future memory accesses. We predict deltas in a range of ±128. The configuration is shown in table III.

TABLE III: Configuration of the implemented GNN models

Model	Layers	Dimensions
GCN	2 GCNConv, 2 Linear	(32,32,64,64)*
GG-NN	2 GatedGraphConv, 2 linear	(32,32,64,64)
GAT	2 GATConv, 2 Linear	(16×2,32×1,64,64)**

*Hidden dimension for (GNN 1, GNN 2, linear 1, linear 2).
**GAT layer is shown as dimension×heads.

C. Metrics

We evaluate the proposed approach using the following metrics: *precision*, *recall*, and *F1-score* [37], to evaluate the memory access prediction performance of the GNN models.

D. Results

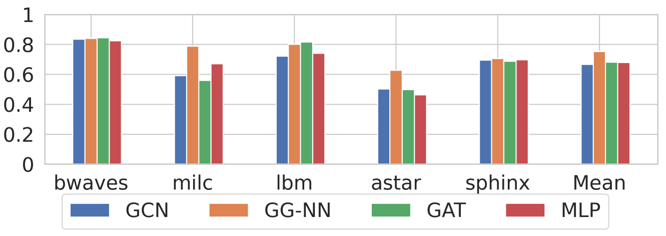


Fig. 4: F1-score of the aggregation-tuned GNNs and MLP.

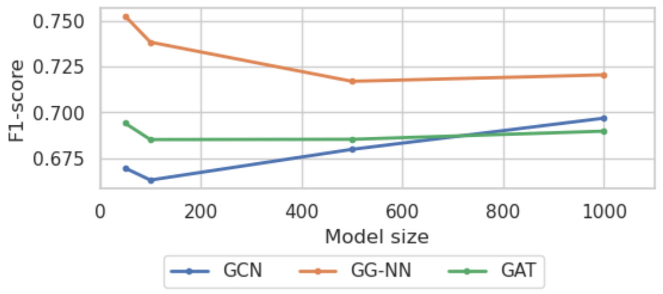


Fig. 5: Influence of model size on model performance.

1) Evaluation of Aggregations: We evaluate the performance of Sum, Max, and Mean aggregation functions in various GNN models for various applications. The precision, recall and F1-score is shown in Figure 3. On average, GCN using Sum, Max, and Mean as the aggregation functions achieves an F1-score of 0.6476, 0.6594 and 0.6359, respectively. Similarly, GG-NN achieves an F1-score of 0.7526, 0.715, and 0.7073, respectively. Under similar settings, GAT achieves an F1-score of 0.6721, 0.6424 and 0.6475, respectively. Overall, Max is the best performing aggregator for GCN, and Sum is the best performing aggregator for GG-NN and GAT.

2) Evaluation of GNN Models: Using the best-performing aggregation functions explored above, the F1-score of all models the and applications are shown in Figure 4. We compare the various GNN models as well as an MLP model that only has two linear layers without graph neural network processing steps. Result show that GG-NN achieves the highest average F1-score across all applications at 0.7526, higher than GAT at 0.6819, GCN at 0.6691, and MLP at 0.6794. GG-NN shows 10.77% improvement compared with MLP without graph mapping and graph-based processing.

3) Influence of GNN Model Sizes: We further explored the influence of GNN model sizes to the prediction performance. By tuning the number of model layes and dimensions, we design four level of model sizes: 50K, 100K, 500K, and 1M parameters. GG-NN outperforms GCN and GAT for all the tested model sizes, demonstrating its high capacity in learning from streaming patterns in a graph. Notably, the model performance does not always improve with larger model size. Using GG-NN, a small model with 50K parameter achieves the highest F1-score among the the models under various sizes.

VI. Conclusion

In this paper, we introduced G-MAP, a novel graph neural network framework for the task of memory access prediction. We represented the memory access sequences as graphs by mapping each memory access as a node and linked both the temporally subsequent memory access and the spatially subsequent access within a page. We also implemented various graph neural network models that take the graph representation of the memory access sequence as input and predict the future memory access deltas. G-MAP, using GG-NN as the backbone achieves an F1-score of 0.7526 on average, which is 10.77% higher than the MLP baseline without any graph mapping and GNN layer. In the future, we will optimize the mapping from memory accesses to graphs by introducing weights and types in edges and building heterogeneous-edge graphs to better capture the memory patterns.

Acknowledgment

This work was supported by Army Resarch Lab (ARL) under award number W911NF2220159 and National Science Foundation (NSF) under award number OAC-2209563.

References

[1] W. A. Wulf and S. A. McKee, “Hitting the memory wall: Implications of the obvious,” *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[2] C. Carvalho, “The gap between processor and memory speeds,” in *Proc. of IEEE International Conference on Control and Automation*, 2002.

[3] M. Dubois, M. Annavaram, and P. Stenström, *Parallel computer organization and design*. cambridge university press, 2012.

[4] S. P. Vander Wiel and D. J. Lilja, “When caches aren’t enough: Data prefetching techniques,” *Computer*, vol. 30, no. 7, pp. 23–30, 1997.

[5] S. Byna, Y. Chen, and X.-H. Sun, "A taxonomy of data prefetching mechanisms," in *2008 International Symposium on Parallel Architectures, Algorithms, and Networks (i-span 2008)*. IEEE, 2008, pp. 19–24.

[6] P. Zhang, A. Srivastava, B. Brooks, R. Kannan, and V. K. Prasanna, "Raop: Recurrent neural network augmented offset prefetcher," in *The International Symposium on Memory Systems*, 2020, pp. 352–362.

[7] P. Zhang, R. Kannan, X. Tong, A. V. Nori, and V. K. Prasanna, "Sharp: Software hint-assisted memory access prediction for graph analytics," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–8.

[8] P. Zhang, A. Srivastava, T.-Y. Wang, C. A. De Rose, R. Kannan, and V. K. Prasanna, "C-memmap: clustering-driven compact, adaptable, and generalizable meta-lstm models for memory access prediction," *International Journal of Data Science and Analytics*, pp. 1–14, 2021.

[9] S. Rahman, M. Burtscher, Z. Zong, and A. Qasem, "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*, Aug. 2015, pp. 383–389.

[10] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," *CoRR*, vol. abs/1803.02329, 2018. [Online]. Available: http://arxiv.org/abs/1803.02329

[11] L. Peled, U. Weiser, and Y. Etsion, "A neural network memory prefetcher using semantic locality," *arXiv preprint arXiv:1804.00478*, 2018.

[12] Y. Zeng and X. Guo, "Long short term memory based hardware prefetcher: a case study," in *Proceedings of the International Symposium on Memory Systems*, 2017, pp. 305–311.

[13] P. Braun and H. Litz, "Understanding memory access patterns for prefetching," in *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA*, 2019.

[14] Z. Shi, A. Jain, K. Swersky, M. Hashemi, P. Ranganathan, and C. Lin, "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, pp. 861–873.

[15] P. Zhang, A. Srivastava, A. V. Nori, R. Kannan, and V. K. Prasanna, "Fine-grained address segmentation for attention-based variable-degree prefetching," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, 2022, pp. 103–112.

[16] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[17] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The world wide web conference*, 2019, pp. 417–426.

[18] W. Fan, Y. Ma, Q. Li, J. Wang, G. Cai, J. Tang, and D. Yin, "A graph neural network framework for social recommendations," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 5, pp. 2033–2047, 2020.

[19] L. Wu, Y. Chen, K. Shen, X. Guo, H. Gao, S. Li, J. Pei, B. Long *et al.*, "Graph neural networks for natural language processing: A survey," *Foundations and Trends® in Machine Learning*, vol. 16, no. 2, pp. 119–328, 2023.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[21] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu, "Vision gnn: An image is worth graph of nodes," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8291–8303, 2022.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[23] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[24] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation," *Web Copy: http://www. glue. umd. edu/ajaleel/workload*, 2010.

[25] G. Ayers, H. Litz, C. Kozyrakis, and P. Ranganathan, "Classifying memory access patterns for prefetching," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 513–526.

[26] D. Joseph and D. Grunwald, "Prefetching using markov predictors," in *Proceedings of the 24th annual international symposium on Computer architecture*, 1997, pp. 252–263.

[27] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," *arXiv preprint arXiv:1803.02329*, 2018.

[28] A. Srivastava, A. Lazaris, B. Brooks, R. Kannan, and V. K. Prasanna, "Predicting memory accesses: the road to compact ml-driven prefetcher," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 461–470.

[29] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," in *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No. 98CB36235)*. IEEE, 1998, pp. 357–368.

[30] P. Zhang, A. Srivastava, A. V. Nori, R. Kannan, and V. K. Prasanna, "Fine-grained address segmentation for attention-based variable-degree prefetching," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, ser. CF '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 103–112. [Online]. Available: https://doi-org.libproxy2.usc.edu/10.1145/3528416.3530236

[31] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[32] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2017.

[33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[34] D. Buterez, J. P. Janet, S. J. Kiddle, D. Oglic, and P. Liò, "Graph neural networks with adaptive readouts," 2022.

[35] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] D. Powers, "Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation," *Mach. Learn. Technol.*, vol. 2, 01 2008.