FastPIC: Fast Position Independent Caching for Multi-Tenant LLM Serving

Anonymous authors
Paper under double-blind review

Abstract

As LLM context windows continue to expand, prefill overhead has become major bottlenecks in long-context serving. Existing context-caching systems rely on prefix matching and cannot effectively reuse repeated content that appears at different positions across multi-tenant requests. Position-Independent Caching (PIC) supports non-prefix reuse, but it remains limited by boundary shifts caused by fixed chunking and attention deviations caused by cross-context KV reuse. This paper presents FastPIC, a position independent KV-cache reuse mechanism for multi-tenant LLM serving. FastPIC uses Token-Defined Chunking to derive boundaries and cache identities from token content, reducing the impact of prefix variations on matching identical token sequences. It further uses Dual-Path Token Selection to recompute attention sinks and high KV-deviation tokens under a limited budget, improving both cache reuse and generation quality. Experiments across multiple models and datasets show that FastPIC reduces TTFT and improves serving throughput while preserving generation quality.

1 Introduction

As LLM context windows expand (Team et al., 2025; GLM et al., 2024), applications such as multi-document question answering, code agents, and retrieval-augmented generation can exploit richer context, but prefill cost and time-to-first-token (TTFT) increase substantially. Context caching reduces this overhead by reusing previously computed KV caches for repeated content (Bang, 2023; Pan et al., 2026; Zheng et al., 2024). However, mainstream prefix-caching schemes (Kwon et al., 2023; Zheng et al., 2024; Liu et al., 2024; Jin et al., 2025) require an identical token prefix and reuse only the longest common prefix, leaving repeated content at different positions or under different prefixes unexploited.

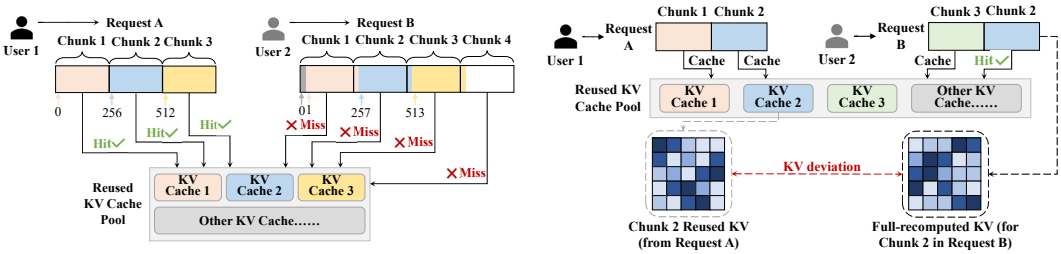
Position-Independent Caching (PIC) (Gim et al., 2024; Yao et al., 2025; Hu et al., 2024) extends the reuse granularity from common prefixes to non-prefix chunks, enabling the reuse of repeated content across requests. Existing PIC methods typically rely on applications to pre-identify reusable immutable chunks, such as RAG documents, few-shot examples, and fixed templates, and partition them for KV-cache storage using predefined boundaries or a fixed chunk size, typically 512 tokens.

However, this assumption rarely holds in general-purpose multi-tenant serving, where requests contain highly variable instructions, contexts, and queries. The system cannot know a priori which tokens will recur or where shared spans begin and end. Even identical passages may be shifted by prefixes or insertions of different lengths, causing them to cross different fixed chunks and receive different chunk identities, as shown in Figure 1a. We call this failure to reuse identical token content due to inconsistent chunk boundaries the **boundary-shift problem**.

Beyond boundary shift, PIC also suffers from the **attention-deviation problem**. A cache hit ensures token-content matching, but not lossless KV reuse. Cached KV states are computed under the source request; when reused in a target request with different instructions, contexts, or queries, their contextual dependencies may change. As shown in Figure 1b, the reused KV states can deviate from the representations that would be computed under the target request, perturbing the attention distribution.

Existing PIC methods mitigate attention deviation through selective KV recomputation with either dynamic (Yao et al., 2025; Zhou et al., 2025; Yang et al., 2025; 2026) or static (Hu et al., 2024;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) The Boundary-Shift Problem in PIC.

(b) The Attention-Deviation Problem in PIC.

Figure 1: Common Challenges for Position-Independent Caching in Multi-Tenant LLM Serving.

Wang et al., 2025; Gim et al., 2024) token selection. CacheBlend selects a fraction p of tokens with the largest deviation between cached KV states and KV states recomputed under the current full prompt. However, the selected tokens still attend over the full context, yielding $O(pN^2)$ complexity, and online selection adds runtime overhead. EPIC adopts a static policy based on attention sinks observed during independent prefilling: it recomputes the first K tokens of each immutable chunk because chunk-initial tokens can disproportionately absorb attention. This chunk-prefix assumption breaks in request-level multi-tenant caching. A reused chunk's KV states come from the full prefill of its original prompt, not from independent chunk prefilling; hence the chunk beginning is not an independent sequence beginning. Context-induced attention sinks may therefore appear anywhere within the chunk, so fixed chunk-prefix recomputation can miss the true sinks and waste computation.

To address boundary shifts and attention deviations in multi-tenant LLM serving, we present **FastPIC**. FastPIC first introduces **Token-Defined Chunking (TDC)**, which derives variable-length chunk boundaries and identities from token content. TDC allows recurring token spans to realign across requests with different prefixes and enables chunk-level hash lookup for KV-cache reuse. FastPIC further introduces **Dual-Path Token Selection (DPTS)** to mitigate attention deviation under a limited recomputation budget. One path uses early-layer attention matrices to compute position-normalized cumulative attention scores and identify sink tokens that abnormally attract attention. The other path measures the deviation between cached KV states and KV states recomputed under the current full prompt to identify critical tokens with distorted representations. DPTS recomputes the union of the two candidate sets, thereby correcting both attention routing and contextual representations while preserving the benefits of cache reuse.

Our contributions. First, we identify and formalize two key challenges for PIC in multi-tenant LLM serving: boundary shifts from fixed chunking and attention deviations from changed contextual dependencies. Second, we propose FastPIC, which combines Token-Defined Chunking and Dual-Path Token Selection to improve cross-request KV reuse while preserving generation quality under a limited recomputation budget. Third, we implement FastPIC and evaluate it on three LLMs (Qwen2.5-7B, Llama3.1-8B, and Llama-70B), six datasets (2WikiMQA, MuSiQue, SAMSum, MultiNews, Needle in a Haystack, and ShareGPT), and both synchronous and asynchronous workloads. FastPIC preserves near-FR generation quality. Compared with state-of-the-art PIC baselines at matched QPS, it delivers up to a 1.95× TTFT speedup and 1.2× higher input throughput, while supporting contexts up to 1.72× longer with lower TTFT.

2 Related Work

LLM Serving Systems. Existing LLM serving optimizations primarily target execution, scheduling, and storage. vLLM (Kwon et al., 2023) improves KV-cache management with PagedAttention, SGLang (Zheng et al., 2024) combines a domain-specific frontend with an optimized runtime, and DeepFlow (Hu et al., 2025) integrates serving optimizations for Ascend accelerators. Other systems study prefill–decode disaggregation, continuous batching, multi-LoRA serving, and KV-cache-centric inference (Zhong et al., 2024; Yu et al., 2022; Sheng et al., 2024; Qin et al., 2025). FastPIC is complementary to these systems: it targets cross-request KV-cache reuse during prefill and can be integrated as a context-caching component for multi-tenant LLM serving.

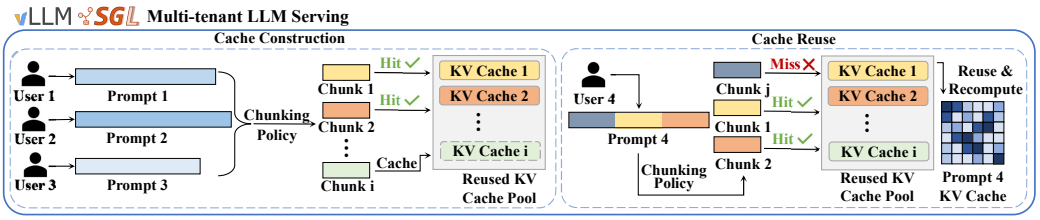


Figure 2: The PIC Architecture in Multi-Tenant LLM Serving.

Position-Independent Caching. PIC extends KV-cache reuse to non-prefix content by partitioning only pre-identified immutable prompt segments into chunks and selectively recomputing reused KV states to preserve generation quality (Gim et al., 2024; Yao et al., 2025; Hu et al., 2024; Yang et al., 2025). EPIC (Hu et al., 2024) adopts a static policy based on attention sparsity, whereas CacheBlend (Yao et al., 2025) dynamically selects tokens with large KV-representation deviations. However, the fixed-length chunking used by most PIC methods is sensitive to variations in request prefixes and structures, causing boundary misalignment and lower cache-hit rates. Adaptive matching (Yang et al., 2025) can alleviate this problem, but its overhead grows with the cache-pool and request sizes, making it unsuitable for latency-sensitive online serving. FastPIC addresses this limitation with Token-Defined Chunking, which constructs content-driven variable-length chunks with stable identities for efficient chunk-level KV reuse.

3 Problem Formulation

As shown in Figure 2, we conceptualize PIC in multi-tenant LLM serving as comprising two stages: cache construction and cache reuse. The former builds reusable KV-cache entries from incoming prompts, while the latter retrieves matched entries and selectively recomputes tokens to construct the KV cache for a new request.

Cache Construction. The system first partitions the prompt token sequence according to a chunking policy and assigns a unique chunk identity to each chunk. It then queries the corresponding cache: on a hit, the existing KV states are retained as reusable entries; on a miss, the system computes the chunk's KV states through prefill and inserts them into the cache. The key challenge in this stage is **how to make chunking robust to prefix differences, so that identical token spans across requests produce consistent boundaries and stable identities that map to the same cache entry.**

Cache Reuse. For subsequent requests, the system loads cached KV states according to matched chunk identities and combines them with the computed states of cache-missed and request-specific tokens to construct the KV cache for the current prompt. Because cached KV states may have been generated under different contexts, the system selectively recomputes the KV states of a subset of tokens to correct representation deviations caused by positional and contextual changes. The central challenge is **how to recalibrate attention toward generation-critical tokens and correct their KV representations under a limited recomputation budget, while preserving both accuracy and caching benefits.**

3.1 The Boundary-Shift Problem

Consider two requests, $P^{(a)}$ and $P^{(b)}$, in a multi-tenant LLM serving system. Suppose that the same token segment S, of length ℓ, occurs at positions i and j, respectively: $P^{(a)}[i : i + \ell) = P^{(b)}[j : j + \ell) = S$. Existing PIC methods typically partition each prompt into fixed-length chunks of B tokens and derive a cache identity from the token content of each chunk:

$$C_k^{(r)} = P^{(r)}[kB : (k+1)B), \qquad \mathrm{id}\left(C_k^{(r)}\right) = H\left(C_k^{(r)}\right), \quad r \in \{a, b\}. \tag{1}$$

Therefore, the identical token segment S can be mapped to the same chunk identity only when its chunk boundaries are aligned across the two requests. We define the chunk phase difference of S as $\delta_B(S) = (j - i) \bmod B$.

Definition 1 (Boundary-Shift Problem). *If an identical token segment S has different chunk phases across two requests, i.e., $\delta_B(S) \neq 0$, we say that S suffers from boundary shift. Fixed-length*

chunking may then group the same tokens with different neighboring content, producing different chunk identities and preventing cache matching. This occurs when the prefix-length difference is not a multiple of B and may affect multiple chunks within a long shared segment.

Let $\mathcal{R}_{\mathcal{C}}(S)$ denote the set of repeated tokens in S that can be mapped to the same cache entry by both requests under chunking strategy $\mathcal{C}$. We define the cache coverage of S as

$$\rho_{\mathcal{C}}(S) = \frac{|\mathcal{R}_{\mathcal{C}}(S)|}{\ell}. \tag{2}$$

Boundary shift arises because fixed-length chunking implicitly ties chunk identities to absolute token positions in the prompt. Changes in prefix length or request structure can therefore repartition identical content, reducing effective cross-request KV-cache reuse.

Design Insight. Chunk boundaries should be determined by local token content rather than absolute positions in the prompt. Under a chunking strategy $\mathcal{C}$, each chunk is represented as $C_k = P[e_k : e_{k+1}), \quad \mathrm{id}(C_k) = H(C_k)$, where e_k is a content-driven boundary satisfying predefined length constraints. This allows identical content preceded by different prefixes to resynchronize within the shared region and, when possible, receive the same chunk identity.

Given a workload distribution over repeated segments in multi-tenant requests, FastPIC seeks to maximize the expected cache coverage under chunk-length and chunking-overhead constraints: $\max_{\mathcal{C}\in\mathcal{F}} \ \mathbb{E}_S\left[\rho_{\mathcal{C}}(S)\right]$, where $\mathcal{F}$ denotes the set of chunking strategies satisfying the prescribed constraints. FastPIC's Token-Defined Chunking pursues this objective through content-driven variable-length chunks and stable chunk identities, reducing prefix-induced boundary mismatches and improving cross-request KV-cache hit rates.

3.2 The Attention-Deviation Problem

Consider a chunk cached from a source request $P^{(a)}$ and reused in a target request $P^{(b)}$. Let $\mathcal{U}$ denote the token positions covered by cache hits in the target request. For each $t \in \mathcal{U}$, we denote the cached KV representation and the corresponding representation under a full prefill of the target request by $\widetilde{\mathbf{Z}}_t = \left(\widetilde{\mathbf{K}}_t, \widetilde{\mathbf{V}}_t\right), \quad \mathbf{Z}_t^* = (\mathbf{K}_t^*, \mathbf{V}_t^*)$. Because the preceding contexts may differ between the source and target requests, these representations generally satisfy $\widetilde{\mathbf{Z}}_t \neq \mathbf{Z}_t^*$.

We define the normalized KV-representation deviation of token t as

$$d_t = \frac{\left\|\widetilde{\mathbf{K}}_t - \mathbf{K}_t^*\right\|_2 + \left\|\widetilde{\mathbf{V}}_t - \mathbf{V}_t^*\right\|_2}{\|\mathbf{K}_t^*\|_2 + \|\mathbf{V}_t^*\|_2 + \epsilon}, \tag{3}$$

where $\epsilon > 0$ prevents division by zero. Let $\mathbf{A}_{l,q}^*$ denote the attention distribution of query token q at layer l under full prefill, and let $\widetilde{\mathbf{A}}_{l,q}$ denote the distribution obtained by directly reusing the cached KV representations. We measure the overall attention deviation as

$$D_{\mathrm{att}} = \frac{1}{L|\mathcal{Q}|} \sum_{l=1}^{L} \sum_{q\in\mathcal{Q}} \left\|\widetilde{\mathbf{A}}_{l,q} - \mathbf{A}_{l,q}^*\right\|_1, \tag{4}$$

where L is the number of model layers and $\mathcal{Q}$ is the set of query tokens in the target request.

Definition 2 (Attention-Deviation Problem). *If cached KV representations are computed under a different context and their direct reuse causes the attention distribution to deviate from that of the target request under full prefill by more than ε, i.e., $D_{\mathrm{att}} > \varepsilon$, the cache-reuse process is said to suffer from the attention-deviation problem. A cache hit guarantees token-content consistency, but not consistency in contextual KV representations or attention behavior. The full-prefill KV representations serve only as an ideal reference and need not be computed online.*

To mitigate attention deviation, PIC selects a recomputation set $\mathcal{R} \subseteq \mathcal{U}$ from the cache-hit tokens and defines the recovered KV representation as $\mathbf{Z}_t^{(\mathcal{R})} = \mathbb{I}[t \in \mathcal{R}]\, \mathbf{Z}_t^* + \mathbb{I}[t \notin \mathcal{R}]\, \widetilde{\mathbf{Z}}_t$. Let $D_{\mathrm{att}}(\mathcal{R})$ denote the resulting attention deviation. Given a recomputation budget K, PIC selects $\mathcal{R}$ by solving $\min_{\mathcal{R}\subseteq\mathcal{U},\,|\mathcal{R}|\leq K} D_{\mathrm{att}}(\mathcal{R})$.

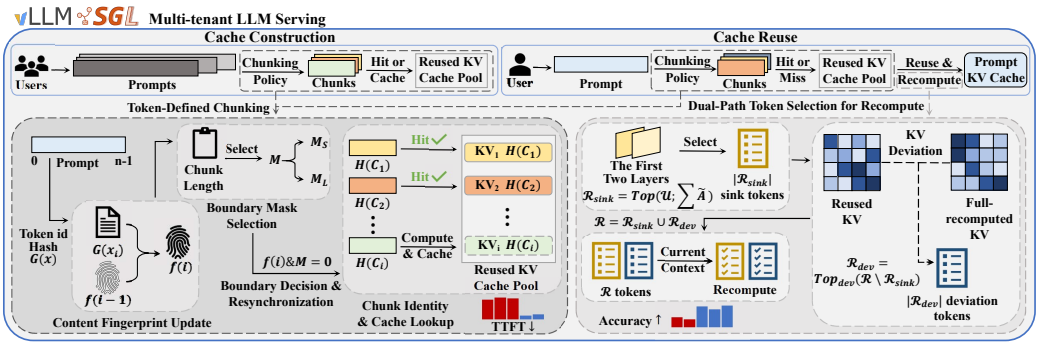


Figure 3: The Workflow of FastPIC.

Design Insight. An effective recomputation strategy should jointly target attention sinks and tokens with large KV-representation deviations. Attention sinks distort attention routing, whereas high-deviation tokens distort generation-critical contextual representations. Selecting both token types under a limited budget more closely approximates the attention behavior and generation quality of full prefill. FastPIC's Dual-Path Token Selection follows this principle.

4 FastPIC

FastPIC instantiates the multi-tenant PIC architecture described in Section 3 by optimizing cache construction and reuse. As shown in Figure 3, Token-Defined Chunking derives variable-length chunks and stable content identities, allowing repeated spans with different prefixes to realign and share KV-cache entries. Dual-Path Token Selection jointly identifies attention sinks and high-deviation cache-hit tokens for recomputation under a bounded budget, using the context of the current request. Together, the two mechanisms improve cross-request position-independent KV-cache reuse while preserving generation quality.

4.1 Token-defined Chunking

To address the boundary-shift problem caused by fixed-length chunking, FastPIC introduces **Token-Defined Chunking (TDC)**. It treats the prompt as a continuous token stream and dynamically determines variable-length chunk boundaries from local token content, without relying on the absolute positions of tokens in the full prompt.

Let the tokenized prompt be $P = (x_0, x_1, \ldots, x_{n-1}), \quad x_i \in \mathcal{V}$, where $\mathcal{V}$ denotes the model vocabulary. FastPIC applies a deterministic mapping $G : \mathcal{V} \to \{0,1\}^w$ to map each token ID to a w-bit value (Muthitacharoen et al., 2001; Xie et al., 2025; Rabin, 1981) and continuously updates the content fingerprint as follows:

$$f_i = ((f_{i-1} \ll 1) + G(x_i)) \bmod 2^w, \qquad f_{-1} = 0^w, \tag{5}$$

where $f_i \in \{0,1\}^w$. The fingerprint state is updated continuously while scanning the prompt. After forming a new chunk, the system resets only the chunk-length counter while preserving the fingerprint state.

Let s denote the start position of the current chunk and i the current position. The current chunk length is $\ell_i = i - s + 1$. Given the minimum, nominal, and maximum chunk lengths $B_{\min}$, B_{norm}, and $B_{\max}$, respectively, the system selects a boundary mask according to the current chunk length:

$$M(\ell_i) = \begin{cases} M_{\text{S}}, & B_{\min} \le \ell_i < B_{\text{norm}}, \\ M_{\text{L}}, & B_{\text{norm}} \le \ell_i < B_{\max}. \end{cases} \tag{6}$$

The current position is identified as a chunk boundary when

$$f_i \,\&\, M(\ell_i) = 0. \tag{7}$$

The system does not perform boundary detection before the chunk reaches $B_{\min}$, thereby avoiding excessively short chunks. Before the nominal length is reached, M_{S}, which contains more effective

bits, suppresses premature segmentation. After the nominal length, M_{L}, which contains fewer effective bits, increases the probability of detecting a boundary. If no boundary is detected by $B_{\max}$, the system forcibly terminates the chunk at the maximum length.

Algorithm 1 implements Token-Defined Chunking. Lines 1–4 initialize the masks, length constraints, and scan state, while Lines 5–7 update the fingerprint fp and chunk length ℓ. Boundary testing is skipped for $\ell < B_{\min}$. Based on ℓ, the algorithm selects M_{S} or M_{L}, which retain nine and six effective bits and trigger boundaries with probabilities 2^{-9} and 2^{-6}, respectively. Thus, M_{S} suppresses premature segmentation, whereas M_{L} promotes the termination of longer chunks. Line 14 declares a boundary when $fp \,\&\, M = 0$, and a boundary is forced when the chunk reaches $B_{\max}$. Lines 20–22 append the final boundary to ensure complete prompt coverage.

Cross-Request Boundary Resynchronization. Token-Defined Chunking determines chunk boundaries from token content rather than absolute positions. Consider two requests that contain the same token segment:

$$P^{(a)}[i : i + \ell) = P^{(b)}[j : j + \ell). \quad (8)$$

Let $f_0^{(a)}$ and $f_0^{(b)}$ denote the fingerprint states immediately before the shared segment in the two requests, respectively. After processing the first r identical tokens in the shared segment, we have

$$f_r^{(a)} - f_r^{(b)} \equiv 2^r \left(f_0^{(a)} - f_0^{(b)} \right) \pmod{2^w}. \quad (9)$$

When $r \geq w$, the initial-state difference vanishes modulo 2^w, so the two requests obtain the same fingerprint state. If they remain in the same length regime without triggering a forced boundary, subsequent identical tokens produce identical boundary decisions. Thus, sufficiently long shared spans can locally resynchronize with high probability. Unlike fixed-length chunking, TDC confines prefix-induced misalignment to a short initial resynchronization period rather than propagating it throughout the shared span.

Algorithm 1: Token-Defined Chunking (256)

Input: Tokenized prompt $P = (x_0, \ldots, x_{n-1})$ and deterministic token hash $G(\cdot)$
Output: Ordered chunk-boundary set $\mathcal{E}$

1 $M_{\mathrm{S}} \leftarrow$ `0x2FF`; `// 9 effective bits`
2 $M_{\mathrm{L}} \leftarrow$ `0x03F`; `// 6 effective bits`
3 $B_{\min} \leftarrow 64$; $B_{\mathrm{norm}} \leftarrow 256$; $B_{\max} \leftarrow 512$;
4 $fp \leftarrow 0$; $s \leftarrow 0$; $\mathcal{E} \leftarrow \{0\}$;
5 **for** $i \leftarrow 0$ **to** $n - 1$ **do**
6 $fp \leftarrow ((fp \ll 1) + G(x_i)) \bmod 2^w$;
7 $\ell \leftarrow i - s + 1$;
8 **if** $\ell \geq B_{\min}$ **then**
9 **if** $\ell < B_{\mathrm{norm}}$ **then**
10 $M \leftarrow M_{\mathrm{S}}$;
11 **else**
12 $M \leftarrow M_{\mathrm{L}}$;
13 **end**
14 **if** $(fp \,\&\, M) = 0 \ \vee \ \ell \geq B_{\max}$ **then**
15 $\mathcal{E} \leftarrow \mathcal{E} \cup \{i + 1\}$;
16 $s \leftarrow i + 1$;
17 **end**
18 **end**
19 **end**
20 **if** $s < n$ **then**
21 $\mathcal{E} \leftarrow \mathcal{E} \cup \{n\}$;
22 **end**
23 **return** $\mathcal{E}$;

Chunk Identity and Cache Lookup. For each generated chunk $C_k = P[e_k : e_{k+1})$, FastPIC derives a stable content identity from its token sequence: $\mathrm{id}(C_k) = H\big(x_{e_k} \| x_{e_k+1} \| \cdots \| x_{e_{k+1}-1}\big)$, where $H(\cdot)$ denotes the SHA-256 content hash function (Yoshida & Biryukov, 2005).

After boundary resynchronization, identical token sequences yield the same chunk identity. FastPIC uses a hash table to directly locate the corresponding KV-cache entry. On a cache hit, it reuses the stored KV representations; on a miss, it computes and inserts them into the cache. This avoids scanning the cache pool and performing request-level sequence matching. For a prompt of length n, chunking requires a single sequential scan with complexity $O(n)$. Each chunk lookup takes $O(1)$ time on average, enabling low-overhead chunk-level KV-cache reuse.

4.2 Dual-Path Token Selection

To mitigate attention deviation caused by context mismatch, FastPIC introduces **Dual-Path Token Selection (DPTS)**. Given the set of cache-hit tokens $\mathcal{U}$ and a recomputation ratio $p \in (0, 1)$, the total recomputation budget is set to $K = p|\mathcal{U}|$. DPTS employs two parallel selection paths. The first identifies attention sinks, whereas the second identifies tokens with large KV-representation deviations. Both paths share the total recomputation budget.

The first path identifies attention sinks from the attention matrices of the first two layers. Prior work shows that early-layer attention is often sparse and concentrated on a small subset of tokens, providing an early signal of anomalous attention accumulation (Ge et al., 2024; Tang et al., 2024).

Let $\mathcal{L}_{\text{early}} = \{1, 2\}$ denote these layers, and let $\widetilde{\mathbf{A}}_{l,q,t}$ denote the attention assigned by query token q to reusable token t at layer l. Let $\mathcal{Q}_t$ be the set of query positions that can attend to token t. We define the position-normalized cumulative attention score of token t as

$$s_{\text{sink}}(t) = \frac{1}{|\mathcal{L}_{\text{early}}|} \sum_{l \in \mathcal{L}_{\text{early}}} \frac{1}{|\mathcal{Q}_t|} \sum_{q \in \mathcal{Q}_t} \widetilde{\mathbf{A}}_{l,q,t}. \tag{10}$$

FastPIC selects the k highest-scoring tokens in $\mathcal{U}$ as attention-sink candidates: $\mathcal{R}_{\text{sink}} = \text{TopK}_k\left(\mathcal{U}; s_{\text{sink}}\right),$. Prior work shows that a small number of high-impact tokens often dominates attention and that additional recomputation provides diminishing returns (Zhang et al., 2023; Hu et al., 2024). We therefore set $k = 8$ to correct sink-induced routing distortion with limited overhead. Recomputing these tokens suppresses abnormal attention concentration and redirects attention toward generation-critical tokens.

The second path uses the remaining budget to identify cache-hit tokens with distorted KV representations. For each token in $\mathcal{U} \setminus \mathcal{R}_{\text{sink}}$, FastPIC ranks candidates by the KV-representation deviation score d_t defined in Section 3.2 and Eq. equation 3. The remaining budget is $K_{\text{dev}} = K - |\mathcal{R}_{\text{sink}}|$. FastPIC then selects the K_{dev} candidates with the largest deviation scores: $\mathcal{R}_{\text{dev}} = \text{TopK}_{K_{\text{dev}}}\left(\mathcal{U} \setminus \mathcal{R}_{\text{sink}}; d_t\right)$. The final recomputation set is the union of the candidates selected by both paths:

$$\mathcal{R} = \mathcal{R}_{\text{sink}} \cup \mathcal{R}_{\text{dev}}, \qquad |\mathcal{R}| \leq K. \tag{11}$$

The system recomputes the KV states of tokens in $\mathcal{R}$ under the current request context and reuses the cached states of the remaining cache-hit tokens. DPTS first restores attention routing by recomputing a small set of high-impact attention sinks, then uses the remaining budget to correct context-critical tokens with large KV-representation deviations. By recomputing only a fraction p of reusable tokens, DPTS jointly reduces anomalous attention concentration and representation distortion, bringing the reused attention behavior closer to full prefill while preserving the benefits of caching.

5 Evaluation

Implementation, Models, and Hardware. We implement FastPIC on vLLM 0.4.1 (Kwon et al., 2023) with approximately 2K additional lines of Python code. We evaluate Qwen2.5-7B, Llama3.1-8B, and Llama-70B on four NVIDIA RTX 6000D GPUs with 364 GB of aggregate GPU memory, a 64-core AMD EPYC 9755 CPU, 188 GB of system memory, and 1 TB of local storage. The software stack uses Ubuntu 22.04, CUDA 12.2, cuDNN 8, and Python 3.10.

Datasets. We use six datasets, each with 200 test cases, covering multi-hop question answering, summarization, long-context retrieval, and multi-tenant serving. The synchronous workloads use *2WikiMQA* (Ho et al., 2020), *MuSiQue* (Trivedi et al., 2022), *SAMSum* (Gliwa et al., 2019), *MultiNews* (Fabbri et al., 2019), and *Needle in a Haystack* (Gao et al., 2026). For the first two datasets, we append `Answer within 5 words.` to each prompt to enforce the short-answer format. We construct asynchronous multi-tenant workloads from *ShareGPT* (ShareGPT Team, 2023) by controlling request arrival rates.

Metrics. We measure efficiency using *Time-to-First-Token* (TTFT), defined as the latency from request arrival to the first generated token, and asynchronous throughput, measured in completed requests or generated tokens per unit time. We evaluate generation quality with F1-score on *2WikiMQA*, *MuSiQue*, and *Needle in a Haystack*, and ROUGE-L on *SAMSum* and *MultiNews*. Lower TTFT is better, whereas higher values are better for all other metrics.

Baselines. We compare FastPIC with *Full Recompute* (FR), *Naive* (Gim et al., 2024), *CacheBlend-r* (Yao et al., 2025), and *EPIC-k* (Hu et al., 2024). FR recomputes all prompt KV states, Naive reuses all matched KV states without recomputation, CacheBlend-r dynamically recomputes an r fraction of tokens, and EPIC-k recomputes the first k tokens of each chunk. We vary r and k to characterize the trade-off between recomputation overhead and generation quality.

Workloads. We evaluate synchronous and asynchronous workloads. In the synchronous setting, requests are processed sequentially to measure the quality–latency trade-off for individual requests.

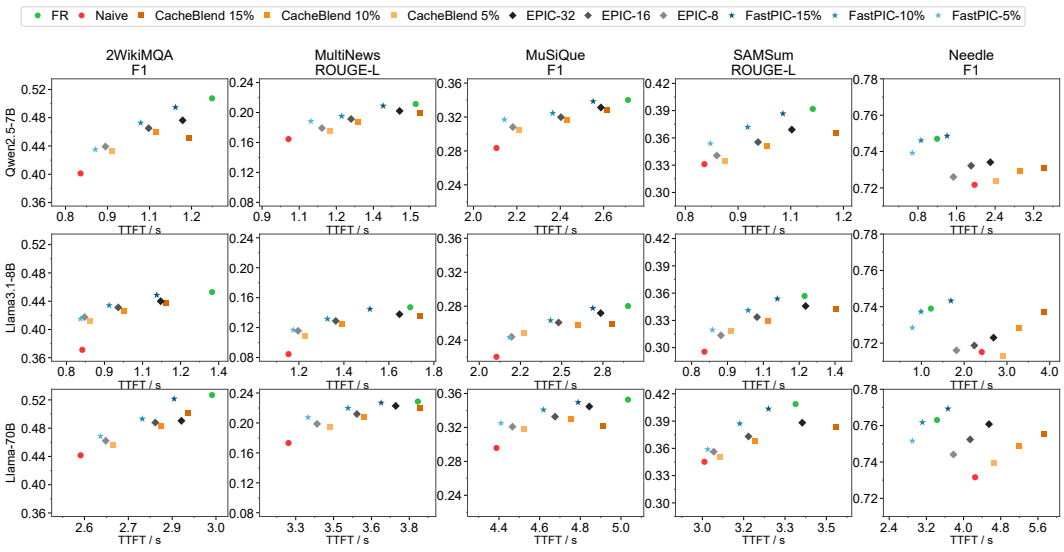


Figure 4: Accuracy–TTFT Trade-offs of FastPIC and Baselines across Models and Tasks.

In the asynchronous setting, we use real instructions from *ShareGPT* and sample request arrivals from a Poisson process while varying the arrival rate to emulate different multi-tenant concurrency levels. Prefix caching and other context-caching mechanisms are disabled.

5.1 Accuracy-Latency Trade-off of FastPIC

Figure 4 presents the accuracy–latency trade-off under the synchronous workload. FastPIC generally lies closer to the upper-left region, approaching FR's quality with substantially lower TTFT. On *2WikiMQA* with Qwen2.5-7B, for example, FastPIC-15 achieves an F1 score of approximately 0.50 versus 0.51 for FR, while reducing TTFT by about 10%. FastPIC benefits from better coordination between KV reuse and selective recomputation. *Naive* performs no recomputation and therefore suffers the largest quality degradation; on *SAMSum*, even FastPIC-5 improves ROUGE-L by approximately 3 percentage points. *EPIC* recomputes fixed chunk-prefix tokens and may miss request-specific attention sinks, whereas *CacheBlend* targets high-deviation tokens without explicitly correcting sink-induced attention routing. Consequently, on *MuSiQue*, both *CacheBlend*-15 and *EPIC*-32 underperform FastPIC-15 despite larger recomputation budgets. By jointly recomputing attention sinks and high-deviation tokens, FastPIC corrects both attention routing and critical KV representations, more reliably recovering the quality loss from position-independent KV reuse.

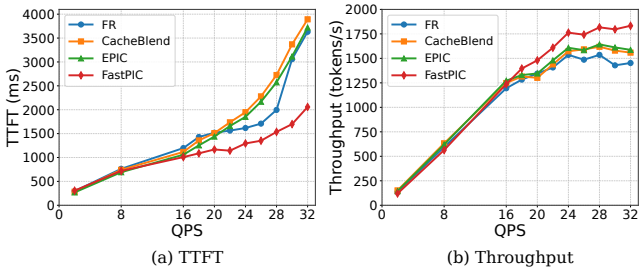
Figure 4 also reveals the trade-off between cache benefits and management overhead. On *Needle in a Haystack*, EPIC and CacheBlend incur substantially higher TTFT; on Llama-70B, FastPIC-15 achieves a F1 score above 0.76 at approximately 3.7 s, whereas CacheBlend-15 requires about 5.6 s and achieves lower quality. When fixed chunking yields few effective cache hits, LMCache lookup, insertion, and selective recomputation can offset the benefits of reuse, causing some EPIC and CacheBlend configurations on *MultiNews* and *SAMSum* to approach or exceed FR in TTFT. Llama-70B incurs approximately $2.5\times$ the TTFT of smaller models because of its greater depth and the inter-GPU communication introduced by PP/TP parallelism. FastPIC nevertheless remains advantageous, indicating that the cache-reuse benefits of TDC extend to larger models.

5.2 Latency and Throughput of FastPIC

To evaluate FastPIC in multi-tenant serving, we use real instructions from *ShareGPT* to construct asynchronous workloads by varying request arrival rates in vLLM. On Llama3.1-8B, we compare FastPIC with FR, CacheBlend-15%, and EPIC-32 under different QPS levels and measure TTFT and input throughput. Figure 5(a) shows that TTFT increases with QPS for all methods. Under FCFS-based continuous batching, higher arrival rates lengthen the waiting queue and increase contention between prefill and decode requests for the batching budget. At low concurrency, the methods

exhibit similar TTFT, but FastPIC gains a clear advantage beyond QPS=16. At QPS=32, FastPIC achieves a TTFT of approximately 2.0 s, corresponding to only 0.56×, 0.55×, and 0.53× the TTFT of FR, EPIC, and CacheBlend, respectively. This advantage stems from Token-Defined Chunking, which mitigates boundary shifts, improves cross-request chunk hit rates, and reduces prefill and cache-management overhead under high load.

Figure 5(b) shows that throughput follows the arrival rate at low QPS, yielding similar performance across methods. As QPS increases, computation and KV-cache capacity become bottlenecks, causing throughput to saturate. At QPS=32, FastPIC reaches approximately 1.84K tokens/s, achieving 1.27×, 1.18×, and 1.16× the throughput of FR, CacheBlend, and EPIC, respectively. Although EPIC uses a smaller recomputation budget, request-level PIC performance is primarily determined by the cache hit rate. Fixed-length chunking misses boundary-shifted content, so the lower recomputation overhead of EPIC and CacheBlend does not translate into end-to-end gains.



(a) TTFT (b) Throughput

Figure 5: TTFT and throughput of FastPIC and baselines across different QPS levels.

5.3 FastPIC's Performance on Long Context

We measure TTFT across context lengths on the *Needle in a Haystack* dataset with Llama-3.1-8B under the synchronous workload. CacheBlend-15% and EPIC-32 use fixed-size chunks of 256 tokens, while FastPIC uses Token-Defined Chunking with $B_{\text{norm}} = 256$; FR is an uncached baseline that performs full prefill. As shown in Figure 6, the TTFT of FR, CacheBlend-15%, and EPIC-32 grows approximately quadratically with context length, whereas FastPIC exhibits near-linear growth. FR processes the full context for every request, while boundary shifts reduce the effective cache-hit rate of the fixed-chunking baselines and erode their reuse benefits, driving their overhead toward that of full prefill.

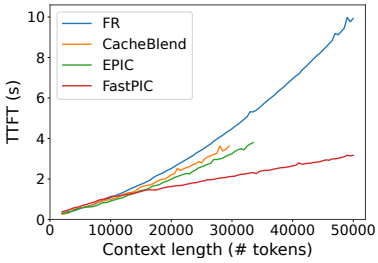


Figure 6: Long-context TTFT scaling of FastPIC and baseline methods.

In terms of GPU-memory scalability, CacheBlend-15% and EPIC-32 encounter out-of-memory errors at approximately 29K and 34K tokens, respectively, whereas FastPIC supports longer contexts. CacheBlend-15% recomputes a larger token fraction, incurring greater runtime intermediate-tensor and memory overhead, while fixed-length chunking introduces redundant KV storage and further pressures the context-cache pool. By aligning repeated content across prefixes through content-driven boundaries and stable chunk identities, FastPIC shares KV representations and limits recomputation overhead, enabling near-linear TTFT scaling and longer-context support.

6 Conclusion

This paper presents FastPIC to improve the efficiency and reliability of position-independent KV-cache reuse in multi-tenant LLM serving. We identify two key challenges in existing PIC methods: how to make chunking robust to prefix variations so that identical token spans across requests produce consistent boundaries and cache identities; and how to recover the generation accuracy degraded by position-independent KV reuse under a limited recomputation budget. FastPIC mitigates boundary shifts through Token-Defined Chunking and uses Dual-Path Token Selection to jointly recompute attention sinks and high-KV-deviation tokens, thereby releasing misallocated attention and correcting critical representations. Experimental results show that FastPIC preserves near-FR generation quality while, at QPS=32, delivering up to a 1.95× TTFT speedup and 1.2× higher input throughput than state-of-the-art PIC baselines. It also maintains lower TTFT at context lengths up to 1.72× longer, enabling efficient cross-request position-independent KV-cache reuse for multi-tenant long-context LLM serving.

ETHICS STATEMENT

This work studies position-independent KV-cache reuse in multi-tenant LLM serving to reduce the computational and resource costs of long-context inference. Because KV states may contain tenant-specific contextual information, cross-request reuse must comply with tenant-isolation and access-control requirements and incorporate cache-namespace isolation and secure eviction to mitigate potential privacy leakage. Our experiments use only publicly available research datasets and public user instructions; no new human-subject data or personally identifiable information was collected.

REPRODUCIBILITY STATEMENT

The Experimental Setup section specifies the FastPIC implementation framework, model and hardware configurations, datasets, data-processing pipeline, workload construction, baselines, key parameters, and evaluation metrics. The experiments use public datasets, publicly available models, and vLLM 0.4.1. We also report the principal configurations for Token-Defined Chunking, Dual-Path Token Selection, and all comparative experiments to support independent verification and reproduction.

AI USE STATEMENT

Generative AI tools were used to assist with code prototyping and debugging, experimental-result organization, manuscript structuring, technical editing, and language polishing. OpenAI GPT-5.5 was also used to review the clarity of the prose and the consistency of notation. The authors designed FastPIC, conducted the experiments, analyzed the data, developed the theoretical derivations, and verified the final conclusions. The authors take responsibility for the manuscript and all reported results.

REFERENCES

Fu Bang. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pp. 212–218, 2023.

Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 1074–1084, 2019.

Yunfan Gao, Yun Xiong, Wenlong Wu, Bohan Li, Yijie Zhong, and Haofen Wang. U-niah: Unified rag and llm evaluation for long context needle-in-a-haystack. *ACM Transactions on Information Systems*, 44(3):1–30, 2026.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. In *International Conference on Learning Representations*, volume 2024, pp. 22975–22988, 2024.

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

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

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Dan Zhang Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. others. 2024. chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, pp. 1763–1769, 2024.

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

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. Epic: Efficient position-independent caching for serving large language models. *arXiv preprint arXiv:2410.15332*, 2024.

Junhao Hu, Jiang Xu, Zhixia Liu, Yulong He, Yuetao Chen, Hao Xu, Jiang Liu, Jie Meng, Baoquan Zhang, Shining Wan, et al. {DEEPSERVE}: Serverless large language model serving at scale. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 57–72, 2025.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 44(1):1–27, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024.

Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pp. 174–187, 2001.

Zaifeng Pan, AJJKUMAR DAHYALAL PATEL, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows. *Advances in Neural Information Processing Systems*, 38: 126246–126265, 2026.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a {KVCache-centric} architecture for serving {LLM} chatbot. In *23rd USENIX conference on file and storage technologies (FAST 25)*, pp. 155–170, 2025.

Michael O Rabin. Fingerprinting by random polynomials. *Technical report*, 1981.

ShareGPT Team. ShareGPT. `https://sharegpt.com`, 2023.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. Slora: Scalable serving of thousands of lora adapters. *Proceedings of machine learning and systems*, 6:296–311, 2024.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Qwen Team et al. Qwq-32b: Embracing the power of reinforcement learning, 2025.

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

Qian Wang, Zahra Yousefijamarani, Morgan Lindsay Heisler, Rongzhi Gu, Bai Xiaolong, Shan Yizhou, Wei Zhang, Wang Lan, Ying Xiong, Yong Zhang, and Zhenan Fan. Mepic: Memory efficient position independent caching for llm serving, 2025. URL `https://arxiv.org/abs/2512.16822`.

Zhixiong Xie, Wenlong Tian, Ruixuan Li, Jianfeng Lu, Weijun Xiao, and Zhiyong Xu. Horseminhash: high-performance and secure jaccard similarity estimation for cloud storage. *IEEE Transactions on Dependable and Secure Computing*, 22(6):6685–6697, 2025.

Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. Kvshare: An llm service system with efficient and effective multi-tenant kv cache reuse. *arXiv preprint arXiv:2503.16525*, 2025.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. Kvlink: Accelerating large language models via efficient kv cache reuse. *Advances in Neural Information Processing Systems*, 38:133797–133824, 2026.

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

Hirotaka Yoshida and Alex Biryukov. Analysis of a sha-256 variant. In *International Workshop on Selected Areas in Cryptography*, pp. 245–260. Springer, 2005.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 193–210, 2024.

Yuechi Zhou, Yi Su, Jianxin Zhang, Juntao Li, Qingrong Xia, Zhefeng Wang, Xinyu Duan, and Baoxing Huai. $a\tilde{3}$: Attention-aware accurate kv cache fusion for fast large language model serving. *arXiv preprint arXiv:2511.17560*, 2025.