Transition Overlap: Construction-Time Conditioning for Position-Independent KV Cache Reuse

Liang Xu[1] **Zhichun Wang**[1,2,3*] **Xuetao Tian**[4]
[1]School of Artificial Intelligence, Beijing Normal University, Beijing 100875, China
[2]Beijing Key Laboratory of Artificial Intelligence for Education, Beijing 100875, China
[3]Engineering Research Center of Intelligent Technology and Educational Application, Ministry of Education, Beijing 100875, China
[4]Faculty of Psychology, Beijing Normal University, Beijing 100875, China
202431081001@mail.bnu.edu.cn {zcwang,xttian}@bnu.edu.cn

Abstract

In long context applications like Retrieval Augmented Generation (RAG), Large Language Models (LLMs) often reuse independently precomputed Key Value (KV) caches to reduce latency. However, directly assembling these cached segments introduces errors that degrade generation quality. We study this problem in the setting of position-independent caching and decompose the reuse error into two components: a geometric error (G) caused by position mismatch, which can be largely removed by position alignment, and a dynamic error (D) caused by context-dependent representation changes. To characterize D, we introduce the smoothness gap, a local diagnostic that measures the discrepancy between local KV-state transitions in isolated caches and those in the full-context baseline. This gap serves as a necessary local consistency diagnostic for reconstruction errors across adjacent positions and reveals that recomputing non-contiguous selected tokens can introduce additional transition discontinuities, thereby amplifying the dynamic error. To overcome this limitation, we propose **Transition Overlap**, a training-free cache-construction protocol that prepends a short temporary overlap to approximate full-context boundary transitions and stores only the KV states of the original segment tokens. We instantiate it under two deployment regimes: Self-Segment Overlap (SSO) for query-independent offline caches and Cross-Segment Overlap (CSO) for cases in which the predecessor segment is known during construction. In both evaluated deployment settings, Transition Overlap improves the measured serving latency–quality trade-off over training-free PIC baselines[1].

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse natural language processing tasks (Bai et al., 2023; Zeng et al., 2025). As context lengths grow in retrieval-augmented question answering (Yi et al., 2025; Rim et al., 2025), multi-document summarization (Belém et al., 2025), and long-context retrieval (Patel, 2025), LLMs face mounting challenges in inference latency. A prevalent optimization is to reuse precomputed key-value (KV) caches across different requests, avoiding redundant encoding of repeated content. The most widely adopted approach is prefix-based reuse (Yu et al., 2025): when a new prompt shares an exact prefix with a previous prompt, systems such as CacheGen (Liu et al., 2024b), vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) can directly reuse the corresponding KV cache states. However, this approach is limited to exact-prefix sharing. In RAG (Lewis et al., 2020; Gao et al., 2024), contexts are often assembled from independently retrieved or pre-segmented text segments that do not share a common prefix. These applications require **Position-Independent Caching (PIC)** strategies that can handle separate, independently encoded contexts.

Existing PIC methods either fine-tune models to tolerate independently constructed caches (Ma et al., 2025; Lu et al., 2025; Yang et al., 2025) or repair reused caches at inference time. The former requires model adaptation and is costly to deploy. The latter, which we call concatenate-then-recompute, first splices cached KV states from different segments and then recomputes selected tokens to restore cross-context dependencies. Prior work often treats attention sinks as the dominant factor. For example, EPIC (Hu et al., 2025) significantly improves over CacheBlend (Yao et al., 2025) by replacing carefully designed token selection

*Corresponding author
[1]Code is available at https://github.com/boyoi/Transition-Overlap

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 32366–32386
October 24-29, 2026 ©2026 Association for Computational Linguistics

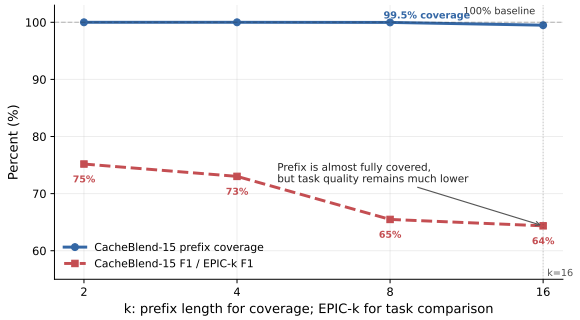


Figure 1: **Coverage–quality mismatch in PIC reuse.** On MuSiQue, CacheBlend-15, with a 15% recomputation budget, covers 99.5% of the boundary-prefix tokens recomputed by EPIC-16 (blue curve), yet reaches only 64% of EPIC-16 F1 (red curve). This mismatch suggests that PIC reuse errors cannot be fully attributed to attention-sink effects.

with naive boundary-token selection. As shown in Fig. 1, a counterintuitive finding on MuSiQue suggests that this token-selection view is incomplete: CacheBlend-15 covers nearly all EPIC-16 boundary-prefix tokens but still performs much worse.

To investigate the sources of PIC reuse error, we decompose it into two components: a geometric error G and a dynamic error D. We find that RoPE position alignment can remove G when the source and target RoPE coordinates are known, while D persists because cached states may be constructed under different local contexts. To characterize this residual context-dependent mismatch, we introduce the smoothness gap as a local diagnostic of D. By exposing local transition inconsistency, this analysis helps explain why scattered inference-time recomputation can be fragile: it may restore some cross-context dependencies but also introduce new discontinuities between recomputed and cached states.

Motivated by this analysis, we propose Transition Overlap, a training-free cache-construction protocol that prepends a short temporary overlap to approximate full-context boundary transitions. Unlike inference-time recomputation methods, Transition Overlap builds more reusable segment caches during construction rather than repairing them after cache assembly. The temporary overlap is used only during the construction forward pass, so the stored cache length and the served prompt remain unchanged. We instantiate this protocol under two deployment regimes: Self-Segment Overlap (SSO) for query-independent offline caches, and Cross-Segment Overlap (CSO) for predecessor-aware construction, including online-parallel serving and fixed source-adjacent caching. In summary, the main contributions of this paper are as follows:

- We analyze PIC reuse by decomposing reuse error into geometric (G) and dynamic (D) components, thereby separating RoPE coordinate mismatch from context-dependent representation mismatch. To diagnose D after RoPE position alignment, we introduce the smoothness gap as a local-consistency diagnostic, showing that recomputing non-contiguous selected tokens can introduce transition discontinuities between recomputed and cached states.
- We propose *Transition Overlap*, a training-free construction-time protocol that conditions each segment on a short temporary overlap, and stores only the original segment KV. We instantiate it in two deployment regimes: SSO for query-independent offline caching, and CSO for online-parallel or fixed-adjacent construction where the predecessor is known.
- Across five RAG serving benchmarks, our method improves the speed–quality trade-off, and our diagnostics show that meaningful transition content is associated with lower key/value dynamic errors and smaller smoothness gaps.

2 Related Work

Position-Independent Caching. PIC is important for optimizing RAG (Lewis et al., 2020; Gao et al., 2024). Existing methods can be broadly divided into model-adaptation and training-free cache-reuse approaches. The former, including KVLink (Yang et al., 2025), TurboRAG (Lu et al., 2025), Block-Attention (Ma et al., 2025), and KVPacket (Chen et al., 2026), requires fine-tuning or architectural modifications to enforce cache compatibility. The latter reuses independently constructed segment caches without modifying the model (Wang et al., 2025; Zhao et al., 2026; Yang et al., 2026). CacheBlend (Yao et al., 2025), EPIC (Hu et al., 2025) and ProphetKV (Wang et al., 2026b) follow a concatenate-then-recompute paradigm: they splice pre-cached segments and recompute selected tokens after cache assembly. FusionRAG (Wang et al., 2026a) improves this line by shifting part of the repair cost offline through similarity-guided cache construction, but it still

relies on query-time recomputation of selected tokens. Our method instead shifts cache compatibility to construction time. The G+D decomposition separates RoPE coordinate mismatch from dynamic representation mismatch; accordingly, SSO/CSO condition segment caches before assembly, enabling direct reuse without post-assembly recomputation.

Cross-Segment KV Cache Coherence and Context Overlap. Cross-segment coherence concerns whether KV states remain consistent when independently constructed segment caches are concatenated. Boundary discontinuities can degrade generation quality (Cai et al., 2025; Zhang et al., 2023). Related non-PIC methods address continuity either by preserving attention sinks, e.g., StreamingLLM (Xiao et al., 2024), or by adding input-level overlap in long-document processing and RAG chunking (Dai et al., 2019; Gao et al., 2024; Chen et al., 2024; Liu et al., 2024a). These mechanisms keep extra tokens in the served sequence or persistent memory. Transition Overlap instead uses extra tokens only as a temporary construction buffer, discarding their KV states after the forward pass so the served prompt and reusable cache length remain unchanged. The closest construction-time baseline is EPIC-0, the zero-recomputation variant of EPIC, which also uses a discardable prefix but fills it with dummy tokens to absorb segment-start attention sinks. Instead, Transition Overlap introduces a short temporary overlap with meaningful self-prefix or predecessor-suffix tokens, allowing segment states to be conditioned before concatenation and better approximate full-context boundary transitions.

3 Problem Setup and Preliminaries

Problem formulation. We consider a reusable segment $\mathcal{B}$ inside an assembled context. It can be encoded either in isolation, producing cached states $K^{(l),\mathrm{iso}}, V^{(l),\mathrm{iso}}$, or in the full context, producing $K^{(l),\mathrm{dir}}, V^{(l),\mathrm{dir}}$. Reusing isolated caches is challenging because the two computation paths may differ in both RoPE coordinates and contextual representations. We distinguish raw and aligned reuse errors. For any layer-wise quantity z, the raw error is $\Delta z_i^{(l),\mathrm{raw}} := z_i^{(l),\mathrm{dir}} - z_i^{(l),\mathrm{iso}}$. When a cached key is explicitly RoPE-aligned before reuse, we denote the resulting key error as $\Delta K_i^{(l),\mathrm{align}} := K_i^{(l),\mathrm{dir}} - \widehat{K}_i^{(l)}$. Sec. 4 uses this notation to separate coordinate-induced error from representation mismatch, and motivates cache-construction transformations that reduce the latter without recomputing cached tokens after prompt assembly. For notation, let global token i be the j-th token of segment r, written $i = (r, j)$. Its target full-context coordinate is $g(i)$, or $g_r(j)$ in segment-local form, and its construction-time source coordinate is $s_r(j)$. For notational simplicity, we use the superscript "iso" to denote the cached construction path. For plain reuse this is isolated segment encoding, and for Transition Overlap it denotes the sliced original token states after augmented overlap construction.

Rotary Position Embedding. Let $h_i^{(l)}$ denote the hidden state at layer l. We write the pre-RoPE key and value projections as $\tilde{k}_i^{(l)} = W_K^{(l)} h_i^{(l-1)}$ and $\tilde{v}_i^{(l)} = W_V^{(l)} h_i^{(l-1)}$. RoPE applies an orthogonal rotation to keys, $K_i^{(l)} = R_{\theta(\mathrm{pos}(i),\alpha_l)} \tilde{k}_i^{(l)}$, where $\mathrm{pos}(i)$ is the coordinate used during construction or reuse. In direct full-context computation, $\mathrm{pos}(i) = g(i)$; in isolated construction it is the local segment index. If an overlap buffer O_r is prepended to segment r, the j-th original token of the segment is constructed at source coordinate $s_r(j) = |O_r| + j$. At reuse time, RoPE alignment maps this source coordinate $s_r(j)$ to the target global coordinate $g(i) = g_r(j)$ for the corresponding global token $i = (r, j)$. We give the frequency-wise RoPE bound and parameter checks in App. B.

4 Error Analysis: Why Cache Construction Matters

This section analyzes why cache construction matters for training-free PIC reuse. We separate RoPE coordinate mismatch from context-dependent representation mismatch, then use the smoothness gap to motivate Transition Overlap for approximating full-context boundary transitions.

4.1 Separating Geometric Error from Dynamic Error

We fix a layer l and token $i \in \mathcal{B}$. The direct full-context key is $K_i^{(l),\mathrm{dir}} = R_{\theta(g(i),\alpha_l^{\mathrm{dir}})} \widetilde{k}_i^{(l),\mathrm{dir}}$. After a possible RoPE alignment transform, the reused key can be written as

$$\widehat{K}_i^{(l)} = R_{\theta(\widehat{g}(i),\bar{\alpha}_l)} \widetilde{k}_i^{(l),\mathrm{iso}}, \tag{1}$$

where $\widehat{g}(i)$ and $\bar{\alpha}_l$ are the RoPE position and scale used by the alignment step. [2] We define the key

[2] Naive concatenation leaves cached keys at their construction coordinates, j or $s_r(j)$. RoPE alignment maps them

reuse error after an optional coordinate transform as $\Delta K_i^{(l),\text{align}} := K_i^{(l),\text{dir}} - \widehat{K}_i^{(l)}$. For readability, we write it as $\Delta K_i^{(l)}$ within this section. Substituting the RoPE forms and factoring out the direct full-context rotation yields

$$\begin{aligned}\Delta K_i^{(l)} &= R_{\theta(g(i),\alpha_l^{\text{dir}})}\left(\widetilde{k}_i^{(l),\text{dir}} - U_i^{(l)}\widetilde{k}_i^{(l),\text{iso}}\right) \\ &= R_{\theta(g(i),\alpha_l^{\text{dir}})}\Big(\underbrace{(I-U_i^{(l)})\,\widetilde{k}_i^{(l),\text{iso}}}_{\textit{geometric error G}} + \underbrace{(\widetilde{k}_i^{(l),\text{dir}} - \widetilde{k}_i^{(l),\text{iso}})}_{\textit{dynamic error D}}\Big),\end{aligned} \tag{2}$$

where $U_i^{(l)} := R^{\top}_{\theta(g(i),\alpha_l^{\text{dir}})} R_{\theta(\widehat{g}(i),\bar{\alpha}_l)}$ is the relative coordinate transform induced by reuse. We denote $G_i^{(l)} := (I - U_i^{(l)})\widetilde{k}_i^{(l),\text{iso}}$ and $D_i^{(l)} := \widetilde{k}_i^{(l),\text{dir}} - \widetilde{k}_i^{(l),\text{iso}}$. The geometric error G is induced by RoPE coordinate mismatch, whereas D denotes the context dependent or path dependent representation residual before RoPE that remains after coordinate effects are separated. Transition Overlap primarily targets the boundary related component of D, but D should not be interpreted as boundary error only. In deeper layers, it may also reflect upstream effects and prompt order effects from prefix mismatch. Since RoPE rotations are orthogonal, left-multiplication by $R_{\theta(g(i),\alpha_l^{\text{dir}})}$ preserves Euclidean norms. Thus, Eq. (2) separates RoPE coordinate mismatch from pre-RoPE representation mismatch. Values do not typically undergo RoPE rotation at the current projection, so their reuse error is treated as dynamic and is given by $\Delta V_i^{(l)} = \widetilde{v}_i^{(l),\text{dir}} - \widetilde{v}_i^{(l),\text{iso}}$.

4.2 Geometric Error Is a Coordinate Problem

The decomposition above makes the role of RoPE alignment explicit. The operator $U_i^{(l)}$ is fully determined by the coordinate alignment. If the reused key is placed at the same RoPE position and scale as in direct full-context computation, then $\widehat{g}(i) = g(i)$ and $\bar{\alpha}_l = \alpha_l^{\text{dir}}$, which gives $U_i^{(l)} = I$ and hence $G_i^{(l)} = 0$. When $U_i^{(l)} \neq I$, the magnitude of $G_i^{(l)}$ also depends on the cached pre-RoPE key $\widetilde{k}_i^{(l),\text{iso}}$. The frequency-wise RoPE bound and parameter checks are provided in App. B and App. G. App. C shows that the two common RoPE alignment implementations are equivalent when they use the same source and target coordinates.

to $\widehat{g}(i) = g(i)$ and $\bar{\alpha}_l = \alpha_l^{\text{dir}}$ without changing pre-RoPE representations.

4.3 Smoothness Gap Exposes a Cost of Inference-Time Repair

After position alignment, we continue to denote post-RoPE key states by K. Let $\bar{K}_i^{(l),M}$ denote the final aligned key state used by method M at token i. Let $K_i^{(l),\text{dir}}$ be the direct full-context key state, and define $\mathcal{E}_i^{(l),M} = \|\bar{K}_i^{(l),M} - K_i^{(l),\text{dir}}\|_2$. For two consecutive tokens i and $i+1$, the reverse triangle inequality gives

$$\begin{aligned}\mathcal{E}_i^{(l),M} + \mathcal{E}_{i+1}^{(l),M} &\geq \left\|\left(\bar{\mathbf{k}}_i^{(l),M} - \bar{\mathbf{k}}_{i+1}^{(l),M}\right) - \left(\mathbf{k}_i^{(l),\text{dir}} - \mathbf{k}_{i+1}^{(l),\text{dir}}\right)\right\|_2 \\ &\geq \left|\underbrace{\left\|\bar{\mathbf{k}}_i^{(l),M} - \bar{\mathbf{k}}_{i+1}^{(l),M}\right\|_2}_{\textit{Assembled Step}} - \underbrace{\left\|\mathbf{k}_i^{(l),\text{dir}} - \mathbf{k}_{i+1}^{(l),\text{dir}}\right\|_2}_{\textit{Full-Context Step}}\right|.\end{aligned} \tag{3}$$

We define the final length discrepancy in Eq. (3) as the Smoothness Gap $\Gamma_i^{(l),M}$. Unlike the preceding vector transition error, $\Gamma_i^{(l),M}$ retains only the mismatch between the assembled and full-context transition lengths. It is therefore a conservative scalar proxy, since small adjacent reconstruction errors imply a small Γ, but the converse does not necessarily hold. Eq. (3) thus gives a necessary local consistency condition, where neighboring reused states cannot both be close to the direct path unless their transition length is also close.

This constraint exposes a limitation of inference-time recomputation. Recomputed tokens may reduce pointwise error but create mixed-path transitions with cached states, increasing local gaps. Thus, after RoPE alignment, cache construction should reduce the boundary-related part of D and make neighboring stored states closer to direct full-context transitions. Section 5 implements this idea with Transition Overlap.

5 Method

We propose **Transition Overlap**, a training-free cache-construction method for reducing dynamic error in PIC reuse.

5.1 Overview and Design Rationale

As summarized in Fig. 2, Transition Overlap avoids post-assembly recomputation by conditioning segment caches during construction. Concatenate-then-recompute methods repair cache mismatch after assembly, which increases decoding TTFT and may introduce discontinuities between recomputed and cached states. Transition Overlap prepends a short temporary buffer, runs a single forward pass, discards the buffer KV states, and stores only the

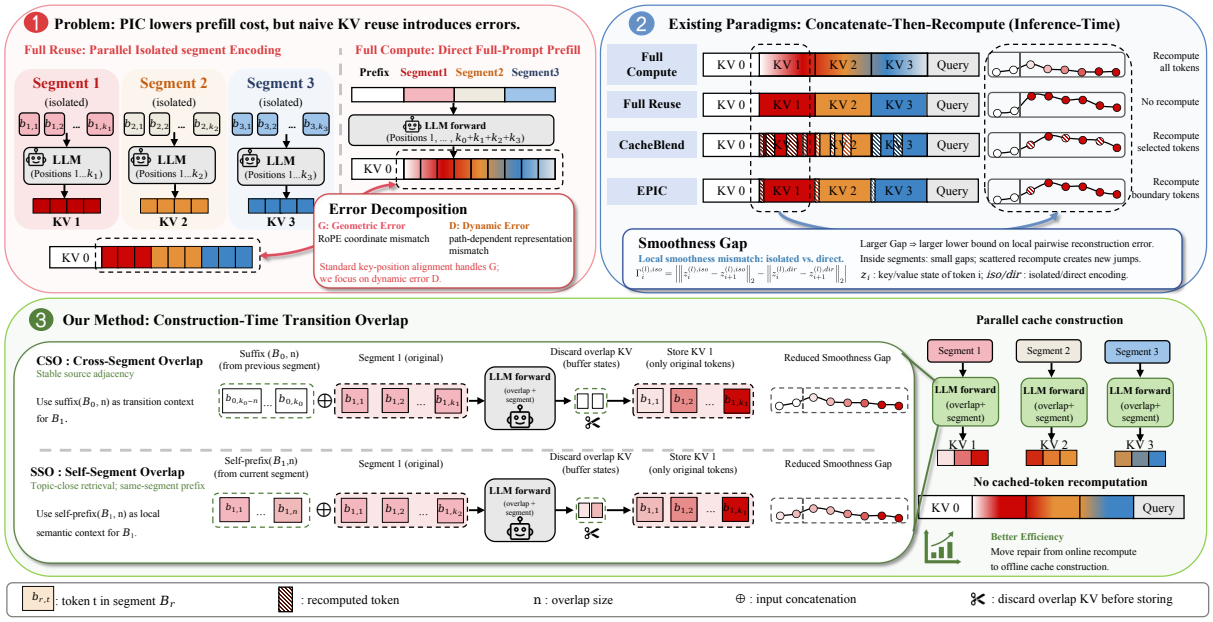


Figure 2: **Construction-time Transition Overlap for position-independent KV cache reuse.** Instead of repairing assembled caches by online recomputation, Transition Overlap prepends a short temporary buffer during cache construction and discards the buffer KV states before storage. CSO uses the suffix of the previous segment when the construction predecessor is known, whereas SSO uses the current segment prefix as a query-independent fallback. Only the original-token KV states are reused at decoding time.

KV states of the original segment tokens. Since the buffer affects only the construction path and is removed before reuse, neither the stored cache length nor the served prompt changes. This shifts repair work from serial online recomputation to a construction stage that can be precomputed in offline stored cache settings or parallelized in online parallel settings. We therefore report both decoding TTFT and sample wall time including construction overhead in Sec. 6.2.

5.2 Transition Overlap

Motivated by the necessary local consistency condition in Sec. 4.3, we now formalize **Transition Overlap**. For each current segment $\hat{B}_r$, we prepend a temporary overlap buffer O_r of length N_{sb} during cache construction. The augmented input is used only for the construction forward pass, and only the KV states of the original segment tokens are retained.

$$O_r = \begin{cases} \text{suffix}_{N_{\text{sb}}}(\hat{B}_{r-1}), & \textit{CSO}, \\ \text{prefix}_{N_{\text{sb}}}(\hat{B}_r), & \textit{SSO}. \end{cases} \tag{4}$$

We then construct and store the cache as follows:

$$\begin{aligned} \tilde{B}_r &= O_r \oplus \hat{B}_r, \\ \{(K^{(l)}_{\tilde{B}_r}, V^{(l)}_{\tilde{B}_r})\}_{l=1}^{L} &= \text{ForwardKV}(\tilde{B}_r), \\ \{(\bar{K}^{(l)}_r, \bar{V}^{(l)}_r)\}_{l=1}^{L} &= \text{Slice}_{\hat{B}_r}\left(\{(K^{(l)}_{\tilde{B}_r}, V^{(l)}_{\tilde{B}_r})\}_{l=1}^{L}\right). \end{aligned} \tag{5}$$

Under this RoPE coordinate convention, if the token at position j in $\hat{B}_r$ is encoded in $\tilde{B}_r = O_r \oplus \hat{B}_r$, its source coordinate is $s_r(j) = |O_r| + j$. At query time, key alignment maps $s_r(j)$ to the target coordinate $g_r(j) = g(i)$ for token $i = (r, j)$. Here, $\oplus$ denotes token concatenation, and $\text{Slice}_{\hat{B}_r}$ retains only the positions of the original segment tokens. The discarded overlap can still influence their hidden states during construction. Fig. 3 summarizes the two deployment regimes that determine the choice of overlap in Eq. (4).

Cross-Segment Overlap (CSO). CSO uses the suffix of a known construction predecessor as the temporary transition context. In Eq. (4), $\hat{B}_{r-1}$ is the predecessor under the construction order. It is the previous retrieved segment in online parallel RAG, or the previous source-adjacent chunk under fixed source offline caching. Thus, CSO is used only when this predecessor is known at construction time. It is not a strict offline one-cache-per-segment method for arbitrary retrieval orders, since a segment may have different query-time predeces-

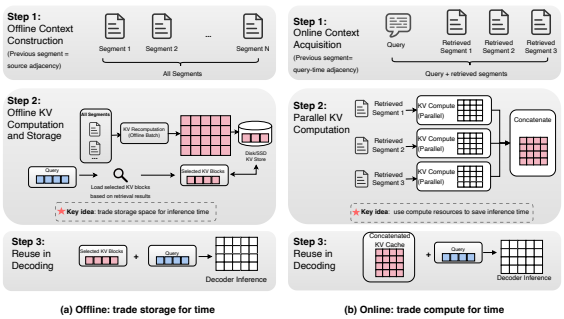


Figure 3: **Deployment modes for Transition Overlap.** In offline stored cache reuse, reusable source segments are precomputed before queries. SSO provides one query-independent cache per segment, while CSO is directly reusable only under fixed source adjacency. In online parallel construction, the retrieved order is known before KV construction, so CSO can use the actual previous segment before KV blocks are concatenated for decoding.

sors across requests. For the first retrieved segment, CSO uses the last N_{sb} tokens of the prompt or instruction prefix and does not fall back to SSO.

Self-Segment Overlap (SSO). SSO uses the current segment prefix as a query-independent fallback. It constructs one-cache-per-segment and is the default choice for offline stored cache reuse, where the query-time predecessor is generally unknown during construction. Therefore, SSO denotes the strict offline one-cache-per-segment variant, while CSO denotes the predecessor known variant for online parallel construction or fixed source-adjacent caching. App. J discusses boundary fragile SSO cases.

6 Experiments

We evaluate four questions: whether Transition Overlap improves quality–efficiency trade-offs (**RQ1**), whether it reduces key/value reuse error (**RQ2**), whether it reduces smoothness gaps (**RQ3**), and whether its benefit depends on meaningful overlap content rather than dummy padding (**RQ4**).

6.1 Experimental Setup

We follow the PIC protocol of EPIC (Hu et al., 2025) and use the same retrieved segments and order for all methods. To avoid mixing assumptions, we distinguish **offline one-cache-per-segment** evaluation, where SSO is applicable, from **online-parallel construction**, where the retrieved order is known before segment-KV construction and CSO is applicable. Recomputation baselines use the same assembled order but repair it after cache assembly; our CSO repairs during a parallelizable construction stage. Details are in App. A.

We evaluate 200 instances each from 2WikiMQA, MuSiQue, HotpotQA, Needle-in-a-Haystack, and SAMSum (Ho et al., 2020; Trivedi et al., 2022; Yang et al., 2018; Gliwa et al., 2019). We report F1 for QA/retrieval tasks and ROUGE-L for summarization. Question tokens are computed online and are not part of reusable context caches. Efficiency is reported as decoding-stage TTFT after reusable caches are available and, when construction or recomputation is part of serving, sample wall-time after retrieval. Retrieval time and standard prefix caching are excluded. We use Qwen3-4B (QwenTeam, 2025), Mistral-7B-Instruct (Jiang et al., 2023), and Llama3.1-70B-Instruct (Grattafiori et al., 2024), comparing Full Reuse, Full Compute, CacheBlend-η_{cb}, EPIC-b_{rec}, ProphetKV, SSO, and CSO. Unless otherwise specified, we use $N_{\text{sb}} = 16$ and perform no online recomputation of cached tokens. We choose 16 as a low-overhead default matched to EPIC-16, not as a universal optimum; matched SSO/CSO sweeps show no consistently best window (App. H.2). All efficiency comparisons are evaluated under identical configurations with single concurrency. This ensures the reported latency reflects pure method-level computation rather than multi-request scheduling effects.

6.2 RQ1: Deployment-Aware Performance Comparison

In the strict offline setting, Full Reuse, EPIC, CacheBlend, ProphetKV, and SSO are compared after reusable caches are available; the primary metric is decoding-stage TTFT, which excludes construction but includes any post-assembly recomputation. If source adjacency is fixed, CSO can also be precomputed for that fixed predecessor. In online-parallel construction, Full Compute, recomputation baselines, and CSO are compared by sample wall-time, which includes construction or recomputation overhead; construction work is reported separately when extra workers are used.

In the offline one-cache-per-segment setting, only SSO is interpreted as the strict offline Transition Overlap variant. Fig. 4 shows that SSO keeps the decoding path close to Full Reuse while improving quality over naive reuse. On Mistral-7B-Instruct, averaged over five datasets, SSO scores only 1.7% and 0.9% below EPIC-64 and EPIC-32, respectively, while eliminating online recomputa-

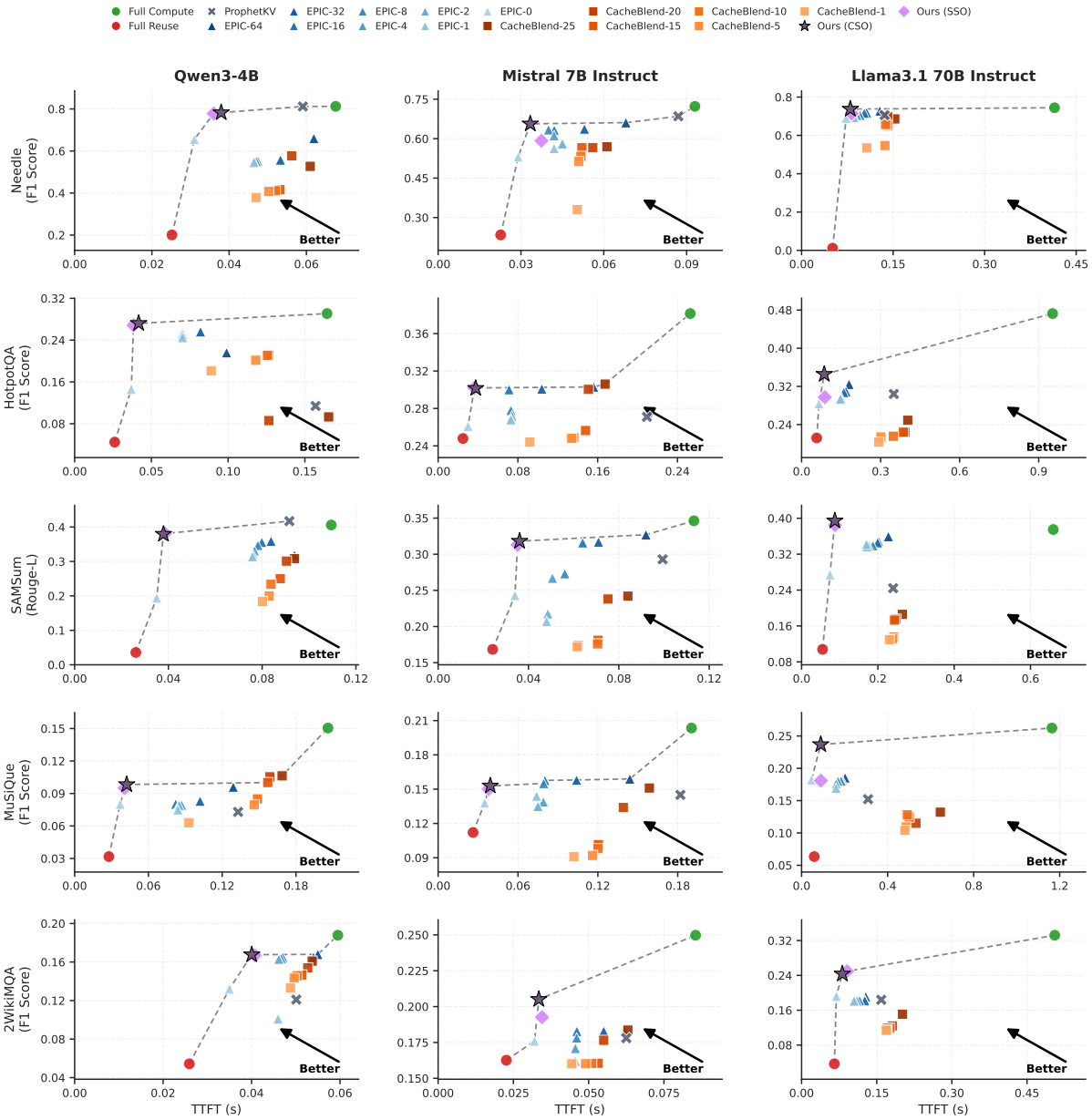


Figure 4: **Accuracy–decoding TTFT trade-off.** Each point shows the mean decoding stage TTFT and task score for one method configuration on a dataset and model pair. Scores are F1 for QA and retrieval tasks and ROUGE-L for SAMSum. SSO denotes strict offline one-cache-per-segment reuse, whereas Ours (CSO) assumes a known predecessor and reports construction costs separately in the wall-time analysis. The upper-left region is better within the same deployment setting.

tion and reducing TTFT by 65.2% and 53.1%. We do not use CSO to claim an offline stored-cache advantage for arbitrary retrieval orders. Tab. 1 reports sample wall-time including CSO construction and baseline recomputation. Across two models and five tasks, paired case-bootstrap tests detect no significant quality degradation versus EPIC or CacheBlend. Five fresh-process repeats with hierarchical paired-bootstrap intervals show significant wall-time speedups in 9/10 settings against each baseline (1.01× to 1.62× over EPIC and 1.04× to 1.62× over CacheBlend, App. H.1). App. H further decomposes construction, TTFT, and decode tail, so lower wall-clock latency with multiple workers is not conflated with lower total compute.

6.3 RQ2: Ablation Study

Fig. 5 separates coordinate and construction effects on Mistral-7B-Instruct. Position alignment greatly reduces key error but leaves value error mostly unchanged because values are not directly RoPE-rotated. CSO without alignment reduces

Method	Hyperparameter	HotpotQA		MuSiQue		2WikiMQA	
		F1↑	Sample Wall-time ↓	F1↑	Sample Wall-time ↓	F1↑	Sample Wall-time ↓
CacheBlend	15	0.236	0.566	0.101	0.656	0.161	0.483
EPIC	16	0.313	0.509	0.158	0.615	0.183	0.471
Ours (CSO)	16	**0.317**	0.477	0.156	**0.557**	**0.215**	0.465
	32	0.312	**0.473**	0.163	0.568	0.198	**0.457**
	64	0.293	0.490	0.149	0.599	0.173	0.458
	128	0.291	0.486	0.153	0.577	0.190	0.468
	256	0.296	0.498	0.157	0.575	0.184	0.470
	512	0.297	0.506	0.166	0.593	0.189	0.494
	1024	0.309	0.518	**0.180**	0.583	0.171	0.494

Table 1: **Online parallel efficiency.** We report average F1 scores and sample wall times across datasets. The hyperparameter column specifies 15 for CacheBlend, 16 for EPIC, and the overlap window N_{sb} for Ours (CSO). Best results appear in bold.

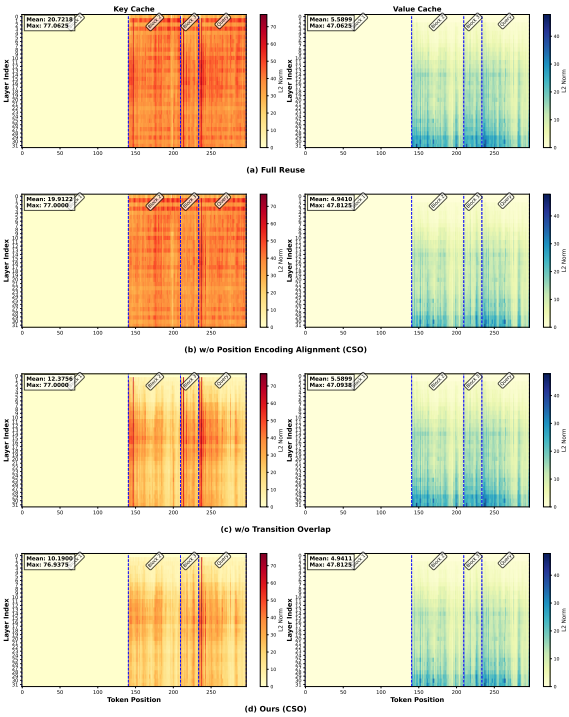


Figure 5: **Layer-wise reuse error ablation.** Heatmaps show L_2 key and value reuse errors across cached blocks and the query. RoPE alignment mainly reduces key error, while Ours (CSO) further lowers key and value errors near boundaries, reducing dynamic error and boundary spikes.

value-side boundary spikes but leaves high key error from unresolved coordinate mismatch. After alignment, CSO further reduces key/value dynamic error near boundaries, showing that overlap reduces boundary-related path-dependent mismatch. App. F demonstrates our method mitigates attention sink artifacts. However, reducing attention sinks is insufficient, as the smoothness gap provides a useful local diagnostic for remaining transition inconsistency. App. D gives a broader error-decomposition diagnostic, and App. E provides multi-example ablation samples.

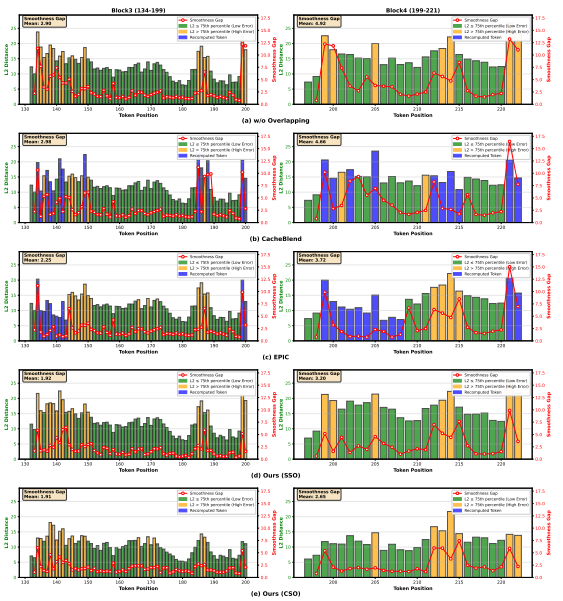


Figure 6: **Segment level consistency analysis.** Bars show aligned key L_2 distance and red lines show key smoothness gap across token positions. CacheBlend and EPIC show larger spikes near recomputed tokens, while Transition Overlap reduces these gaps.

6.4 RQ3: Smoothness Gap Reduction

Fig. 6 uses the smoothness gap as a diagnostic metric rather than a complete causal explanation for task accuracy. According to Eq. (3), two neighboring reused states cannot both remain close to the direct path if their transition length differs significantly from the direct transition. Consequently, recomputation-based methods often create local spikes around recomputed tokens and reveal mixed-path discontinuities. Transition Overlap completely avoids post-assembly recomputation and yields lower mean gaps. Notably, CSO reduces the Block 3 mean gap to 1.91 while maintaining stable curves. Beyond the segment-level analysis above, we further examine whether the smoothness gap also tracks local reconstruction difficulty at dataset scale. On MuSiQue, we compute Γ_i^{iso} and compare it with four local aligned-state error summaries. As shown in Fig. 7, larger gaps significantly correlate with larger local errors across all evaluated metrics ($\hat{\rho} \in [0.69, 0.75]$, $p < 10^{-7}$). After within-example aggregation over 200 examples, the correlations remain strong (Tab. **??**), and a matched-budget MuSiQue replication shows the same trend (App. I).

6.5 RQ4: Prefix-Content Control

To determine if Transition Overlap is merely temporary padding, we fix the prefix length ($b_{\text{pref}} = 4$),

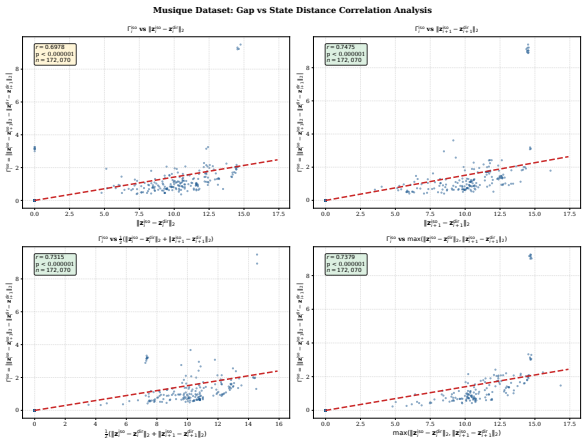


Figure 7: **Smoothness gap correlates with local key error.** Pearson correlations between $\Gamma_i^{(l)}$ and four local aligned-key error summaries across layers. The significant positive correlations ($\hat{\rho} > 0.69, p < 10^{-7}$) indicate that larger smoothness gaps consistently track harder local cache transitions, supporting Γ as a practical diagnostic signal for cache construction.

insertion position, and trimming rule while varying only the prefix content. Fig. 8 demonstrates that random tokens outperform a content-free dummy prefix, while SSO yields further improvements.

Furthermore, App. I verifies the same-budget correlation between lower proxy discrepancies and higher end-to-end F1 on HotpotQA. These findings reinforce our earlier token-accounting distinction. Unlike standard kept overlap methods that retain duplicated tokens in the served context, Transition Overlap strictly discards all overlap KV states prior to reuse.

7 Conclusion

This paper investigates PIC for RAG serving. We decompose reuse error into RoPE coordinate and path dependent components, using the smoothness gap to diagnose local consistency between neighboring reused states. Our Transition Overlap protocol improves cache construction directly and avoids serially repairing cached tokens after assembly. We introduce SSO for strict offline independent segment caching and CSO for settings with known predecessors via fixed source adjacency or online parallel construction. Crucially, neither variant retains overlap KV states in the served prompt. Experiments show improved measured latency–quality trade-offs under the corresponding deployment assumptions, while construction cost, decoding TTFT, and sample wall-time are reported separately.

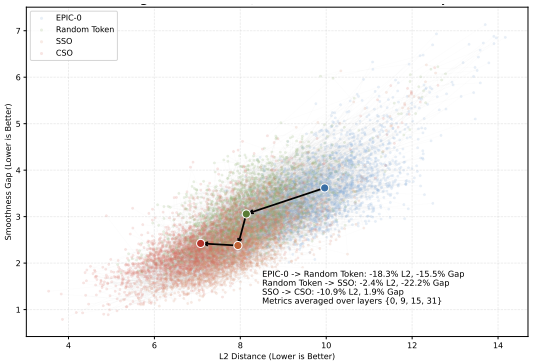


Figure 8: **Prefix content control with fixed budget and position.** All variants use the same prefix budget $b_{\text{pref}} = 4$ and insertion position, changing only prefix content. Metrics average aligned key L_2 distance and smoothness gap over the first 32 tokens and layers $\ell \in \{0, 9, 15, 31\}$. Lower is better. SSO and CSO improve over dummy and random prefixes.

8 Limitations

Transition Overlap reduces boundary-related representation mismatch but does not remove all context-dependence in reusable KV states. CSO is not an offline one-cache-per-segment method for arbitrary retrieval orders unless the predecessor is fixed; otherwise storing predecessor-specific caches is impractical. SSO is deployable in strict offline reuse, but may be weaker on boundary-fragile segments whose first tokens are numbers, entities, or short spans affected by repeated self-prefixes. Although the overlap is discarded and never served, it can still affect stored KV states. Our diagnostics rely on direct full-context references and are intended for analysis, not online deployment. Broader evaluation over retrievers, chunking policies, delimiters, hardware stacks, and production workloads remains future work.

9 Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 62276026) and the Fundamental Research Funds for the Central Universities (No. 2253500001)

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *Preprint*, arXiv:2309.16609.

Catarina G Belém, Pouya Pezeshkpour, Hayate Iso, Seiji

Maekawa, Nikita Bhutani, and Estevam Hruschka. 2025. From single to multi: How LLMs hallucinate in multi-document summarization. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5291–5324, Albuquerque, New Mexico. Association for Computational Linguistics.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2025. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *Preprint*, arXiv:2406.02069.

Chuangtao Chen, Grace Li Zhang, Xunzhao Yin, Cheng Zhuo, Bing Li, and Ulf Schlichtmann. 2026. Kv packet: Recomputation-free context-independent kv caching for llms. *Preprint*, arXiv:2604.13226.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. 2024. Dense X retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15159–15177, Miami, Florida, USA. Association for Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey. *Preprint*, arXiv:2312.10997.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, page 70–79. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. 2025. Epic: Efficient position-independent caching for serving large language models. *Preprint*, arXiv:2410.15332.

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. *Preprint*, arXiv:2310.06825.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

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*, 12:157–173.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024b. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 38–56, New York, NY, USA. Association for Computing Machinery.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. 2025. 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*, pages 6588–6601, Suzhou, China. Association for Computational Linguistics.

Dongyang Ma, Yan Wang, and Lan Tian. 2025. Block-attention for efficient prefilling. *Preprint*, arXiv:2409.15355.

Piyushkumar Patel. 2025. Graph-enhanced retrieval-augmented question answering for e-commerce customer support. *arXiv preprint arXiv:2509.14267*.

QwenTeam. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Daniel Rim, Minsoo Cho, Changwoo Chun, and Jaegul Choo. 2025. To chat or task: a multi-turn dialogue generation framework for task-oriented dialogue systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 576–592, Vienna, Austria. Association for Computational Linguistics.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Jiahao Wang, Weiyu Xie, Mingxing Zhang, Boxin Zhang, Jianwei Dong, Yuening Zhu, Chen Lin, Jingqi Tang, Yaochen Han, Zhiyuan Ai, Xianglin Chen, Yongwei Wu, and Congfeng Jiang. 2026a. From prefix cache to fusion rag cache: Accelerating llm inference in retrieval-augmented generation. *Proceedings of the ACM on Management of Data*, 4(1):1–28.

Qian Wang, Zahra Yousefijamarani, Morgan Lindsay Heisler, Rongzhi Gu, Bai Xiaolong, Shan Yizhou, Wei Zhang, Wang Lan, Ying Xiong, Yong Zhang, and Zhenan Fan. 2025. Mepic: Memory efficient position independent caching for llm serving. *Preprint*, arXiv:2512.16822.

Shihao Wang, Jiahao Chen, Yanqi Pan, Hao Huang, Yichen Hao, Xiangyu Zou, Wen Xia, Wentao Zhang, Chongyang Qiu, and Pengfei Wang. 2026b. Prophetkv: User-query-driven selective recomputation for efficient kv cache reuse in retrieval-augmented generation. *Preprint*, arXiv:2602.02579.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

Bin Yang, Qiuyu Leng, Jun Zeng, and Zhenhua Wu. 2026. Cacheclip: Accelerating rag with effective kv cache reuse. *Preprint*, arXiv:2510.10129.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. 2025. Kvlink: Accelerating large language models via efficient kv cache reuse. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 133797–133824. Curran Associates, Inc.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

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*, EuroSys '25, page 94–109, New York, NY, USA. Association for Computing Machinery.

Zihao Yi, Jiarui Ouyang, Zhe Xu, Yuwen Liu, Tianhao Liao, Haohao Luo, and Ying Shen. 2025. A survey on recent advances in llm-based multi-turn dialogue systems. *ACM Comput. Surv.*, 58(6).

Lingfan Yu, Jinkun Lin, and Jinyang Li. 2025. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 144–158. ACM.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, and 151 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *Preprint*, arXiv:2508.06471.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Shiju Zhao, Junhao Hu, Jiaqi Zheng, and Guihai Chen. 2026. You need an encoder for native position-independent caching. *Preprint*, arXiv:2602.01519.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. Sglang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

A Detailed Experimental Setup

A.1 Evaluation Protocol for Position-Independent Caching (PIC)

We evaluate PIC methods under a deployment-aware protocol that mirrors two common RAG serving modes:

Offline stored-cache mode. The system precomputes and stores one KV cache per reusable segment before any query. A segment may have different query-time predecessors across requests, so this mode cannot in general use an arbitrary query-time predecessor during cache construction. Therefore, SSO is the strict one-cache-per-segment Transition Overlap variant. CSO is directly reusable offline only when the predecessor is fixed by source adjacency, such as fixed intra-document chunking.

Online-parallel construction mode. Given a query, the system retrieves an ordered list of segments $\{\hat{B}_1, \ldots, \hat{B}_k\}$ and constructs the selected segment caches in parallel before decoding. This mode trades additional construction work for lower online wall-clock latency. In this mode, CSO uses the immediately previous segment in the query-time assembled order, because that order is known at construction time. We therefore evaluate online-parallel CSO with sample wall-time and construction-work accounting rather than treating its construction stage as free.

Transition Overlap construction. For our method, Transition Overlap is applied only during cache construction: we prepend an overlap buffer, run a single forward pass, and then discard overlap KV states before storage or assembly (Sec. 5.2). All reuse methods are evaluated on the same retrieved segments and the same concatenation order; the only difference is how KV caches are aligned/transformed and whether any tokens are recomputed after assembly.

A.2 Datasets and Preprocessing

RAG serving benchmarks. We follow the evaluation protocol of EPIC (Hu et al., 2025) and report results on five representative tasks (200 instances each): (i) *Multi-document QA*: 2WikiMQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), HotpotQA (Yang et al., 2018); (ii) *Long-context retrieval*: Needle-in-a-Haystack; (iii) *Instruction-following summarization*: SAMSum (Gliwa et al., 2019). For multi-document QA, each provided document is treated as an immutable segment. For Needle and SAMSum, we split the immutable contexts into fixed 512-token chunks to enable modular KV reuse. [3] We disable alternative caching mechanisms (e.g., standard prefix caching) to isolate position-independent reuse effects.

Dataset	Task	Chars	Tokens	#Seg
2WikiMQA	MD-QA	29.6K	4.9K	9.9
MuSiQue	MD-QA	69.4K	11.2K	11.1
HotpotQA	MD-QA	56.6K	9.1K	8.6
Needle	Ret.	55.1K	9.2K	30.0
SAMSum	Summ.	34.0K	6.1K	–

Table 2: RAG evaluation dataset statistics.

A.3 Prompting and Context Assembly

We use a unified prompt template across reuse methods to ensure comparability. For QA/retrieval tasks, the prompt order is: (1) a short instruction, (2) concatenated retrieved segments separated by a fixed delimiter, and (3) the question. This order is important for PIC: reusable context-segment KV states are not conditioned on query tokens, and the question tokens are appended after the cached context and computed online. For SAMSum, the prompt contains: (1) an instruction (summarize the conversation), and (2) the segmented conversation context. We use greedy decoding (temperature $= 0$) to remove sampling variance and cap the maximum generation length at 32 tokens for all methods.

A.4 Models and Inference Configuration

We evaluate three instruction-tuned backbones: Qwen3-4B, Mistral-7B-Instruct, Llama3.1-70B-Instruct. All methods run with identical inference settings (dtype, batch size, decoding strategy, and maximum generation length). We run in batch size 1 for TTFT measurements to match interactive serving. Additional diagnostic experiments in Appendices G and E use LLaMA-3.1-8B-Instruct and Qwen-2.5-7B-Instruct, though these evaluations remain separate from the main RQ1 benchmark comparison.

A.5 Metrics and TTFT Measurement

Accuracy metrics. For QA and Needle, we report token-level F1 computed on normalized answers, following standard SQuAD-style evaluation. For SAMSum, we report ROUGE-L.

[3] We keep the segmentation rule fixed across all methods so that accuracy differences are attributable to cache reuse rather than chunking.

Serving metrics and cost accounting. We report two latency metrics.

Decoding-stage TTFT is measured as wall-clock time from the start of cache assembly/decoding to the emission of the first decoded token, assuming that reusable segment caches are already available. For methods with stored segment caches, this includes KV cache loading/assembly, RoPE alignment, any recomputation required after assembly, and the first decode step. This metric captures interactive latency after cache availability.

Sample wall-time includes all LLM-side work needed to serve one sample after retrieval: cache construction if the method constructs query-specific caches, KV alignment/assembly, online recomputation if any, the first decode step, and the decode tail. For a construction stage parallelized over p workers, we report the wall-clock construction time and, when discussing resource use, the corresponding construction work summed over workers. Retrieval time is excluded because the retrieved segments and retrieval pipeline are identical across methods. We average all latency metrics over the 200 instances per dataset. Each latency configuration is measured in five fresh-process repeats. In each repeat, vLLM initialization and model warm-up are completed before timing, after which all 200 evaluation instances are timed. No additional request-level warm-up instances are excluded.

A.6 Implementation Details and Baselines

Serving stack. We implement all RAG-serving methods in vLLM v0.10.2 (Kwon et al., 2023) by porting the official EPIC codebase.[4] All methods share the same cache format and attention kernel to ensure fair TTFT comparison.

Baselines. We evaluate the following baseline configurations:

- **Full Reuse:** Concatenate cached KV segments for decoding without recomputation.
- **Full Compute:** Recompute all KV states online (full-context inference).
- **CacheBlend-η_{cb}:** Recompute tokens selected by CacheBlend's discrepancy score, with $\eta_{\text{cb}} \in \{1\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$.
- **EPIC-b_{rec}:** Recompute the first b_{rec} tokens at each segment boundary, with $b_{\text{rec}} \in \{0, 1, 2, 4, 8, 16, 32, 64\}$.
- **ProphetKV:** We implement ProphetKV ourselves based on the algorithmic description provided in the original ProphetKV paper. Following the recomputation-ratio setting reported in that paper, we use a 20% recomputation ratio unless otherwise specified.

Our methods. We evaluate **CSO** and **SSO** (Sec. 5.2). Unless otherwise specified, we use overlap length $N_{\text{sb}} = 16$ tokens in RAG and perform no online recomputation of cached tokens.

Hardware. Unless otherwise noted, the reported runs are executed on a single node with 2× NVIDIA H100 80GB GPUs, 1× Intel Xeon Platinum 8468 CPUs.

A.7 Additional Artifact and Reproducibility Notes

Artifact licenses and intended use. We use publicly released benchmark datasets, pretrained models, and open-source serving/evaluation software under their original licenses and terms. Our use is limited to non-commercial research evaluation. We do not redistribute model weights or raw benchmark data. The KV caches produced in this work are intermediate inference artifacts used only for measuring latency and task quality.

Data privacy and content. We do not collect new human-subject data. We use existing public benchmark datasets and report only aggregate evaluation results. We reviewed the dataset documentation and manually inspected examples shown in the paper or supplementary material; examples containing obvious personally identifying information or offensive content are excluded or sanitized.

Compute budget and package settings. The main experiments use Qwen3-4B, Mistral-7B-Instruct, and Llama3.1-70B-Instruct on a single node with 2 NVIDIA H100 80GB GPUs and one Intel Xeon Platinum 8468 CPU. The total computational budget is approximately 12 H100 GPU-hours. We implement serving experiments in vLLM v0.10.2, compute token-level F1 with SQuAD-style answer normalization for QA/retrieval tasks, and compute ROUGE-L F1 for SAMSum using the `rouge` package by pltrdy (v1.0.1) with the default `Rouge()` configuration; for multiple references, we report the highest score.

[4] https://github.com/DerekHJH/epic/

Method	Recomp. Budget	Overlap N_{sb}
Full Reuse	0	0
Full Compute	all tokens	0
EPIC-k	$k \in \{0, 1, 2, 4, 8, 16, 32, 64\}$/boundary	0
CacheBlend-r	$r \in \{1, 5, 10, 15, 20, 25\}\%$ tokens	0
ProphetKV	20% query-driven tokens	0
Ours (SSO)	0	16
Ours (CSO)	0	16

Table 3: Key hyperparameters and budget settings for PIC methods.

B Quantitative Error Bounds

This appendix gives the frequency-wise bound for the geometric error G used in Sec. 4.2. The main text uses only the resulting zero-error condition, while the full derivation is kept here to avoid interrupting the main argument. We use the convention $\theta_m(p, \alpha) = \omega_m p/\alpha$, where $\omega_m = b^{-2m/d_k}$. If no RoPE scaling is used, $\alpha = 1$. We keep the subscript l only to allow implementation-specific layer-wise scaling.

Componentwise key-error bound. From Eq. (2), the key error at layer l and token i is

$$\Delta K_i^{(l)} = R_{\theta(g(i),\alpha_l^{\text{dir}})}\left(G_i^{(l)} + D_i^{(l)}\right). \tag{6}$$

Here

$$\begin{aligned} G_i^{(l)} &:= (I - U_i^{(l)})\widetilde{k}_i^{(l),\text{iso}}, \\ D_i^{(l)} &:= \widetilde{k}_i^{(l),\text{dir}} - \widetilde{k}_i^{(l),\text{iso}}. \end{aligned} \tag{7}$$

Since RoPE rotations are orthogonal, they preserve Euclidean norms. Applying the triangle inequality gives

$$\|\Delta K_i^{(l)}\|_2 \le \|G_i^{(l)}\|_2 + \|D_i^{(l)}\|_2. \tag{8}$$

Thus, the aligned key reuse error is bounded by the geometric error G and the dynamic error D.

Exact form of the geometric term. RoPE acts independently on $d_k/2$ two-dimensional frequency blocks. For compactness, write

$$\begin{aligned} \widetilde{k}_{i,m}^{(l),\text{iso}} &:= \widetilde{k}_i^{(l),\text{iso},(\text{m})} \in \mathbb{R}^2, \\ m &= 0, \dots, d_k/2 - 1. \end{aligned} \tag{9}$$

Then

$$\widetilde{k}_i^{(l),\text{iso}} = \left[\widetilde{k}_{i,0}^{(l),\text{iso}}, \dots, \widetilde{k}_{i,d_k/2-1}^{(l),\text{iso}}\right]. \tag{10}$$

Because $U_i^{(l)}$ is the relative RoPE transform between the reuse coordinates and the direct full-context coordinates, its m-th block is a two-dimensional rotation by

$$\begin{aligned} \delta\theta_{i,m} &:= \theta_m(\widehat{g}(i), \bar{\alpha}_l) - \theta_m(g(i), \alpha_l^{\text{dir}}) \\ &= \omega_m \left(\frac{\widehat{g}(i)}{\bar{\alpha}_l} - \frac{g(i)}{\alpha_l^{\text{dir}}}\right). \end{aligned} \tag{11}$$

Here $\widehat{g}(i)$ and $\bar{\alpha}_l$ are the position and scale used by the reuse transform, while $g(i)$ and α_l^{dir} are those of direct full-context computation. Therefore,

$$\begin{aligned} \|G_i^{(l)}\|_2^2 &= \sum_{m=0}^{d_k/2-1} \left\|(I - R_{\delta\theta_{i,m}})\widetilde{k}_{i,m}^{(l),\text{iso}}\right\|_2^2 \\ &= 4 \sum_{m=0}^{d_k/2-1} \sin^2\left(\frac{\delta\theta_{i,m}}{2}\right) \left\|\widetilde{k}_{i,m}^{(l),\text{iso}}\right\|_2^2. \end{aligned} \tag{12}$$

The second equality follows from

$$(I - R_\phi)^\top (I - R_\phi) = 4\sin^2\left(\frac{\phi}{2}\right) I_2 \tag{13}$$

for any two-dimensional rotation matrix R_ϕ.

Using $|\sin(x)| \le |x|$, we obtain the frequency-wise upper bound

$$\|G_i^{(l)}\|_2 \le \left(\sum_{m=0}^{d_k/2-1} (\delta\theta_{i,m})^2 \left\|\widetilde{k}_{i,m}^{(l),\text{iso}}\right\|_2^2\right)^{1/2}. \tag{14}$$

Combining Eq. (8) and Eq. (14) gives

$$\begin{aligned} \|\Delta K_i^{(l)}\|_2 \le &\left(\sum_{m=0}^{d_k/2-1} (\delta\theta_{i,m})^2 \left\|\widetilde{k}_{i,m}^{(l),\text{iso}}\right\|_2^2\right)^{1/2} \\ &+ \|D_i^{(l)}\|_2. \end{aligned} \tag{15}$$

Zero geometric-error condition. Eq. (14) shows that $G_i^{(l)} = 0$ whenever $\delta\theta_{i,m} = 0$ for every frequency band m. A simple sufficient way to ensure this is to place the reused key at the same RoPE position and scale as direct full-context computation, i.e.,

$$\widehat{g}(i) = g(i), \quad \bar{\alpha}_l = \alpha_l^{\text{dir}}. \tag{16}$$

Then $U_i^{(l)} = I$, so

$$\begin{aligned} G_i^{(l)} &= (I - U_i^{(l)})\widetilde{k}_i^{(l),\text{iso}} \\ &= (I - I)\widetilde{k}_i^{(l),\text{iso}} = 0. \end{aligned} \tag{17}$$

This removes the geometric error G, but the dynamic error D may still remain.

Value states. For value states, RoPE is typically not applied at the current projection, so their reuse error has no direct geometric term. We treat value reuse error as a value-side dynamic error, although deeper-layer values may still reflect upstream positional effects through $h^{(l-1)}$.

$$\Delta V_i^{(l)} = \widetilde{v}_i^{(l),\mathrm{dir}} - \widetilde{v}_i^{(l),\mathrm{iso}}. \tag{18}$$

In this decomposition, value error is treated as value-side dynamic error, although deeper-layer values may still reflect upstream positional effects.

C Equivalence of Alignment Implementation Strategies

This section shows that, for the same underlying pre-RoPE key, two RoPE alignment implementations are equivalent: storing the pre-RoPE key and applying the target RoPE, or storing the post-RoPE key and applying a relative source-to-target rotation. This equivalence concerns coordinates only; it does not turn an isolated representation into the direct full-context representation.

We consider two distinct storage strategies common in inference systems:

- **Strategy 1 (Pre-Encoding Cache):** The system stores the raw projection $\widetilde{k}_i^{(l)}$. During reuse, the target positional encoding $\mathcal{P}_{\mathrm{tgt}}^{(l)}$ is applied directly.
- **Strategy 2 (Post-Encoding Cache):** The system stores the encoded key $K_i^{(l),\mathrm{cache}} = \mathcal{P}_{\mathrm{src}}^{(l)}(\widetilde{k}_i^{(l)})$ computed during the initial prefill. During reuse, alignment is performed by effectively removing the source encoding and applying the target encoding.

Lemma 1 (Invariance of Alignment Strategies)**.** *Let $\mathcal{P}_{src}^{(l)}$ and $\mathcal{P}_{tgt}^{(l)}$ denote the positional operators for the source (cached) and target (reuse) contexts, respectively. Assuming $\mathcal{P}_{src}^{(l)}$ is a bijective mapping (a property held by RoPE as an orthogonal rotation), Strategy 1 and Strategy 2 yield identical aligned representations.*

Proof. Let $\widetilde{k}_i^{(l)}$ be the underlying content representation.

Under Strategy 1, the aligned key is defined directly as:

$$K_{\mathrm{S1}}^{(l),\mathrm{aligned}} := \mathcal{P}_{\mathrm{tgt}}^{(l)}\left(\widetilde{k}_i^{(l)}\right). \tag{19}$$

Under Strategy 2, the cached term is $C_i = \mathcal{P}_{\mathrm{src}}^{(l)}(\widetilde{k}_i^{(l)})$. The alignment operation applies the inverse source operator followed by the target operator:

$$\begin{aligned} K_{\mathrm{S2}}^{(l),\mathrm{aligned}} &:= \mathcal{P}_{\mathrm{tgt}}^{(l)} \circ \left(\mathcal{P}_{\mathrm{src}}^{(l)}\right)^{-1}(C_i) \\ &= \mathcal{P}_{\mathrm{tgt}}^{(l)}\left(\left(\mathcal{P}_{\mathrm{src}}^{(l)}\right)^{-1}\left(\mathcal{P}_{\mathrm{src}}^{(l)}(\widetilde{k}_i^{(l)})\right)\right). \end{aligned} \tag{20}$$

Since $\left(\mathcal{P}_{\mathrm{src}}^{(l)}\right)^{-1} \circ \mathcal{P}_{\mathrm{src}}^{(l)} = I$, where I is the identity map, we obtain:

$$K_{\mathrm{S2}}^{(l),\mathrm{aligned}} = \mathcal{P}_{\mathrm{tgt}}^{(l)}\left(I \cdot \widetilde{k}_i^{(l)}\right) = \mathcal{P}_{\mathrm{tgt}}^{(l)}\left(\widetilde{k}_i^{(l)}\right). \tag{21}$$

Comparing Eq. (19) with the result above confirms that $K_{\mathrm{S1}}^{(l),\mathrm{aligned}} \equiv K_{\mathrm{S2}}^{(l),\mathrm{aligned}}$.

Application to RoPE. In RoPE, each operator is an orthogonal rotation matrix: $P_\theta^{(l)} = R_{\theta(i)}$, and hence $(P_\theta^{(l)})^{-1} = R_{-\theta(i)}$. Therefore,

$$P_{\mathrm{tgt}} \circ (P_{\mathrm{src}})^{-1} = R_{\theta_{\mathrm{tgt}}} R_{\theta_{\mathrm{src}}}^{\top} = R_{\theta_{\mathrm{tgt}} - \theta_{\mathrm{src}}}. \tag{22}$$

For Transition Overlap, the source operator must use the construction-time source coordinate. For the j-th token of segment r (corresponding to global token $i = (r, j)$) encoded after an overlap buffer O_r, we have $P_{\mathrm{src}} = R_{\theta(s_r(j),\alpha_{\mathrm{src}})}$ with $s_r(j) = |O_r| + j$, rather than $R_{\theta(j,\alpha_{\mathrm{src}})}$. This confirms that applying the relative rotation difference in Eq. (22) is mathematically exact and equivalent to pre-alignment. □

D Error Decomposition Diagnostics

Dataset and Scale. We conduct our validation on the complete LongBench MuSiQue benchmark, comprising 200 multi-hop question-answering instances. Unlike prior pilot studies using sampled subsets, this experiment processes the entire standard evaluation set to ensure statistical robustness. The dataset contains 2,218 segments (3–20 segments per question, variable length), yielding 1,997 analysis samples after filtering the lowest 10% error cases to remove trivial reuse scenarios.

Error Decomposition Framework. We decompose the KV cache reuse error into two complementary sources: a geometric-error component (captured by the upper bound RE_G) and a dynamic error component (RE_D), corresponding to the normalized magnitude of the dynamic error D defined

Layer	RE_K Measured	RE_G Bound	RE_D Measured	Coverage
0	0.962	0.952	0.081	100%
6	1.175	0.990	0.869	100%
13	1.015	0.882	0.606	100%
27	1.015	0.862	0.492	100%

Table 4: **Layer-wise error statistics on LongBench MuSiQue.** RE_D denotes the measured dynamic error component, while RE_G is a theoretical upper bound on the geometric error component. RE_K is the observed total reuse error.

in Sec. 4.1. In addition, we report the observed total reuse error RE_K. At each layer l and token position i, our decomposition satisfies:

$$\mathrm{RE}_K^{(l)}(i) \leq \mathrm{RE}_G^{(l)}(i) + \mathrm{RE}_D^{(l)}(i). \qquad (23)$$

Coverage check. The 100% coverage is expected from the additive upper bound in Eq. (23) and serves as an implementation sanity check. It does not by itself prove that the decomposition is complete or tight. Bound tightness is better assessed by reporting the slack $\mathrm{RE}_G + \mathrm{RE}_D - \mathrm{RE}_K$ or the ratio $\mathrm{RE}_K/(\mathrm{RE}_G + \mathrm{RE}_D)$.

Quantitative Layer-Wise Statistics

Tab. **??** summarizes the error statistics across the four representative layers. Several critical observations emerge from this data:

Layer 0 (Input Layer—Geometry Dominance): With $RE_G = 0.952$ and $RE_D = 0.081$, the dynamic-error component is small in this diagnostic. In an idealized deterministic first-layer pre-RoPE projection with identical token inputs, this residual would be expected to be close to zero. The small nonzero value here should be interpreted as an implementation- and measurement-level residual, e.g., due to normalization/extraction conventions, finite-precision arithmetic, and the normalized error statistic, rather than as strong evidence of boundary drift at the input projection. This is consistent with the decomposition: shallow key errors are strongly affected by RoPE coordinate mismatch. The close overlap between the geometric error region (blue) and the observed error line (black) in Fig. 9(a) suggests that RoPE position alignment is especially important at the input layer.

Layer 6 (Early Middle Layer—Large Residual Contribution): This layer exhibits the highest total error ($RE_K = 1.175$) and a large increase in the dynamic-error component ($RE_D = 0.869$ vs. 0.081 at Layer 0). Here, RE_D becomes comparable to RE_G ($RE_G = 0.990$), indicating a change in the observed error profile. The expanded orange region in Fig. 9(b) visualizes this larger dynamic-error contribution.

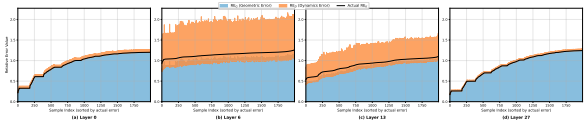


Figure 9: **Error decomposition diagnostics across representative layers.** The plots visualize the geometric-error bound (RE_G, blue) and dynamic-error component (RE_D, orange) stacked against the observed total reuse error (RE_K, black). **Consistency with Eq. (23):** All samples fall within the additive bound $\mathrm{RE}_K \leq \mathrm{RE}_G + \mathrm{RE}_D$. (a) **Layer 0:** Coordinate error is prominent. (b) **Layer 6:** The dynamic-error component becomes larger. (c) **Layer 13:** Both terms contribute. (d) **Layer 27:** The dynamic-error component decreases in this diagnostic.

Layer 13 (Middle Layer—Balanced Regime): Both terms contribute substantially ($RE_G = 0.882$, $RE_D = 0.606$), with total error stabilizing at $RE_K = 1.015$. This suggests that middle layers may benefit from both coordinate alignment and construction-time reduction of dynamic error.

Layer 27 (Output Layer—Residual Decay): The dynamic-error component decreases to $RE_D = 0.492$ (a 43% reduction from Layer 6), while total error remains stable at $RE_K = 1.015$. This suggests that later layers are less sensitive to segmentation artifacts in this diagnostic. The stacked regions continue to cover the observed error line in Fig. 9(d), as expected from the additive bound.

Implications for Cache Optimization

These diagnostics suggest practical directions for cache optimization: shallow layers are strongly affected by coordinate alignment, middle layers show larger dynamic-error components, and later layers appear less sensitive in this setting. These observations motivate construction-time methods such as Transition Overlap, but layer-specific optimization remains future work.

E Validation Across Multiple SAMSum Examples

To test whether the ablation patterns in Sec. 6.3 generalize beyond a single example, we repeat the layer-wise reuse error analysis on the first 7 SAMSum instances. We compare Naive Full Reuse w/o RoPE position alignment (direct concatenation without RoPE position alignment) against CSO, and visualize the L_2 reuse error for both Key and Value caches across all layers: 32 lay-

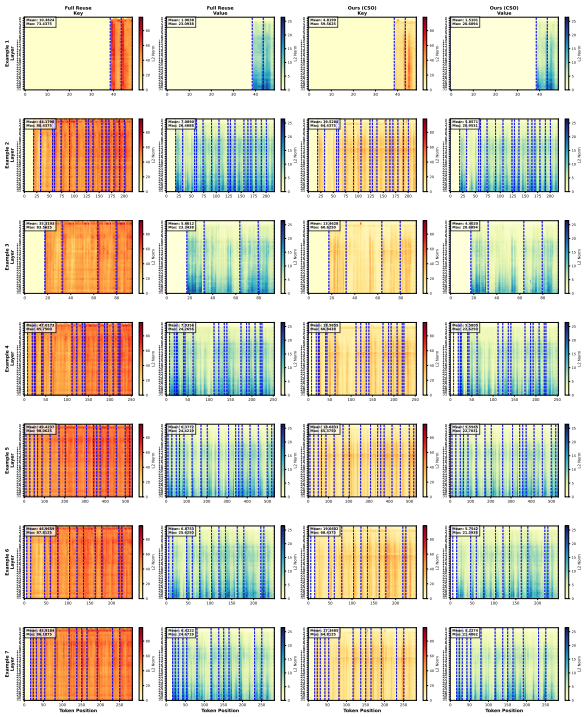


Figure 10: **Reuse error across 7 SAMSum examples (LLaMA-3.1-8B-Instruct).** Full Reuse shows pronounced boundary artifacts in both Key/Value caches, while CSO consistently reduces these errors across all layers and examples.

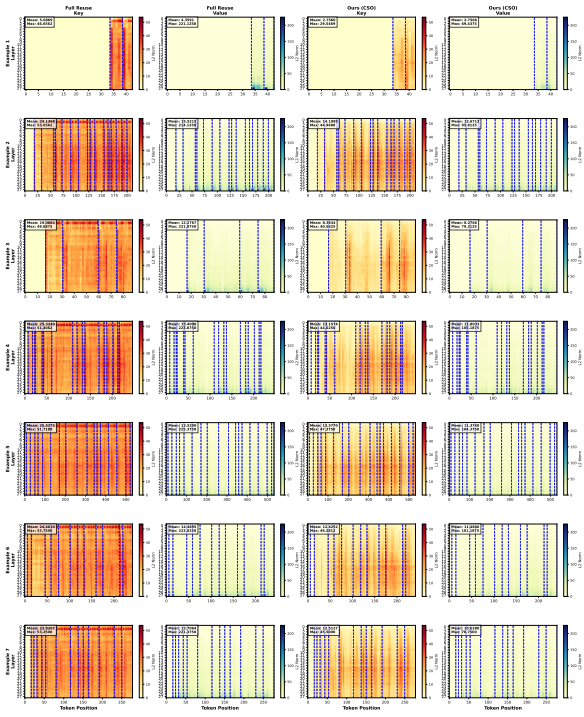


Figure 11: **Reuse error across 7 SAMSum examples (Qwen-2.5-7B-Instruct).** The same boundary-error reduction pattern holds on Qwen, indicating that our method is model-agnostic.

ers for LLaMA-3.1-8B-Instruct and 28 layers for Qwen-2.5-7B-Instruct.

Figures 10 and 11 show consistent trends across LLaMA-3.1-8B-Instruct and Qwen-2.5-7B-Instruct: Full Reuse exhibits prominent boundary spikes (segment-transition artifacts) in both Key and Value, whereas CSO substantially attenuates these spikes and lowers reuse error across layers. This confirms that the benefits of Transition Overlap are robust across samples and model families.

App. F provides an additional qualitative attention-map diagnostic showing how the temporary overlap can absorb segment-start attention-sink artifacts before the overlap states are discarded.

F Attention Sink Mitigation via Transition Overlap

Independently encoding each segment can introduce start-of-sequence artifacts, including attention sinks where early tokens receive disproportionately high attention. When segments are later concatenated for KV reuse, these artifacts appear as boundary discontinuities. Transition Overlap can mitigate this effect in our diagnostic by prepending a short overlap buffer during cache construction, so that any sink behavior is absorbed by the overlap region that will be trimmed before storage. Concretely, we encode an augmented segment $\tilde{\mathcal{B}} = \mathcal{O} \oplus \mathcal{B}$ and then discard the overlap cache, retaining only the KV states for the original segment (Sec. 5.2).

G Validation of Geometric Error Bounds Under RoPE Parameter Variations

This section checks the geometric error bound in Eq. (14) under two common RoPE parameter variations: (i) a scaling factor α that rescales rotation angles, and (ii) the RoPE frequency base b that determines $\omega_m = b^{-2m/d_k}$. We run the same bound-checking pipeline on MuSiQue using Qwen-2.5-7B-Instruct and report results for layers $l \in \{0, 6, 13, 27\}$.

G.1 Experiment A: Alpha Scaling

We vary $\alpha \in \{0.5, 1.0, 2.0, 4.0\}$ in the RoPE angle definition (consistent with Sec. 3), which directly changes the angular mismatch term $\delta\theta_{i,m}$ in Eq. (14). Fig. 14 plots the observed total error $\|\Delta K^{(l)}\|_2$ against the bound $\|G^{(l)}\|_2 + \|D^{(l)}\|_2$ across layers and α values. Across all settings, every sample satisfies the additive bound, as expected from the derivation. This also checks that our implementation of the bound is consistent across α

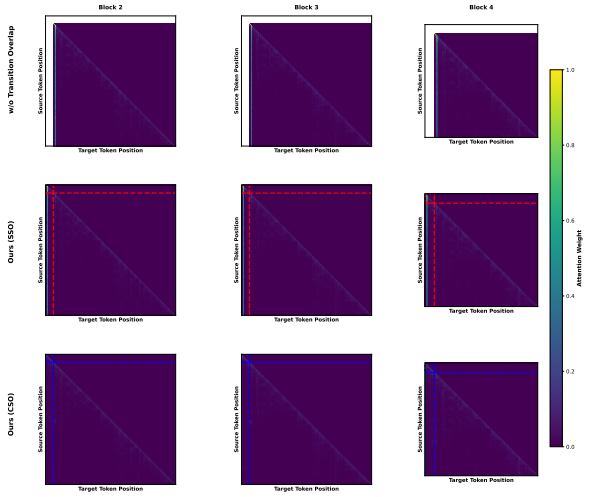


Figure 12: Attention maps (Layer 15, query position 40) on a SAMSum segment. The dashed line indicates the trimming boundary. Without overlap, attention concentrates near the segment start. With Ours (SSO)/(CSO), the concentration shifts into the overlap region and is removed after trimming.

values. We also observe that sensitivity to α is layer-dependent: shallow layers show more pronounced changes, while mid layers are often dominated by dynamic error.

G.2 Experiment B: Frequency Base Variations

We vary the RoPE base $b \in \{10^4, 10^5, 5 \times 10^5, 10^6\}$, which changes the frequency schedule $\omega_m = b^{-2m/d_k}$ and hence the band-wise contribution in Eq. (14). Fig. 15 reports the same bound check under different b. Again, the additive bound holds with 100% coverage across all layers and base values. Changing b produces distinct layer-wise effects compared to α, consistent with the frequency-weighted structure of Eq. (14). These experiments show that the additive bound remains satisfied under the tested RoPE bases, and the observed changes are consistent with the frequency-wise form of Eq. (14).

H Parallel Construction and Wall-Time Breakdown

Transition Overlap adds computation only during cache construction. This stage is outside online decoding and can be parallelized across cached segments. To make this cost explicit, we decompose the per-sample wall-time into three parts: the construction-parallelizable stage, the time to first token (TTFT), and the decode tail. This analysis complements the TTFT results in the main paper by showing where the extra construction cost appears and how it changes under parallelism.

Fig. 13 reports the breakdown on MuSiQue. Increasing offline parallelism mainly reduces the construction-parallelizable stage component, while TTFT and the decode tail remain relatively stable. Across all tested parallelism levels, CSO achieves the lowest average sample wall-time. This indicates that moving boundary repair to cache construction does not create a large online bottleneck, and that the added construction cost can be effectively amortized in practical serving settings.

H.1 Statistical Reliability Across Models and Tasks

For task quality, we use paired case bootstrap over the same 200 instances and report 95% confidence intervals for the CSO score and paired differences from each baseline. For latency, each configuration is measured in five fresh-process repeats, and hierarchical paired bootstrap over repeats and matched instances is used to obtain 95% confidence intervals for speedup ratios. Across two models and five tasks, we detect no significant quality degradation relative to EPIC or CacheBlend. CSO is significantly faster in 9/10 settings against each baseline, with mean speedups of 1.01–$1.62\times$ over EPIC and 1.04–$1.62\times$ over CacheBlend.

H.2 Matched SSO/CSO Overlap-Length Sweep

We additionally sweep matched overlap windows for SSO and CSO on HotpotQA, MuSiQue, and 2WikiMQA. Across 21 matched comparisons, the F1 differences between CSO and SSO range from -0.058 to $+0.024$, with no significant difference after Holm correction. Their wall-time ratios range from 0.939 to 1.088, with dataset-dependent fluctuations and no consistent winner. These results do not identify a universally optimal overlap window and support using $N_{\text{sb}} = 16$ as a practical low-overhead default.

I Matched-Budget Link Between Proxy Discrepancy and End-to-End Quality

The smoothness gap and the L2 distance are diagnostic quantities computed with a direct full-context reference, so they are not used as online inputs during deployment. However, they can still be useful if reducing them is associated with better end-to-end task quality. To test this link under a

Task	CSO score [95% CI]	Δ EPIC [95% CI]	Δ CacheBlend [95% CI]	Wall time	Speedup EPIC	Speedup CacheBlend
Mistral-7B-Instruct						
HotpotQA	0.317 [0.272, 0.364]	+0.004 [−0.030, +0.039]	+0.081 [+0.037, +0.126]	0.477 ± 0.011 s	1.07 × [1.01, 1.12]	1.19 × [1.13, 1.25]
MuSiQue	0.156 [0.132, 0.181]	−0.002 [−0.023, +0.019]	+0.055 [+0.033, +0.078]	0.557 ± 0.008 s	1.10 × [1.06, 1.14]	1.18 × [1.14, 1.23]
2WikiMQA	0.215 [0.189, 0.242]	+0.032 [+0.014, +0.050]	+0.054 [+0.032, +0.076]	0.465 ± 0.023 s	1.01 × [0.94, 1.09]	1.04 × [0.97, 1.11]
SAMSum	0.329 [0.308, 0.341]	+0.004 [−0.013, +0.021]	+0.144 [+0.123, +0.166]	1.507 ± 0.106 s	1.20 × [1.12, 1.27]	1.49 × [1.37, 1.63]
Needle	0.661 [0.615, 0.680]	−0.012 [−0.038, +0.015]	−0.012 [−0.044, +0.022]	0.575 ± 0.011 s	1.59 × [1.54, 1.64]	1.62 × [1.56, 1.70]
Qwen3-4B						
HotpotQA	0.270 [0.216, 0.296]	+0.011 [+0.003, +0.021]	+0.004 [−0.006, +0.010]	0.466 ± 0.049 s	1.28 × [1.24, 1.34]	1.49 × [1.40, 1.59]
MuSiQue	0.101 [0.052, 0.141]	+0.007 [−0.003, +0.017]	−0.008 [−0.023, +0.008]	0.504 ± 0.024 s	1.42 × [1.35, 1.49]	1.60 × [1.52, 1.67]
2WikiMQA	0.167 [0.155, 0.180]	+0.001 [−0.010, +0.011]	+0.005 [−0.005, +0.010]	0.503 ± 0.031 s	1.42 × [1.35, 1.50]	1.49 × [1.41, 1.58]
SAMSum	0.397 [0.377, 0.419]	+0.002 [−0.009, +0.011]	+0.152 [+0.126, +0.178]	1.712 ± 0.080 s	1.28 × [1.22, 1.35]	1.18 × [1.11, 1.26]
Needle	0.793 [0.778, 0.816]	−0.006 [−0.018, +0.012]	−0.004 [−0.014, +0.010]	0.537 ± 0.018 s	1.62 × [1.53, 1.71]	1.61 × [1.56, 1.66]

Table 5: CSO scores, quality differences, wall-clock time, and speedups across tasks.

Dataset	N_{sb}	SSO F1	CSO F1	ΔF1	Adjusted p	CSO/SSO Wall-time
HotpotQA	16	0.311	0.318	+0.007	1.000	1.031
	32	0.302	0.316	+0.013	1.000	0.998
	64	0.315	0.295	-0.021	1.000	0.968
	128	0.289	0.291	+0.002	1.000	1.021
	256	0.305	0.297	-0.008	1.000	0.988
	512	0.281	0.295	+0.014	1.000	0.964
	1024	0.299	0.306	+0.007	1.000	0.969
MuSiQue	16	0.163	0.158	-0.005	1.000	1.017
	32	0.158	0.163	+0.005	1.000	1.017
	64	0.143	0.150	+0.007	1.000	0.995
	128	0.149	0.152	+0.003	1.000	1.007
	256	0.155	0.160	+0.005	1.000	0.982
	512	0.147	0.165	+0.018	1.000	0.955
	1024	0.157	0.181	+0.024	1.000	0.939
2WikiMQA	16	0.204	0.215	+0.012	1.000	1.000
	32	0.187	0.200	+0.013	1.000	0.983
	64	0.207	0.174	-0.033	1.000	0.982
	128	0.242	0.190	-0.053	0.292	1.088
	256	0.231	0.188	-0.042	0.769	1.056
	512	0.235	0.192	-0.043	1.000	1.003
	1024	0.227	0.168	-0.058	0.234	0.970

Table 6: **Matched overlap-length sweep for SSO and CSO.** For each dataset and overlap length, SSO and CSO are evaluated under identical settings. ΔF1 denotes $\mathrm{F1}_{\mathrm{CSO}} - \mathrm{F1}_{\mathrm{SSO}}$ and is computed from the unrounded scores. The final column reports the wall-time ratio $\mathrm{WallTime}_{\mathrm{CSO}}/\mathrm{WallTime}_{\mathrm{SSO}}$.

controlled setting, we conduct a matched-budget intervention on HotpotQA.

For each prefix budget k, we keep both the insertion position and the number of prefix tokens fixed. We then change only the prefix content: Dummy-k uses content-free tokens, while CSO-k uses the matched previous-context overlap. This design controls for token budget and position, so each arrow in Fig. 16 represents a direct intervention on prefix content rather than a change in compute.

Across all evaluated budgets, replacing Dummy-k with CSO-k moves the system toward lower proxy discrepancy and higher Mean F1. This does not claim that the smoothness gap alone is a complete causal explanation for task accuracy. Rather, it provides controlled operational evidence that reducing the proxy discrepancies used in our analysis is aligned with better end-to-end performance under the same budget and position. We observe the same trend on MuSiQue: averaged across budgets, CSO reduces the smoothness gap and aligned L_2 distance by 1.13 and 1.93, respectively, while increasing F1 by 0.119.

J Boundary Fragility of Self-Segment Overlap

This appendix clarifies a boundary-sensitive limitation of Self-Segment Overlap (SSO). As described in Sec. 5.2, SSO is used when the true preceding segment is unavailable or when the system stores only one query-independent cache per segment. It prepends a short prefix copied from the current segment during cache construction, and

Relationship	$\hat{\rho}$	95% CI	p
Γ_i^{iso} vs. $\left\|\mathbf{z}_i^{\text{iso}} - \mathbf{z}_i^{\text{dir}}\right\|_2$	0.9564	[0.9428, 0.9669]	7.747×10^{-108}
Γ_i^{iso} vs. $\left\|\mathbf{z}_{i+1}^{\text{iso}} - \mathbf{z}_{i+1}^{\text{dir}}\right\|_2$	0.9567	[0.9431, 0.9671]	4.439×10^{-108}
Γ_i^{iso} vs. $\frac{1}{2}\left(\left\|\mathbf{z}_i^{\text{iso}} - \mathbf{z}_i^{\text{dir}}\right\|_2 + \left\|\mathbf{z}_{i+1}^{\text{iso}} - \mathbf{z}_{i+1}^{\text{dir}}\right\|_2\right)$	0.9566	[0.9430, 0.9670]	5.754×10^{-108}
Γ_i^{iso} vs. $\max\left(\left\|\mathbf{z}_i^{\text{iso}} - \mathbf{z}_i^{\text{dir}}\right\|_2, \left\|\mathbf{z}_{i+1}^{\text{iso}} - \mathbf{z}_{i+1}^{\text{dir}}\right\|_2\right)$	0.9596	[0.9470, 0.9693]	5.019×10^{-111}

Table 7: Example-level Pearson correlations between the smoothness gap and local aligned-state error summaries.

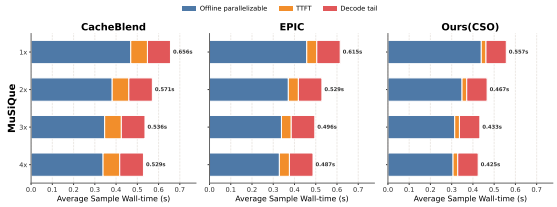


Figure 13: **MuSiQue wall-time breakdown across offline-parallelism levels.** We compare CacheBlend, EPIC, and Ours (CSO) under $1\times$, $2\times$, $3\times$, and $4\times$ offline parallelism. Each bar is decomposed into the construction-parallelizable stage, TTFT, and the decode tail. Higher parallelism mainly reduces the construction component, while TTFT and the decode tail remain stable. Ours (CSO) achieves the lowest average sample wall-time at all parallelism levels.

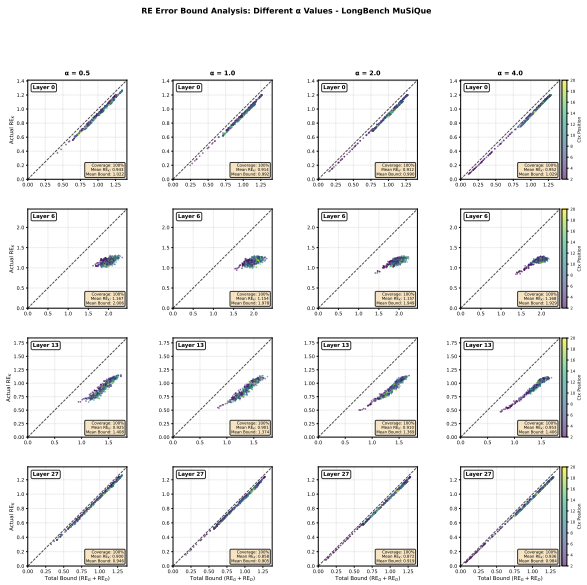


Figure 14: **Geometric error bound check under alpha scaling.** Scatter plots compare observed total error RE_K (y-axis) versus the additive bound $\mathrm{RE}_G + \mathrm{RE}_D$ (x-axis) across representative layers and scaling factors $\alpha \in \{0.5, 1.0, 2.0, 4.0\}$ on MuSiQue. Colors denote different α values, and the diagonal dashed line represents equality. **Key observations:** (1) **Bound check:** All points satisfy $\mathrm{RE}_K \leq \mathrm{RE}_G + \mathrm{RE}_D$, as expected from the additive bound. (2) **Layer behavior:** Shallow and deep layers show smaller slack than some middle layers in this diagnostic. (3) **Parameter effect:** Changing α changes the coordinate term through $\delta\theta_{i,m}$.

then stores only the KV states of the original segment. This construction is intended to approximate the transition context that CSO would provide in online-parallel construction or fixed source-adjacent caching. It is not designed to make the augmented construction sequence linguistically natural.

Representation smoothness is different from textual fluency. The Smoothness Gap in Eq. (3) measures a mismatch between local representation transitions under a reused-cache path and direct full-context computation. It is therefore a cache-level diagnostic, not a measure of natural language fluency. This distinction matters in RAG, where retrieved segments may already be semantically non-contiguous even under direct full-context inference. Our goal is to make the stored cache of each segment closer to its direct-path representation, rather than to enforce discourse continuity between retrieved segments.

Construction-time sequence vs. served prompt. SSO may create an unnatural temporary construction sequence. For example, if a segment begins with "10 dollars" and the copied prefix contains "10", the augmented construction input may contain a repeated surface form. This is a real construction-time concern, because the temporary prefix can influence the stored KV states of the original segment.

However, this repetition is not inserted into the served prompt. Given a segment B and a self-overlap O copied from the prefix of B, SSO encodes $\widetilde{B} = O \oplus B$, where $\oplus$ denotes input concatenation. After this forward pass, all KV states corresponding to O are discarded, and only the KV states of the original segment B are stored. At query time, the system assembles caches of the original segments only. Thus, the final generation-time context contains the original text once, although its stored KV states may have been affected by the temporary construction prefix.

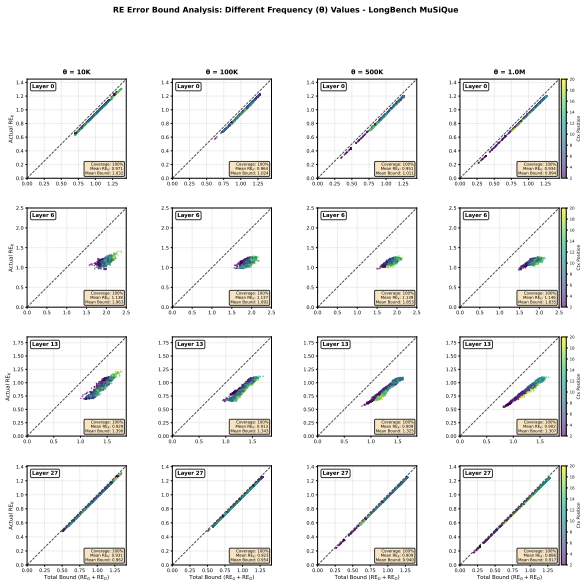


Figure 15: **Geometric error bound check under frequency-base variations.** Scatter plots analogous to Fig. 14, but varying the RoPE frequency base $b \in \{10^4, 10^5, 5 \times 10^5, 10^6\}$. **Key observations:** (1) Layer 27 shows lower observed error for larger tested bases in this diagnostic. (2) Layer 13 shows a non-monotonic pattern across tested bases. (3) The results are consistent with Eq. (14), where $\omega_m = b^{-2m/d_k}$ controls the per-band contribution.

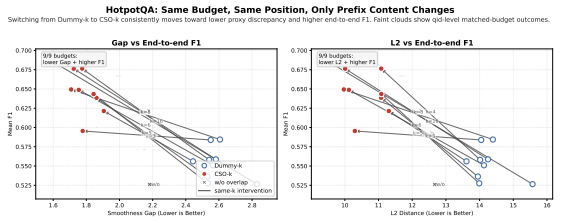


Figure 16: **Matched-budget intervention linking proxy reduction to end-to-end gains.** On HotpotQA, for each prefix budget k, we keep the insertion position and token budget fixed and change only the prefix content. Dummy-k uses content-free tokens, while CSO-k uses matched previous-context overlap. Each arrow is therefore a controlled intervention from Dummy-k to CSO-k. Across all evaluated budgets, this intervention consistently moves the system toward lower front smoothness gap or lower front L2 distance and higher end-to-end Mean F1.

Boundary-sensitive diagnostic. To quantify this limitation, we construct a targeted diagnostic subset of high-risk boundaries. A boundary is selected if the beginning of a segment contains a fragile surface form, such as a number, an entity mention, or a short span whose meaning can be changed by partial repetition. We further keep only cases where Full Compute answers correctly but removing the boundary tokens makes the task answer wrong. This filtering step ensures that the selected boundaries are not merely surface-level artifacts; the boundary tokens are also relevant to the downstream prediction. This procedure yields 23/19/29/12/9 cases on MuSiQue, HotpotQA, 2WikiMQA, SAMSum, and Needle, respectively. For each method, we report the task's standard metric on this boundary-sensitive subset: F1 for QA and retrieval tasks, and ROUGE-L for summarization. Scores are normalized by the corresponding Full Compute score on the same subset, so Full Compute equals 1.0. Higher values indicate better preservation of boundary-sensitive information.

Method	Budget 16	Budget 32	Budget 64
w/o overlap	0.232	0.232	0.232
SSO	0.342	0.356	0.428
CSO	0.377	0.447	0.467
EPIC	0.374	0.464	0.549
CacheBlend	0.317	0.327	0.323

Table 8: **Boundary-sensitive diagnostic results.** Scores are normalized by the Full Compute score on the same selected subset, so higher is better. For SSO and CSO, the budget denotes the overlap length. For EPIC, it denotes the number of recomputed boundary tokens. For CacheBlend, the three columns correspond to approximately 15%, 20%, and 25% recomputation rates, respectively.

Tab. 8 shows three findings. First, boundary-sensitive examples are difficult for position-independent cache reuse: without overlap, the normalized score is only 0.232. Second, SSO improves over no overlap at all budgets, indicating that a self-prefix can still provide useful construction-time context. Third, SSO does not fully remove boundary fragility. Its score increases from 0.342 at budget 16 to 0.428 at budget 64, suggesting that a longer self-overlap can reduce, but not eliminate, the effect of fragile boundary tokens. CSO performs better than SSO when source adjacency is available, because it uses matched previous-context tokens rather than a repeated self-prefix. EPIC is also strong at larger budgets, but it requires online boundary-token recomputation, whereas SSO and CSO keep the cached-token reuse path recomputation-free.