Neural Hierarchical Sequence Model for Irregular Data Prefetching

Zhan Shi
The University of Texas at Austin
zshi17@cs.utexas.edu

Akanksha Jain
The University of Texas at Austin
akanksha@cs.utexas.edu

Kevin Swersky
Google Research
kswersky@google.com

Milad Hashemi
Google Cloud
miladh@google.com

Parthasarathy Ranganathan
Google Cloud
parthas@google.com

Calvin Lin
The University of Texas at Austin
lin@cs.utexas.edu

Abstract

Data prefetchers are common hardware structures that improve processor performance by hiding the long latency of DRAM memory accesses. In particular, data prefetchers predict the memory addresses of data that the program will access in the near future. From a machine learning perspective, the challenge is to accurately predict memory accesses that span across a huge address space. To address this challenge, this paper introduces a neural hierarchical sequence model that can accommodate the vast input space and that makes pure data addresses a viable feature for neural networks.

1 Introduction

The performance of many programs is limited by the high cost of accessing memory. Prefetchers mitigate this overhead by predicting the memory addresses that the processor will need and fetching them ahead of time. While prefetchers that target sequential or strided memory accesses are widely deployed in commercial microprocessors, prefetching for *irregular* memory accesses—memory accesses that arise from indirect arrays or pointer-based data structures, such as linked-lists, trees and graphs—has not been as successful.

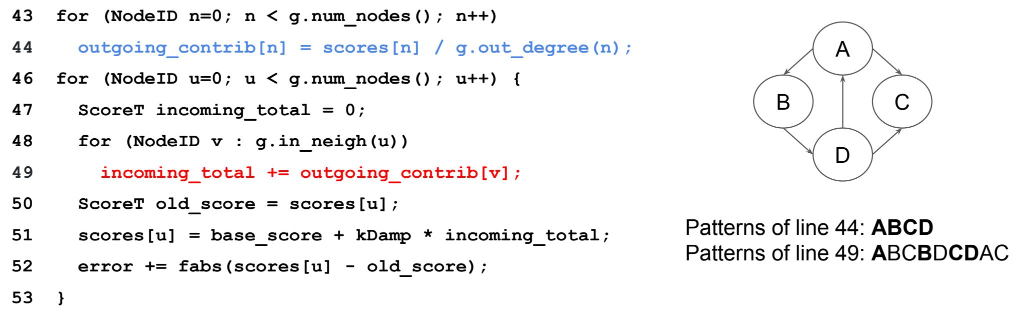
In this paper, we use deep learning to improve prefetching for these hard-to-predict irregular memory sequences. In particular, we view irregular prefetching as a sequence prediction task: Given a sequence of past memory accesses A, B, C, predict the next addresses in the sequence, say, D, E, F. Unfortunately, formulating prefetching as a learning problem presents two significant challenges. First, the vocabulary size is enormous, as the prefetcher must choose from among of millions of possible addresses [2]. By contrast, the vocabulary size of natural language tasks is only 60K [1]. Second, it is non-trivial to associate the most predictable future memory access with a given sequence of memory accesses. For example, the input sequence A, B, C can be associated with D, E, or F, but some of those output addresses may be more predictable than others depending on the underlying correlation in the program that generated this sequence.

To reduce the vocabulary size, Hashemi et al., instead predict deltas between consecutive addresses in the sequence, and they partition the address space spatially, such that, addresses that are close together numerically belong to the same cluster [2]. While their scheme effectively captures local and delta-based correlations, their formulation precludes the prediction of irregular acceses that often span the entire memory footprint.

To model the entire memory space with one model, we predict absolute addresses, and to manage the complexity of the large vocabulary size, we present a neural hierarchical sequence prefetching model (NHS). The key idea is to predict memory addresses hierarchically by first predicting the page of the predicted address and then predicting the offset within the predicted page. Although not practical for depolyment in hardware, NHS significantly outperforms state-of-the-art irregular prefetchers, improving prediction accuracy from 59.4% to 75.1% and improving performance from 28.3% to 40.1%. We also show that the problem formulation is the key to designing an efficient model, as we show that compared to a non-hierarchical model, NHS reduces training cost by $10\times$ and model size by $25\times$.

2 Related Work and Motivation

A common technique to predict irregular accesses is to learn pairwise correlation between consecutive elements in an addresss sequence. Such prefetchers are commonly referred to as *address correlation-based prefetchers* [7, 3]. For example, if address A is likely to be followed by address B, then an address correlation-based prefetcher will prefetch B when it sees A. Unfortunately, state-of-the-art address correlation-based prefetchers are limited because they consider a short history of just one memory address [7, 3], even though long histories have been found useful for other hardware prediction problems, such as branch prediction [5] and cache replacement [6].



```
43  for (NodeID n=0; n < g.num_nodes(); n++)
44    outgoing_contrib[n] = scores[n] / g.out_degree(n);
46  for (NodeID u=0; u < g.num_nodes(); u++) {
47    ScoreT incoming_total = 0;
48    for (NodeID v : g.in_neigh(u))
49      incoming_total += outgoing_contrib[v];
50    ScoreT old_score = scores[u];
51    scores[u] = base_score + kDamp * incoming_total;
52    error += fabs(scores[u] - old_score);
53  }
```

A
B
C
D
Patterns of line 44: **ABCD**
Patterns of line 49: **A**BC**B**D**CD**AC

Figure 1: Core code snippet of PageRank from the GAP benchmark suite (left) and a simple input graph example (right). We focus on the two lines that appear in blue and red. A limited history is sufficient for the memory load on Line 44, which represnts a simple access pattern that traverses all nodes in the graph. By contrast, the acceses on Line 49 are more complex, as it traverses the neighboring nodes of each node in the graph; thus, the memory acceses on Line 49 require an understanding of the neighboring sequences.

2.1 The Importance of a Long History

Figure 1 shows that prefetching can benefit from using a long history of memory addresses as its feature. In particular, we focus on line 49, which traverses the neighbor nodes of each node in the graph. To better predict the access sequence of line 49, the prefetcher needs to understand the sequence of neighbor nodes. For example, for the simple input graph shown in Figure 1, the sequences of neighbor nodes are ABC for A, BD for B, C for C, and DAC for D. Thus, the last address is not sufficient for predicting the next access. Instead, the prefetcher needs to look at a longer sequence of past accesses.

2.2 Modeling Memory Space

A deep sequence model can effectively leverage long histories, but it is difficult to apply a deep sequence model on a memory access history when the addresses come from a vast memory address space [2, 6]. Recent studies [2] formulate data prefetching as a machine learning problem, observing that the address space is extremely sparse, with only O(10M) cache block miss addresses appearing out of the entire 2^{64} physical address space; unfortunately, this problem is a poor fit for normalization because prefetching becomes ineffective with any loss of precision in the address bits.

Therefore, Hashemi et al. treat the address space as a huge and discrete vocabulary to perform classification [2]. This formulation is similar to non-learning based irregular prefetchers [7, 3], where each address is represented as a separate symbol. However, the order of magnitude of addresses prevents the direct use of a classification model. For example, on a short 100M trace of *mcf*, an off-the-shelf LSTM model model with a 2M vocabulary takes 12 hours for each epoch on 1 Nvidia P100 GPU, thus takes months to converge. Moreover, to accommodate the 2M vocabulary, the embedding size is reduced to 64, which will otherwise exhaust the 16G memory.

Partitioning the memory space and using frequent deltas [2] cannot meet our goal of capturing irregular access patterns that span a huge memory region with rare deltas. Thus, to make use of data addresses as a feature, we aim to reduce the number of unique classes in the classification model while still modeling the entire memory space with absolute addresses.

3 Our Approach

3.1 Neural Hierarchical Sequence Model

Figure 2 shows our new Neural Hierarchical Sequence Model (NHS). PC [1] and address sequences are used to represent the memory access stream, where to reduce the number of unique classes, the address sequence is split into a page sequence and an offset sequence that are embedded separately. A page-aware offset embedding is obtained through a conditional attention embedding layer, which is concatenated with the pc embedding and page embedding to form the input representation. Two separate attention-based LSTM layers take the same input representation to independently predict an output page and offset, which form the final prediction.

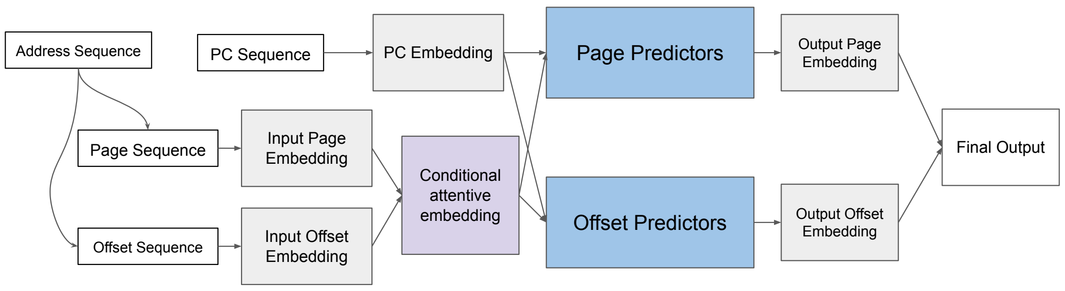


Figure 2: Neural Hierarchical Sequence Model.

3.2 Offset Aliasing and Conditional attention Embedding

Offset aliasing. Splitting an address into its page and offset effectively reduces the number of classes in the input and output space by 30-60×, but induces a problem during learning that we refer as "offset aliasing". As a simple example, there are two addresses X and Y that have different page numbers P_X and P_Y but the same offset O. At a high level, the same offset across pages may have different patterns that requires different representations depending on the page number. In reality, during the optimization process, the embedding of offset O can be updated in different directions by addresses X and Y. This becomes a severe problem as applications often have hundreds of thousands or even millions unique pages, and more extreme for some GAP benchmarks where every offset is toggled by nearly all pages, which leads to an offset aliasing among nearly all pages.

Conditional attention embedding layer. We propose a conditional attention embedding (CAE) layer to resolve the offset aliasing issue. The high-level idea is to have an offset embedding that is conditioned on both the offset and the page without the needs for embedding each (page, offset) pair, which is equivalent to embedding all unique addresses. To this end, we added an attention layer between the page and offset embedding and set the size of offset embedding to be 5-1000× of that of

[1]the address of the instruction that issued the memory request, also called the *program counter* (PC)

the page embedding. Figure 3 illustrates the CAE layer with a 200-dimension page embedding and 1000-dimension offset embedding. The 1000-d offset embedding is divided into 5 sub-embeddings, which is of the same size of the page embedding to perform attention. Attention weights are computed based on the page embedding and each offset sub-embedding with dot product, and a final page-aware offset embedding is obtained by weighted summing all the offset sub-embeddings.

In our design, as the number of offsets are fixed (64) for all programs and is much less than the number of pages for all programs, conditional attention embedding significantly reduces the number of classes in both the input and output space, thus reduces the model size the training cost.

3.3 Training Labels and Metrics

The benefits of providing a good data address as training label are two-fold. First, using more predictive data addresses improves model accuracy. Second, using timely and not-too-early data addresses as label translates the same accuracy into better performance improvements. Unlike previous work [2] that trains the model to predict the next address in the global access stream, we train NHS to predict the next address that will be accessed by the last PC in the sequence. This labeling scheme is similar to PC localization used in Irregular Stream Buffer (ISB) [3] in training, but in testing we explicitly measure the accuracy of predicted addresses being used by the specific PC, where ISB only cares about if the predicted address is used by any PC.

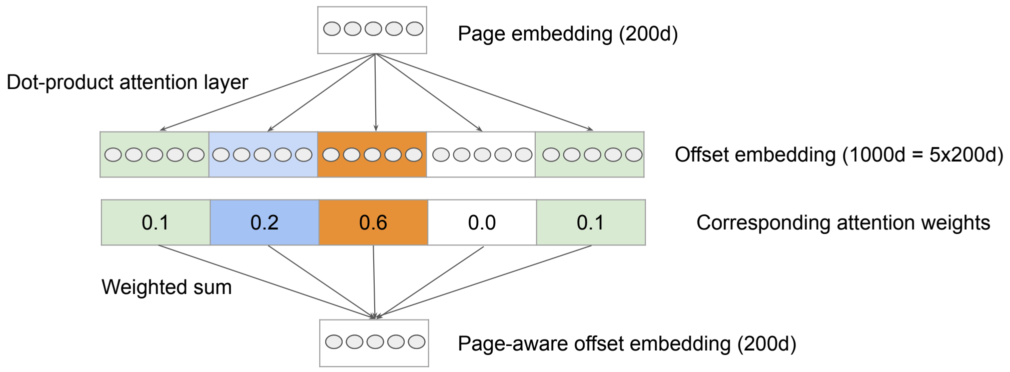

Page embedding (200d)
Dot-product attention layer
Offset embedding (1000d = 5x200d)
0.1
0.2
0.6
0.0
0.1
Corresponding attention weights
Weighted sum
Page-aware offset embedding (200d)

Figure 3: Conditional attention embedding.

4 Experiments

4.1 Data Collection

We evaluate our model with traces of last-level cache (LLC) accesses, which are generated by running applications through ChampSim [4]. For every LLC access, the trace contains a (PC, address) tuple. We run our offline learning models on 250 millions of instruction for a subset of single-core benchmarks. To evaluate, we use the first 80% of each trace for training and the last 20% for testing.

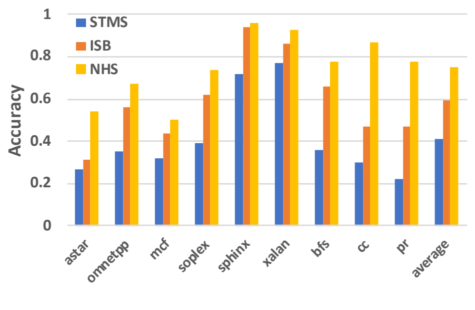


Figure 4: Accuracy.

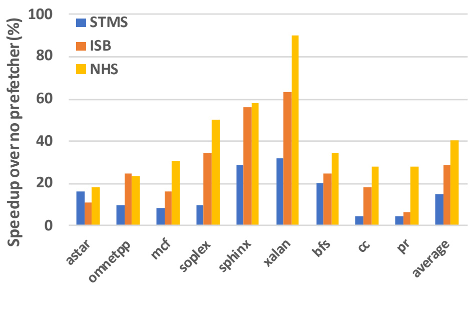


Figure 5: Performance.

4.2 Model Comparisons

We compare against the state-of-the-art irregular prefetchers, Sampled Temporal Memory Streaming (STMS) [7] and Irregular Stream Buffer (ISB) [3] on a selection of benchmarks from SPEC06 and GAP benchmark suites that feature irregular memory access patterns. We measure the prediction accuracy and performance improvements over no prefetching, as shown in Figure 4 and 5. An accurate prediction for ISB and NHS is defined as the predicted address being the next address accessed by the last PC of the sequence, and is defined for STMS as the next address accessed by any PC. As can be seen, NHS achieves superior prediction accuracy, which is translated into significant performance improvements.

4.3 Ablation Studies

In the conditional attention embedding (CAE) layer, the ratio between the offset embedding size and the page embedding size is critical and tuned for each benchmark. Figure 6 shows the learning curves of different ratios for the *cc* benchmark. This benchmark is on an extreme end as all offsets are toggled by nearly all pages, thus requires the ratio to be sufficiently large to represent the various behaviors across pages.

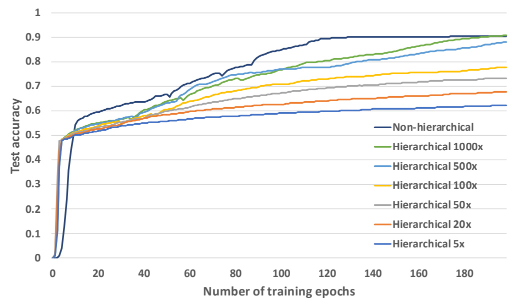


Figure 6: Learning curve of different offset-page ratios for the cc benchmark.

Compared to the non-hierarchical model, NHS obtains similar accuracy with much lower training cost and model size. In particular, for the *cc* benchmark, NHS achieves 10× lower training cost from 10 min to 60 sec to finish an epoch on a single P100 GPU, and reduces the model size by 25× from around 250 MB to 10 MB.

5 Conclusions

In this paper, we have shown how deep learning models can improve the state-of-the-art in address correlation-based data prefetching. The key is a novel hierarchical model that can handle the vast space of memory addresses, making data addresses an available feature for neural networks. Though impractical for implementation in a hardware microprocessor, these results provide guidance for hardware designers who wish to produce the next generation of data prefetchers.

References

[1] Wenhu Chen, Yu Su, Yilin Shen, Zhiyu Chen, Xifeng Yan, and William Wang. How large a vocabulary does text classification need? a variational approach to vocabulary selection. *arXiv preprint arXiv:1902.10339*, 2019.

[2] Milad Hashemi, Kevin Swersky, Jamie A Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. *arXiv preprint arXiv:1803.02329*, 2018.

[3] Akanksha Jain and Calvin Lin. Linearizing irregular memory accesses for improved correlated prefetching. In *The International Symposium on Microarchitecture*, 2013.

[4] Aamer Jaleel, Robert S Cohn, Chi-Keung Luk, and Bruce Jacob. Cmp$im: A Pin-based on-the-fly multi-core cache simulator. In *Proceedings of the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*, pages 28–36, 2008.

[5] Daniel A Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *The International Symposium on High-Performance Computer Architecture*, 2001.

[6] Zhan Shi, Xiangru Huang, Akanksha Jain, and Calvin Lin. Applying deep learning to the cache replacement problem. In *The International Symposium on Microarchitecture*, 2019.

[7] Thomas F Wenisch, Michael Ferdman, Anastasia Ailamaki, Babak Falsafi, and Andreas Moshovos. Practical off-chip meta-data for temporal memory streaming. In *The International Symposium on High Performance Computer Architecture*, 2009.