Recency/Frequency Adaptive KV Caching for Large Language Model Serving

Yang Shen [1] **Meghana Madhyastha**[† 2] **Robert Underwood** [3] **Bogdan Nicolae** [3] **Randal Burns** [1]

Abstract

Key-value (KV) caching is a powerful technique for accelerating large language model inference and generation. Inference workloads are large and diverse, which makes them difficult to cache effectively. Existing cache management strategies adopt the least-recently-used policy for evicting cache blocks. However, LRU leads to multiple unrelated workloads flushing each other's caches. To address this, we integrate adaptive caching that dynamically allocates cache space between recently and frequently occurring KV blocks. Evaluations show that it improves the KV cache hit rate by up to 10.8% and reduces time to first token by up to 12.6% over naive vLLM on synthetic document question answering workloads, and 2.1% and 2.0% respectively on real-world conversation workloads. The method generalizes well to batch inference and demonstrates clear interpretability while effectively accommodating diverse workloads. Our open-source implementation is available at https://github.com/Y-aang/vllm-ARC.

1. Introduction

Large language models (LLMs) are widely deployed across a variety of applications, ranging from conversational agents to enterprise-level automation workflows. In production environments, the efficiency of inference and generation has emerged as a central concern (Wan et al., 2023; Zhou et al., 2024; Zhen et al., 2025; Miao et al., 2025). The KV cache is a fundamental mechanism for reducing redundant computation during decoding (Yu et al., 2022; Kwon et al., 2023; Zheng et al., 2024). However, as model sizes and context lengths continue to grow, the limited capacity of GPU memory has become a bottleneck for KV cache and raises new challenges for its effective management (Zhang et al., 2023; Feng et al., 2024; Jiang et al., 2025).

Prefix caching and reuse between requests significantly reduces computation, and its hit rate directly affects both system throughput and latency (Liu et al., 2025). In many scenarios, different requests share a common prefix. For example, in multi-turn dialogue systems, a user's historical conversation appears as a prefix in subsequent requests (Chiang et al., 2023). In question answering, certain documents or passages may act as hotspots frequently queried by different users (Yang et al., 2015; Pang et al., 2022). These observations suggest that designing efficient KV cache management strategies that capture such reuse behaviors between requests remains an open and impactful problem.

However, most existing LLM serving frameworks, such as vLLM (Kwon et al., 2023), primarily adopt simple least-recently-used (LRU) eviction strategies. Meanwhile, real workloads often exhibit more complex access characteristics: some present strong locality patterns, while others demonstrate a mixture of both recency and frequency properties. These mixed patterns in LLM serving are not fully captured by purely recency-driven eviction policies, motivating the exploration of more expressive strategies (Wang et al., 2025a).

Therefore, we explore dynamic hybrid recency-frequency strategies as a foundation for adaptive KV cache management. We integrate the Adaptive Replacement Cache (ARC) (Megiddo & Modha, 2003) into the LLM serving framework (vLLM). ARC implements a two-level split cache that separates recent and frequent requests; workload adaptive allocation of cache space between the two caches; and, ghost-caching to track misses that fall outside of the cache but could have been hits if cache space was allocated differently. We evaluate optimizations under diverse workload settings. Our study covers a long-context question answering (QA) workload derived from document QA datasets and a trace-driven conversational workload from a large-scale LLM service. On the document QA workload, we observe up to a 10.8% improvement in hit rate and a 12.6% reduction in time to first token (TTFT); on the conversation workload, we observe up to a 2.1% improvement in hit rate and a 2.0%

[†] Work done while at Johns Hopkins University. [1]Department of Computer Science, Johns Hopkins University, Baltimore, USA [2]Parasail, Inc., San Mateo, USA [3]Argonne National Laboratory, Lemont, USA. Correspondence to: Yang Shen <yaangysshen@gmail.com>, Randal Burns <randal@cs.jhu.edu>.

Published at the Resource-Adaptive Foundation Model Inference (AdaptFM) Workshop, ICML 2026, Seoul, South Korea.

reduction in TTFT. These results indicate the potential of adaptive KV cache eviction and provide insights for future LLM serving systems.

2. Background and Related Work

LLM Serving. The LLM inference process is autoregressive, consisting of two stages: the prefill stage and the decoding stage (Dao et al., 2022; Zhong et al., 2024). During the prefill stage, the model processes the entire input prompt in a single forward computation to obtain its hidden representations; during the decoding stage, the model generates new tokens one at a time conditioned on all preceding context.

To improve inference efficiency, modern LLM serving systems employ prefix caching and paged attention (Kwon et al., 2023; Zheng et al., 2024; Qin et al., 2024; Gao et al., 2024; Srivatsa et al., 2024). Prefix caching reduces redundant computation by reusing key and value of self-attention for a shared prefix. It reuses the prefix within the same request during the decoding process, and also reuses the shared prefix across requests during the prefill process. Paged attention organizes the KV cache into fixed-size KV blocks and manages them through a paging-style allocation scheme (Kwon et al., 2023), which avoids the need for contiguous memory, mitigates fragmentation, and improves efficiency.

For cache replacement, most systems (Kwon et al., 2023; Zheng et al., 2024) adopt recency-based policies such as LRU. When the cache becomes full, the system evicts the least-recently-used units (e.g., KV blocks) to make room for new requests. Marconi (Pan et al., 2024) further explores FLOP-aware eviction over LRU for state-space models. RAGCache (Jin et al., 2025) designs a prefix-aware Greedy-Dual-Size-Frequency replacement policy for retrieval-augmented generation. MoonCake (Qin et al., 2024) introduces a hierarchical KV cache but focuses on prefill-decode disaggregation. Learning-based approaches (Yang et al., 2025b) learn a continuation predictor to evict prefixes, while works like (Wang et al., 2025a) propose workload-aware caching focusing on cloud use cases.

Applications. LLMs are now widely deployed in various scenarios, including document QA and multi-turn conversation. In document QA (Yang et al., 2015; Pang et al., 2022), the input typically consists of a long document paired with a question, requiring the model to process the entire document before producing an answer. In multi-turn conversation (Chiang et al., 2023), the dialog history grows with each turn, forming an increasingly large context that the model must repeatedly incorporate during generation.

Prefix caching and reuse are useful in those cases to improve inference efficiency. In document QA, the long document prefix can be reused through KV cache across requests. In multi-turn conversation, the accumulated dialogue history can be directly retrieved from the cache in each turn, reducing the cost of repeatedly reprocessing the entire context.

3. Method

Modern LLM serving systems face large and dynamic workloads that exhibit a mixture of access patterns beyond simple locality. For instance, in document QA scenarios, specific documents often act as "hotspots" that are frequently queried by different users (Breslau et al., 1999; Crovella & Bestavros, 2002). In multi-turn chatbot scenarios, sessions' temporal locality and varying user activity levels lead to complex and interleaved access patterns. These mixed patterns are not fully captured by purely recency-driven eviction policies. Motivated by this, we investigate hybrid eviction policies that dynamically partition cache space, allowing the system to adaptively fit diverse and shifting workloads.

Figure 1 illustrates the overall architecture of the adaptive KV caching system for LLM serving. Requests from applications, such as document QA characterized by hot documents or chatbots characterized by active users, are handled by the inference service. The service routes requests to instances and the GPUs that process requests. Our technique addresses the management of the GPU caches, modifying the evictor module of a vLLM serving architecture. This applies to complex serving frameworks, including collaborative caching across multiple instances or multiple GPUs (Qin et al., 2024; Liu et al., 2025), and inference services that co-schedule or route requests based on knowledge of existing cache contents (Srivatsa et al., 2024; Cao et al., 2025). In these cases, there is a module that does per-GPU memory management and eviction. Our memory manager utilizes a specialized evictor based on the Adaptive Replacement Caching (Megiddo & Modha, 2003) to dynamically manage KV blocks based on real-time access patterns.

Figure 2 illustrates the structure of ARC. This strategy automatically balances cache space between recency and frequency through two key features:

- **Recency-Frequency Dual-Queue:** The cache is logically split into a low-frequency queue (L1 = T1 + B1) for items accessed once and a high-frequency queue (L2 = T2 +B2) for items accessed multiple times. Items in the low-frequency queue are promoted to the high-frequency queue upon a hit.
- **Dynamic Cache Partitioning with Ghost Caches:** ARC divides L1 and L2 into physical caches T1 and T2 that refer to KV blocks in memory and ghost caches B1 and B2 that extend the MRU/LRU history of T1 and T2. They store only the metadata for evicted KV

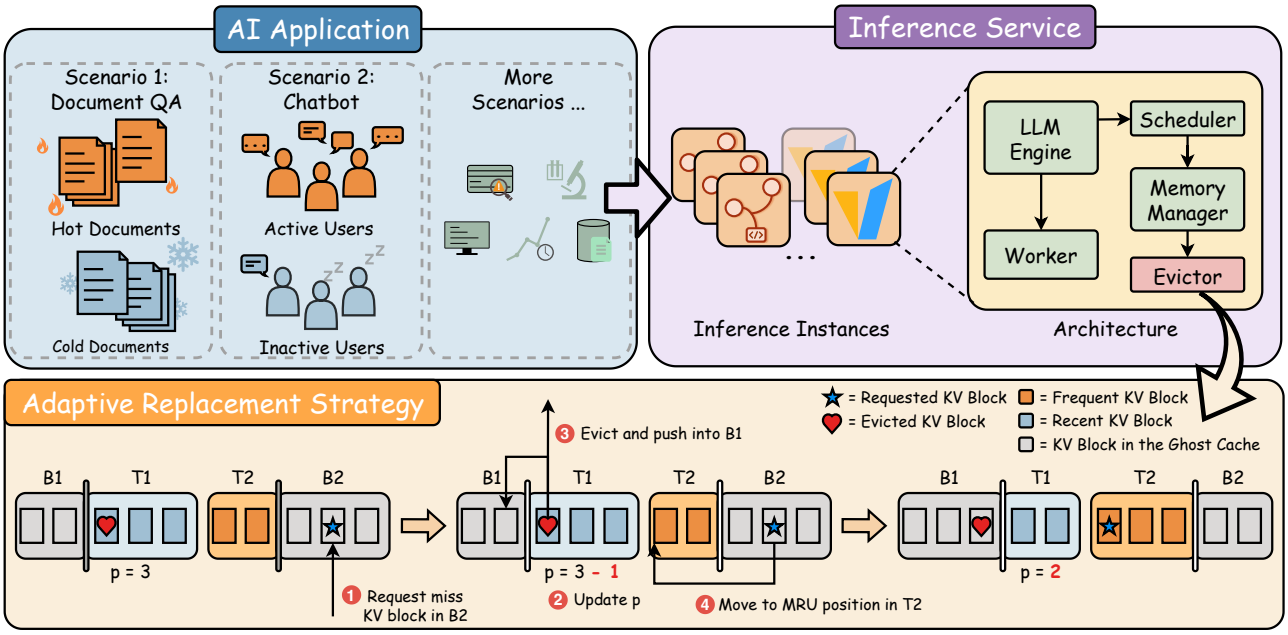


Figure 1. Overview of adaptive KV caching for LLM serving.

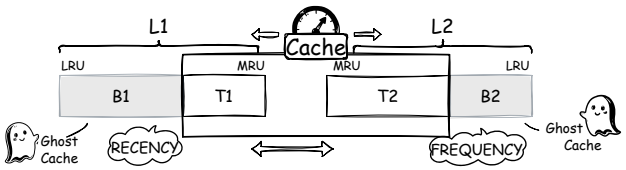


Figure 2. Structure of the Adaptive Replacement Cache. The cache consists of a recency queue (L1) and a frequency queue (L2) that are split into physical caches T1 and T2 and ghost caches B1 and B2 that track metadata. Requests that hit in the ghost caches are used to adapt the length of the T1 and T2 queues.

blocks of each queue. Hits in the ghost cache serve as feedback signals to increase the size of the corresponding queue and dynamically adjust the partition of memory between two queues (T1 and T2).

The lower pane of Figure 1 shows the operation of ARC. A request for a KV block that appears in B2 is a miss. The block is no longer cached. But, it has been referenced before and could have been a hit in a larger cache. The LLM engine processes the miss, regenerating the block. ARC puts this KV block to the frequency cache T2, evicts the LRU block from T1. The combination of T1 and T2 is of fixed size. Misses that hit in B1 indicate that a larger T1 cache would have resulted in a hit and the algorithm should invest more space in T1. ARC (Megiddo & Modha, 2003) gives full details of adaptation and learning rate for a page cache.

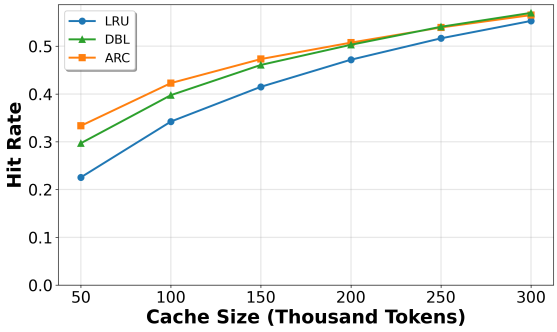
The design makes the cache flush-resistant during workload shifts. A large number of new, unseen requests will clear T1, but the contents of T2 remain in cache. It adapts to workload shifts more quickly than a static policy by growing T1 when there is reuse of recently referenced KV blocks. It adapts to hotspots by growing T2 when KV blocks are used more than twice.

The caching strategy is parameter-free and integrates naturally with the LLM inference engine. The KV cache is structured into fixed-size KV blocks (e.g., 16 tokens), aligning with the paged attention mechanism widely adopted in modern serving systems. All eviction decisions of the cache strategy are performed at the KV block level.

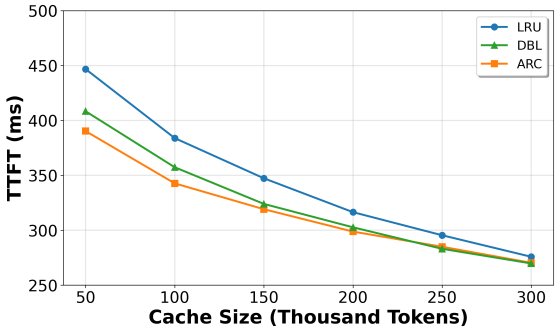
4. Evaluation

We integrate the adaptive caching strategy into vLLM (Kwon et al., 2023) and evaluate performance for two document QA workloads and a multi-turn conversation trace-driven workload. Experiments measure the cache hit rate and correlate improvements to the performance metric TTFT during generation. We isolate the contributions of both aspects of ARC: two-queue caching and adaptive sizing. We demonstrate effectiveness across different batch sizes. Finally, we examine the sizes of the recency and frequency caches over time to demonstrate the importance of adaptation.

Model and Hardware Configuration. We evaluate all eviction strategies using Qwen3 (Yang et al., 2025a) with 14B parameters. Qwen3 is a popular open-source model, and 14B is a representative model size for modern LLM serving. All the main experiments are conducted on a single NVIDIA

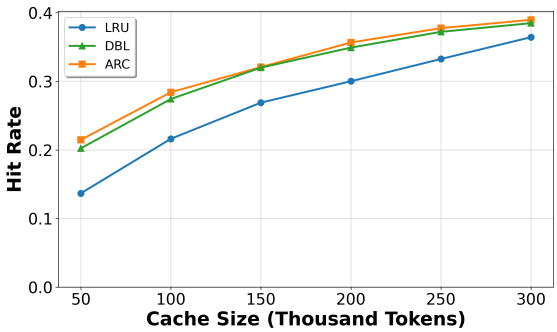


(a) KV Cache Hit Rate

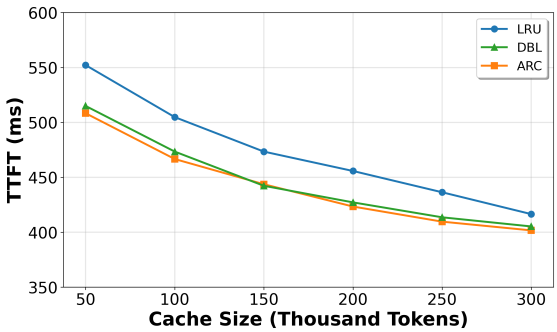


(b) Average TTFT

Figure 3. Document QA workload on QuALITY that compares static two-queue replacement (DBL) and adaptive two-queue replacement (ARC) with LRU.



(a) KV Cache Hit Rate



(b) Average TTFT

Figure 4. Document QA workload on WikiQA.

H100 PCIe GPU with 80 GB of memory on Lambda GPU Cloud.

Inference Configuration. To isolate the effect of cache eviction strategies from other runtime factors, we standardize the inference behavior:

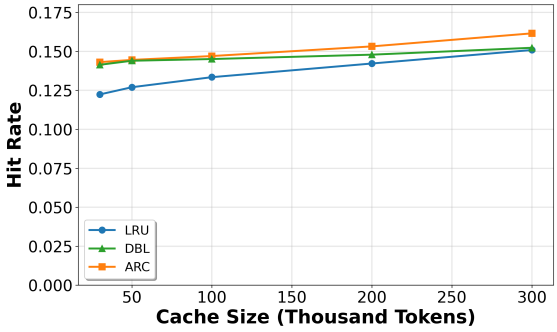
- The offline inference mode is used. Except when noted, the batch size is fixed to 1, eliminating requests waiting on prior requests in the batch, ensuring that all TTFT variation reflects cache behavior.
- Each request generates exactly one token, removing variability from the decoding phase to prevent generating responses of different lengths.
- The KV block size is set to 16 tokens, following the standard paged-attention layout used in modern LLM serving systems.
- Prefix caching is enabled, allowing shared prefixes across requests to be reused by the eviction strategies.

4.1. Document Question Answering

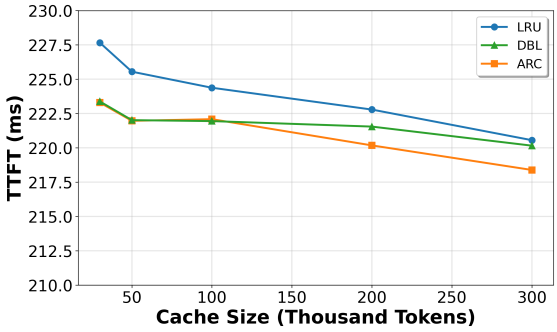
We evaluate cache eviction strategies using two widely-adopted long-context document QA datasets: QuALITY (Pang et al., 2022) and WikiQA (Yang et al., 2015). QuALITY is a long-document reading comprehension dataset. We apply HTML-stripped v1.0.1 from the official repository, which contains 381 unique long documents paired with 761 questions with passages over 5k tokens on average. WikiQA is an open-domain question-answering dataset built from Bing search queries and their linked Wikipedia pages. It includes 3,047 user questions mapped to their associated long passages, along with 29,258 question-sentence pairs.

These are question answer sets, not workloads. We know of no publicly available trace-driven QA workloads. We generate synthetic workloads from them using windowed Zipf sampling, which is widely used to generate workloads with realistic distribution for LLMs (Srivatsa et al., 2024; Wang et al., 2025b; Ye et al., 2025). Specifically, we draw 3,072 requests in total and partition them into windows of 512 requests each. Within each window, requests are sampled according to a Zipf distribution (exponent = 1.0), while the underlying items are reshuffled across windows to introduce shifts in popularity.

We evaluate and compare ARC with the vLLM LRU replacement algorithm. We also include **DBL**, a variant of ARC that



(a) KV Cache Hit Rate



(b) Average TTFT

Figure 5. Performance comparison on the multi-turn conversation workload.

employs a fixed 1:1 ratio between recency and frequency queues, to isolate the contribution of recency/frequency separation from memory adaptation.

Figure 3 and 4 show that ARC improves the KV cache hit rate and that translates to an overall improvement in TTFT. ARC increases the hit rate by 1.2%-10.8% on QuALITY and 2.5%-7.8% on WikiQA. The average TTFT under ARC decreases by 2.0%-12.6% and 3.6%-7.9% respectively. As the cache size grows, the improvement of the adaptive eviction policy diminishes. This is expected as larger caches capture a larger fraction of the workload and replacement decisions are less frequent and less important. In these workloads, improvements from the adaptive properties of ARC less important and more pronounced at smaller cache sizes.

4.2. Multi-Turn Conversation

We use Trace A from the Qwen-Bailian Anonymous Dataset to evaluate the adaptive cache eviction under a multi-turn conversation scenario (Wang et al., 2025a). This trace contains a two-hour sample of anonymized KV cache requests served by a single Qwen deployment on Aliyun Bailian, one of the world's largest cloud providers. Trace A corresponds to a ChatGPT-style, consumer-facing service, capturing real user interactions at production scale. It contains 43,058 requests, and the average input length is around 2.3k tokens. It provides a practical view of multi-turn conversation scenario. To ensure stable evaluation, we warm up the cache with the first 20,000 requests and report hit rate and TTFT on the rest.

On a trace-driven conversational workload, ARC improves performance compared with the baseline LRU (Figure 5). ARC achieves a 1.1%-2.1% increase in hit rate, which translates into a 1.0%-2.0% reduction in average TTFT. In this case, the contribution from adaptive queue sizing grows as the cache grows in size and static two-queue caching (DBL) degrades for caches larger than 200K tokens. This represents workload shifts on a longer scale that arise only in larger caches (see Section 4.4).

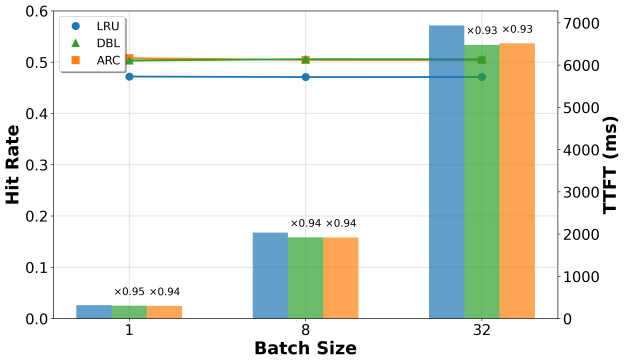


Figure 6. KV cache hit rate and average TTFT under different batch sizes on the QuALITY dataset. The cache size is 20k tokens.

4.3. Batch Inference

To evaluate whether adaptive KV caching remains effective under batch inference, we conduct additional experiments using the same synthetic document QA workload as in Section 4.1, while varying the batch size. Batch inference is commonly adopted in offline serving and throughput-oriented scenarios, where multiple requests are processed simultaneously to maximize throughput. In this experiment, the cache size is set as 20 thousand tokens.

Figure 6 reports the KV cache hit rate and average TTFT under different batch sizes on the QuALITY dataset. The cache hit rate remains stable as the batch size increases for all eviction strategies. This trend is expected, as the increase of batch size does not change much the prefix reuse pattern between requests. The only difference would be if the same prefix arose in the same batch.

The relative advantage of ARC persists as the batch size grows compared to LRU with a 5-7% reduction in TTFT and benefits grow slightly as batch size increases. The increase in the time to first token as batch size increases reflects that

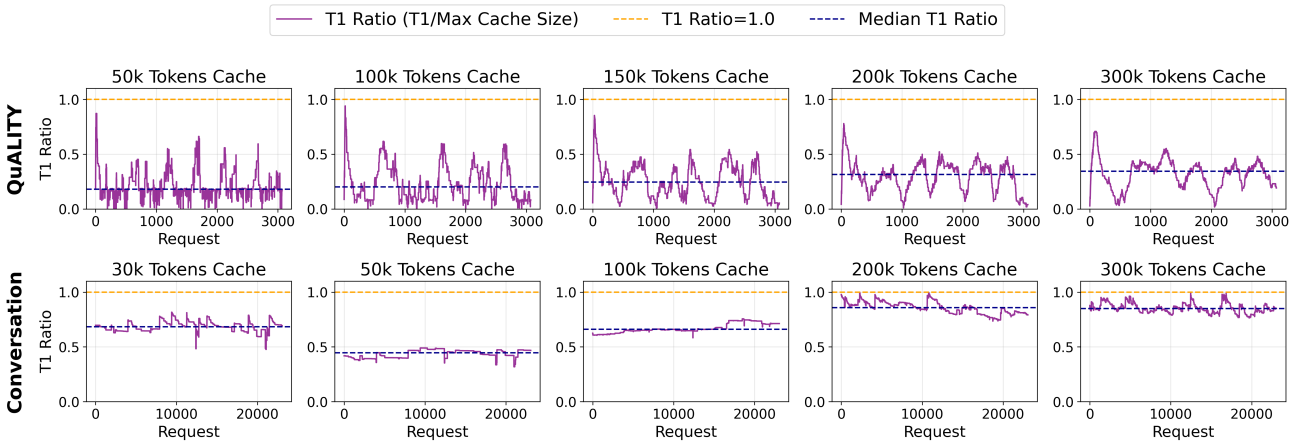


Figure 7. Dynamic evolution of the T1 Ratio across different workloads and cache capacities.

the larger batch delays all requests. Throughput is higher on aggregate, but TTFT is reduced.

4.4. Impact of Adaptive Partitioning

ARC's improvement comes in part from the dynamic allocation of memory between the two caches. Examining the evolution of the T1 and T2 cache sizes over time provides insight into how adaptive memory allocation improves performance. It also characterizes workloads, revealing dynamics in reuse and workload shifts. Figure 7 shows the ratio of T1 (recency queue) and T2 (frequency queue) over the duration of the experiment.

The conversation dataset reveals the operation of a cache under a trace-driven workload. As the cache increases in size, the balance of space is reserved for recency. This hints at a smaller working set maintained for frequency that is already captured at smaller cache sizes. The 30K cache size seems to be an outlier. Likely, 30K tokens is not large enough to store the working set and the cache is unable to populate T2. This reflects the poor hit rate seen at this cache size (Figure 5).

We also see that the cache sizes change more dramatically at 200K and 300K in contrast to the 100K token cache. ARC captures long-scale workload shifts that are not captured by smaller ARC caches or other methods. The adaptation leads to a divergence in performance between ARC and DBL (Figure 5).

The QuALITY dataset shows the cache migrating between recency and frequency. At the start, the cache is empty and there are no frequency hits. The cache invests all space in the recency queue. Over many requests, the cache migrates between frequency and recency. This corresponds to 512-request windowed Zipf sampling used to generate the workload. Again, the median shifts to recency as the cache size increases.

In both workloads, the deviation of the T1 ratio from the 50% baseline aligns with the performance gain, which shows that the adaptive strategy outperforms a static split.

5. Conclusion

This paper presents adaptive KV caching for LLM serving, integrating hybrid recency-frequency strategies (ARC) into vLLM. By adaptively capturing prefix-reuse patterns beyond simple recency, these strategies improve cache hit rate and reduce TTFT across both synthetic document QA workloads and real multi-turn conversation traces. Our method generalizes naturally to batch inference and exhibits clear interpretability, demonstrating that adaptive KV cache management is a practical and effective way to enhance LLM serving performance.

6. Future Work

An extension of this work is to expand the scope of adaptability within the eviction framework. Beyond recency and frequency, other factors such as prompt-length-dependent latency savings and positional variance of reuse probability could be explored. Designing eviction strategies that explicitly balance length, recency, and frequency is a direction for further performance improvements.

Other directions include evaluating and designing caching strategies in more sophisticated serving architectures. While our design and evaluation are primarily based on offline inference, production systems often involve online inference, hierarchical CPU/Disk offloading, multi-GPU distributed setups, and prefill-decoding disaggregated architectures. Exploring diverse scenarios presents opportunities to further refine system design and optimize end-to-end performance.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning and system for machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357 and by the National Science Foundation under Grant NSF OAC-02103874. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or Department of Energy.

References

Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM*, volume 1, pp. 126–134, 1999.

Cao, S., Wang, Y., Mao, Z., Hsu, P.-L., Yin, L., Xia, T., Li, D., Liu, S., Zhang, Y., Zhou, Y., et al. Locality-aware Fair Scheduling in LLM Serving. *arXiv preprint arXiv:2501.14312*, 2025.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., et al. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

Crovella, M. E. and Bestavros, A. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 2002.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Feng, Y., Lv, J., Cao, Y., Xie, X., and Zhou, S. K. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. *arXiv preprint arXiv:2407.11550*, 2024.

Gao, B., He, Z., Sharma, P., Kang, Q., Jevdjic, D., Deng, J., Yang, X., Yu, Z., and Zuo, P. Cost-Efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 111–126, 2024.

Jiang, J., Yang, P., Zhang, R., and Liu, F. Towards efficient large language model serving: A survey on system-aware KV cache optimization. *Authorea Preprints*, 2025. URL https://doi.org/10.36227/techrxiv.176046306.66521015/v1.

Jin, C., Zhang, Z., Jiang, X., Liu, F., Liu, S., Liu, X., and Jin, X. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 44(1):1–27, 2025.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Liu, Y., Cheng, Y., Yao, J., An, Y., Chen, X., Feng, S., Huang, Y., Shen, S., Zhang, R., Du, K., et al. Lmcache: An efficient KV cache layer for enterprise-scale LLM inference. *arXiv preprint arXiv:2510.09665*, 2025.

Megiddo, N. and Modha, D. S. ARC: A self-tuning, low overhead replacement cache. In *2nd USENIX Conference on File and Storage Technologies (FAST 03)*, 2003.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Jin, H., Chen, T., and Jia, Z. Towards efficient generative large language model serving: A survey from algorithms to systems. *ACM Computing Surveys*, 58(1):1–37, 2025.

Pan, R., Wang, Z., Jia, Z., Karakus, C., Zancato, L., Dao, T., Wang, Y., and Netravali, R. Marconi: Prefix caching for the era of hybrid LLMs. *arXiv preprint arXiv:2411.19379*, 2024.

Pang, R. Y., Parrish, A., Joshi, N., Nangia, N., Phang, J., Chen, A., Padmakumar, V., Ma, J., Thompson, J., He, H., et al. Quality: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5336–5358, 2022.

Qin, R., Li, Z., He, W., Cui, J., Tang, H., Ren, F., Ma, T., Cai, S., Zhang, Y., Zhang, M., et al. Mooncake: A KVcache-centric disaggregated architecture for LLM serving. *ACM Transactions on Storage*, 2024.

Srivatsa, V., He, Z., Abhyankar, R., Li, D., and Zhang, Y. Preble: Efficient distributed prompt scheduling for LLM serving. *arXiv preprint arXiv:2407.00023*, 2024.

Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., Liu, J., Qu, Z., Yan, S., Zhu, Y., Zhang, Q., et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 2023.

Wang, J., Han, J., Wei, X., Shen, S., Zhang, D., Fang, C., Chen, R., Yu, W., and Chen, H. KVCache cache in the wild: Characterizing and optimizing KVCache cache at a large cloud provider. *arXiv preprint arXiv:2506.02634*, 2025a.

Wang, Y., Chen, Y., Li, Z., Kang, X., Fang, Y., Zhou, Y., Zheng, Y., Tang, Z., He, X., Guo, R., et al. BurstGPT: A real-world workload dataset to optimize LLM serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5831–5841, 2025b.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Yang, D., Li, A., Li, K., and Lloyd, W. Learned prefix caching for efficient LLM inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b.

Yang, Y., Yih, W.-t., and Meek, C. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 2013–2018, 2015.

Ye, Z., Chen, L., Lai, R., Lin, W., Zhang, Y., Wang, S., Chen, T., Kasikci, B., Grover, V., Krishnamurthy, A., et al. Flashinfer: Efficient and customizable attention engine for LLM inference serving. *arXiv preprint arXiv:2501.01005*, 2025.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H_2O: Heavy-Hitter Oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Zhen, R., Li, J., Ji, Y., Yang, Z., Liu, T., Xia, Q., Duan, X., Wang, Z., Huai, B., and Zhang, M. Taming the titans: A survey of efficient LLM inference serving. *arXiv preprint arXiv:2504.19720*, 2025.

Zheng, L., Yin, L., Xie, Z., Sun, C. L., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024.

Zhou, Z., Ning, X., Hong, K., Fu, T., Xu, J., Li, S., Lou, Y., Wang, L., Yuan, Z., Li, X., et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.