# Plug-and-Play KV Cache Reuse for Low-Latency Retrieval-Augmented Generation


Ye Yue
*School of EECS*
*Ohio University*
Athens, OH, USA
0009-0005-2250-7536

Zhewei Wang
*School of EECS*
*Ohio University*
Athens, OH, USA
0009-0005-4777-5625

Fabio Agosto
*Digital Transformation Office*
*Air Force Materiel Command (AFMC)*
Dayton, OH, USA
0009-0007-7818-6537

Jundong Liu*
*School of EECS*
*Ohio University*
Athens, OH, USA
0000-0002-9049-2832



***Abstract*—Retrieval-augmented generation (RAG) systems often suffer from high latency during the prefilling stage, when large language models compute key–value (KV) caches for long prompts. Although caching shared documents can reduce redundant computation, directly reusing precomputed KV blocks typically causes substantial accuracy loss.**

**In this work, we study KV cache reuse for RAG under a strictly plug-and-play regime, where the base language model is left unchanged and no retraining, prompt redefinition, or selective recomputation is performed. Within this setting, we propose a fine-tuning-free approach that reuses KV states across queries through an anchor-block and position-adjustment strategy. Our design preserves alignment with the model's attention mechanism while enabling efficient recombination of cached blocks.**

**Experiments on multiple QA benchmarks demonstrate up to a 5.9× reduction in time-to-first-token (TTFT) with only moderate accuracy degradation, demonstrating a practical efficiency–accuracy trade-off and establishing KV reuse as a promising path for real-time RAG applications where system simplicity and inference speed are the primary priorities.**



***Index Terms*—KV cache, RAG, TTFT, LLMs.**


## I. Introduction

Over the past three years, large language models (LLMs) have driven rapid progress in artificial intelligence, particularly in natural language understanding and generation. Despite these advances, LLMs remain prone to hallucinations, often producing plausible but factually incorrect outputs. These errors arise from several factors, including reliance on statistical correlations rather than verified knowledge, training objectives that reward confident guessing, and the lack of explicit grounding in external information sources [1]. Retrieval-Augmented Generation (RAG) has emerged as an effective approach to mitigate these issues by combining LLMs with real-time retrieval from curated knowledge bases or document collections. By grounding generation in external evidence, RAG significantly improves factual accuracy and reliability.

While RAG improves output quality, it introduces additional computational overhead during inference. In many real-world deployments, queries often retrieve overlapping or identical document chunks due to skewed retrieval distributions and recurring information needs. As a result, the same retrieved passages may be repeatedly processed by the language model across different queries, sessions, or users. This observation suggests an opportunity to reuse intermediate model states to reduce redundant computation.

Modern LLM inference pipelines typically consist of a prefilling stage followed by autoregressive decoding. During prefilling, the model processes all input tokens to compute internal attention states. To accelerate decoding, most LLM systems store the key and value tensors generated during prefilling in a key–value (KV) cache. This cache enables subsequent tokens to reuse previously computed attention states, significantly reducing computational cost during generation.

Recent LLM serving systems have explored extending KV reuse across requests. For example, systems such as vLLM [2] and SGLang [3] enable prefix-based KV sharing, where requests with identical token prefixes can reuse cached attention states at the block level rather than recomputing them. These approaches demonstrate that cross-request KV reuse can substantially improve throughput and latency in multi-request serving environments.

However, directly applying prefix-based KV reuse to RAG workloads remains challenging. In RAG pipelines, prompts are dynamically constructed by combining user queries with retrieved passages. Because KV states are both context-dependent and position-dependent, even small changes in query wording, retrieval ordering, or passage selection can lead to different KV representations. As a result, simple prefix matching is insufficient for enabling effective KV reuse across RAG generations.

To overcome the limitations of prefix-based KV reuse in dynamic RAG settings, several approaches have been proposed to enable cache reuse under non-identical contexts. CacheBlend [4] improves reuse reliability by selectively recomputing a small subset of tokens within reused chunks to restore missing cross-attention interactions with preceding text. Similarly, Cache-Craft [5] identifies which cached chunks remain valid under prefix shifts and performs minimal recomputation only for tokens that lose contextual alignment. While effective, these approaches typically require tight integration with the LLM runtime and fine-grained control over attention computation.

Other methods focus on restructuring prompts or modifying attention computation to facilitate modular reuse. Prompt


Corresponding author: Dr. Jundong Liu. Email: liuj1@ohio.edu.


This work is supported in part by Stocker Fellowship program.

Cache [6] introduces modular attention reuse by decomposing prompts into reusable modules through a Prompt Markup Language (PML). However, this requires users or systems to follow a predefined prompt structure, which may reduce usability in general RAG deployments. Block-Attention [7] restructures attention computation into independent blocks and retrains the LLM to operate under this modified mechanism, eliminating training–inference mismatch issues. TurboRAG [8] similarly relies on fine-tuning to stabilize cache reuse behavior. Although effective, such approaches introduce additional training cost and may create trade-offs between reuse efficiency and the base model's general-purpose performance.

Recent work has also explored accelerating RAG through direct manipulation of KV representations associated with retrieved contexts. For example, Superposition Prompting [9] processes input documents through independent parallel prompt paths and reduces computation through context pruning. However, this approach alters the effective token composition and context structure seen by the model, potentially introducing semantic approximation effects, and its effectiveness is tied to specific positional encoding assumptions, limiting portability across models.

Despite these advances, most existing methods rely on at least one of the following: model retraining, architectural modification, prompt structure constraints, or semantic alteration of input tokens. These requirements limit portability across models and deployment environments. In contrast, we propose a new design paradigm, termed Pure KV Manipulation, which achieves inference acceleration solely through cache-level operations without modifying model weights, prompt formats, or token semantics. By demonstrating that substantial speedups are achievable through weight-agnostic KV operations, we establish a new baseline for portable RAG acceleration and open a new research direction focused on cache-native optimization.

Our approach enables plug-and-play KV cache reuse by exploiting the structural regularity of RAG prompts. We treat the fixed system instruction as an anchor block and precompute candidate context blocks together with this anchor so that their KV states align with the prompt structure. During inference, the cached anchor and retrieved blocks are concatenated, after positional re-encoding, to form a ready-made prefix cache, allowing the model to skip recomputation and begin directly from the user query. We further evaluate two positional re-encoding techniques to correct misalignment during context integration and demonstrate experimentally that this approach improves both efficiency and accuracy in RAG tasks.

Under the Pure KV Manipulation paradigm, we deliberately operate under a strict plug-and-play design regime. Specifically, we impose the following constraints: (i) no modification to model weights or attention mechanisms, (ii) no retraining or fine-tuning, (iii) no prompt redefinition or markup language, and (iv) no selective token recomputation during inference. These constraints reduce system complexity and deployment cost, but also restrict how much contextual adaptation can be achieved when reusing cached KV states across different prompts.

Under this regime, attention and positional mismatches may introduce accuracy degradation. The goal of this work is therefore to control the efficiency–accuracy trade-off under pure KV-state manipulation. We demonstrate that meaningful latency reductions are achievable while keeping accuracy degradation within a bounded and predictable range. This framing distinguishes our work from prior approaches that relax these constraints through model modification, prompt engineering, or selective recomputation.

## II. Background

### A. Retrieval-Augmented Generation

RAG enhances LLMs by incorporating external authoritative knowledge. During inference, a retriever selects relevant documents and concatenates them with the system instruction and user query to form the model input. This design improves factual grounding but also creates very long prompts. Across queries, the same text often appears repeatedly: retrieved passages may recur for different questions, and the system instruction is typically fixed. Consequently, the prefill stage, where the model encodes the entire input to build KV caches, must frequently recompute identical information, resulting in significant latency and wasted computation.

### B. KV Cache in LLMs

During autoregressive decoding, transformers rely on self-attention, which scales quadratically with input length. To accelerate inference, modern LLMs cache the hidden key–value states of processed tokens, allowing each new token to attend to stored representations rather than recomputing attention over the entire sequence. This mechanism improves efficiency within a single generation. However, standard KV caching is limited to one prompt and does not leverage repeated content across multiple generations.

### C. KV Reuse for RAG and Its Challenges

Reusing cached blocks in RAG introduces several challenges. The first is attention mismatch: when context chunks are precomputed independently in a context-free manner, tokens within a block can only attend to others within the same block. In the ground-truth full sequence, however, each token should also attend to all preceding tokens across earlier blocks. This discrepancy produces inaccurate KV states and degrades generation quality.

The second is position encoding mismatch: in models with rotary position embeddings (RoPE) [10], the relative positions of tokens during precomputation differ from their actual positions when blocks are concatenated into the final prompt. Without correction, the cached states become inconsistent with the model's positional encoding, further compounding accuracy loss.

## III. Method

In this work, our goal is to enable efficient, plug-and-play KV cache reuse in RAG systems without fine-tuning the underlying language model. With the observation that system

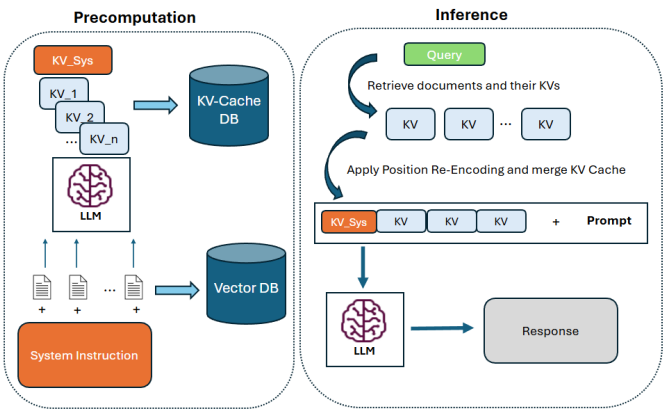


Fig. 1. Overview of our method: Each context block is precomputed along with the anchor block. During inference, the cached anchor and the kv cache of the retrieved blocks are combined after positional re-encoding to form a ready-made prefix cache.

instructions are typically fixed across queries [6], as shown in Fig. 1, we designed our approach by treating the system instruction block as an anchor. Each candidate context block is precomputed together with the anchor block so that their KV states align with the prompt structure. At inference time, the KV states of the anchor and retrieved blocks are concatenated to form a prefilled cache, allowing the model to skip prefix recomputation and begin directly from the user query. A positional re-encoding step is applied to ensure consistency of rotary embeddings when cached blocks are inserted into their proper locations in the final prompt.

Fig. 2 illustrates the conceptual attention mask when pairing the anchor (orange text and box) with a context block (blue text and box). The shaded region indicates how tokens within each block attend during precomputation and how the block-specific KV rows are extracted for reuse. Tokens in the query (green text and box) can attend to all preceding tokens.

### A. Block Precomputation

Let $c_0$ be the anchor block that always appears at the beginning of the prompt. During Precomputation, each context block $c_k$ is paired with the anchor and evaluated once. Let the tokenizations be

$$c_0 = \{a_1, \ldots, a_{n_0}\}, \quad c_k = \{b_1, \ldots, b_{n_k}\}$$
$$s_k = c_0 \oplus c_k = \{a_1, \ldots, a_{n_0}, b_1, \ldots, b_{n_k}\}$$

where $s_k$ denotes the concatenated sequence. A single forward pass over $s_k$ yields, for each transformer layer $\ell = 1, \ldots, L$, key and value matrices, with per-token key/value dimension $d_v$, are defined as,

$$\mathbf{K}^{(\ell)}_{s_k} = \mathbf{K}^{(\ell)}_{[c_0,c_k]} \text{ and } \mathbf{V}^{(\ell)}_{s_k} = \mathbf{V}^{(\ell)}_{[c_0,c_k]} \in \mathbb{R}^{(n_0+n_k)\times d_v}$$

whose rows align with the token positions in $s_k$. The first $n_0$ rows correspond to $c_0$ and the next $n_k$ rows correspond to $n_k$. To form the cache for $c_k$, we only keep the rows associated with the block $c_k$. Let $\mathcal{B}_k = \{n_0 + 1, \ldots, n_0 + n_k\}$ denote the row indices within $s_k$ that select the tokens of $c_k$. The block-specific cache extracted is

$$\begin{aligned} KV_{c_k} &= \left\{\left(\mathbf{K}^{\ell}_{c_k}, \mathbf{V}^{\ell}_{c_k}\right)\right\}_{\ell=1}^{L} \\ &= \left\{\left(\mathbf{K}^{\ell}_{s_k}[\mathcal{B}_k, :], \mathbf{V}^{\ell}_{s_k}[\mathcal{B}_k, :]\right)\right\}_{\ell=1}^{L} \end{aligned}$$

Intuitively, we "drop" the anchor rows and keep only the $c_k$ rows, but those rows were computed in the presence of $c_0$, aligning attention with the anchor (the attention-sink).

This procedure ensures that every block is encoded in the presence of the anchor, aligning its attention distribution with how it would appear in a real prompt. In our method, treating the system instruction as the anchor block—naturally fulfills the role of an attention sink. Because it is always present and placed at the start of the prompt, it becomes the consistent reference point that absorbs attention and anchors subsequent blocks [11], [12]. By pairing every block with this anchor during precomputation, we ensure that the KV states of each block are encoded under realistic attention conditions. This aligns with the insights from Star Attention [13]: leveraging the sink phenomenon leads to more stable attention distributions and improves the reliability of KV cache reuse.

### B. Position Encoding Handling

A fundamental challenge in precomputing blocks independently is the misalignment of positional indices. Popular LLMs typically employ RoPE for position encoding, where each token's embedding depends on its position in the full prompt sequence. However, when a block is precomputed in isolation, each token is assigned a position relative to its local offset $|c_0|$ + offset. Once integrated into the complete sequence, the token's true absolute position should be the sum of all preceding block lengths plus its local offset. This mismatch invalidates the cached states due to incorrect positional data. To address this issue, we adopted two approaches.

**Position Re-encoding** RoPE encodes positional information by rotating the query and key vectors according to their position indices. Since this rotation is a linear transformation, it can be reversed and reapplied for new positions. Specifically, for a token in block $c_k$ at local offset $j$, its position during precomputation was:

$$p_{\text{old}} = |c_0| + j$$

while in the full prompt its true position is

$$p_{\text{new}} = p_{\text{pre}} + j$$

where pre is the number of tokens preceding $c_k$ in the constructed prompt. The constant shift is therefore

$$\delta p = p_{\text{new}} - p_{old} = p_{\text{pre}} - |c_0|$$

The cached $\mathbf{K}$ is corrected as

$$\widetilde{\mathbf{K}}^{(\ell)}_{c_k} = \mathcal{R}_{\Delta p}(\mathbf{K}^{(\ell)}_{c_k})$$

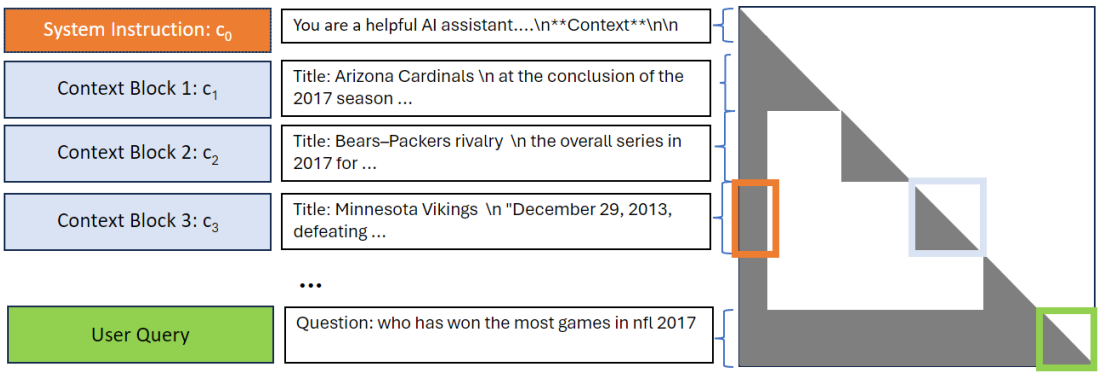


Fig. 2. Conceptual attention mask for one generation. Each candidate block $c_k$ is paired with the anchor $c_0$. The attention mask (shown on the right) illustrates the token-level attention pattern during precomputation. Block-specific KV states are obtained by discarding the anchor rows. See text for details.

TABLE I
ACCURACY COMPARISON OF DIFFERENT KV CACHE REUSE METHODS ACROSS FOUR QA DATASETS. EM DENOTES *Exact Match*.

| | TQA | | Musique | | 2Wiki | | NQ | |
|---|---|---|---|---|---|---|---|---|
| | EM | F1 | EM | F1 | EM | F1 | EM | F1 |
| Full Recompute | 69.5 | 66.3 | 22.0 | 22.4 | 32.0 | 32.0 | 48.0 | 44.7 |
| **Direct Reuse** | **50.5** | **53.2** | **5.0** | **10.2** | **15.0** | **18.0** | **28.0** | **33.8** |
| Anchor + Re-Encoding | **61.0** | 61.0 | **13.5** | **15.2** | 26.5 | 26.9 | 36.5 | **39.2** |
| Anchor + Decoupling | **61.0** | **61.4** | 13.0 | 14.6 | **27.0** | **27.4** | **37.0** | 39.1 |

where $\mathcal{R}_{\Delta p}$ is the RoPE rotation by shift $\Delta p$. Cached $\mathbf{V}$ remain unchanged:

$$\widetilde{\mathbf{V}}_{c_k}^{(\ell)} = \mathbf{V}_{c_k}^{(\ell)}.$$

In our implementation, we adopt the same strategy as [7]: the cached key vectors are first rotated back to zero to remove the old positional encoding, and then rotated forward to their actual positions to apply the correct encoding.

**RoPE De-coupling** Another way to address the positional encoding mismatch is to decouple positional information from the token embeddings. This can be done by caching the KV states without applying any positional encoding during precomputation. At runtime, once the full prompt is constructed, the correct positional encodings are then applied to the cached states based on their true positions in the sequence.

### C. Generation Procedure

When a user query arrives, the retriever selects the relevant context blocks as well as their precomputed caches. After positional adjustment of each block's keys, the full prefilled cache is formed by concatenating the anchor cache with the corrected block caches in order.

The combined cache, consisting of the anchor KV together with the selected context KVs, forms the prefilled cache. Newly generated tokens then attend to this prefilled cache during autoregressive decoding, enabling efficient inference without recomputing the long context.

## IV. EXPERIMENTS AND RESULTS

We evaluated our plug-and-play fast RAG design on four open-domain QA benchmarks: TriviaQA (TQA) [14], MusiQue [15], 2WikiMultiHopQA (2Wiki) [16], and Natural Questions (NQ) [17], each with 200 randomly sampled examples. The base model is `Llama-3.2-3B-Instruct` with greedy decoding. All experiments were run on a single NVIDIA RTX A6000 GPU. Passages are retrieved by `Contriever-MSMARCO` [18] and split into blocks of approximately 512 tokens [19], [20] using `LangChain`. Each prompt begins with a fixed system anchor block of 332 tokens, followed by the retrieved contexts and the user query.

We employed two complementary metrics, computed after normalizing both answers and predictions, to evaluate answer quality. *Subspan EM* (Exact Match) counts a prediction as correct if it contains any ground-truth answer as a contiguous substring, ensuring the essential answer is present. *F1* computes the token-level harmonic mean of precision and recall, rewarding partial overlaps. While F1 reflects overlap, it alone cannot distinguish between answers that fully contain the correct span and those that only share some tokens. Using both metrics together provides a stricter and more comprehensive assessment of answer quality.

Table I reports the accuracy of different cache reuse methods. **Full Recompute** serves as the ground-truth reference, where LLMs prefill the KV cache by performing attention over the entire input sequence. **Direct Reuse** is the naive baseline in which caches are pre-computed independently for each context block and then concatenated at inference. This approach is efficient but ignores both anchor information and positional consistency. Our proposed methods, grouped under **Anchor**, address these limitations by incorporating anchor information and maintaining positional consistency. **Anchor + Re-Encoding** applies positional re-encoding during inference

TABLE II
TIME-TO-FIRST-TOKEN (TTFT) COMPARISON OF DIFFERENT KV CACHE REUSE METHODS ACROSS FOUR QA DATASETS.

| TTFT (seconds) | TQA | Musique | 2Wiki | NQ |
|---|---|---|---|---|
| Full Recompute | 0.489 | 0.422 | 0.430 | 0.49 |
| Direct Reuse | 0.056 | 0.049 | 0.049 | 0.055 |
| Anchor + Re-Encoding | 0.084 | 0.073 | 0.073 | 0.083 |
| Anchor + Decoupling | 0.085 | 0.073 | 0.074 | 0.084 |

to restore alignment, while **Anchor + Decoupling** introduces an alternative strategy that separates position encoding from content encoding to further enhance robustness.

It is evident from the results that our proposed methods under the *Anchor* group significantly outperform the baseline *Direct Reuse* model, demonstrating the effectiveness of our cache reuse design. In theory, *Re-Encoding* and *Decoupling* are equivalent approaches, and the minor performance differences observed between them, as shown in Table I, can be attributed to engineering disparities rather than fundamental design differences.

Time consumption is reported as TTFT in seconds in Table II. *Direct Reuse* is the most straightforward and lightweight approach, as clearly reflected in the results. Our proposed Anchor-based methods, while slightly slower than Direct Reuse due to additional overhead, still achieve substantial speedup of approximately $5.9\times$ over the *Full Recompute* baseline. Our methods introduce a modest latency overhead relative to *Direct Reuse*. The additional TTFT is less than 6% of *Full Recompute* on average across the four datasets. However it recovers a substantial amount of answer quality. Figure 3 visualizes the efficiency–accuracy trade-off and shows that our method shifts the operating point closer to the upper-left performance region. Compared to *Direct Reuse*, our approach substantially improves answer quality, while remaining significantly faster than *Full Recompute*. Overall, our method achieves a more favorable balance between latency and accuracy, capturing most of the accuracy benefits of full recomputation while retaining much of the efficiency advantage of cache reuse.

## V. DISCUSSIONS AND CONCLUSIONS

In this work, we presented a plug-and-play, fine-tuning-free method for KV cache reuse in retrieval-augmented generation. By introducing an anchor block with position adjustment, our approach enables precomputed context caches to be recombined across queries while remaining compatible with the model's native attention mechanism. Experiments show that substantial latency reductions can be achieved while maintaining answer quality under strict plug-and-play constraints.

The experimental results illustrate the efficiency–accuracy trade-off inherent to strict plug-and-play KV reuse. Full recomputation provides an accuracy upper bound at high latency, whereas direct KV reuse minimizes latency but introduces substantial accuracy degradation due to attention isolation and positional mismatch. Anchor-based reuse operates between these extremes, recovering much of the lost accuracy while preserving most of the latency advantage.

These findings suggest that non-intrusive KV-state reuse can reduce inference latency, and that lightweight alignment mechanisms can partially compensate for context misalignment without requiring model modification. However, Our study evaluates only a limited set of operating points and does not attempt to exhaustively characterize the full accuracy–latency frontier. Beyond the specific method, this work establishes pure KV-state manipulation as a viable design paradigm for portable RAG acceleration. Our results suggest that meaningful efficiency gains are possible without modifying model weights, prompt formats, or inference pipelines, providing a practical path toward low-latency, real-time RAG systems.

We also observe that the effectiveness of positional correction varies across datasets, suggesting that task structure and reasoning depth may influence sensitivity to positional mismatch. A more fine-grained analysis of these factors, along with broader exploration of hybrid approaches beyond strict plug-and-play reuse, represents a promising direction for future work.

## REFERENCES


[1] A. T. Kalai, O. Nachum, S. S. Vempala, and E. Zhang, "Why language models hallucinate," 2025.

[2] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," 2023. [Online]. Available: https://arxiv.org/abs/2309.06180

[3] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "Sglang: Efficient execution of structured language model programs," 2024. [Online]. Available: https://arxiv.org/abs/2312.07104

[4] J. Yao, H. Li, Y. Liu, S. Ray, Y. Cheng, Q. Zhang, K. Du, S. Lu, and J. Jiang, "CacheBlend: Fast large language model serving for RAG with cached knowledge fusion." [Online]. Available: http://arxiv.org/abs/2405.16444

[5] S. Agarwal, S. Sundaresan, S. Mitra, D. Mahapatra, A. Gupta, R. Sharma, N. J. Kapu, T. Yu, and S. Saini, "Cache-craft: Managing chunk-caches for efficient retrieval-augmented generation," 2025. [Online]. Available: https://arxiv.org/abs/2502.15734

[6] I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong, "Prompt cache: Modular attention reuse for low-latency inference." [Online]. Available: http://arxiv.org/abs/2311.04934

[7] E. Sun, Y. Wang, and L. Tian, "Block-attention for efficient RAG." [Online]. Available: http://arxiv.org/abs/2409.15355

[8] S. Lu, H. Wang, Y. Rong, Z. Chen, and Y. Tang, "TurboRAG: Accelerating retrieval-augmented generation with precomputed KV caches for chunked text," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng, Eds. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 6588–6601. [Online]. Available: https://aclanthology.org/2025.emnlp-main.334/

[9] T. Merth, Q. Fu, M. Rastegari, and M. Najibi, "Superposition prompting: Improving and accelerating retrieval-augmented generation," in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. PMLR, 21–27 Jul 2024, pp. 35 507–35 527. [Online]. Available: https://proceedings.mlr.press/v235/merth24a.html

[10] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," 2023. [Online]. Available: https://arxiv.org/abs/2104.09864

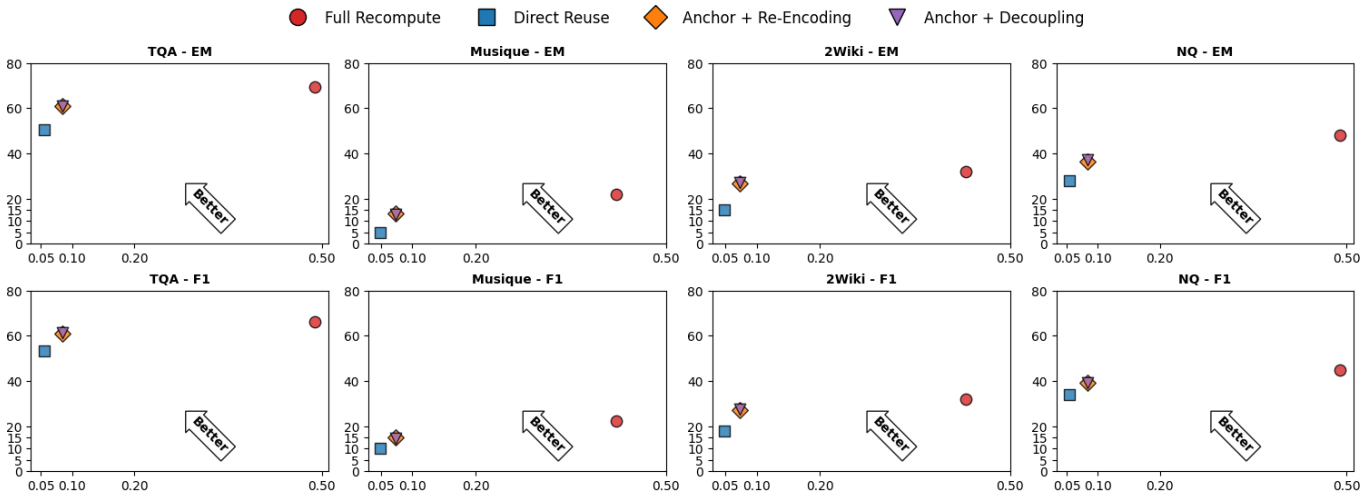


Fig. 3. Efficiency–accuracy trade-off of KV-cache reuse methods on four QA benchmarks. The x-axis reports TTFT and the y-axis reports answer quality (Subspan EM/F1). The arrow indicates the direction of improved performance (upper-left: lower latency, higher accuracy). Our method lies closer to this upper-left performance region than *Direct Reuse* and achieves a more favorable efficiency–accuracy balance than *Full Recompute*.


[11] C. Xiao, P. Zhang, X. Han, G. Xiao, Y. Lin, Z. Zhang, Z. Liu, and M. Sun, "Infllm: training-free long-context extrapolation for llms with an efficient context memory," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS '24. Red Hook, NY, USA: Curran Associates Inc., 2025.

[12] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," 2024. [Online]. Available: https://arxiv.org/abs/2309.17453

[13] S. Acharya, F. Jia, and B. Ginsburg, "Star attention: Efficient llm inference over long sequences," *arXiv preprint arXiv:2411.17116*, 2024.

[14] M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer, "TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, R. Barzilay and M.-Y. Kan, Eds. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1601–1611. [Online]. Available: https://aclanthology.org/P17-1147/

[15] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal, "MuSiQue: Multihop questions via single-hop question composition," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 539–554, 2022. [Online]. Available: https://aclanthology.org/2022.tacl-1.31/

[16] X. Ho, A.-K. Duong Nguyen, S. Sugawara, and A. Aizawa, "Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps," in *Proceedings of the 28th International Conference on Computational Linguistics*, D. Scott, N. Bel, and C. Zong, Eds. Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 6609–6625. [Online]. Available: https://aclanthology.org/2020.coling-main.580/

[17] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, K. Toutanova, L. Jones, M. Kelcey, M.-W. Chang, A. M. Dai, J. Uszkoreit, Q. Le, and S. Petrov, "Natural questions: A benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 452–466, 2019. [Online]. Available: https://aclanthology.org/Q19-1026/

[18] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave, "Unsupervised dense information retrieval with contrastive learning," 2021. [Online]. Available: https://arxiv.org/abs/2112.09118

[19] S. R. Bhat, M. Rudat, J. Spiekermann, and N. Flores-Herr, "Rethinking chunk size for long-document retrieval: A multi-dataset analysis," 2025. [Online]. Available: https://arxiv.org/abs/2505.21700

[20] R. Theja, "Evaluating the ideal chunk size for a rag system using llamaindex," Oct. 2023. [Online]. Available: https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5