A^3: Attention-Aware Accurate KV Cache Fusion for Fast Large Language Model Serving

Anonymous ACL submission

Abstract

Large language models (LLMs) have demonstrated strong capabilities in processing long contexts, enabling them to tackle tasks involving long textual inputs such as multi-turn conversations, legal documents, or retrieved documents in Retrieval-Augmented Generation (RAG) systems. However, despite their ability to handle long sequences, the resulting decoding latency and memory overhead remain substantial, posing challenges for real-world deployment. Recent advances in KV Cache reuse have shown potential to mitigate these costs, but still suffer from notable performance degradation. To address this issue, we conduct an in-depth investigation of recomputation-based reuse methods and observe that the recomputed tokens often fail to align with the context segments most relevant to the question. This misalignment hinders proper updates to the critical contextual representations. Therefore, we propose the **A**ttention-**A**ware **A**ccurate KV Cache Fusion algorithm (A^3), which precomputes and selectively fuses the KV Cache of text chunks based on their relevance to the question, achieving accurate integration with minimal computational overhead. Extensive experiments on various benchmarks and LLMs demonstrate that A^3 achieves the best task performance compared to four baselines while reducing the time-to-first-token (TTFT) by 2×.

1 Introduction

Large language models (LLMs) (Yang et al., 2024; Zhao et al., 2024; Guo et al., 2025; Dubey et al., 2024) have demonstrated remarkable capabilities in processing long-context inputs across a wide range of tasks, such as multi-turn dialogue generation (Yi et al., 2024), long-document question answering (Dasigi et al., 2021), or evidence aggregation in RAG systems (Wang et al., 2024; Li et al., 2024b).

Nevertheless, processing long textual inputs introduces substantial time and memory costs (Wu et al., 2024; Zhong et al., 2024) during autoregressive decoding, which degrades the user experience. To accelerate inference, researchers adopt techniques such as model pruning (Ma et al., 2023; Frantar and Alistarh, 2023), model quantization (Frantar et al., 2022; Lin et al., 2024), and token eviction (Zhang et al., 2023; Chen et al., 2024). Recently, a promising solution is cross-request KV Cache reuse, which precomputes the KV Cache of potentially repeated text chunks across different inputs and reuses them in the future to reduce KV Cache computation.

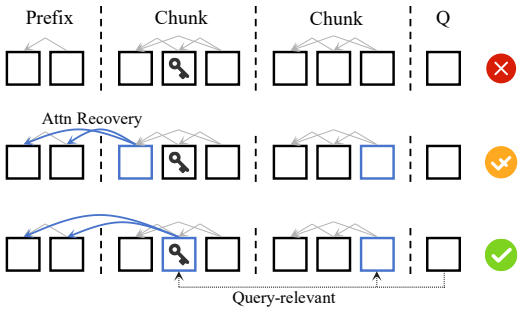


Figure 1: Blue blocks means the corresponding KVs are recomputed. (Top) Without recomputation. (Middle) Recomputation applied but key information missed. (Bottom) Key information updated based on query-aware attention.

However, KV Cache reuse is lossy, with the loss primarily arising from the mismatch between the encoded positions during precomputation and the actual positions during inference (position loss), as well as the attention loss where later text blocks miss the attention from earlier ones. Ratner et al. (2023); He et al. (2025) utilize the properties of ROPE to achieve position recovery, while Yao et al. (2025); Hu et al. (2025) use recomputation-based methods to update the KV Cache for important tokens to approximate attention recovery. Nevertheless, these methods still exhibit a significant performance gap compared to not using any reuse

methods, particularly in long-text key information retrieval tasks.

We believe that position recovery is indispensable, but further recomputation yields limited improvements. We find that whether it is Yao et al. (2025) updating tokens according to the largest value difference or Hu et al. (2025) using the sink rule (Xiao et al., 2024) to update the head and tail tokens of chunks, both approaches neglect the role of the user's query. This leads to a misalignment between the recomputed tokens and the tokens focused on by the user query, which is important in retrieval tasks. High-attention tokens are not involved in the recomputation, while the updated tokens may be irrelevant to the query.

Therefore, we propose Attention-Aware Accurate KV Cache Fusion, which selects recomputation tokens based on the attention from the user query to the documents, enabling accurate KV Cache reuse. Figure 1 provides a brief visual illustration. Our experiments across three LLMs and three benchmarks demonstrate that A^3 achieves the best task performance among four baselines, while maintaining a comparable computational overhead. Furthermore, A^3 is orthogonal to KV Cache compression methods, such as the quantization method KIVI (Liu et al., 2024b) and the eviction method SnapKV (Li et al., 2024a). Given that KV reuse only optimizes the time latency of the first token, we additionally combined the token eviction method to optimize TPOT as an acceleration extension for A^3. This yields greater throughput and reduced decoding time at a minimal cost.

In summary, our contributions are as follows:

- We propose A^3, a novel Attention-Aware Accurate KV Cache reuse algorithm that selects recomputation tokens based on the attention from the user query to the document, effectively bridging the mismatch between updated tokens and query-relevant content.
- We conduct extensive experiments on three LLMs across three long-context benchmarks, showing that A^3 achieves the best performance among baselines while maintaining comparable computational overhead.
- We further design a lightweight acceleration extension by integrating token eviction, which significantly reduces the TPOT and improves inference throughput, making A^3 practical for real-world deployment.

2 Related Work

2.1 KV Cache Reuse

Work on KV Cache reuse can be roughly categorized by both the position of the reused chunks and whether additional training is required. The first is prefix caching (Zheng et al., 2023; Jin et al., 2024; Liu et al., 2023), which reuses only the prefix text chunk. In contrast, in this paper, we focus on reusing text chunks from arbitrary positions within the input. For example, *Prompt Cache* (Gim et al., 2024) proposes a modular caching scheme based on Prompt Markup Language. APE (Yang et al., 2025b) restores accuracy by incorporating a shared prefix and using tunable attention temperatures and scaling factors. Furthermore, PCW (Ratner et al., 2023) and PIE (He et al., 2025) try to solve position loss based on ROPE (Su et al., 2024). Yao et al. (2025) and Hu et al. (2025) developed CacheBlend and LegoLink, employing recomputation techniques in an attempt to restore the cross-attention between text chunks. CacheBlend recomputes tokens where the concatenated value states exhibit significant deviation compared to true values. LegoLink, leveraging the properties of Attention Sink (Xiao et al., 2024), recomputes a few initial and final tokens within each text chunk.

There are also training-based methods for KV reuse, such as BlockAttention (Ma et al., 2025) and KVLink (Yang et al., 2025a). Our work follows the recomputation-based approach, aiming to identify a tuning-free, efficient, and accurate reuse paradigm.

2.2 KV Cache Compression

As the length of texts continues to increase, the memory overhead of KV Cache becomes increasingly non-negligible (Su et al., 2025). To address this issue, researchers have explored two main directions. One line of work focuses on compressing the KV Cache through quantization techniques, reducing high-precision representations to lower-precision ones. Representative methods include KIVI (Liu et al., 2024b), KVQuant (Hooper et al., 2024), and GEAR (Kang et al., 2024). Another line of work adopts token eviction strategies, which selectively retain the KV Cache of important tokens. For example, StreamingLLM (Xiao et al., 2024) preserves the initial tokens in the prompt, H2O (Zhang et al., 2023) retains tokens with higher accumulated attention scores, and SepLLM (Chen et al., 2024) prioritizes special tokens. Our method runs orthogonally with these approaches but has

been rarely explored in practice. In this work, we attempt to integrate it with SnapKV (Li et al., 2024a) as an extension of our proposed method.

3 Method

3.1 Background

In long-context scenarios, the input typically consists of three components: a system prompt $\mathcal{S}$, a question Q, and a long text passage D relevant to the question. These passages are often composed of multiple smaller segments, i.e., $D = \{D_1, D_2, \cdots, D_k\}$. Given a transformer-based (Vaswani, 2017) LLM with parameters θ, the input $X = (\mathcal{S}, D, Q)$ of length n, and the corresponding hidden state representation $\boldsymbol{X}_l \in \mathbb{R}^{b \times n \times h}$ in layer l, where b is the batch size and h represents the hidden size, the model first computes and caches the KV tensors during the prefill phase. Specifically, to eliminate redundant computation of key and value vectors, LLMs typically store the KV Cache of the input in advance, i.e., $K_l = \boldsymbol{X}_l W_l^k$ and $V_l = \boldsymbol{X}_l W_l^v$ are computed and cached, where $W_l^k, W_l^v \in \mathbb{R}^{h \times h}$ are the weight matrices at layer l. After prefilling, LLM autoregressively decodes the next token with its KV Cache ($\in \mathbb{R}^{b \times 1 \times h}$), where the generated KV vectors are appended to the existing KV Cache before performing the attention computation.

3.2 Attention-Aware Accurate Fusion

Our method consists of three main stages: precomputation, position recovery, and attention recovery, along with an optional acceleration extension.

Precomputation. When reusing KV Cache, we first precompute and store the KV Cache of the system prompt $\mathcal{S}$ and reusable documents D, which can be formulated as:

$$K_i, V_i = \text{LLM}_\theta(i),\ i \in \{\mathcal{S}, D_1, D_2, \cdots, D_k\}, \tag{1}$$

where $\text{LLM}_\theta(\cdot)$ returns the KV Cache of text chunks for all layers. We concatenate each segment of $\{K, V\}_i \in \mathbb{R}^{b \times L \times |i| \times h}$ along the length dimension to obtain the K_{cat} and V_{cat}, where L represents the layer numbers of an LLM. During precomputation, documents are precomputed for their corresponding KV Cache, and the KV tensors are stored in persistent storage (e.g., SSD). Notably, we store the Keys before applying RoPE (pre-RoPE), allowing for correct position injection during the subsequent reuse phase.

Position Recovery. Precomputing all chunks independently leads to incorrect positional encoding, since in the concatenated input to LLMs, a document's position does not necessarily start from zero as it does during precomputation. For example, for document D_2 in input $X = (\mathcal{S}, \{D_1, D_2, \cdots, D_k\}, Q)$, its position during precomputation ranges from 0 to $|D_2| - 1$, while its position in X starts from $|\mathcal{S}| + |D_1|$. Here, we leverage the properties of Rotary Position Embedding (RoPE) (Su et al., 2024), which is widely used in most LLMs, to achieve position recovery.

For the embedding vector $\boldsymbol{x}_m \in \mathbb{R}^h$ at position m in input X and given $\Theta \in \left\{\theta_i = 10000^{\frac{-2i}{h}}, i \in [0, 1, \ldots, \frac{h-1}{2}]\right\}$, multiplying $\boldsymbol{x}_m W_k$ on the left by a rotation matrix $R^h_{\Theta,m}$ can incorporate the positional information, where

$$R^h_{\Theta,m} = \begin{pmatrix} \cos m\theta_0 & -\sin m\theta_0 & \cdots & 0 & 0 \\ \sin m\theta_0 & \cos m\theta_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cos m\theta_{\frac{h}{2}-1} & -\sin m\theta_{\frac{h}{2}-1} \\ 0 & 0 & \cdots & \sin m\theta_{\frac{h}{2}-1} & \cos m\theta_{\frac{h}{2}-1} \end{pmatrix}.$$

Therefore, if we store the KV Cache of documents without positional embedding during precomputation and apply a global positional embedding to the concatenated KV Cache, i.e., applying the rotation matrix $R^d_{\Theta,m}$ to each position based on the length from 0 to $|X| - 1$, the positions can be recovered. The concatenated Key tensor after position recovery can be formulated as:

$$K'_{cat} = R^h_{\Theta,m} K_{cat},\ m = \{0, 1, 2, \cdots\}. \tag{2}$$

Attention Recovery. Now, we have obtained the KV pair $\langle K'_{cat},\ V_{cat}\rangle$ with recovered positions, but the attention from the later chunks to the earlier ones is still missing. Based on the example from Qasper (Bai et al., 2024) shown in Figure 2, we observe that ①**existing recomputation-based methods tend to miss high-attention tokens**, achieving hit rates of only 30% and 37.5%, respectively. This limitation leads to suboptimal performance in the attention recovery process. As illustrated in Figure 3, we plot the L2 loss between the recomputed attention maps and the ground truth with two recomputation strategies. Both methods marginally reduce the attention loss compared to the baseline without recomputation, but the improvement is not substantial. In contrast, ②**we observe a more consistent reduction in attention loss when high-attention tokens are explicitly selected for recomputation**, across shallow (layer 4), middle (layers 13 and 22), and deep (layer 31) layers of the

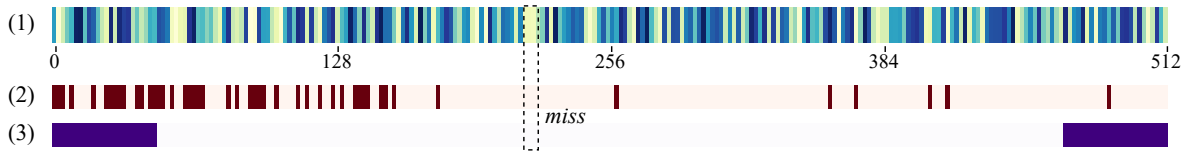


Figure 2: The mismatch between high-attention tokens and recomputed tokens. (1) The attention heatmap between the question and the document. We observe that both (2) the strategy of selecting tokens with the largest KV differences and (3) the strategy of selecting head and tail tokens of the document result in limited coverage of the high-attention tokens.

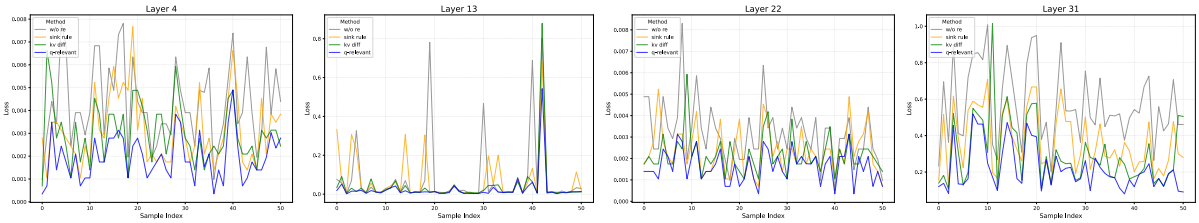


Figure 3: Attention recovery performance under different reuse strategies. The attention map obtained by recomputing question-relevant tokens is more consistent with the original attention map compared to other.

LLaMA3-8B-Instruct (Dubey et al., 2024), as evaluated on sampled data from the Qasper. These two observations support the rationale behind our proposed method, and its effectiveness on specific tasks will be detailed in the experiment section. In the following, we proceed to describe our approach formally.

For an L-layer decoder-only LLM, we index the layer number l from 0 to $L-1$. The input X is first passed through the embedding layer and decoder block 0, producing the actual query (Q_{true}), key (K_{true}), and value (V_{true}) representations at layer 1. At this point, we aim to identify the document tokens $\{d_1, d_2, ...\}$ that are semantically relevant to the question $Q = \{q_0, q_1, ...\}$, based on the shallow representations. We compute the attention score between each question token q_j and the document token d_i, yielding a question-relevant score s for d_i and Re for the recomputation tokens:

$$s(d_i) = \sum_j \text{Softmax}(\frac{q_j K_{true}^T}{\sqrt{h}})|_{d_i},$$
$$Re = top_p(s(d_i)),\ p = rn, \tag{3}$$

where r is a hyperparameter that determines the recomputation ratio and n is the input length. Therefore, the attention output of layer 1 is:

$$O_1 = \text{Softmax}(Q_{true}[Re, Q]K_{true}^T/\sqrt{h}) \cdot V_{true}. \tag{4}$$

Therefore, the states updated at layer l (> 1) is given by:

$$\boldsymbol{X}_l = H(O_{l-1}), Q_l = \boldsymbol{X}_l W_l^q,$$
$$K'_{cat_l}[Re, Q] \leftarrow \boldsymbol{X}_l W_l^k,$$
$$V_{cat_l}[Re, Q] \leftarrow \boldsymbol{X}_l W_l^v,$$
$$O_l = \text{Softmax}(\frac{Q_l K'_{cat_l}}{\sqrt{h}}) \cdot V_{cat}, \tag{5}$$

where $H(\cdot)$ denotes the composition of layer normalization and the feedforward network applied to the attention output.

Extention. Moreover, we investigate the integration of our method with orthogonal eviction strategies to further optimize the generation latency of non-initial tokens and reduce overall KV Cache usage. In this work, we instantiate this extension with KV eviction; the compatibility with other compression techniques such as quantization is discussed in the Appendix but not empirically evaluated here. Inspired by Xiao et al. (2024), we first retain the tokens corresponding to the system prompt $\mathcal{S}$ and the question Q. Since the KV Cache for $\mathcal{S}$ is lossless and typically small (usually $|\mathcal{S}| < 50$), we choose to preserve it entirely without imposing a fixed-size constraint (e.g., the first 4 tokens). Given a maximum KV Cache capacity C, we then adopt the strategy proposed in SnapKV (Li et al., 2024a) to adaptively select the top $C - |\mathcal{S}| - |Q|$ tokens at each layer and each attention head for retention. After the first token is generated, a unified KV Cache eviction is performed.

Methods	Single-Doc QA		Multi-Doc QA		Summarization		Few-shot Learning			Others		Avg.
	Qasper	MultiQ	Hotpot	2WikiM	GovRep	MultiN	TREC	TriviaQ	SAMSum	PCount	LCC	
LLaMA3-8B-Instruct												
Vanilla	40.52	45.65	48.81	33.51	33.11	25.59	68.33	89.45	39.82	12.06	57.41	44.93
FullReuse	4.63	5.27	3.48	1.71	13.96	7.48	0.67	1.55	1.89	4.25	25.43	6.39
PIE	34.71	30.07	38.55	30.29	27.55	23.52	61.33	88.30	39.30	5.67	64.12	40.31
CacheBlend	35.03	32.82	36.97	28.53	28.47	24.45	67.67	87.58	39.07	5.67	56.29	40.23
LegoLink	34.53	30.79	36.15	30.46	27.49	23.96	67.00	87.20	39.16	7.00	56.52	40.02
ours	**39.07**	**36.46**	**42.17**	**32.56**	**28.88**	**24.55**	**68.67**	**89.25**	**39.33**	**8.67**	**65.39**	**43.18**
Mistral-7B-Instruct-v0.2												
Vanilla	20.09	42.59	24.10	18.37	32.35	25.43	62.33	88.60	40.47	3.92	62.68	38.27
FullReuse	5.00	5.42	0.97	0.39	22.37	21.20	0.33	2.36	2.38	1.83	27.80	8.19
PIE	20.26	38.85	24.14	17.15	31.73	25.62	59.33	**76.95**	39.86	5.30	62.34	36.50
CacheBlend	**21.72**	39.17	25.19	18.19	32.24	**25.74**	59.67	75.04	40.16	3.39	61.82	36.58
LegoLink	21.38	**40.14**	25.46	17.13	32.03	25.72	59.67	75.82	40.16	**4.34**	63.57	36.86
ours	21.12	39.31	**25.52**	**18.38**	**32.41**	25.28	**59.67**	76.44	**40.25**	3.15	**65.22**	**36.98**

Table 1: Performance comparison of our method with *FullReuse*, *PIE*, *CacheBlend*, *LegoLink*, and up-bound method Vanilla on LongBench for LLaMA3-8B-Inst and Mistral-7B-Inst-v0.2. The best results are highlighted in **bold**.

4 Experiments

4.1 Experiment Setup

Datasets. We select the following three widely used long-text benchmarks that are commonly adopted in both industry and academia to evaluate advanced models:

- **LongBench** (Bai et al., 2024): a multi-task long-context benchmark. We select eleven representative datasets from it, covering single- and multi-document question answering, summarization, few-shot learning, code completion, and more.
- **Needle-in-a-Haystack** (Liu et al., 2024a): a retrieval benchmark that tests whether models can recover a target fact embedded in long contexts.
- **Ruler** (Hsieh et al., 2024): a benchmark for evidence localization and reasoning over long unannotated contexts.

Models and Baselines. We adopt LLaMA3-8B-Inst (Dubey et al., 2024), Mistral-7B-Inst-v0.2 (Jiang et al., 2023), and Qwen2.5-7B-Inst (Yang et al., 2024) as backbone models. For fair comparison, we primarily consider five baseline methods, including the following:

- Vanilla: It does not employ any modifications, such as KV Cache reuse. Vanilla performs inference with FP16 precision (applied consistently across all methods) and serves as the upper bound for all following reuse approaches.
- *FullReuse*: It represents the most basic reuse method, where the saved KV Caches corresponding to $\mathcal{S}$ and $\mathcal{D}$ are simply concatenated without any further modifications.
- *PIE* (He et al., 2025): Building upon *FullReuse*, it encodes the correct positional information for the concatenated KV Cache without any additional recomputation.
- *CacheBlend* (Yao et al., 2025): A recomputation-based method, selectively recomputing the KV Cache for a subset of tokens after position recovery. Specifically, it updates the tokens with the largest discrepancies between the concatenated values and the ground-truth values.
- *LegoLink* (Hu et al., 2025): A recomputation-based method that, following the findings of StreamingLLM (Xiao et al., 2024), selects several tokens at the beginning and end of each document for KV Cache recomputation.

Details. We split the input into several chunks for all benchmarks with a chunk size of 512, which is consistent with *CacheBlend*. For a fair comparison, we use the same recomputation ratio r=0.15 for both *CacheBlend* and A^3 in the main experiments. This choice serves as a representative trade-off between accuracy and recomputation overhead. Similarly, for *LegoLink*, we recompute the first and last 20 tokens of each chunk.

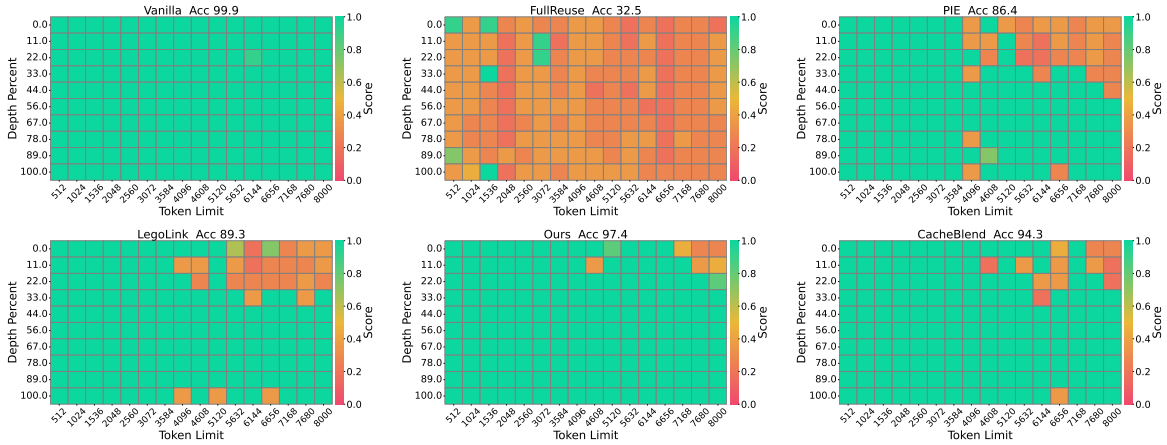


Figure 4: Results of Needle-in-a-Haystack on LLaMA-3-8B-Instruct. The vertical axis of the figure represents the depth percentage, and the horizontal axis represents the token length. Our method achieves the best retrieval performance.

Methods	Single NIAH			Multi-key NIAH								
	Single-1	Single-2	Single-3	Multi-1	Multi-2	Multi-3	MQuery	MValue	CWE	FWE	VT	Avg.
Vanilla	99.67	98.67	100.0	100.0	100.0	99.33	93.25	99.08	97.80	94.67	100.0	98.41
FullReuse	69.00	78.67	27.67	23.67	0.00	1.33	35.42	38.08	67.30	92.78	41.60	43.23
PIE	98.00	**92.67**	94.00	66.33	82.67	73.00	65.00	40.92	86.10	97.00	50.93	76.97
CacheBlend	**100.0**	87.00	96.00	83.00	83.89	**78.67**	**78.17**	**57.75**	**94.50**	94.44	69.27	83.88
LegoLink	91.33	83.33	82.67	78.00	79.67	74.33	74.58	46.17	94.00	97.00	66.60	78.88
ours	98.00	91.33	**98.67**	**83.67**	**86.00**	76.33	75.92	56.00	92.57	**97.11**	**70.27**	**84.17**

Table 2: Performance comparison of our method with *FullReuse*, *PIE*, *CacheBlend*, *LegoLink*, and up-bound method Vanilla on Ruler for Qwen2.5-7B-Inst. The best results are highlighted in **bold**.

4.2 Task Performance

Due to space constraints, several results on specific models are provided in the Appendix (Supplementary Material) for reference, such as LongBench results for Qwen2.5-7B-Inst.

Results on LongBench. Table 4 presents results of various models and reuse methods on the LongBench. We report F1 scores for Single-Document QA, Multi-Document QA tasks, and TriviaQA; ROUGE-L (Lin, 2004) for summarization tasks and SAMSum; Exact Match for TREC and PassageCount; and fuzzy string matching for evaluating code similarity in the code completion task (LCC). Our method achieves **the best** average performance, particularly on LLaMA3-8B-Inst, where it consistently outperforms all other reuse baselines across every sub-dataset.

Moreover, compared to the Vanilla setting, our approach achieves near lossless performance (36.98 vs 38.27 in Mistral). From Table 4, we observe that position recovery is essential, as evidenced by the poor performance of *FullReuse*. In addition, reuse methods perform well on summarization tasks such as MultiNews and SAMSum, likely because missing some key information has a limited impact on overall generation quality. In contrast, on QA tasks like Qasper and HotpotQA, baseline methods exhibit a noticeable drop in performance (40.52 $\rightarrow$ 35.03). While our method significantly outperforms them (39.07 $\rightarrow$ 35.03), indicating that our recomputation strategy effectively captures critical information necessary for producing correct answers.

Results on Needle-in-a-Haystack. The six subfigures in Figure 4 present results of six methods. We report the ROUGE score between the generated response and the inserted needle text. The x-axis represents the input text length in the current experiment, while the y-axis indicates the relative position at which the needle is inserted into the input. The observed results align well with intuition: position recovery is essential (*PIE* $\gg$ *FullReuse*);

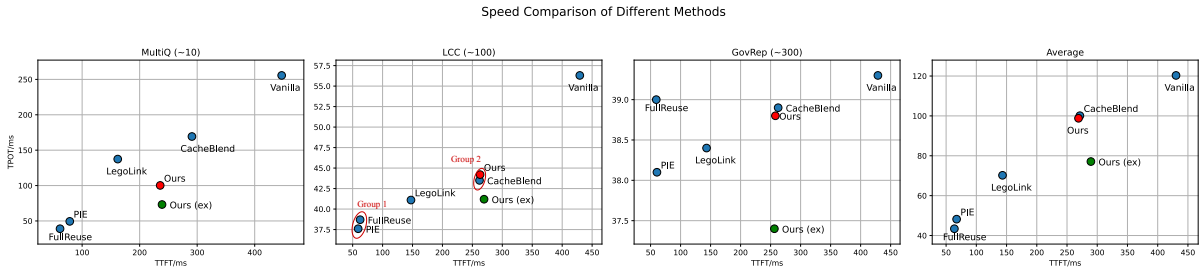
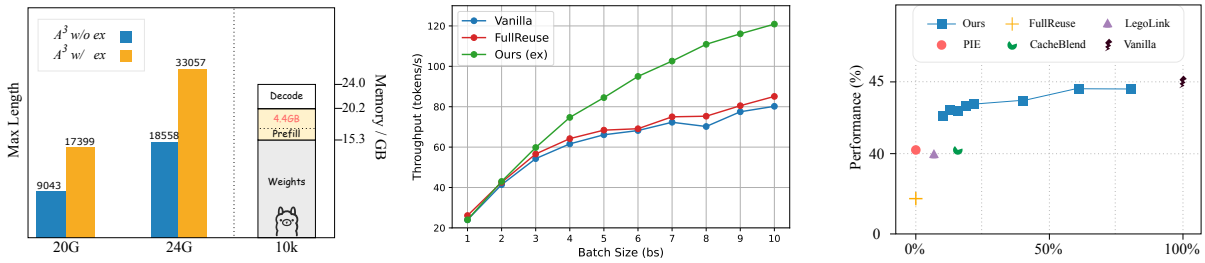


Figure 5: Comparison of inference performance among different reuse methods.



(a) Longer generation and GPU memory savings with extension.

(b) Throughput between vanilla, the most efficient method, and A^3 (ex).

(c) Impact of recomputation ratio of our method on LongBench.

Figure 6: Inference performance analysis of A^3, illustrating latency variations with the acceleration extension and ablation studies on key hyperparameters.

attention recovery is also needed (*LegoLink* > *PIE*); the data-driven approach outperforms the rule-based one (*CacheBlend* > *LegoLink*); and the attention-guided recomputation method outperforms the one driven by KV Cache discrepancy (*Ours* > *CacheBlend*). Nevertheless, none of the reuse methods reach the upper bound (Vanilla > *all*). In summary, our method achieves **the best** performance in Ruler, which demonstrates the key information retrieval capability of A^3.

Results on Ruler. We report the Exact Match score for each sub-dataset, as shown in Table 2. Our method achieves **the highest** average accuracy (84.17%) among all KV reuse baselines on Qwen2.5-7B-Inst. Compared to *FullReuse* (43.23%) and *PIE* (76.97%), our method significantly improves robustness across both Single and Multi-key NIAH tasks. While *CacheBlend* (83.88%) and *LegoLink* (78.88%) perform well, our method consistently outperforms them, especially on challenging tasks like FWE and VT, highlighting its advantage in maintaining high accuracy under efficient KV reuse.

4.3 Inference Performance

During inference, we conduct experiments using the Transformers framework in combination with the FlashInfer (Ye et al., 2025) acceleration module on NVIDIA 3090 GPUs. As shown in Figure 5, we evaluate the LLaMA3-8B-Inst model and report the TTFT and TPOT under different reuse strategies for generation lengths of 10, 100, and 300 tokens.

The six methods can be grouped based on their reuse and recomputation strategies. Group 1 includes *PIE* and *FullReuse*, which require no recomputation and therefore appear near the bottom-left corner of the plot. Group 2 includes A^3 and *CacheBlend*, which recompute the same number of tokens, with the only difference being the overhead in KV difference computation and attention computation (Eq. 3). *LegoLink* lies between Group 1 and Group 2, representing a trade-off between reuse granularity and recomputation. Lastly, *Vanilla*, which recomputes KV caches for the entire sequence, appears at the top-right of the figure due to its highest memory and computation cost.

Notably, our method **achieves efficiency comparable** to *CacheBlend*, while outperforming it in task performance. In particular, our method yields a speedup of about 2× over the vanilla approach, and up to 3.3× acceleration can be observed on larger models (e.g., 70B) (Yao et al., 2025). We also separately measure the time cost of computing Re (Eq. 3), which takes only 0.1 ms on average for an input length of 6,000 tokens. This aligns with the observation in Figure 5, where our method and

	Evict	Qasper	MultiQ	Hotpot	2WikiM	GovRep	MultiN	TREC	TriviaQ	SAMSum	PCount	LCC	Avg.
LLaMA	✗	39.07	36.46	42.17	32.56	28.88	24.55	68.67	89.25	39.33	8.67	65.39	43.18
	✓	36.19	35.05	42.31	32.68	25.84	22.90	67.00	89.25	37.64	8.67	61.34	41.72
Mistral	✗	21.12	39.31	25.52	18.38	32.41	25.28	59.67	76.44	40.25	3.15	65.22	36.98
	✓	18.87	39.42	25.67	17.28	30.03	24.70	58.67	82.65	39.74	2.67	62.89	36.60
Qwen	✗	38.96	41.18	38.92	25.77	31.04	22.61	64.00	86.56	40.42	4.87	62.44	41.52
	✓	33.50	39.38	39.21	26.04	28.55	21.55	61.67	85.98	39.39	6.03	59.17	40.04

Table 3: The impact of eviction-based extension on the task performance of our method. Experiments are conducted on three models and the LongBench benchmark.

CacheBlend appear close together.

When combined with eviction-based extension (*ex*), we observe a shift toward the bottom-right in the plot (red dot → green dot), indicating a slight increase in TTFT (approximately 5 ms) in exchange for **improved further decoding efficiency**. It is worth emphasizing that KV reuse itself does not bring any memory savings or significant throughput improvements. For instance, in the MultiQA dataset, this leads to a 25% reduction in TPOT. In our experiments, the maximum KV capacity C is set to 1024, which corresponds to approximately 17% of the input length. Additionally, we evaluate the maximum generation length and GPU memory savings before and after using *ex*. It is important to note that the reuse methods themselves do not introduce additional GPU memory overhead. As shown in Figure 6a, when the maximum allowed GPU memory is 20GB, the maximum generation length achievable with *ex* is comparable to that of the 24GB setting without *ex* (with an input length of 7044 tokens). Furthermore, when the input length reaches 10K tokens, A^3 (*ex*) reduces GPU memory consumption by approximately 4.4GB, which can then be repurposed for decoding.

Furthermore, we compare the throughput of *Vanilla*, the most efficient method *FullReuse* (disregarding task performance), and our method with extension. The input length is 2,000 tokens, and the output length is 100 tokens. From the results shown in Figure 6b, we can observe that the extension improves the throughput of A^3 and as the batch size increases, the extension leads to a significant improvement in throughput.

But what is the cost? We evaluate the impact of our extension strategy on three models across the LongBench and Ruler datasets, reporting results before and after applying KV eviction. A dash ("–") indicates that the model fails to generate intelligible output. As shown in Table 3, extension strategy does not lead to catastrophic performance degradation—despite being a combination of two lossy mechanisms. The overall performance **drop remains controllable**. For instance, on LongBench, the Mistral model experiences only a 1% decrease (36.98 →36.60) in average accuracy. Considering the inference speedup by *ex* discussed earlier, this represents a practical trade-off between performance and efficiency. To the best of our knowledge, this is the first work to demonstrate the feasibility of jointly optimizing KV reuse and KV eviction.

We further analyze the impact of different recomputation ratios in Figure 6c. Due to space constraints, we only plot the ratio sweep for A^3; *CacheBlend* is governed by the same accuracy-efficiency trade-off, and we discuss our choice of r=0.15 for it in the Appendix.

5 Conclusion

In this work, we propose A^3, an Attention-Aware Accurate KV Cache Fusion algorithm designed to address the performance degradation of existing KV reuse methods in long-context LLM inference. By leveraging query-aware attention to guide selective recomputation, A^3 effectively aligns KV updates with user-relevant information, mitigating attention loss without incurring significant overhead. Extensive experiments across multiple models and benchmarks demonstrate that A^3 outperforms prior reuse strategies in task performance while maintaining competitive decoding efficiency. Furthermore, we introduce an acceleration extension via token eviction, which complements A^3 by improving TPOT and overall throughput. Our findings highlight the feasibility and effectiveness of jointly optimizing KV reuse and eviction for practical long-context LLM applications.

Limitations

Although A^3 has achieved excellent results, there are still some limitations:

- Our experiments are conducted exclusively on LLMs based on the Transformer architecture. Our framework is not directly compatible with models built on alternative architectures or those that do not produce KV Caches, such as Mamba. Moreover, if a model does not use ROPE, position recovery during KV reuse becomes significantly more challenging.

- Additionally, KV reuse requires the offline storage (e.g., save on SSD) of a large number of KV Cache tensors. As a result, for large-scale external databases, the storage overhead may be substantial. Furthermore, loading precomputed KV Caches into the GPU imposes a heavy burden on the CPU, and high-concurrency requests may severely strain CPU performance.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3119–3137. Association for Computational Linguistics.

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. 2024. Sepllm: Accelerate large language models by compressing one segment into one separator. *arXiv preprint arXiv:2412.12094*.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zhenyu He, Jun Zhang, Shengjie Luo, Jingjing Xu, Zhi Zhang, and Di He. 2025. Let the code llm edit itself when you edit the code. In *The Thirteenth International Conference on Learning Representations*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Junhao Hu, Wenrui Huang, Haoyi Wang, Weidong Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. 2025. Epic: Efficient position-independent context caching for serving large language models. *Proceedings of the 42nd International Conference on Machine Learning*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2024. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. GEAR: an efficient KV cache compression recipe for near-lossless generative inference of LLM. *CoRR*, abs/2403.05527.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024a. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. 2024b. Retrieval augmented generation or long-context llms? a comprehensive study and hybrid approach. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 881–893.

Chin Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the Workshop on Text Summarization Branches Out, 2004*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 11:157–173.

Yuhan Liu, Hanchen Li, Kuntai Du, Jiayi Yao, Yihua Cheng, Yuyang Huang, Shan Lu, Michael Maire, Henry Hoffmann, Ari Holtzman, and 1 others. 2023. Cachegen: Fast context loading for language model applications. *CoRR*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Dongyang Ma, Yan Wang, and Lan Tian. 2025. Block-attention for efficient prefilling. *The Thirteenth International Conference on Learning Representations*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Nir Ratner, Yoav Levine, Yonatan Belinkov, Ori Ram, Inbal Magar, Omri Abend, Ehud Karpas, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. Parallel context windows for large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6383–6402.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Yi Su, Yuechi Zhou, Quantong Qiu, Juntao Li, Qingrong Xia, Ping Li, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. Accurate KV cache quantization with outlier tokens tracing. pages 12895–12915.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, and 1 others. 2024. Searching for best practices in retrieval-augmented generation. *arXiv preprint arXiv:2407.01219*.

Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 640–654.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 13 others. 2024. Qwen2 technical report. *CoRR*, abs/2407.10671.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. 2025a. Kvlink: Accelerating large language models via efficient kv cache reuse. *39th Conference on Neural Information Processing Systems*.

Xinyu Yang, Tianqi Chen, and Beidi Chen. 2025b. Ape: Faster and longer context-augmented generation via adaptive parallel encoding. In *The Thirteenth International Conference on Learning Representations*.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2025. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 94–109.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and 1 others. 2025. Flashinfer: Efficient and customizable attention engine for llm inference serving. In *Eighth Conference on Machine Learning and Systems*.

Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. 2024. A survey on recent advances in llm-based multi-turn dialogue systems. *arXiv preprint arXiv:2402.18013*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2024. A survey of large language models. *Preprint*, arXiv:2303.18223.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody_Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2023. Efficiently programming large language models using sglang.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210.

A Discussion and Future Work

Compatibility with KV Compression. As noted in our Limitations section, storage overhead is an inherent challenge for *all* KV reuse methods rather than a drawback unique to A^3. To mitigate this issue at the system level, we explicitly designed and evaluated an eviction-based extension in the main paper. Since KV eviction is itself a form of KV compression, our results already provide empirical evidence that A^3 can be effectively combined with compression-style techniques. More broadly, A^3 is also compatible with KV quantization methods such as KIVI, because our method mainly determines *which* tokens should be recomputed, whereas quantization determines *how* KV states are stored. Therefore, A^3 is orthogonal to both eviction-based and quantization-based compression, making reuse + eviction + quantization a natural unified direction for future system design. A full empirical study of this combined setting is left to future work, since jointly optimizing multiple compression paradigms would introduce additional implementation and tuning complexity beyond the scope of this paper.

Applicability to Multi-turn Dialogue. Our method is naturally compatible with multi-turn scenarios. At the end of each turn, the KV cache of the newly observed user query and the generated assistant response can be retained as reusable context. When a new turn arrives, the previous dialogue history can be treated as a set of cached documents, while the latest user utterance serves as the current query for A^3. The method then selects and recomputes the history tokens that are most relevant to the new query, avoiding repeated full prefilling of the entire conversation window. This suggests that A^3 is especially promising for long multi-turn interactions with recurring topics, although a dedicated system study on dialogue memory management remains future work.

B Clarification on LongBench Evaluation Settings

Model version mismatch with KVLink. We clarify that our LongBench results for HotpotQA and 2WikiMQA are obtained with LLaMA3-8B-Instruct, rather than Llama-3.1-8B-Instruct. Therefore, these numbers are not directly comparable to the higher results reported by KVLink, which are obtained on Llama 3.1. Moreover, even under the same Llama 3.1 backbone, the results reported by LegoLink and KVLink are themselves not identical. More generally, we acknowledge that the baseline comparison in our paper is conducted under a unified experimental setting across methods.

Prompting and evaluation protocol. For HotpotQA and 2WikiMQA, we follow the standard LongBench evaluation pipeline. The model receives the concatenated context and question in a single-turn QA prompt and is asked to produce a short answer directly. We do not use chain-of-thought prompting, self-consistency, retrieval augmentation beyond the benchmark input, or task-specific prompt tuning. The reported scores are computed with the official LongBench metrics.

Decoding and chunk-level reuse settings. As stated in the main paper, all LongBench inputs are split into chunks of size 512. We use greedy decoding (temperature $= 0$), and for a fair comparison, both *CacheBlend* and A^3 use the same recomputation ratio r=0.15, while *LegoLink* recomputes the first and last 20 tokens of each chunk.

C Other Results

For LongBench. We present the results of Qwen2.5-7B-Inst on LongBench. The results in Table 4 show that our method achieves comparable performance. Compared to selective recomputation methods such as *CacheBlend* and A^3, *LegoLink* even achieves better performance in some cases, which may be attributed to the more pronounced attention sink phenomenon in the Qwen model.

For Needle-in-a-Haystack. We present the results on the Needle-in-a-Haystack task using Qwen2.5-7B-Instruct and Mistral-7B-Instruct-v0.2 in Figure 7. Our method consistently achieves the best performance. Notably, for the Qwen model, using position recovery alone is sufficient to achieve the same 100% accuracy as our method.

For Ruler. We present the experimental results on the Ruler task using LLaMA3-8B-Instruct and Mistral-7B-Instruct-v0.2. Our method achieves the best average performance.

Performance Variance Across Models. We observe that while A^3 achieves substantial gains on LLaMA3, the improvements on Qwen2.5 and Mistral are more modest compared to strong baselines like *CacheBlend* and *LegoLink*. This discrepancy can be attributed to two primary factors. First, different models exhibit varying sensitivities to attention patterns. LLaMA3 appears to rely heavily on precise attention allocation for long-context retrieval, making our attention-aware recomputation particularly effective. In contrast, models like Qwen2.5 demonstrate a more robust attention mechanism (potentially due to stronger attention sinks or different training data distributions), where heuristic-based methods like *LegoLink* already capture sufficient information, leaving less room for improvement. Second, the baseline performance of Qwen2.5 is inherently high (e.g., nearing the upper bound on several tasks), which naturally compresses the potential margin for gain. Nevertheless, our method consistently matches or exceeds the best baselines, demonstrating its reliability across different architectures.

D Choice of Recomputation Ratio

Figure 6c reports the ratio sweep for A^3. *CacheBlend* also varies with the recomputation ratio by construction: increasing r causes more tokens to be recomputed after position recovery, which typically improves accuracy but also raises the recomputation overhead and weakens the latency advantage of reuse. Therefore, for the main comparison we follow common practice and adopt r=0.15 for *CacheBlend* as a representative trade-off point between accuracy and efficiency, rather than tuning it toward either extreme.

E About Extension

We report the results of A^3 with the extension strategy on the Ruler benchmark in Table 6.

Performance Variance on Ruler. We note a visual performance drop on Ruler compared to LongBench. This is primarily due to the distinct evaluation metrics employed by the two benchmarks. Ruler typically adopts a strict "hit-or-miss" criterion (binary 0 or 1), meaning that any failure to retrieve the exact information results in a zero score. In contrast, LongBench utilizes continuous metrics such as F1 or ROUGE (ranging from 0 to 1), which allow for partial credit. Consequently, a small deviation in the model's internal states—common when applying approximation techniques like eviction—might lead to a complete zero in Ruler, making the performance drop appear more drastic than in LongBench.

Additive vs. Multiplicative Loss. Furthermore, we hypothesize that the information loss induced by KV reuse and KV eviction operates in an *additive* rather than multiplicative manner. As evidenced by the results, applying eviction strategies alone (without reuse) results in a comparable magnitude of performance degradation. The combined performance decline in our extension roughly corresponds to the sum of the individual losses from reuse and eviction. This suggests that our reuse method does not exacerbate the model's sensitivity to token eviction, but rather, the two sources of error accumulate linearly.

F Validity of Shallow-Layer Signals

One critical design choice in A^3 is estimating token relevance based on the attention outputs from the first layer. While we do not provide a population-level validation across all possible configurations, Figure 3 serves as strong indirect evidence for "cross-layer consistency." It demonstrates that recomputing tokens selected via shallow-layer semantics significantly reduces the attention loss across deeper layers (e.g., layers 13, 22, 31), aligning the recovered attention patterns closer to the ground truth. Furthermore, we acknowledge that

Methods	Single-Doc QA		Multi-Doc QA		Summarization		Few-shot Learning			Others		Avg.
	Qasper	MultiQ	Hotpot	2WikiM	GovRep	MultiN	TREC	TriviaQ	SAMSum	PCount	LCC	
Qwen2.5-7B-Instruct												
Vanilla	40.10	50.58	53.76	43.73	33.70	23.78	67.00	88.79	42.82	8.78	62.99	46.91
FullReuse	9.02	23.82	27.10	16.83	29.67	22.09	16.33	65.40	13.17	2.81	64.39	26.42
PIE	35.37	40.06	38.49	23.74	30.37	22.19	60.67	84.25	40.14	4.29	58.97	39.87
CacheBlend	37.07	41.44	**42.54**	26.57	31.29	22.42	61.33	89.78	40.37	4.94	60.38	41.65
LegoLink	37.67	**42.72**	40.98	**28.70**	**32.03**	22.51	60.67	**89.79**	**40.96**	**5.47**	62.13	**42.15**
ours	**38.96**	41.18	38.92	25.77	31.04	**22.61**	**64.00**	86.56	40.42	4.87	**62.44**	41.52

Table 4: Performance comparison of our method with *FullReuse*, *PIE*, *CacheBlend*, *LegoLink*, and up-bound method Vanilla on LongBench for Qwen2.5-7B-Inst. The best results are highlighted in **bold**.

Methods	Single NIAH			Multi-key NIAH			MQuery	MValue	CWE	FWE	VT	Avg.
	Single-1	Single-2	Single-3	Multi-1	Multi-2	Multi-3						
LLaMA3-8B-Instruct												
Vanilla	100.0	100.0	100.0	99.33	99.67	98.0	91.33	99.08	99.73	93.11	99.53	98.16
FullReuse	71.67	83.67	21.00	29.33	2.00	0.00	27.25	41.50	37.23	96.33	59.13	42.65
PIE	94.33	81.00	99.00	56.67	55.00	59.33	57.08	40.92	99.40	98.44	64.80	73.27
CacheBlend	**100.0**	97.00	99.67	78.33	**67.00**	69.67	**81.17**	58.67	99.70	98.22	82.87	84.75
LegoLink	99.67	90.00	99.33	71.67	58.67	65.67	68.00	54.75	99.60	**98.44**	73.40	79.93
ours	98.33	**97.33**	**100.0**	**79.00**	65.00	**71.00**	80.00	**65.50**	**99.70**	98.22	**88.33**	**85.67**
Mistral-7B-Instruct-v0.2												
Vanilla	94.67	95.0	63.67	96.0	8.33	-	83.50	53.25	67.93	89.00	85.60	67.00
FullReuse	97.33	-	-	-	0.67	-	-	0.08	87.90	67.33	43.33	26.97
PIE	99.00	-	-	0.33	53.33	-	0.08	0.42	90.60	71.56	14.07	29.94
CacheBlend	99.33	-	-	-	43.33	-	**0.17**	0.17	87.80	**73.33**	28.87	30.27
LegoLink	99.33	-	-	0.33	51.00	-	-	0.33	89.93	70.44	30.93	31.12
ours	**99.33**	**0.33**	-	**0.33**	**56.67**	-	0.08	**0.50**	**90.70**	70.56	**46.53**	**33.18**

Table 5: Performance comparison of our method with *FullReuse*, *PIE*, *CacheBlend*, *LegoLink*, and up-bound method Vanilla on Ruler for LLaMA3-8B-Instruct and Mistral-7B-Instruct-v0.2. The best results are highlighted in **bold**.

this strategy is empirically driven by the necessity of efficiency. To achieve meaningful acceleration, we must identify important tokens early in the forward pass. Relying on deep-layer signals would require executing the model through those layers, defeating the purpose of computational reuse. This trade-off—sacrificing absolute theoretical guarantee for substantial speedup—is consistent with prior efficient inference works like *CacheBlend* (Yao et al., 2025), which similarly relies on heuristic metrics to guide computation.

G Statistical Significance

All experiments are conducted using greedy decoding (temperature set to 0), ensuring that the generation process is fully deterministic. Given the identical input and model weights, the output remains constant across multiple runs, eliminating variance caused by sampling randomness. Therefore, statistical significance tests and confidence intervals are not applicable in this context. The reported scores directly reflect the stable and reproducible performance of the evaluated methods.

H Overhead Analysis

Regarding the pre-computation and storage overheads, we emphasize that **all evaluated reuse baselines** (including *FullReuse*, *CacheBlend*, etc.) share the identical I/O mechanism: loading pre-computed KV caches from SSD to GPU memory. Therefore, the transfer latency is a common constant across methods and does not affect the relative

	Evict	Single-1	Single-2	Single-3	Multi-1	Multi-2	Multi-3	MQuery	MValue	CWE	FWE	VT	Avg.
LLaMA	✗	98.33	97.33	100.0	79.00	65.00	71.00	80.00	65.50	99.70	98.22	88.33	85.67
	✓	98.33	91.67	3.33	76.67	48.67	5.00	78.17	53.08	97.10	94.00	88.67	66.79
Mistral	✗	99.33	0.33	-	0.33	56.67	-	0.08	0.50	90.70	70.56	46.53	33.18
	✓	99.33	-	-	-	16.67	-	11.08	1.33	85.63	91.00	49.13	32.20
Qwen	✗	98.00	91.33	98.67	83.67	86.00	76.33	75.92	56.00	92.57	97.11	70.27	84.17
	✓	98.00	77.67	12.00	61.00	4.33	0.33	37.25	9.08	82.63	90.44	71.00	49.43

Table 6: The impact of eviction-based extension on the task performance of our method. Experiments are conducted on three models and the Ruler benchmark.

performance comparison. While we acknowledge that SSD-to-GPU bandwidth can be a bottleneck in real-world deployments (as noted in Limitations), it is an infrastructure-level challenge orthogonal to our algorithmic contributions.

Furthermore, the computational cost of **RoPE re-rotation** during the reuse phase is negligible. RoPE operations are element-wise multiplications applied only to the Key vectors, which constitute a tiny fraction of the total inference FLOPs compared to the heavy matrix multiplications in the Feed-Forward Networks (FFN) and Attention projections. Our latency measurements in Figure 5 already include these overheads, confirming that they do not hinder the overall speedup.

I Scalability

Scaling with Sequence Length, Attention Heads, and Batch Size. The additional overhead of A^3 mainly comes from computing query-aware token scores and recomputing the selected subset of tokens. For sequence length, the scoring step computes interactions between the short query and the document tokens, and therefore scales as $O(|Q|N)$ with $|Q| \ll N$ in our setting. After selecting the top-rN tokens, the recomputation cost is restricted to the selected subset, yielding an additional cost of approximately $O((rN)^2)$ rather than recomputing the full context. In the main experiments we use r=0.15, so this extra cost remains a small fraction of full prefill. For attention heads, the score extraction is carried out together with the model's native multi-head attention and reduced across heads in parallel on GPU, so it does not introduce an extra sequential bottleneck with respect to the number of heads. For batch size, Figure 6b shows that the method remains effective under larger batches; in particular, the throughput gain becomes more pronounced as the batch size increases, indicating that the added overhead does not dominate the computation under batching.

Scalability to Larger Models. Due to computational resource constraints, our current evaluation is focused on models in the 7B-8B parameter range (LLaMA3-8B, Mistral-7B, Qwen2.5-7B). However, we emphasize that the core principle of our method—leveraging shallow attention patterns to guide KV reuse—is architecture-agnostic and likely to generalize to larger models. In fact, prior studies (Yao et al., 2025) have observed that the relative speedup of KV reuse methods tends to increase with model size (e.g., up to 3.3× on 70B models), as the computational overhead of the prefill stage grows, while the I/O bottleneck becomes more pronounced. Therefore, we anticipate that A^3 would yield even greater efficiency gains on larger-scale models like LLaMA3-70B, making it a promising direction for future work with sufficient hardware support.

J Code Appendix

Installation

To set up the environment, please use the provided `requirements.txt` file:

```
pip install -r requirements.txt
```

Usage

All experiments are conducted under the `a3kv` directory.

1. **Chunking Long Contexts**

   We provide scripts to chunk the datasets used in our evaluation:

```
cd a3kv
python ./chunk_longbench.py
python ./chunk_needle.py
```

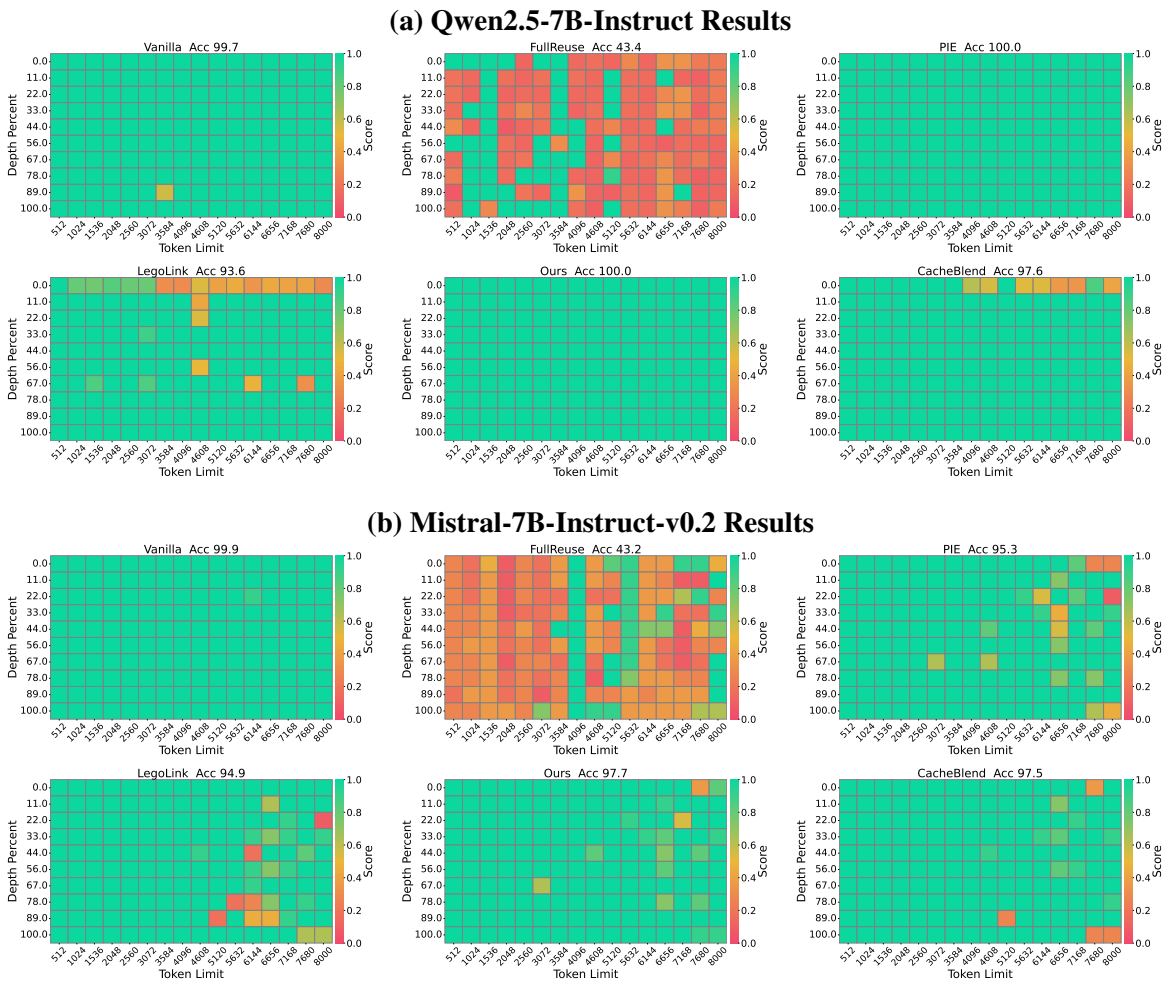


Figure 7: Needle-in-a-Haystack results on Qwen2.5-7B-Instruct and Mistral-7B. The vertical axis represents depth percentage, and the horizontal axis represents token length. Our method achieves the best retrieval performance.

```
python ./chunk_ruler.py
```

2. **Precomputation**

   The following script precomputes the attention cache for each chunked document:

```
bash ./scripts/precompute.sh
```

3. **Evaluation**

   We provide separate scripts to evaluate each benchmark:

```
# for LongBench
bash ./scripts/eval_longbench.sh

# for Needle-in-a-Haystack
bash ./scripts/eval_needle.sh
python ./visualize.py

# for Ruler
bash ./scripts/eval_ruler.sh
python ./scripts/copy_results_ruler.py
```