An Early Exploration of Deep-Learning-Driven Prefetching for Far Memory

Yutong Huang[1] **Zhiyuan Guo**[1] **Yiying Zhang**[1,2]
[1]University of California, San Diego San Diego, CA, USA
[2]GenseeAI Inc. San Diego, CA, USA
{yutonghuang,z9guo,yiying}@ucsd.edu

Abstract

Far-memory systems, where applications store less-active data in more energy-efficient memory media, are increasingly adopted by datacenters. However, applications are bottlenecked by on-demand data fetching from far- to local-memory. We present ***Memix***, a far-memory system that embodies a deep learning–system co-design for efficient and accurate prefetching, minimizing on-demand far-memory accesses. One key observation is that memory accesses are shaped by both application semantics and runtime context, providing an opportunity to optimize each independently. Preliminary evaluation of Memix on data-intensive workloads shows that it outperforms the state-of-the-art far-memory system by up to 42%.

1 Introduction

Applications in today's datacenters, such as data analytics, large-language model inference, and vector information retrieval [1, 2, 3], exhibit unprecedented memory demands. However, scaling server-local memory (CPU DRAM, GPU HBM) to meet these demands is untenable due to limited capacity, high cost per GB, and steep energy overheads. In response, major datacenter providers such as Google and Microsoft increasingly augment hosts with more energy-efficient, slower, and non-local memory tiers, *e.g.*, network-attached pooled memory and non-volatile memory [4, 5]. Such memory systems are commonly referred to as *far memory*. Far-memory systems offer lower cost/GB, higher energy efficiency, lower carbon footprint, and elastic capacity. Prior works show that the reuse of out-dated DDR4 memory as far memory could save the carbon footprint of the datacenter up to 20% [6, 7]. The main challenge of practicing far memory is the large latency gap: An RDMA access to remote memory can be over 20× slower than local memory [8], resulting in substantial slowdowns and negating energy-saving benefits [4, 9, 10].

An effective way to hide this latency is prefetching, which predicts future accesses and fetches data from far to local memory in advance. However, inaccurate prefetching can undermine energy savings by incurring unnecessary remote transfers and cache pollution, increasing both bandwidth and memory energy consumption without improving performance [11, 12]. Existing far-memory prefetching systems only perform simple rule-based predictions, such as sequential and stride, to achieve the accuracy and efficiency requirements [9, 11, 13]. In real-world applications, complex access patterns such as pointer chasing and tree traversals amplify the cost of on-demand far-memory fetches, causing significant slowdowns in many data-center workloads [14, 15].

We argue that, to harvest the energy benefit with far memory, prefetching demands both *prediction accuracy* and *execution efficiency*. We propose *Memix*, a Linux-based far-memory system encorporating a novel deep-learning (DL) guided prefetching mechanism. Our key insight is that memory access behavior is governed by both application semantics (*e.g.*, algorithmic logic) and input-dependent runtime context (*e.g.*, memory layout). The semantics tend to generalize across inputs and can be learned offline, but the actual memory addresses are input-specific and best handled at runtime.

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025).

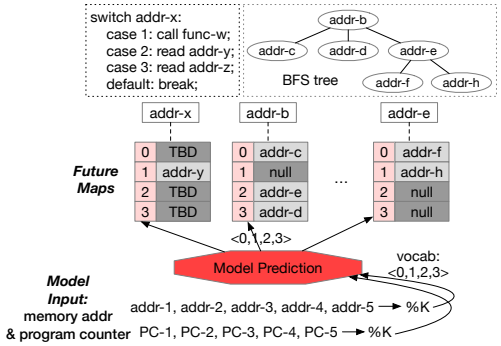

Figure 1: **Memix prediction representation** *An example of vocabulary size (K) being 4. The top part shows code/algorithm corresponding to the accesses of chunks* `addr-x`*,* `addr-b`*, and* `addr-e`*. The bottom shows the input to the model: the chunk addresses and PCs of the 5 previous misses.*

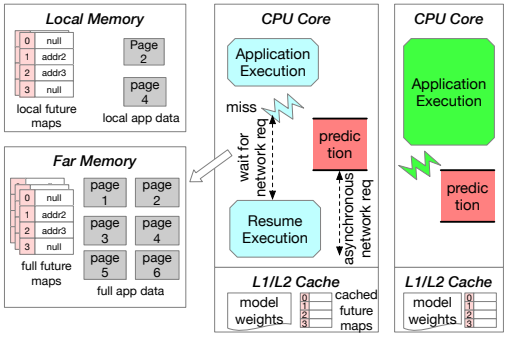

Figure 2: **Memix threading model** *The left panel shows the Memix system's memory layout. The middle panel shows local memory misses during single-core execution, where Memix waits only for on-demand requests. The right panel shows that prefetch hits incur no network request.*

Memix exploits this separation by training a DL model to learn semantic patterns offline while tracking and predicting the actual access address efficiently at runtime.

We implement Memix's prediction and prefetching system in the Linux kernel. We build a predictor model based on RetNet [16], a transformer [17] variant with constant-time inference. Our implementation achieves an average inference latency of $1\mu s$ on a single CPU core. Memix leverages an energy- and carbon-efficient setup by reusing older-generation devices as far memory, and outperforms state-of-the-art systems by up to 42%.

2 Two Key Ideas

Decoupling Access Pattern and Addresses Prediction. The first idea centers on improving prediction through a deeper understanding of the memory access process. We observe that an application's memory access patterns are often repetitive due to code patterns like loops and recursion. However, these patterns are typically complex and nonlinear [18], making them difficult to model with simple rules or heuristics. DL models, on the other hand, are well-suited to capturing long-range, nonlinear dependencies. To maintain efficiency, specially with smaller models that can run within a single CPU core and are trained offline, we must simplify the problem space and minimize runtime variability.

Our idea is to decouple application memory access semantics from the actual runtime memory layout by using DL prediction for the former and mapping tables for the latter. Specifically, we use a small DL model to predict memory access relationships in terms of *abstract ordinals*—representing the possible memory-access outcomes after a short history of memory behavior—rather than concrete memory addresses or offsets, which are highly input- and environment-dependent. At runtime, we construct *future maps*: mapping tables that resolve these predicted ordinals to actual memory addresses observed during the program's first access, thus capturing the true memory layout dynamically. Figure 1 illustrates this idea with an example.

Overlapping I/O and In-Kernel Model Inference. Our second main idea focuses on the execution efficient of the DL-based prediction framework. The latency introduced by PCIe transfers and kernel launch overhead makes GPU-based prediction unsuitable for our use case [19]. Thus, Memix performs its model prediction in CPU. To avoid the energy and performance cost of additional CPU cores, Memix performs prediction for an application on the same core it runs on. Our idea to avoid application performance overhead is to hide model prediction behind far-memory I/O time. Specifically, Memix performs prediction when a foreground page fault is being handled with far-memory data read. We then issue asynchronous prefetch requests in the background.

3 Memix Design

Memix is a swap-based far-memory system that prefetches memory pages (instead of individual addresses) from far memory with an DL-based memory access predictor. Memix consists of an offline training component, an online predictor, and prefetcher in the Linux kernel, as seen in Figure 2.

Model Inputs. Memix uses page miss history as the input to the DL model instead of full memory access history. The reason is that, by being in the swap system, Memix can observe and log miss addresses on every page fault without incurring additional overhead. In contrast, capturing the full memory access stream would introduce substantial runtime overhead and is therefore avoided. In addition to using page miss addresses, we associate every miss with the faulting program counter (PC), as doing so can incorporate program execution information with memory access history, and recording and using PC incurs no additional overhead.

To fit the two types of inputs into the vocabulary, we take the mod of their value to the vocabulary size, K. Just like graph traversal can be recorded using neighbor IDs instead of absolute node IDs, page accesses can be tracked using page mod K to capture relative page transition. We then use a history sequence of h pairs of the modulo of miss page address and PCs as the model input, as shown at the bottom of Figure 1. Even though taking a mod is inevitably a lossy process, a history sequence and two types of information still allow our model to make accurate predictions.

Model Outputs and Future Maps. We choose to predict page misses (*i.e.*, accesses to memory pages not in local memory), rather than attempting to predict every individual memory access—which would be computationally intensive and unnecessary. Essentially, Memix uses page miss sequence in recent history to predict page miss sequence in the future. This approach significantly reduces the computational load on the DL model and the monitoring overhead.

Our solution is to label possible outcomes of memory access as ordinals. Specifically, we record a vocabulary size (*i.e.*, K) of possible next memory page misses after a miss happens at page X. Based on the model inputs as described above, our model predicts an ordinal from 0 to $K-1$, corresponding to one of the likely next page misses. We dynamically maintain a *future map* for each page X. Each entry in the future map represents one possible page to be accessed after the miss of page X, and the value of the entry is the runtime virtual memory address of the page. A null future map entry represents an outcome that has not occured at the runtime yet.

Vocabulary Size. Naturally, a program can have fewer or more possible memory pages to access than K after a page is accessed. If there are fewer possibilities (*e.g.*, Figure 1, pages `addr-b` only have three possible outcomes), the model will just not yield the remaining as a predicted value. If there are more possibilities than K, the model will not properly capture the less frequently occurring accesses. We set the default configurable K to 64, which balances memory overhead with prediction accuracy. This choice is effective because small future maps can fit into CPU L1/L2 caches, reducing prediction latency. In addition, applications with repeatable behavior typically exhibit only a few possible outcomes after a page fault [20]. Finally, since memory allocators often place nearby requests within the same memory page, $K = 64$ is sufficient to capture access patterns across pages.

4 Evaluation Results

Implementation. We integrated Memix into the Linux kernel and implemented RetNet in kernel space with AVX-512 instructions on x86. The model consists of two layers, each with a hidden dimension of 8, resulting in 2.5K trainable parameters. This compact size allows the parameters to reside entirely in CPU cache, enabling an average inference latency of $1\mu s$.

Experiment Setup. We evaluate Memix on our private clusters, where a compute node, equipped with a 28-core Intel Xeon Gold 5512U CPU and DDR5 memory, is connected via 100Gbps RDMA to a memory node, equipped with a 16-core Intel Xeon Gold 5218 CPU with DDR4 memory. This emulates a common data center scenario where new servers leverage older servers' memory for additional capacity, while being carbon-efficient.

We compare Memix with two baselines: FastSwap [9] and Hermit [10]. FastSwap is a swap-based far-memory system implemented in the Linux kernel. Hermit builds on top of FastSwap and improves its swap-out procedure to avoid swap-out being the application performance bottleneck. Both systems use the Linux prefetching policy, which only follows simple and strict rules for issuing prefetches for sequential accesses. We evaluate Memix on three applications: XGBoost [1], a machine learning framework for gradient boosted decision trees (GBDTs), PageRank in GAP benchmark suite [21], MCF in SPEC 2006 benchmark [22]. These three applications serve to illustrate a representative subset of data analytics and machine learning applications, highlighting the types of computational patterns and data access behaviors that Memix is designed to handle efficiently.

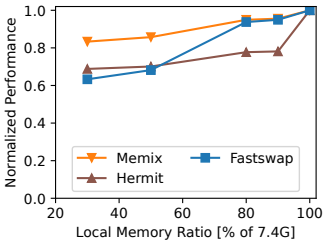

Figure 3: **XGBoost performance**

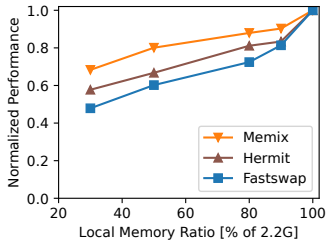

Figure 4: **Pagerank performance**

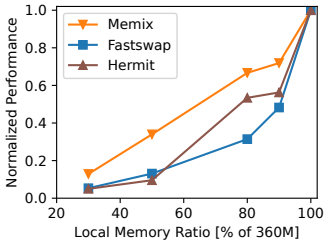

Figure 5: **MCF performance**

Application performance. Figures 3, 4, 5 present the end-to-end application performance of XGBoost, PageRank, and MCF compared to state-of-the-art systems. These preliminary results show that Memix outperforms existing systems by up to 42%. For each set of experiments, we change the application server's local memory size from 30% to 90% of the total application memory size (x axis) and measure the total application execution time (y axis). For each result, we normalize execution time against that of running at full local-memory capacity (higher y-axis are better). Comparing to both systems, Memix generates up to 3x more useful prefetch in 30% local memory size.

The improved performance of Memix yields energy savings. By reducing memory stalls and enabling faster data access, CPUs spend more time idling and can enter low-power states more often. Centralizing far memory in disaggregated pools reduces overprovisioning, eliminates duplicate data, and lowers energy use. Consequently, the observed speedup enhances both throughput and energy efficiency, making Memix beneficial for performance and energy optimization in data centers.

5 Discussion and Conclusion

We presented Memix, a DL-based far-memory prefetching system in the Linux kernel. Memix's core idea is to decouple the learning of application semantics from the runtime capturing of memory accesses. By doing so, Memix achieves overall application performance benefits over two recent far-memory systems. Below we further discuss our setup, findings and future directions:

Target Applications. Memory intensive applications with repeated memory accesses, such as graph processing, data analytics, and machine learning, are well-suited for Memix. Hash map is a counterexample, as the memory accessed is mostly dependent on the hash function.

Target Environment. Memix is designed for data center with tens to hundreds of enterprise applications. Each application requires only a one-time training run unless its input changes substantially. In our experiments, training each application took about 30 minutes on a single A6000 GPU.

Lesson Learned. Memix's approach of separating access patterns and overlapping interference reveals further opportunities, one critical observation is *swap-out management*. Our aggressive prefetching boosts performance but also increases swap-out activity, as each prefetch requires an empty page frame for new data. Even with perfect prefetching, evicting useful pages can degrade performance due to flushed data. This underscores that swap-out policy is equally critical and could be jointly optimized with I/O and prefetching, potentially guided by deep learning.

Future Directions. CXL (Compute Express Link) [23] enables memory expansion by attaching standalone memory devices over a PCIe-like interconnect, even with heterogeneous memory controllers (e.g., DDR4 via CXL with a DDR5 CPU). Similarly, CXL incurs higher latency than DRAM and thus sensitive to memory placement and prefetching. Its effectiveness relies on application semantic and placement-aware management, which Memix could be extended to support.

6 Acknowledgment

We would like to thank Ryan Lee, Geoff Voelker, Zijian He, Vikranth Srivatsa, and Reyna Abhyankar for their valuable contributions and feedback on this paper. This material is based upon work supported by funding from the PRISM center (part of SRC's JUMP 2.0), the National Science Foundation under the grant NSF 2504468 and grant NSF 2403253, and gifts from Google and Meta. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these institutions.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[2] Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A. Rossi, Subhabrata Mukherjee, Xianfeng Tang, Qi He, Zhigang Hua, Bo Long, Tong Zhao, Neil Shah, Amin Javari, Yinglong Xia, and Jiliang Tang. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921*, 2024.

[3] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Guoyang Chen, Ximing Liu, Mark D. Hill, and Michael M. Swift. Pond: Cxl-based memory pooling and disaggregation. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2023.

[5] Baotong Lu, Kaisong Huang, Chieh-Jan Mike Liang, Tianzheng Wang, and Eric Lo. Dex: Scalable range indexing on disaggregated memory. *Proc. VLDB Endow.*, 17(10):2603–2616, June 2024.

[6] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, Ryan Wee, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D. Hill, Mosharaf Chowdhury, and Asaf Cidon. Managing memory tiers with CXL in virtualized environments. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 37–56, Santa Clara, CA, July 2024. USENIX Association.

[7] Jialun Lyu, Jaylen Wang, Kali Frost, Chaojie Zhang, Celine Irvene, Esha Choukse, Rodrigo Fonseca, Ricardo Bianchini, Fiodar Kazhamiaka, and Daniel S. Berger. Myths and misconceptions around reducing carbon embedded in cloud platforms. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, HotCarbon '23, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Teng Ma, Kang Chen, Shaonan Ma, Zhuo Song, and Yongwei Wu. Thinking more about rdma memory semantics. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 456–467, 2021.

[9] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can far memory improve job throughput? In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, 2020.

[10] Yifan Qiao, Chenxi Wang, Zhenyuan Ruan, Adam Belay, Qingda Lu, Yiying Zhang, Miryung Kim, and Guoqing Harry Xu. Hermit: Low-Latency, High-Throughput, and transparent remote memory via Feedback-Directed asynchrony. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA, April 2023.

[11] Hasan Al Maruf and Mosharaf Chowdhury. Effectively prefetching remote memory with leap. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*, pages 843–857. USENIX Association, 2020.

[12] Shao-Peng Yang, Minjae Kim, Sanghyun Nam, Juhyung Park, Jin yong Choi, Eyee Hyun Nam, Eunji Lee, Sungjin Lee, and Bryan S. Kim. Overcoming the memory wall with CXL-Enabled SSDs. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 601–617, Boston, MA, July 2023. USENIX Association.

[13] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, Boston, MA, March 2017. USENIX Association.

[14] Yupeng Tang, Seung-seob Lee, Abhishek Bhattacharjee, and Anurag Khandelwal. pulse: Accelerating distributed pointer-traversals on disaggregated memory. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '25, page 858–875, New York, NY, USA, 2025. Association for Computing Machinery.

[15] Rongxin Cheng, Yifan Peng, Xingda Wei, Hongrui Xie, Rong Chen, Sijie Shen, and Haibo Chen. Characterizing the dilemma of performance and index size in billion-scale vector search and breaking it with second-tier memory, 2024.

[16] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[18] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. In *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research, 10–15 Jul 2018.

[19] Lingqi Zhang, Mohamed Wahib, and Satoshi Matsuoka. Understanding the overheads of launching cuda kernels. *ICPP19*, pages 5–8, 2019.

[20] Quang Duong, Akanksha Jain, and Calvin Lin. A new formulation of neural data prefetching. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.

[21] Scott Beamer, Krste Asanović, and David Patterson. The gap benchmark suite, 2017.

[22] Standard Performance Evaluation Corporation (SPEC). SPEC CPU2006 Benchmark Description: 429.mcf. `https://www.spec.org/cpu2006/Docs/429.mcf.html`, 2006. Accessed: 2025-04-17.

[23] CXL Consortium. `https://www.computeexpresslink.org/`.