FAST AND QUALITY-PRESERVING NON-PREFIX KV CACHE REUSE FOR LARGE LANGUAGE MODEL SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Key-Value (KV) Cache reuse reduces prefill computation for long-context large language models (LLMs), but reusing independently encoded text chunks in non-prefix contexts degrades generation quality because cross-chunk attention is missing. Selective recomputation can mitigate this problem, but it requires both selecting effective tokens for recomputation and preventing the costs of recomputation and cache loading from offsetting the benefits of reuse. We observe that the frequency-domain structure of KV provides an effective signal for identifying tokens that help restore cross-chunk dependencies. Based on this observation, we propose CacheTune, a frequency-guided method for non-prefix KV Cache reuse. CacheTune analyzes independently encoded KV Chunks offline to generate recomputation indices shared across layers, recomputes the selected KVs in the new context, and reuses the remaining cache, without additional training or online importance analysis. The same indices also determine which KVs must be loaded, so the volume of cache transfers decreases as the recomputation ratio increases, while allowing transfer to overlap with computation. CacheTune further calibrates the recomputation ratio based on hardware computation costs and cache access costs to accommodate different storage tiers. Across mainstream models and long-context tasks, CacheTune achieves a 3.72–4.86$\times$ speedup in time-to-first-token (TTFT) and 3.93–6.21$\times$ the throughput of full recomputation while maintaining comparable generation quality. With SSD/HDD caches, the TTFT speedup still reaches 2.34–2.36$\times$. The code is available at `https://anonymous.4open.science/r/CacheTune-2123/`.

1 INTRODUCTION

Large language model (LLM) serving is gradually shifting from short conversations to long-context applications such as retrieval-augmented generation (RAG) (Lewis et al., 2020), long-document question answering (Bai et al., 2024), and agent reasoning (Yao et al., 2023). In these scenarios, input prompts are typically assembled dynamically from retrieved documents, conversation histories, or reusable knowledge segments. Prefill must process the entire context, and its computational cost grows rapidly with input length. The resulting time-to-first-token (TTFT) has become a major performance bottleneck in long-context serving (Gim et al., 2024; Yao et al., 2025). The same document or knowledge segment may appear in multiple requests. Encoding them anew each time also repeats the corresponding Key-Value (KV) generation, further increasing the prefill workload.

A natural way to eliminate redundant prefill computation is to materialize the KV Cache generated from reusable text segments as independently cacheable units, hereafter called KV Chunks, and reuse them across requests (Ye et al., 2024; Liu et al., 2024). Prefix caching follows this approach and has been adopted in systems such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024). However, it requires the KV Chunks to form a strict prefix of the new request. In RAG and multi-document workloads, reusable content typically consists of independent text chunks whose number, order, and positions change dynamically, preventing prefix caching from fully exploiting these repeated data.

Allowing independent KV Chunks to be reused at arbitrary positions can improve cache hit rates, but substantially degrades generation quality. The fundamental reason is that a Transformer's causal self-

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.

attention mechanism makes a token's KV depend not only on its own content but also on all preceding context. Text chunks encoded independently in advance lack the cross-chunk attention that should have been established in the complete prompt. Their KV therefore represents a context-mismatched materialized state relative to the current global context. We refer to this inter-chunk attention as **cross-attention**. Directly concatenating these chunk caches causes the resulting discrepancies to propagate into subsequent attention computations and the final generated output. Non-prefix KV Cache reuse must therefore repair the invalidated contextual dependencies (Hu et al., 2025).

Selecting important KV for recomputation is a key approach to restoring cross-attention. Existing recomputation methods organize cross-attention repair based on online KV deviations, chunk boundary positions, or learned link tokens (Yao et al., 2025; Hu et al., 2025; Yang et al., 2025b). These methods respectively rely on online analysis, fixed positional priors, or additional training. We consider another question: can effective selection signals be extracted solely from an independent chunk's KV, so that the KV indices to be recomputed can be determined when the cache is generated and reused across requests? We observe that KV energy along the token dimension is concentrated in low-frequency components. After the low-frequency signal is reconstructed at the original positions, tokens selected by their magnitude contribute more to restoring attention distributions and task quality under the same recomputation budget. This observation provides a basis for an offline token importance ranking independent of the current query. In addition, recomputation index selection must account for cache access costs. As reusable KV Caches exceed GPU memory capacity and expand to lower storage tiers or even external caching systems (Lin et al., 2026; Gao et al., 2024), cache transfer and online recomputation become two competing latency paths. The computational savings from KV reuse may be offset by additional cache-loading overhead (Qin et al., 2026; Liu et al., 2025). Efficient reuse therefore requires dynamically balancing cache transfer and online recomputation.

Based on these observations, we propose CacheTune, which manages the repair, transfer, and reuse of non-prefix KV Caches across storage tiers while preserving generation quality. During cache generation, CacheTune performs a one-time offline frequency-domain analysis of each independently encoded KV Chunk to identify the tokens most critical to global semantic recovery, enabling targeted repair of cross-chunk attention. To hide cross-tier I/O overhead, CacheTune uses an asynchronous pipeline with multiple CUDA streams to overlap sparse KV transfer, online recomputation, and deferred positional-encoding recovery. Building on this pipeline, a hardware-aware search for the recomputation ratio adaptively balances computation and data movement across heterogeneous storage tiers. CacheTune thus enables high-quality non-prefix KV reuse with accuracy close to full recomputation and substantially reduces TTFT across different storage configurations. Our contributions are summarized as follows:

- We propose a frequency-guided method for KV importance modeling and selective recomputation in non-prefix KV Cache reuse. Using the intrinsic frequency-domain structure of independently encoded KV Chunks, the method identifies offline the KV indices most critical to restoring cross-chunk semantic dependencies, effectively recovering the inter-chunk cross-attention missing in non-prefix reuse at a low recomputation cost.
- We propose an algorithm–system co-design framework that translates frequency-guided indices into a complementary recomputation and reuse plan. CacheTune transfers only the KV that does not require recomputation, asynchronously overlaps sparse KV transfer, selective recomputation, and deferred positional-encoding recovery, and further adjusts the recomputation ratio to the computation and I/O characteristics of the target cache tier. By coordinating data movement across cache tiers with recomputation on the compute device, CacheTune reduces TTFT in multi-tier caching environments.
- We evaluate CacheTune across multiple models, long-context tasks, and KV Cache storage tiers. While maintaining generation quality close to full recomputation, CacheTune achieves a 3.72–$4.86\times$ TTFT speedup and 3.93–$6.21\times$ the serving throughput, offering a better quality–latency trade-off than state-of-the-art baselines. With I/O-bound SSD and HDD cache pools, CacheTune still achieves a 2.34–$2.36\times$ TTFT speedup.

2 Related Work

Prefix caching systems, including vLLM, SGLang, and ChunkAttention, reuse only matching prefixes (Kwon et al., 2023; Zheng et al., 2024; Ye et al., 2024). Non-prefix methods relax this constraint

through modular caching or cross-chunk repair. PromptCache uses predefined prompt schemas (Gim et al., 2024); CacheBlend selects tokens with large KV deviations through online first-layer recomputation (Yao et al., 2025); EPIC repairs fixed attention-sink positions (Hu et al., 2025); and KVLink introduces learned link tokens (Yang et al., 2025b). CacheSlide addresses positional drift when reusable segments preserve their relative order (Liu et al., 2026b). CacheClip (Yang et al., 2025a), Cache-Craft (Agarwal et al., 2025), and ProphetKV (Wang et al., 2026) respectively use auxiliary models, historical information, or query-dependent signals.

In contrast, CacheTune constructs query-independent recomputation indices offline directly from each independent KV Chunk and uses them to guide KV transfer. Unlike FreqKV, which uses the frequency domain to compress KV storage (Kai et al., 2026), CacheTune uses frequency-domain structure only to extract the positions of tokens important to restoring global cross-attention. Appendix A discusses broader related work and background, with an illustrative comparison in Figure 7.

3 MOTIVATION

Restoring cross-chunk dependencies. At layer ℓ, the KV of token i are derived from its hidden state, namely $k_i^\ell = h_i^\ell W_K^\ell$ and $v_i^\ell = h_i^\ell W_V^\ell$. This hidden state has already aggregated the preceding context through causal self-attention. Encoding a text chunk independently therefore omits its dependencies on preceding chunks, producing KVs different from those obtained by encoding the complete prompt.

We analyze KV Caches along the token dimension and observe that their energy is concentrated in low-frequency components. Appendix E.1 reports KV spectrum measurements for different models. This observation motivates us to use low-frequency structure to rank token positions for recomputation. Supplementary experiments in Appendix E.2 further show that low-frequency KV indices obtained through frequency-domain analysis provide high coverage of the KV positions that change most between independent chunk encoding and full-context encoding.

To test whether this ranking helps restore missing dependencies, we compare low-frequency- and high-frequency-guided selection under the same 15% recomputation budget, using full KV reuse and full recomputation as references. Full recomputation provides the exact full-context attention distribution. Figure 1(a,b) shows that low-frequency-guided recomputation more closely matches the attention distribution of full recomputation across different queries and numbers of chunks. Figure 1(c) further tests whether this improvement is reflected in the model's final accuracy. On MuSiQue, we fix the document chunk caches and indices while varying the query, chunk order, and length of the preceding context. Across all four context settings, low-frequency selection achieves higher F1 than full reuse, high-frequency selection, and random selection, approaching full recomputation. These results show that indices generated from chunk content can be reused across queries and context layouts without being selected again when a request arrives.

Balancing computation and transfer. Reusable KV Chunks may reside in CPU memory or on SSD/HDD. Increasing the recomputation ratio increases GPU workload but reduces the KV that must be loaded and allows more GPU computation to overlap with cache loading. In Appendix E.3, we measure overall latency when different recomputation ratios are selected for KV Chunks across storage tiers. The results show that a higher reuse ratio is more suitable when cache access is inexpensive on fast storage media, whereas slower media may benefit from more recomputation. The selection of KV indices for recomputation must therefore be combined with complementary cache transfer, and the balance between them must be adjusted for the target storage tier.

4 METHODOLOGY

CacheTune operates in two stages, as shown in Figure 2. Offline frequency analysis ranks token positions in each independently encoded KV Chunk and determines the indices of positions that require recomputation to restore cross-attention. During online inference, KV at the selected positions is recomputed in the assembled context, while the complementary KV is transferred from the cache pool and merged with the recomputed results. The execution pipeline coordinates sparse transfer, selective recomputation, and positional recovery. A hardware-aware scheduler adjusts the recomputation ratio to balance these operations and reduce overall TTFT.

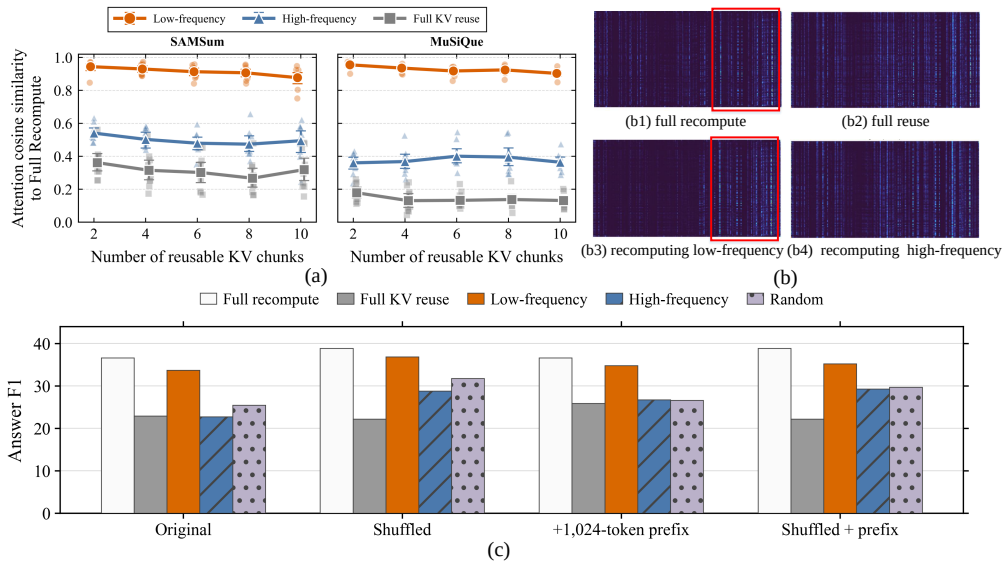

Figure 1: Reusing low-frequency indices across contexts. (a) Attention similarity across different queries and numbers of chunks; curves and error bars show means and 95% confidence intervals. (b) Representative attention distributions. (c) Mean F1 on MuSiQue with fixed caches and indices. All three selection strategies recompute 15% of chunk tokens, and the added preceding context contains 1,024 tokens. The model is Mistral v0.3-7B.

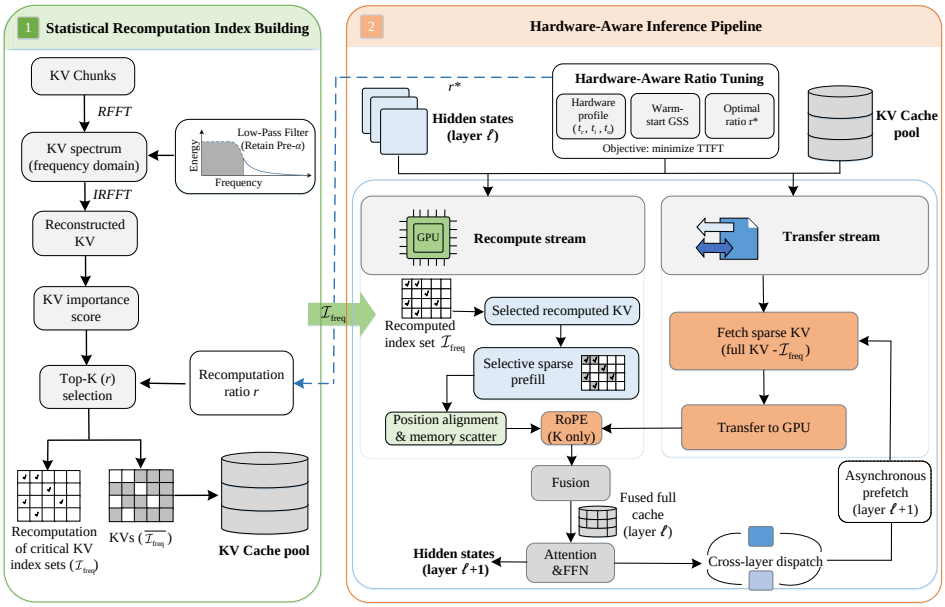

Figure 2: Overall architecture of CacheTune.

4.1 Frequency-Domain Selective Recomputation

During offline cache generation, CacheTune constructs a token-importance ranking for each independently encoded KV Chunk. When the chunk appears in a new context, this ranking is used to select the KV indices that need to be recomputed to restore cross-attention. The frequency-domain observations in Section 3 provide the basis for this ranking. The procedure comprises spectral reconstruction, token scoring, and cross-layer selection.

Recovering low-frequency structure aligned with tokens. For layer ℓ, let $\mathbf{K}^{(\ell)}, \mathbf{V}^{(\ell)} \in \mathbb{R}^{N\times H\times D}$ denote the independently encoded Key and Value tensors, where N is the chunk length, H is the

number of KV heads, and D is the head dimension. We apply a real fast Fourier transform along the token axis separately for each head and feature channel (Williamson, 2012):

$$\widehat{\mathbf{K}}^{(\ell)} = \mathrm{rfft}(\mathbf{K}^{(\ell)}, \mathrm{dim} = 0), \qquad \widehat{\mathbf{V}}^{(\ell)} = \mathrm{rfft}(\mathbf{V}^{(\ell)}, \mathrm{dim} = 0). \tag{1}$$

Both spectra belong to $\mathbb{C}^{F\times H\times D}$, where $F = \lfloor N/2 \rfloor + 1$ contains the nonredundant frequencies of the real-valued sequence. Frequency indices describe variation across token positions; they are not themselves token indices.

We retain the lowest α fraction of frequencies and reconstruct a signal aligned with the original N positions. Let $\mathbf{X} \in \{\mathbf{K}, \mathbf{V}\}$. The same operation applies to both Key and Value:

$$c = \lfloor \alpha F \rfloor, \qquad \widehat{\mathbf{X}}^{(\ell)}_{\mathrm{LP}}[k,:,:] = \begin{cases} \widehat{\mathbf{X}}^{(\ell)}[k,:,:], & k < c, \\ \mathbf{0}, & k \geq c, \end{cases}$$
$$\widetilde{\mathbf{X}}^{(\ell)} = \mathrm{irfft}(\widehat{\mathbf{X}}^{(\ell)}_{\mathrm{LP}}, n = N, \mathrm{dim} = 0). \tag{2}$$

Thus, the shapes of $\widetilde{\mathbf{K}}^{(\ell)}$ and $\widetilde{\mathbf{V}}^{(\ell)}$ return to $N \times H \times D$. This inverse transform allows us to assign importance scores to individual token positions based on the retained spectral structure. The cutoff ratio α controls which frequency components contribute to scoring, whereas the recomputation ratio r controls how many token positions are updated. The former controls the basis for index selection, and the latter controls the online recomputation budget.

Scoring and selecting positions shared across layers. For token i, we compute the ℓ_2 norm over its $H \times D$ elements to measure the magnitudes of the reconstructed Key and Value:

$$s^{(\ell)}_{i,K} = \|\widetilde{\mathbf{K}}^{(\ell)}_i\|_2, \qquad s^{(\ell)}_{i,V} = \|\widetilde{\mathbf{V}}^{(\ell)}_i\|_2,$$
$$s^{(\ell)}_i = \tfrac{1}{2}\big(s^{(\ell)}_{i,K} + s^{(\ell)}_{i,V}\big). \tag{3}$$

The joint score gives equal weight to KV magnitudes. Because numerical scales may differ across layers, we first normalize the scores within each layer and then average them over all L layers:

$$\widetilde{s}^{(\ell)}_i = \frac{s^{(\ell)}_i}{\sum_{j=1}^{N} s^{(\ell)}_j + \epsilon}, \qquad \overline{s}_i = \frac{1}{L}\sum_{\ell=1}^{L} \widetilde{s}^{(\ell)}_i. \tag{4}$$

Here, ϵ prevents division by zero. For recomputation ratio r, the shared index set and its complement are:

$$\mathcal{I}_{\mathrm{freq}} = \mathrm{TopK}(\{\overline{s}_i\}_{i=1}^{N}, rN), \qquad \overline{\mathcal{I}}_{\mathrm{freq}} = \{1, \ldots, N\} \setminus \mathcal{I}_{\mathrm{freq}}. \tag{5}$$

Here, TopK returns the token positions with the highest aggregated scores. Sharing these positions across layers allows updated hidden states to propagate continuously through the Transformer. Independently selecting unrelated positions in each layer would not form the same continuous recomputation path. The ranking depends only on the independently encoded text chunk and can therefore be reused directly by subsequent requests. Changing r only changes the length of the prefix selected from this ranking; it does not require spectral analysis to be repeated.

Recomputing in the new context. During inference, local indices within each chunk are mapped to global positions in the assembled prompt. CacheTune propagates the hidden states at the selected positions forward layer by layer and regenerates their KV from the current layer's input. At each layer, this KV is merged at the original positions with the cached KV in the complement $\overline{\mathcal{I}}_{\mathrm{freq}}$. The selected queries therefore attend to the assembled preceding context under the causal mask, and their updated hidden states continue to the next layer. Both reused and recomputed Keys use the global positions described in Section 4.2.

4.2 KV Cache Offloading and Sparse Reuse

This section uses the offline indices obtained in Section 4.1 to optimize non-prefix KV Cache reuse in two ways: reducing cache transfer volume and overlapping computation with transfer. These optimizations reduce the latency overhead of external cache access.

CacheTune stores reusable KV Chunks in cache pools outside the GPU, including CPU memory and SSD/HDD. The shared index set $\mathcal{I}_{\mathrm{freq}}$ guides both recomputation and cache loading: the system

loads only the KV in the complement $\overline{\mathcal{I}}_{\text{freq}}$, reducing the number of KV entries transferred per chunk per layer from N to $(1-r)N$. Dedicated transfer and recomputation streams coordinate with the forward-computation stream through per-layer CUDA events, allowing asynchronous prefetching to overlap with computation. Reusable per-layer working buffers support scatter-based merging, placing transferred and recomputed KV back at their corresponding positions in the complete KV Cache.

To accommodate changes in chunk positions across requests, CacheTune stores Keys offline before rotary position embedding (RoPE) is applied. After reused Keys are transferred to the GPU, the recomputation stream applies RoPE according to their global positions in the assembled prompt; newly computed Keys use the same position mapping. Positional encoding is thus incorporated into the asynchronous pipeline and completed before attention uses the cache. Appendix B and Figure 8 further detail the dependencies between streams, buffer merging, and positional alignment.

4.3 Hardware-Aware Joint Scheduling of Recomputation and Transfer

The pipeline in Section 4.2 overlaps recomputation with cache transfer, but the time taken by each path varies with the GPU and cache medium. A fixed recomputation ratio may leave a long wait for transfer on slower KV storage paths. Subject to an empirical lower bound for quality preservation, this section therefore treats the recomputation ratio as a joint decision variable for computation and transfer, aiming to shorten the critical path of the entire prefill pipeline.

The recomputation ratio r trades GPU computational workload against cache transfer volume. Let t_c and t_i denote the per-token, per-layer computation cost and effective transfer cost obtained through profiling, respectively, and let t_o denote the fixed overhead per layer. For L layers and N reusable tokens, the workloads of the recomputation and loading paths are proportional to rN and $(1-r)N$, respectively. Approximating the main online prefill latency with a steady-state pipeline gives:

$$T_{\text{TTFT}}(r) \approx L \max(rNt_c, (1-r)Nt_i) + Lt_o, \qquad r_0 = \frac{t_i}{t_c + t_i}. \tag{6}$$

The maximum term corresponds to whichever of computation and transfer takes longer. Equating the times of the two paths yields the initial ratio r_0: the more expensive cache access is, the more beneficial it is to replace reads with recomputation. This approximation locates the initial search region; startup, synchronization, and pipeline bubbles in complete requests are accounted for by the measured objective below.

Selecting r solely to minimize latency may leave too little recomputation for fast cache configurations. To limit generation-quality degradation, we set the lower search bound to $r_{\min} = 15\%$. The quality–latency results in Appendix E.4 support this choice. We search over $r \in [r_{\min}, r_{\max}]$, where $r_{\max} \leq 1$, and minimize the measured mean TTFT on a small calibration set $\mathcal{S}_{\text{cal}}$:

$$f(r) = \frac{1}{|\mathcal{S}_{\text{cal}}|} \sum_{s \in \mathcal{S}_{\text{cal}}} \text{TTFT}(s; r), \qquad r^* = \underset{r \in [r_{\min}, r_{\max}]}{\arg\min} f(r). \tag{7}$$

CacheTune clips r_0 to the search interval and uses this value as the initial point for a warm-started golden-section search (Kochenderfer & Wheeler, 2019). Measuring TTFT in this way accounts for transfer startup, synchronization, and pipeline effects omitted from the theoretical cost model. The computation–transfer trade-off is approximately unimodal, providing a basis for this derivative-free search. Calibration is performed once when deploying to a given hardware/storage configuration, and the selected ratio can be used for subsequent requests. Appendix C provides the detailed derivation and search pseudocode.

5 Experimental Evaluation

5.1 Experimental Setup

We use Mistral v0.3-7B, Llama 3-8B, and Qwen 2.5-32B (Jiang et al., 2023; Llama Team, 2024; Qwen Team, 2024). The datasets comprise SAMSum and Multi-News for summarization, evaluated using ROUGE-L, and MuSiQue, WikiMQA, and HotpotQA for multi-hop question answering, evaluated using F1. Experiments run on two platforms, with 2×A100-40GB and 2×RTX 4090-24GB GPUs, respectively; both connect to the host through PCIe Gen3×16. Cache media include GPU memory, CPU memory, and HDD and SSD cache pools on the A100 and RTX 4090 platforms, respectively.

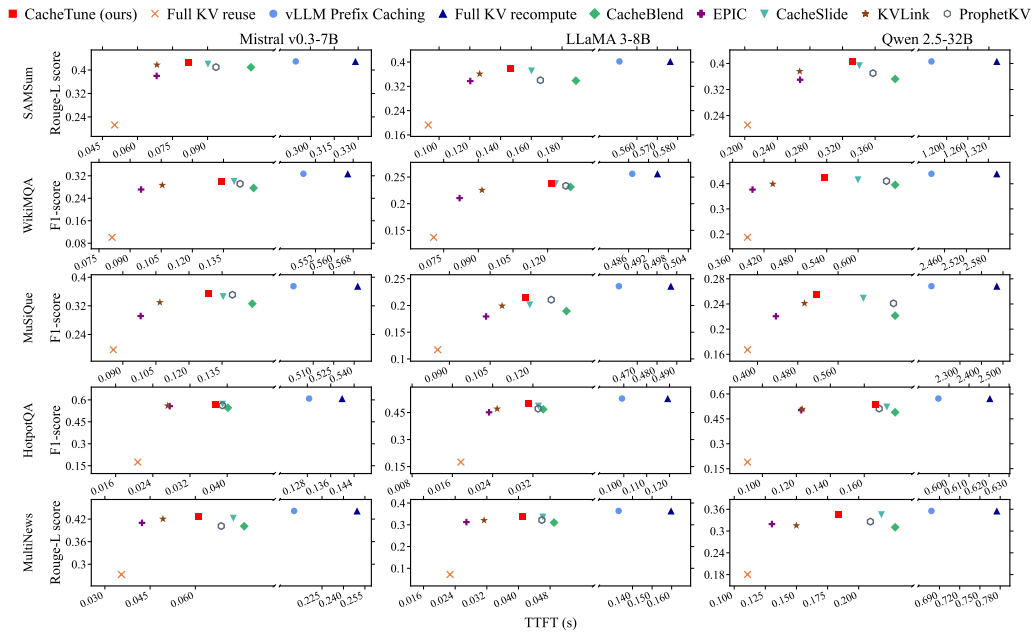

Figure 3: Accuracy–TTFT trade-off of CacheTune and the baselines across models and dataset tasks.

The baselines include Full Recompute, Full KV Reuse, vLLM Prefix Caching (Kwon et al., 2023), CacheBlend (Yao et al., 2025), EPIC (Hu et al., 2025), KVLink (Yang et al., 2025b), CacheSlide (Liu et al., 2026b), and ProphetKV (Wang et al., 2026). Full Recompute processes the entire prompt; Full KV Reuse concatenates independently encoded chunk caches; Prefix Caching reuses only matching prefixes. CacheBlend uses a recomputation ratio of 15% after selection in the first layer; EPIC recomputes the first 16 tokens of each chunk, and KVLink uses 5 link tokens per chunk. For CacheSlide, reusable documents are treated as fixed segments and queries as updated segments. ProphetKV aggregates query attention scores across layers and recomputes the top-scoring 15% of context tokens. In the main CPU-cache comparison, CacheTune uses $\alpha = 0.5$ and $r = 15\%$; the storage-tier experiments also evaluate calibrated ratios. Appendix D provides detailed baseline descriptions, dataset construction, and software versions.

5.2 Generation Quality–TTFT Trade-off

We compare CacheTune with the baselines on generation quality and TTFT. Methods such as EPIC, KVLink, and ProphetKV are not optimized for multi-tier KV Cache storage, so we keep all their reusable KV Chunks in GPU memory, the fastest-access tier, representing their best-case TTFT. CacheTune offloads reusable KV Chunks to CPU memory using the optimized pipeline in Section 4.2. With fast CPU-cache access, the hardware-aware scheduler selects the quality-preserving lower bound, $r = r_{\min} = 15\%$. We set the low-frequency cutoff ratio to $\alpha = 0.5$; Appendix E.4 evaluates sensitivity to both r and α.

Figure 3 shows the overall trade-off. Full Recompute provides the accuracy upper bound but incurs the highest TTFT; Full KV Reuse has the lowest TTFT but severely reduces accuracy because cross-attention is missing. vLLM Prefix Caching preserves Full Recompute accuracy by losslessly reusing matching prefixes. However, many non-prefix chunks still require recomputation, resulting in higher TTFT than CacheTune. CacheTune achieves accuracy close to Full Recompute, preserving an average of 94.99% of its generation quality across five datasets and three models while delivering 3.72–4.86× TTFT speedup. At the same recomputation ratio of 15%, CacheTune achieves both higher accuracy and lower TTFT than CacheBlend and ProphetKV. Its overall accuracy–TTFT trade-off also surpasses CacheSlide. CacheSlide primarily targets RPDC-style agent prompts, where reusable segments retain their relative order and positional drift is the main concern. CacheTune supports more general non-prefix reuse of independently encoded chunks with varying orders and positions. Together, these

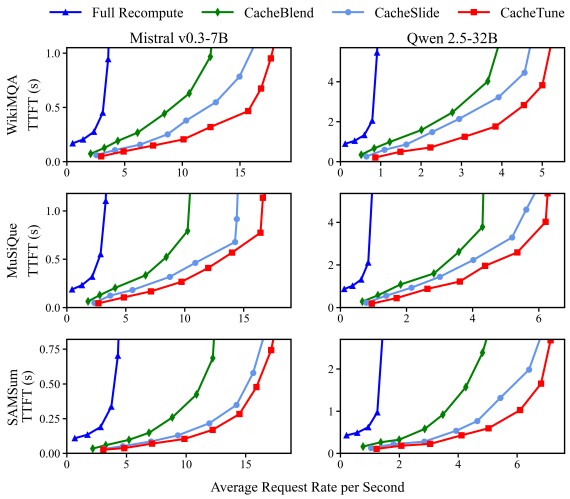

Figure 4: TTFT as the request rate increases. Curves extending farther to the right at low TTFT indicate higher effective throughput under low-latency serving conditions.

Table 1: TTFT (s) comparison between GPU-resident reuse and the optimized CPU offloading pipeline.

Method	SAMSum	WikiMQA	MuSiQue
GPU			
Full Recompute	0.3283	0.5657	0.5426
Prefix Caching	0.2908	0.5469	0.4955
CacheBlend	0.1086	0.1495	0.1487
CacheSlide	0.0902	0.1401	0.1355
CacheTune	0.0899	0.1485	0.1398
CPU			
Full Recompute	0.3283	0.5657	0.5426
Prefix Caching	0.3012	0.5496	0.5071
CacheBlend	0.1113	0.1768	0.1723
CacheSlide	0.0995	0.1404	0.1492
CacheTune	0.0817	0.1342	0.1289

results show that CacheTune's frequency-domain selection restores cross-attention more effectively, while its sparse offloading and prefetching pipeline hides most transfer overhead.

CacheBlend and CacheSlide identify recomputation KVs through online first-layer recomputation, and ProphetKV selects recomputation targets by computing and aggregating query-to-context attention scores across layers online. CacheTune instead obtains recomputation indices offline during KV Chunk generation, avoiding online token-selection overhead. These indices also allow CacheTune to skip loading KV pairs that will be recomputed. Very fast methods such as EPIC and KVLink achieve lower TTFT but suffer noticeable accuracy degradation. EPIC updates only a few fixed positions at the beginning of each KV Chunk, which may miss semantically important tokens in other regions and lead to accuracy loss across datasets. KVLink relies on pretrained link tokens, introducing additional offline computation. In contrast, CacheTune's frequency-domain analysis is lightweight: generating the recomputation index set for a 3K-token KV Chunk takes only about 0.69 ms on average on a GPU, and the indices can be reused across inference tasks.

5.3 Inference Serving Throughput

As shown in Figure 4, we compare the TTFT of different methods at varying request rates. Full Recompute saturates at relatively low request rates, causing TTFT to rise rapidly. CacheBlend and CacheSlide improve throughput, but their latency still increases considerably under heavy load. CacheTune maintains low TTFT while shifting saturation to higher request rates, thereby achieving higher throughput. Relative to Full Recompute, CacheTune achieves 3.93–6.21$\times$ throughput, compared with 2.71–5.16$\times$ for CacheBlend and 3.79–5.94$\times$ for CacheSlide. This shows that CacheTune reduces per-request prefill latency and improves effective serving capacity across load levels.

5.4 Effectiveness of Frequency-Guided Selection

To assess whether frequency-domain importance scores identify the KV pairs most critical to restoring global cross-attention, we compare five strategies at a fixed 15% token-selection ratio: random selection, L2-norm selection, high-frequency-guided selection, LF-only prefill, and CacheTune. LF-only prefill retains the top 15% of tokens ranked by low-frequency importance and discards the rest before prefill. L2-norm selection uses the same joint K/V norm and cross-layer aggregation as in Section 4.1, but omits frequency-domain analysis. As shown in Figure 5, random, L2-norm, and high-frequency-guided selection recover only a limited portion of the quality lost through non-prefix KV reuse and do not effectively restore cross-chunk semantic dependencies. LF-only prefill performs relatively well on SAMSum, where summarization mainly requires understanding

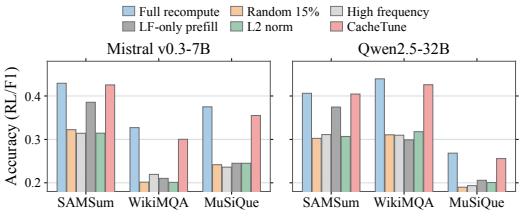

Figure 5: Effectiveness of different strategies for selecting tokens for recomputation.

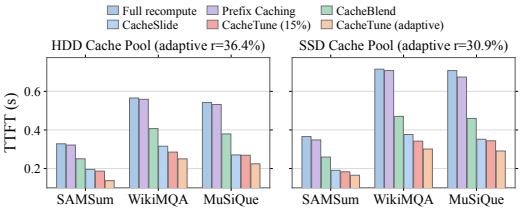

Figure 6: TTFT with different external KV Cache media and recomputation strategies.

the overall contextual semantics, suggesting that low-frequency analysis can identify tokens with strong semantic representativeness. However, its performance drops markedly on WikiMQA and MuSiQue, which require fine-grained evidence and multi-hop reasoning, and remains below that of CacheTune across all settings in Figure 5. These results show that tokens selected by low-frequency analysis help restore cross-chunk attention but cannot alone form a complete context meeting task requirements. The discarded tokens still contain important factual details, local semantics, and intermediate reasoning evidence. CacheTune consistently produces results closest to Full Recompute, showing that recomputing the identified critical KV pairs in the global context while reusing the remaining KV pairs helps restore cross-chunk dependencies without discarding the rest of the context.

5.5 Adaptability Across Cache Tiers

GPU and CPU caches. This experiment examines whether the external-cache pipeline preserves latency gains after moving the cache out of GPU memory. Table 1 compares Mistral with GPU-resident and CPU-resident caches, with CacheTune's recomputation ratio fixed at 15%. Within each comparison, CacheBlend, CacheSlide, and Prefix Caching use the same cache medium. With CPU-resident caches, CacheTune reduces average TTFT by 25.3%, 11.9%, and 74.4% relative to these three baselines, respectively. Across the three tasks, its CPU-cache TTFT is comparable to or lower than that of its own GPU-resident reuse path. This is because the CPU configuration enables the pipeline that overlaps transfer and recomputation, while sparse loading and deferred RoPE further reduce GPU-side data processing and computation. These end-to-end measurements show that the optimized CPU-cache path preserves the benefits of reuse despite the additional data movement.

SSD/HDD caches. Using Mistral, we further compare TTFT with a fixed $r = 15\%$ and with hardware-aware calibration on HDD and SSD configurations to examine the effect of adjusting the recomputation budget. In Figure 6, the fixed $r = 15\%$ achieves 1.92× and 2.05× TTFT speedup over Full Recompute on the HDD and SSD configurations, respectively. Calibration selects $r = 36.4\%$ for the HDD and $r = 30.9\%$ for the SSD, increasing the respective speedups to 2.36× and 2.34×. More recomputation reduces costly cache reads, further lowering end-to-end latency. These ratios are calibrated for latency using the first 10 SAMSum samples. Calibration takes approximately 3–4 minutes for each hardware and storage configuration, after which the ratios are reused across datasets without recalibration. This calibration estimates the balance between computation and I/O, while the lower bound on r prevents severe generation-quality degradation from insufficient recomputation. Appendix E.5 provides further analysis of these results and quantifies the latency benefits of the cache-loading optimizations introduced in Section 4.2 through cumulative ablation.

Appendix F further discusses the experimental results and the method's limitations.

6 Conclusion

We present CacheTune to address cross-chunk attention loss and cross-tier I/O bottlenecks in non-prefix KV Cache reuse for long-context LLM serving. CacheTune uses the low-frequency structure of independently encoded KV Chunks to identify the KV pairs most critical to restoring global semantics. It coordinates sparse transfer, selective recomputation, deferred RoPE restoration, and hardware-aware scheduling across GPU, CPU, and SSD/HDD cache tiers. Across multiple models and long-context tasks, CacheTune preserves generation quality close to Full Recompute while achieving 3.72–4.86× TTFT speedup and 3.93–6.21× inference serving throughput.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing, LaTeX typesetting, and coding assistance. The core research ideas and scientific contributions were developed by the authors themselves. We have thoroughly reviewed all AI-assisted work. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

An anonymous link to the source code is provided in the abstract. Section 4 describes the method, while Appendices B and C detail the implementation and hardware-aware calibration procedure. Appendix D describes the experimental setup, datasets, and baselines. Additional experiments and ablation studies are presented in Appendix E.

REFERENCES

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. Cache-Craft: Managing chunk-caches for efficient retrieval-augmented generation. *Proceedings of the ACM on Management of Data*, 3(3): 1–28, 2025. doi: 10.1145/3725273. URL https://doi.org/10.1145/3725273.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jing Liu, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Bailu Ding, Xiao Yan, Jiawei Jiang, Chen Chen, Mingxing Zhang, Cheng Li, Yuqing Yang, Fan Yang, and Mao Yang. RetroInfer: A vector storage engine for scalable long-context LLM inference. *Proceedings of the VLDB Endowment*, 19(5):1016–1031, May 2026. ISSN 2150-8097. doi: 10.14778/3796195.3796212. URL https://doi.org/10.14778/3796195.3796212.

Debendra Das Sharma, Robert Blankenship, and Daniel Berger. An introduction to the Compute Express Link (CXL) interconnect. *ACM Computing Surveys*, 56(11):1–37, July 2024. ISSN 0360-0300. doi: 10.1145/3669900. URL https://doi.org/10.1145/3669900.

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. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1102. URL https://aclanthology.org/P19-1102/.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-Efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 111–126. USENIX Association, 2024. URL https://www.usenix.org/conference/atc24/presentation/gao-bin-cost.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt Cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 325–338, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

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. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-5409. URL https://aclanthology.org/D19-5409/.

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. International Committee on Computational Linguistics, 2020. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

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. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24391–24402. PMLR, 2025. URL https://proceedings.mlr.press/v267/hu25j.html.

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. Mistral 7B. *CoRR*, abs/2310.06825, 2023. doi: 10.48550/ARXIV.2310.06825. URL https://doi.org/10.48550/arXiv.2310.06825.

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. doi: 10.1145/3768628. URL https://doi.org/10.1145/3768628.

Jushi Kai, Yixuan Wang, Boyi Zeng, Haoli Bai, Bo Jiang, Ziwei He, and Zhouhan Lin. FreqKV: Key-value compression in frequency domain for context window extension. In *International Conference on Learning Representations*, volume 2026, pp. 105568–105587, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ac4c13a5c6ec1bba68c9ce1d908321d2-Abstract-Conference.html.

Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. The MIT Press, 2019. URL https://mitpress.mit.edu/9780262039420/algorithms-for-optimization/.

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. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

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. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81. Association for Computational Linguistics, 2004. URL https://aclanthology.org/W04-1013/.

Jian Lin, Jiazhi Mi, Zicong Hong, Haodong Wang, Qianli Liu, Haoyue Zhang, Peng Li, and Song Guo. KVDrive: A holistic multi-tier KV cache management system for long-context LLM inference. *Proceedings of the ACM on Management of Data*, 4(3):1–25, 2026. doi: 10.1145/3802077. URL https://doi.org/10.1145/3802077.

Song Liu, Fei Li, Chenyu Zhao, Qin Xia, Shiqiang Nie, Jinyu Wang, and Weiguo Wu. FDSR: Efficient model training via adaptive tensor quantization based on frequency domain division and similarity data reuse. *ACM Transactions on Architecture and Code Optimization*, 23(2):1–26, 2026a. doi: 10.1145/3802593. URL https://doi.org/10.1145/3802593.

Yang Liu, Yunfei Gu, Liqiang Zhang, Chentao Wu, Guangtao Xue, Jie Li, Minyi Guo, Junhao Hu, and Jie Meng. CacheSlide: Unlocking cross Position-Aware KV cache reuse for accelerating LLM serving. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 83–99. USENIX Association, 2026b. URL https://www.usenix.org/conference/fast26/presentation/liu-yang.

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. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56. Association for Computing Machinery, 2024. doi: 10.1145/3651890.3672274. URL https://doi.org/10.1145/3651890.3672274.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. *arXiv preprint arXiv:2510.09665*, 2025. doi: 10.48550/arXiv.2510.09665. URL https://arxiv.org/abs/2510.09665.

Llama Team. The Llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A KVCache-centric disaggregated architecture for LLM serving. *ACM Transactions on Storage*, 22 (4):1–38, 2026. doi: 10.1145/3773772. URL https://doi.org/10.1145/3773772.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. doi: 10.48550/arXiv.2412.15115. URL https://arxiv.org/abs/2412.15115.

Minseok Seo, Jungi Hyun, Seongho Jeong, Xuan Truong Nguyen, Hyuk-Jae Lee, and Hyokeun Lee. OASIS: Outlier-aware KV cache clustering for scaling LLM inference in CXL memory systems. *IEEE Computer Architecture Letters*, 24(1):165–168, 2025. doi: 10.1109/LCA.2025.3567844. URL https://doi.org/10.1109/LCA.2025.3567844.

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. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Guanchu Wang, Zirui Liu, Zhimeng Jiang, Ninghao Liu, Na Zou, and Xia Hu. DIVISION: Memory efficient training via dual activation precision. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 36036–36057. PMLR, 2023. URL https://proceedings.mlr.press/v202/wang23s.html.

Shihao Wang, Jiahao Chen, Yanqi Pan, Hao Huang, Yichen Hao, Xiangyu Zou, Wen Xia, Chongyang Qiu, Wentao Zhang, and PengFei Wang. ProphetKV: User-query-driven selective recomputation for efficient KV cache reuse in retrieval-augmented generation. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=SwGwmKIgUb.

Darrell Williamson. *Discrete-time Signal Processing: An Algebraic Approach*. Advanced Textbooks in Control and Signal Processing. Springer London, 1 edition, 2012. doi: 10.1007/978-1-4471-0541-1. URL https://link.springer.com/book/10.1007/978-1-4471-0541-1. eBook publication; original edition published in 1999.

Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. Learning in the frequency domain. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1740–1749, 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Xu_Learning_in_the_Frequency_Domain_CVPR_2020_paper.html.

Bin Yang, Qiuyu Leng, Jun Zeng, and Zhenhua Wu. CacheClip: Accelerating RAG with effective KV cache reuse. *arXiv preprint arXiv:2510.10129*, 2025a. doi: 10.48550/arXiv.2510.10129. URL https://arxiv.org/abs/2510.10129.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. In *Advances in Neural Information Processing Systems*, volume 38, pp. 133797–133824. Curran Associates, Inc., 2025b. doi: 10.52202/085713-4461. URL https://papers.nips.cc/paper_files/paper/2025/hash/c217d123885341853cecbdd2be809983-Abstract-Conference.html.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1259. URL https://doi.org/10.18653/v1/d18-1259.

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. Association for Computing Machinery, 2025. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Lu Ye, Ze Tao, Yong Huang, and Yang Li. ChunkAttention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11608–11620. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.623. URL https://aclanthology.org/2024.acl-long.623/.

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. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A Supplementary Background on Cache Reuse and Frequency-Domain Methods

As context lengths and serving scales grow, the KV Cache is evolving from temporary state within a single inference run into data that can be retained, retrieved, and migrated across requests. Tiered caching studies how to use the capacities and access speeds of different media (Gao et al., 2024; Jin et al., 2025), global scheduling coordinates requests and cache resources (Qin et al., 2026), and vector indexing and multi-tier data placement support cache organization and access at larger scales (Chen et al., 2026; Lin et al., 2026). On top of these management mechanisms, reusing existing KVs can eliminate prefill computation for repeated text. How this reuse is implemented depends on whether the context in which the cache was generated matches that of the current request.

Prefix caching uses this consistency of context: when two requests share the same prefix, the preceding context of each token in that prefix remains unchanged, allowing the corresponding KVs to be reused directly. vLLM, SGLang, and ChunkAttention all use prefix sharing to reduce repeated computation (Kwon et al., 2023; Zheng et al., 2024; Ye et al., 2024). In RAG and multi-document summarization, however, reusable content often consists of independent document chunks whose order and positions vary across requests. When independently encoded chunks enter a new prompt, they need to establish dependencies on preceding chunks, giving rise to the problem of cross-chunk repair in non-prefix cache reuse.

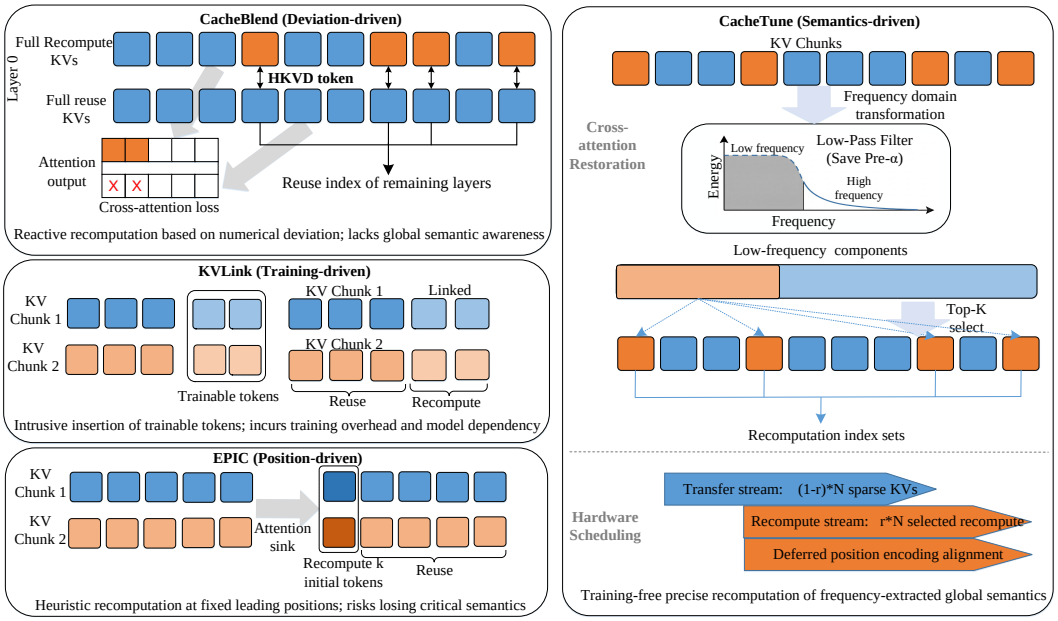

Figure 7: Comparison of cross-attention recovery strategies for non-prefix KV Cache reuse.

Non-prefix reuse methods adopt different organization and repair mechanisms. PromptCache uses predefined prompt schemas to organize KVs into modules that can be cached and composed independently (Gim et al., 2024). To address the loss of cross-attention after independent chunks are concatenated, CacheBlend fully recomputes the first layer at runtime and uses KV deviations to select tokens to update in subsequent layers (Yao et al., 2025); EPIC concentrates recomputation at attention-sink positions at the beginning of each chunk (Hu et al., 2025); and KVLink learns link tokens at chunk boundaries through additional training to establish connections between chunks (Yang et al., 2025b). These designs organize repair using numerical changes in the current context, fixed positional priors, and learned connection mechanisms, respectively. Prior methods also adjust the recomputation ratio r based on estimated computation and cache-loading costs (Yao et al., 2025). CacheTune further considers how r affects complementary transfer volume and calibrates the ratio using measured end-to-end TTFT, accounting for factors not captured by these estimates, such as pipeline startup, synchronization, and overlap effects, to achieve more precise adjustment of the recomputation ratio.

Other methods organize updates around the reuse scenario or selection signal. CacheSlide studies relative-position-dependent caching, where reusable segments retain their relative order while their absolute positions shift, and supports reuse through chunked contextual positional encoding, weighted correction attention, and cache management optimizations (Liu et al., 2026b). CacheClip uses a fine-tuned auxiliary model to guide token selection (Yang et al., 2025a), Cache-Craft estimates chunk reusability from prefix overlap and historical attention (Agarwal et al., 2025), and ProphetKV combines query-induced attention with multi-layer metrics to prioritize recomputation (Wang et al., 2026). These studies expand the structural, historical, and query information available for cache reuse.

CacheTune extracts a ranking of positions from the KV frequency-domain structure of independently encoded chunks, constructing the basis for selection during cache generation. When a request arrives, it recomputes a subset of tokens according to this ranking so that their representations incorporate the current global context, and loads cached KVs for the remaining positions. Because the ranking is generated from the chunk itself, the same chunk can retain its existing indices across different queries, and the cache system can use them to determine in advance which complementary portion to transfer. Figure 7 contrasts the positions selected and updated within a chunk by several cross-chunk repair strategies, illustrating the differences in their selection criteria and recomputation schemes.

In image and signal processing, different frequency components describe variations at different scales, and their energy distribution can guide representation and compression (Xu et al., 2020;

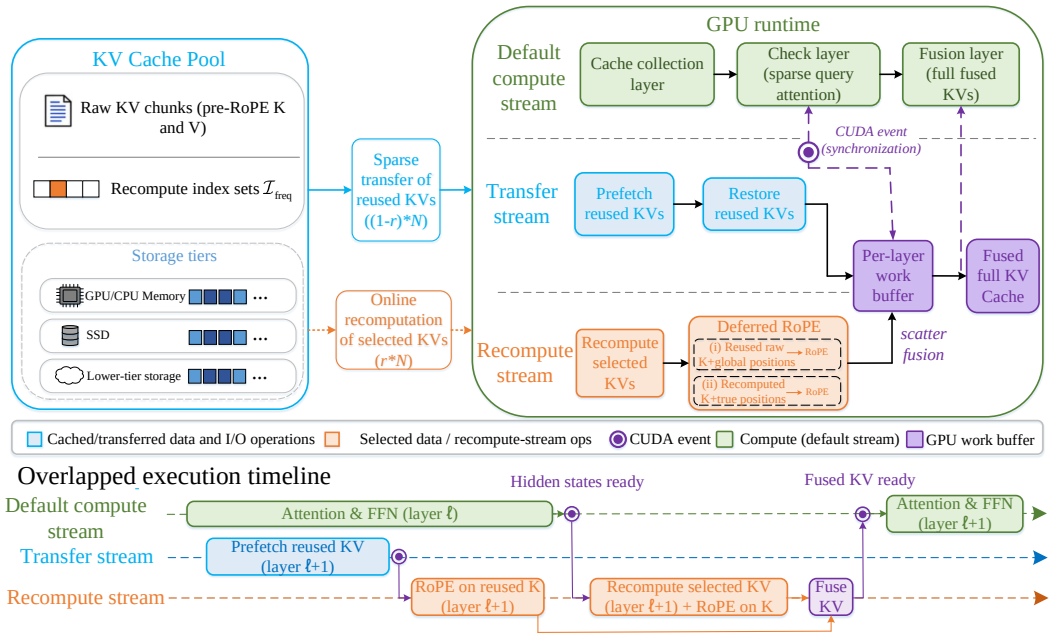

Figure 8: KV Cache offloading and sparse reuse pipeline.

Wang et al., 2023; Liu et al., 2026a). FreqKV applies this idea to the KV Cache, reducing storage overhead by retaining low-frequency structure and compressing high-frequency components (Kai et al., 2026). CacheTune instead uses low-frequency structure to identify recomputation positions that are important for recovering cross-attention. The KVs at every original position are supplied through either recomputation or reuse, preserving the full context while concentrating the limited recomputation budget on repairing important cross-chunk cross-attention.

B Offline Preparation and Runtime Implementation

CacheTune uses independently encoded KV Chunks together with frequency-guided selection of important KV indices for online prefill. The important KV indices determine which tokens to update at each layer, the cache pool supplies KVs for the remaining positions, and the runtime fuses the two parts according to their positions in the composed prompt. The same index set thus connects the selection rule in Section 4.1 with the execution pipeline in Section 4.2: once a position is selected, its KVs are generated by forward computation in the current context; the remaining positions follow the sparse loading path. Figure 8 shows the layer-by-layer execution of this process.

B.1 Offline Preparation

For a reusable text chunk containing N tokens, CacheTune first encodes it independently and collects the Keys and Values from all L layers. The tensor at each layer has shape $N \times H \times D$, where H is the number of KV heads and D is the head dimension. Spectral analysis is performed along the token axis, so the KV representations that vary across positions in each head and channel form a sequence of length N. Because the sequences are real-valued, rFFT produces $\lfloor N/2 \rfloor + 1$ non-redundant frequencies. CacheTune retains the lowest α fraction of frequency components and applies the inverse transform with the output length set to N to recover low-frequency tensors aligned with the original positions.

Following the importance scoring rule in Section 4.1, CacheTune computes the magnitudes of the low-frequency Key and Value at each position separately, combines them with equal weights, normalizes the scores within each layer, and averages them across layers to form a unified importance ranking within the chunk. Normalization within each layer makes the numerical scales of different layers comparable, while cross-layer aggregation selects the same set of positions for continuous sparse forward execution. For a given recomputation ratio r, the top-ranked rN positions form the

recomputation set $\mathcal{I}_{\text{freq}}$, and its complement is denoted by $\overline{\mathcal{I}}_{\text{freq}}$. The hidden states of the former propagate continuously across Transformer layers, while the latter supply context through cached KVs at each layer.

The ranking is computed during cache generation, and subsequent requests select the corresponding number of positions according to r. The spectral transform for a single head and channel has a complexity of $O(N \log N)$. Accounting for the number of layers, KV heads, and head dimension gives a total complexity of $O(LHDN \log N)$ for this step, while scoring and aggregation require $O(LHDN)$. This preprocessing overhead is amortized over repeated uses of a cached chunk. Section 5.2 reports that generating recomputation indices on the GPU for a cached chunk containing 3K tokens takes only about 0.69 ms on average.

Once a new composed prompt is formed, CacheTune maps local indices within each chunk to global indices using the chunk's starting position in the sequence. If chunk b starts at global position o_b, its local position i corresponds to $o_b + i - 1$. The recomputation set and its reuse complement use the same mapping for query filtering, cache loading, fusion writeback, and positional encoding. Changing the order or offsets of chunks therefore requires only an update to the index mapping, while adjusting the recomputation ratio changes the set size along the stored ranking.

B.2 Sparse KV Reuse Pipeline

Online prefill assembles KVs from two sources at each layer: KVs in $\mathcal{I}_{\text{freq}}$ are recomputed in the current context, and KVs in $\overline{\mathcal{I}}_{\text{freq}}$ are loaded from the external cache pool. CacheTune applies this partition at the start of loading, reducing the transfer volume for each chunk at each layer from N to $(1-r)N$ KV pairs. Online computation retains the state-update path of the selected tokens, while the KVs accessed by attention cover the full sequence. To connect these two granularities, the runtime uses a collection layer, a check layer, and subsequent fusion layers, with reusable per-layer working buffers for KV assembly.

The collection layer organizes the offline KV Chunks to be reused by the current request and establishes the positional correspondence between these chunks and the composed prompt. The precomputed ranking of each chunk is thereby converted into request-level recomputation and loading indices. Cached chunks may reside in CPU memory, SSDs, or HDDs. The runtime uses the loading indices to obtain KVs in the reuse complement at each layer, leaving the selected positions to be generated by the recomputation path. Chunk content, layer number, and global position are associated at this stage for subsequent layer-by-layer use.

The check layer applies the precomputed indices on the query side to select tokens in reusable chunks that require further updates. Queries at the selected positions participate in the current layer's attention computation and aggregate preceding context over the complete KVs of that layer. After attention and the remaining forward computation, the hidden states at these positions pass to the next layer to continue generating the required queries and KVs. The same index set is retained across layers so that every selected position has a continuously updated hidden state; uncached content, such as new queries, still follows the normal forward path.

Layers after the check layer perform fusion. At layer ℓ, the recomputation path generates KVs in $\mathcal{I}_{\text{freq}}$ from the updated input hidden states, while the transfer path obtains the corresponding layer's cached KVs in $\overline{\mathcal{I}}_{\text{freq}}$. After global positional encoding is applied to both sets of Keys, the system writes reused KVs into the working buffer according to their position indices, then scatters recomputed KVs into the remaining selected positions. These two complementary index sets cover all positions in the reusable chunks at that layer and, together with KVs generated for uncached content such as new queries, form the complete cache. The selected queries of the current layer access the preceding context under a causal mask, and the results undergo operations within the layer, such as feed-forward computation, to form the input to the next layer. Reusable buffers allow this assembly process to proceed layer by layer while reducing repeated allocation and intermediate data organization overhead.

These operations are coordinated across three CUDA streams. The model's native forward stream performs the current layer's forward computation; the transfer stream asynchronously prefetches the reused KVs needed by the next layer; and the recomputation stream generates selected KVs from ready hidden states and performs positional recovery for reused Keys. The streams hand off data

through per-layer CUDA events: after transfer completes, reused Keys undergo positional recovery; once both the recomputed and reused portions are ready, the fused result is consumed by the current layer's attention; and the layer's output then supplies the input for recomputation at the next layer.

The pipeline advances continuously through the layers. While the forward stream processes layer ℓ, the transfer stream prefetches the reuse complement for layer $\ell + 1$, overlapping external cache access with the current layer's computation. Once the output of layer ℓ is ready, the recomputation stream can use it to generate the selected KVs for the next layer and combine them with the prefetched result to assemble that layer's cache. Per-layer events schedule handoffs at the points where data are consumed, while prefetching uses computation time in adjacent layers to complete loading in advance. The reduction in transfer volume enabled by offline indices thus works together with cross-layer overlap to shorten the time online prefill spends waiting for external cache loading.

B.3 Deferred RoPE for Positional Alignment

The local positions within an independent chunk usually differ from its global positions in the new context. In models that use RoPE, positional rotation is applied to Keys after linear projection. If cached Keys have already been rotated according to their positions within the chunk, moving the chunk directly carries the original positional encoding into the new sequence. CacheTune therefore stores raw pre-RoPE Keys in the offline cache and defers positional rotation until online reuse, allowing the same chunk to be used at different global offsets.

After the transfer stream loads the raw reused Keys of layer ℓ onto the GPU, the recomputation stream reads their corresponding global positions p_i^{global} and performs positional recovery as follows:

$$\mathbf{K}^{(\ell)}_{\text{reuse},i} = \text{RoPE}(\mathbf{K}^{(\ell)}_{\text{raw},i}, p_i^{\text{global}}), \qquad i \in \overline{\mathcal{I}}_{\text{freq}}. \tag{8}$$

Newly generated Keys at selected positions likewise receive RoPE using their true global positions, while Values participate directly in fusion according to their corresponding indices. Both types of Keys are thus expressed in the same positional coordinate system of the composed prompt, allowing queries at the current layer to perform causal attention over the fused KVs. Positional recovery aligns the chunk's coordinates within the new prompt, while selective recomputation restores cross-chunk dependencies by updating hidden states; the two operate together within the same fusion process.

Deferred RoPE uses the same index mapping as sparse loading and is integrated into the asynchronous path of the recomputation stream. Reused Keys are handed off in the order of transfer completion, positional recovery, and consumption during fusion, interleaved with ready computation or transfer tasks from other layers. Positional encoding is thus recovered along the online data flow, while chunks in the cache pool retain a uniform form that can be reused across requests.

C Hardware-Aware Calibration of the Recomputation Ratio

The complementary construction in Section 4.2 makes the recomputation ratio r control two costs simultaneously: increasing r raises the number of tokens that need to be updated on the GPU and reduces the number of KVs loaded from the external cache. These costs vary in opposite directions with r, and their relative magnitudes depend on the model, GPU, and cache path. Reuse has a low access cost in CPU memory; on SSDs/HDDs or other external storage media, increasing recomputation may reduce TTFT by replacing expensive reads. Section 4.3 therefore adopts a scheduling method that combines analytical estimation with empirical calibration: it first locates the region where computation and transfer are balanced, then determines the deployment ratio from measured latency for complete requests.

C.1 Computation and Transfer Cost Model

Consider processing N reusable cached tokens through L Transformer layers. Given ratio r, each layer recomputes rN tokens and loads the remaining $(1 - r)N$ KV pairs. Let t_c denote the effective per-token, per-layer recomputation cost, which is mainly affected by model width and GPU compute capability; t_i denotes the effective loading cost of the corresponding KVs, jointly reflecting cache-medium bandwidth, DMA granularity, and PCIe utilization; and t_o denotes fixed per-layer overhead,

including kernel launches, CUDA event synchronization, and buffer management. These quantities are estimated during deployment initialization by profiling the target model and cache path.

The model treats t_c, t_i, and t_o as amortized constants for a given configuration and uses a steady-state pipeline approximation that ignores startup and drain bubbles. The per-layer times of the computation and transfer paths are, respectively:

$$T_c(r) = rNt_c, \qquad T_i(r) = (1-r)Nt_i \tag{C.1}$$

In the cross-layer pipeline in Appendix B, cache transfer for the next layer overlaps with computation at the current layer, so the per-layer execution pace is determined by the slower of the two paths. Including the fixed overhead gives:

$$T_{\text{layer}}(r) \approx \max\{rNt_c, (1-r)Nt_i\} + t_o \tag{C.2}$$

Summing over L layers yields the latency approximation used in Section 4.3:

$$T_{\text{TTFT}}(r) \approx L \cdot \max\{rNt_c, (1-r)Nt_i\} + Lt_o \tag{C.3}$$

The model defines two operating regions. When r is small, cache transfer dominates, and increasing r reduces the KVs waiting to be loaded; when r is large, GPU recomputation dominates, and further increasing r lengthens the computation path. The two paths intersect at $rNt_c = (1-r)Nt_i$, giving the analytical balance point:

$$r_0 = \frac{t_i}{t_c + t_i} \tag{C.4}$$

Thus, a higher cache access cost shifts the balance point toward a larger recomputation ratio, whereas a higher per-unit recomputation cost makes greater cache reuse more attractive. N and L determine the total workload, while t_c and t_i characterize the relative costs of the target configuration. The actual search uses TTFT for complete requests, including computation for new queries and uncached content, transfer startup, synchronization, and pipeline bubbles in the measurements.

C.2 Warm-Started Golden-Section Search

Calibration first establishes the allowed recomputation interval $[r_{\min}, r_{\max}]$. Based on the quality–latency experiments in Appendix E.4, we use $r_{\min} = 15\%$ as an empirical lower bound for quality preservation and set $r_{\max} \leq 1$. This lower bound reserves a sufficient cross-chunk repair budget when optimizing latency on fast media. The analytical balance point is then clipped to this interval to serve as the initial probe for empirical search:

$$r_{\text{init}} = \min\{r_{\max}, \max\{r_{\min}, r_0\}\} \tag{C.5}$$

Let $\mathcal{S}_{\text{cal}}$ be a calibration sample set with representative request lengths. Each candidate ratio selects recomputation positions along the same offline ranking and runs the complete prefill process on these requests. The calibration objective is to minimize average TTFT:

$$F(r) = \frac{1}{|\mathcal{S}_{\text{cal}}|} \sum_{s \in \mathcal{S}_{\text{cal}}} \text{TTFT}(s; r) \tag{C.6}$$

$$r^* = \underset{r \in [r_{\min}, r_{\max}]}{\arg\min} \; F(r) \tag{C.7}$$

The computation–transfer balance model has a unimodal structure that first decreases and then increases, providing a basis for one-dimensional derivative-free search. CacheTune uses golden-section search (Kochenderfer & Wheeler, 2019), warm-started from the analytical initial value. Specifically, it first pairs r_{init} from Equation (C.5) with another probe in the interval to form an ordered pair, compares their measured average TTFT, and retains the more favorable subinterval; it then reinitializes the standard golden-section points within that subinterval. Algorithm 1 denotes this clipped initial value by r_0.

In the standard update phase, let $\varphi = (\sqrt{5}-1)/2$ and choose $x_1 = b - \varphi(b-a)$ and $x_2 = a + \varphi(b-a)$ within the current interval $[a, b]$. After comparing $F(x_1)$ and $F(x_2)$, the search retains the interval on the lower-latency side and reuses one existing probe and its function value, requiring only one new candidate-ratio evaluation per iteration. When the interval width is less than ε, the algorithm returns the midpoint of the remaining interval as the deployment ratio. The analytical model supplies

Algorithm 1 Hardware-Aware Calibration of the Recomputation Ratio

Input: calibration set $\mathcal{S}_{\text{cal}}$; model M; roofline prior r_0, obtained by clipping $t_i/(t_c + t_i)$ to $[r_{\min}, r_{\max}]$; semantic constraint bounds $r_{\min}$ and $r_{\max}$, satisfying $r_{\min} \le r_{\max}$; tolerance threshold ε.
Output: selected recomputation ratio r^*.

$F(x) \triangleq \text{EvalTTFT}(M, \mathcal{S}_{\text{cal}}, x)$ ▷ Average TTFT on the calibration set
$a \leftarrow r_{\min}$; $b \leftarrow r_{\max}$; $\varphi \leftarrow (\sqrt{5}-1)/2$ ▷ Initialize the search
If $r_0 \le (a+b)/2$, then $(x_1, x_2) \leftarrow (r_0, a + \varphi(b-a))$;
otherwise $(x_1, x_2) \leftarrow (b - \varphi(b-a), r_0)$ ▷ Construct ordered warm-start probes
$(f_1, f_2) \leftarrow (F(x_1), F(x_2))$
If $f_1 \le f_2$, then $(a, b) \leftarrow (a, x_2)$; otherwise $(a, b) \leftarrow (x_1, b)$
▷ Shrink the interval using unimodality
$x_1 \leftarrow b - \varphi(b-a)$; $x_2 \leftarrow a + \varphi(b-a)$ ▷ Reinitialize standard GSS
$(f_1, f_2) \leftarrow (F(x_1), F(x_2))$
While $b - a \ge \varepsilon$:
  If $f_1 < f_2$:
    $b \leftarrow x_2$; $x_2 \leftarrow x_1$; $f_2 \leftarrow f_1$
    $x_1 \leftarrow b - \varphi(b-a)$; $f_1 \leftarrow F(x_1)$
  Else:
    $a \leftarrow x_1$; $x_1 \leftarrow x_2$; $f_1 \leftarrow f_2$
    $x_2 \leftarrow a + \varphi(b-a)$; $f_2 \leftarrow F(x_2)$
  End if
End while
Return $(a+b)/2$

the starting point, while the empirical objective feeds the actual costs of transfer and pipelining into the choice of ratio.

During the golden-section phase, the interval width shrinks by a factor of φ in each iteration. If its initial width is Δr, reaching a ratio tolerance of ε requires $O(\log(\Delta r/\varepsilon))$ iterations. Each candidate ratio is evaluated on $|\mathcal{S}_{\text{cal}}|$ requests, so this phase requires $O(|\mathcal{S}_{\text{cal}}| \cdot \log(\Delta r/\varepsilon))$ TTFT measurements, plus a fixed number of initial probes. The entire process is performed during deployment calibration, and the resulting ratio is used for subsequent requests. Online execution directly uses this ratio to select token importance indices and generate the complementary KV loading plan.

D Experimental Setup and Evaluation Methodology

We evaluate CacheTune in terms of generation quality and prefill efficiency. The quality experiments examine how much cross-chunk information selective recomputation can recover after independently cached chunks are concatenated into a new context. The performance experiments examine the combined overhead of repair operations and cache transfer, comparing time to first token and serving throughput across storage tiers.

D.1 Models, Hardware, and Software Environment

We use three models, Mistral v0.3-7B, Llama 3-8B, and Qwen 2.5-32B (Jiang et al., 2023; Llama Team, 2024; Qwen Team, 2024), to evaluate CacheTune across model sizes. The hardware comprises two platforms: one with 2×NVIDIA A100-40GB GPUs and another with 2×NVIDIA GeForce RTX 4090-24GB GPUs. Both platforms connect to the host through PCIe Gen3×16. Reusable KV Caches are placed in GPU memory, CPU memory, and disk cache pools. The disk experiments use an HDD cache pool on the A100 platform and an SSD cache pool on the RTX 4090 platform. These configurations provide different GPU computing capabilities and external-cache access costs for hardware-aware scheduling.

All experiments run on Ubuntu 22.04.1 with Linux kernel 6.8.0, PyTorch 2.2.1, and CUDA 12.4.

D.2 Data Preparation

The experiments cover five long-context datasets, comprising two summarization tasks and three multi-hop question-answering tasks. SAMSum constructs summarization contexts by concatenating multi-turn dialogues, while Multi-News constructs multi-document summarization contexts by

concatenating news articles. MuSiQue, WikiMQA, and HotpotQA combine retrieved fragments from multiple documents into question-answering contexts. All these tasks require the model to use information from multiple reusable segments, and multi-hop question answering additionally requires connecting evidence distributed across segments. Table 2 summarizes the task type, context construction, and evaluation metric for each dataset.

Table 2: Summary of evaluation datasets and metrics.

Dataset	Task and Context Construction	Metric ↑
SAMSum (Gliwa et al., 2019)	Multi-turn dialogue summarization; concatenated dialogues	ROUGE-L
MuSiQue (Trivedi et al., 2022)	Multi-hop question answering; concatenated retrieved chunks	F1
WikiMQA (Ho et al., 2020)	Multi-hop question answering; concatenated retrieved chunks	F1
HotpotQA (Yang et al., 2018)	Multi-hop question answering; concatenated retrieved chunks	F1
Multi-News (Fabbri et al., 2019)	Multi-document summarization; concatenated news articles	ROUGE-L

SAMSum and Multi-News use ROUGE-L to evaluate summarization quality (Lin, 2004); MuSiQue, WikiMQA, and HotpotQA use answer F1. For both metrics, higher values indicate closer agreement between the generated output and the reference answer. Methods are compared on the same model and task, with Full Recompute providing a quality reference based on full-context encoding.

D.3 Comparison Methods, Metrics, and Cache Placement

We use eight baselines. Full Recompute performs standard prefill over the entire prompt; Full KV Reuse directly concatenates independently encoded chunk caches. vLLM Prefix Caching uses native Automatic Prefix Caching to reuse only strictly matching prefixes, recomputing the remaining content through the default prefill path. These three methods correspond to full-context computation, direct non-prefix reuse, and strict prefix reuse, respectively.

The remaining baselines restore cross-chunk dependencies in different ways. Following the released CacheBlend implementation, we use the first layer as the check layer to identify HKVD tokens from KV deviations, and then selectively recompute the same selected 15% of positions in subsequent layers. EPIC updates the attention-sink positions at the beginning of each chunk, set to the first 16 tokens in our experiments. KVLink inserts 5 trained link tokens at each chunk boundary. For CacheSlide, we treat reusable documents as fixed segments and queries as updated segments. We use Weighted Correction Attention for selective recomputation and cache fusion, adapting it to the multi-chunk contexts in our experiments. We use an adapted implementation of ProphetKV based on its original method. In the first stage, query tokens are processed through successive layers to compute their attention to context tokens in the reused cache, and the scores are averaged over query positions and layers. In the second stage, the aggregated scores determine a shared token set whose KVs are recomputed at each layer, while the remaining KVs are reused directly. For a fair comparison, we also set ProphetKV's recomputation ratio to 15%.

CacheTune uses a low-frequency cutoff ratio of $\alpha = 0.5$ and a recomputation ratio of $r = 15\%$ in the main CPU-cache experiments. Offline ranking determines which positions to update; online execution recomputes the KV pairs at these positions and loads the cache for the remaining positions. Appendix E.4 varies α and r separately to analyze the effects of the spectral selection range and recomputation budget. The external-disk experiments additionally use calibrated recomputation ratios.

System performance is measured primarily by time to first token (TTFT) to assess the effect of prefill overhead on response latency. Figure 3 jointly compares task quality and TTFT. Methods without multi-tier offloading optimizations, such as EPIC and KVLink, use GPU-resident caches, while CacheTune places reusable chunks in CPU memory and enables the sparse offloading pipeline. Figure 4 further varies the request rate to examine how TTFT changes with increasing load and compare serving throughput across methods.

The storage-tier experiments compare methods using the same cache medium. In Table 1, CacheTune, CacheBlend, CacheSlide, and Prefix Caching are each run with GPU and CPU cache pools. CacheTune's GPU configuration uses the native cache reuse path, whereas its CPU configuration enables sparse transfer, asynchronous prefetching, and deferred RoPE restoration. Both use a fixed recomputation ratio of 15%. Figure 6 compares fixed and adaptive ratios on the A100/HDD and RTX 4090/SSD configurations, with TTFT speedups measured relative to Full Recompute on the corresponding configuration.

D.4 Deployment Calibration of the Recomputation Ratio

We use the first 10 SAMSum samples as the calibration set. Calibration measures computation, transfer, and synchronization costs without using reference answers or task-specific quality scores. Other datasets with comparable cache reuse patterns can therefore also serve as calibration workloads. Each candidate ratio is evaluated on the same calibration requests, and mean TTFT is used as the search objective. Hardware profiling and ratio search follow the procedure described in Appendix C.

The fast-cache experiments use $r = 15\%$, which also serves as the empirical lower bound for disk calibration. The disk experiments use the calibrated ratios of $r = 36.4\%$ for A100/HDD and $r = 30.9\%$ for RTX 4090/SSD. These ratios remain fixed during subsequent evaluation on each configuration. The corresponding latency results are reported in Appendix E.5.

E Supplementary Experiments and Analysis

CacheTune's frequency selection and hardware scheduling determine the positions and budget for recomputation, respectively. Our experiments examine these two decisions: spectral distributions and parameter studies assess the basis of the positional importance ranking; reuse of fixed indices tests recovery across contexts; and measurements across cache granularities and storage tiers examine how the recomputation budget affects data transfer and execution latency.

E.1 Frequency-Domain Energy Distribution of the KV Cache

Frequency-guided selection starts from the premise that KV sequences contain structure that can characterize token importance. To examine this structure, we run MuSiQue and SAMSum on Mistral v0.3-7B and Llama 3-8B, apply the fast Fourier transform (FFT) (Williamson, 2012) separately to Keys and Values along the token dimension, and measure the energy fractions in different frequency bands. Low-frequency components correspond to slower variations along the sequence, whereas high-frequency components correspond to faster local variations. Figure 9 presents the measurements for the two models and two task types.

All groups show a consistent concentration at low frequencies: the high-frequency energy fractions of Keys and Values decrease as frequency increases, and the lowest 20% of the spectrum carries the vast majority of the total energy. This indicates that the main variations in KVs along the sequence can be compactly described by low-frequency structure. The phenomenon occurs in both summarization and multi-hop question answering, providing a basis for constructing an offline ranking from the frequency-domain features of each chunk itself.

CacheTune applies the inverse transform to the retained spectrum to recover a low-frequency signal aligned with the original sequence positions, then computes importance from the reconstructed magnitude at each token. The overall structure in the spectrum is thus converted into a ranking of positions that online recomputation can use directly. The spectral measurements motivate the use of low-frequency signals.

E.2 Basis for Low-Frequency Selection and Robustness of Reuse across Contexts

Independent encoding omits cross-chunk context, causing the KVs at some positions to deviate from full encoding. Using these positions with larger changes as a reference, we examine whether low-frequency reconstruction magnitudes can provide an effective KV importance ranking for

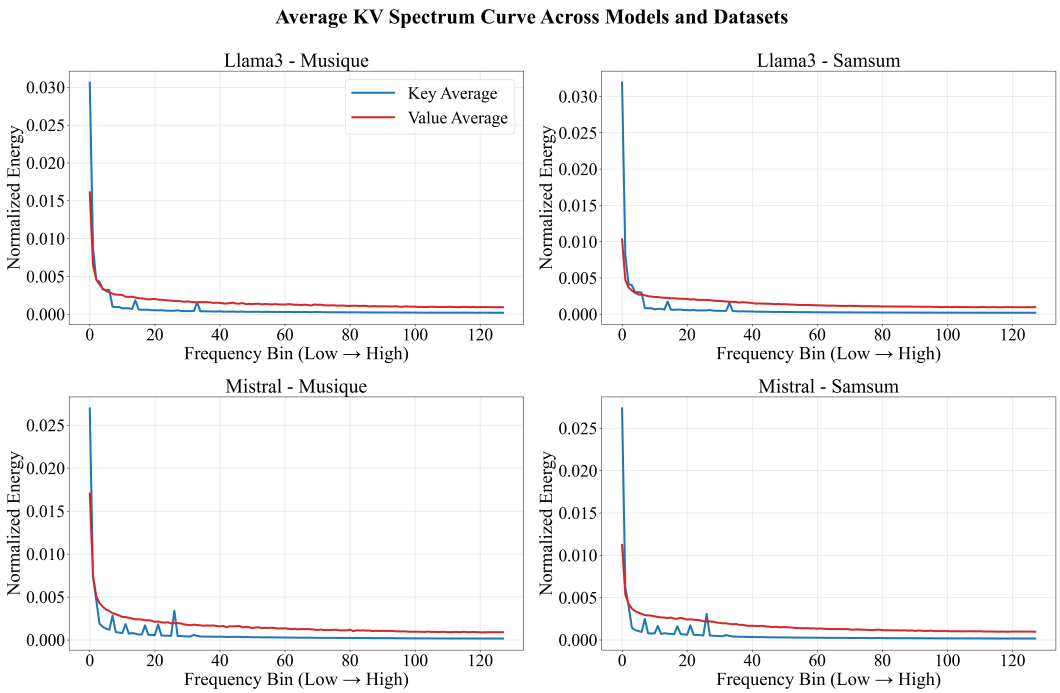

Figure 9: Frequency-domain energy distribution of the KV Cache along the sequence dimension.

recomputation, and further test how well the ranking recovers attention distributions and final answers.

Using Mistral v0.3-7B on SAMSum and MuSiQue, we compare the Keys and Values of the same token under independent chunk encoding and full-context encoding. Drift is computed from KV differences normalized within each layer, and Keys are compared before RoPE to exclude the effect of positional rotation. We use the 15% of tokens with the largest drift as the reference set. The low-frequency, high-frequency, and random strategies each also select 15% of positions; Recall@15% is the intersection of the two sets divided by the size of the high-drift set.

Table 3: Coverage of high-drift KVs by different selection strategies. Only 15% of tokens are selected; Recall@15% is reported.

Metric	SAMSum	MuSiQue
LF Recall@15%	90.23%	86.31%
HF Recall@15%	37.64%	24.90%
Random Recall@15%	15.92%	14.62%

In Table 3, low-frequency selection achieves Recall@15% of 90.23% and 86.31% on the two tasks, exceeding 37.64% and 24.90% for high-frequency selection and 15.92% and 14.62% for random selection. This indicates that low-frequency reconstruction magnitudes can identify, from independently cached representations, positions that change substantially in the full context. We further recompute the positions selected by the low-frequency strategy to test whether updating these KVs brings attention closer to the result of full-context encoding.

On SAMSum and MuSiQue, we use the attention distribution from full recomputation as the reference and compare its cosine similarity with those from full KV reuse, low-frequency-guided recomputation, and high-frequency-guided recomputation. Both the low-frequency and high-frequency strategies select 15% of positions from each chunk, update their KVs in the composed context, and directly reuse the remaining positions. This allows attention recovery from different position selections to be compared under the same recomputation budget. Figure 1(a) presents the results across different queries and numbers of chunks.

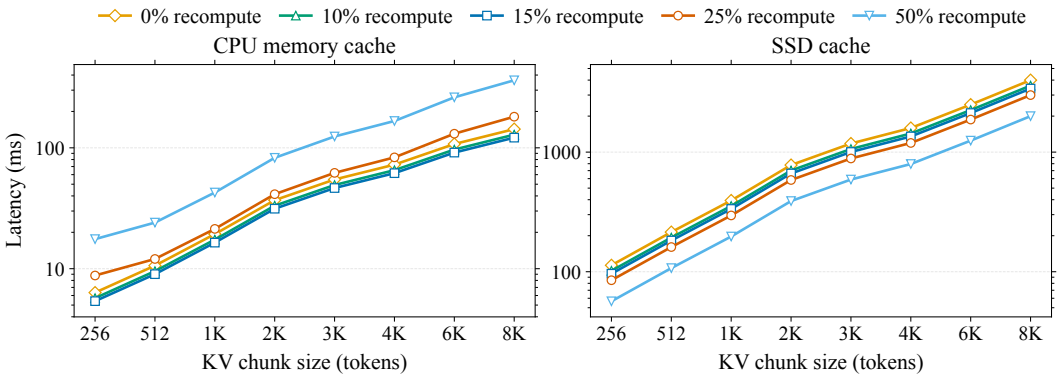

Figure 10: Effect of the recomputation ratio on KV Cache reuse latency.

Figure 1(a) shows that directly concatenating independently cached chunks leaves the attention distribution consistently different from that of full recomputation. The high-frequency strategy alleviates some of this deviation, whereas low-frequency recomputation maintains higher and more stable similarity across different queries and numbers of chunks. The representative heatmaps in Figure 1(b) show the same trend: full reuse and the high-frequency strategy exhibit clear shifts in attention positions, while the low-frequency strategy recovers a distribution closer to the reference. The preceding coverage experiment shows that the low-frequency ranking identifies high-drift positions; the attention results here further show that updating KVs according to this ranking helps restore the use of cross-chunk information.

To test whether this recomputation strategy improves final answers and whether offline indices remain effective in new contexts, we fix the cached document chunks and their within-chunk indices on MuSiQue, reuse them across different queries, and construct four context settings: the baseline order, random reordering, addition of a 1,024-token preceding context, and reordering followed by the addition of preceding context. When the global positions of chunks change, RoPE is applied to both reused and recomputed Keys using their corresponding new positions. The low-frequency, high-frequency, and random strategies all recompute 15% of document tokens, with full KV reuse and full recomputation serving as references for direct reuse and full encoding, respectively. Figure 1(c) presents the overall results. The low-frequency strategy achieves the highest reuse quality in all four settings. In the order shown in Figure 1(c), its F1 differs from the corresponding full-recomputation result by 2.94, 1.98, 1.86, and 3.63 points, respectively, and improves over random selection by 8.21, 5.10, 8.14, and 5.54 points. In particular, after chunks are reordered or preceding context is added, retaining the existing indices preserves its advantage over the high-frequency, random, and full-reuse strategies. This indicates that positions identified from chunk content still help recover missing dependencies in new contexts.

Low-frequency reconstruction magnitudes provide a basis for selecting context-sensitive positions, and recomputing these positions brings attention and answer quality closer to full encoding. Frozen indices retain these benefits after changes to the query and layout, making this signal suitable for offline extraction during independent chunk encoding and subsequent use with the cache in later requests.

E.3 Impact of Storage Tiers

Cached chunks are both the content units of non-prefix reuse and the basic units of organization and access in external storage. As chunk size increases, full recomputation must process more tokens, while direct reuse must read more KVs. To examine how these two costs vary with chunk granularity, we measure the total overhead of recomputation and loading the complementary KVs from the cache pool under ideal conditions. Cache-loading costs are measured using a baseline implementation without optimizations such as cross-layer prefetching.

Figure 10 shows that the absolute latency of each path increases with chunk length, while the relative performance of different recomputation ratios depends on the cache medium. CPU memory has a low access cost, so a small recomputation ratio keeps latency low: most KVs come from the cache, and

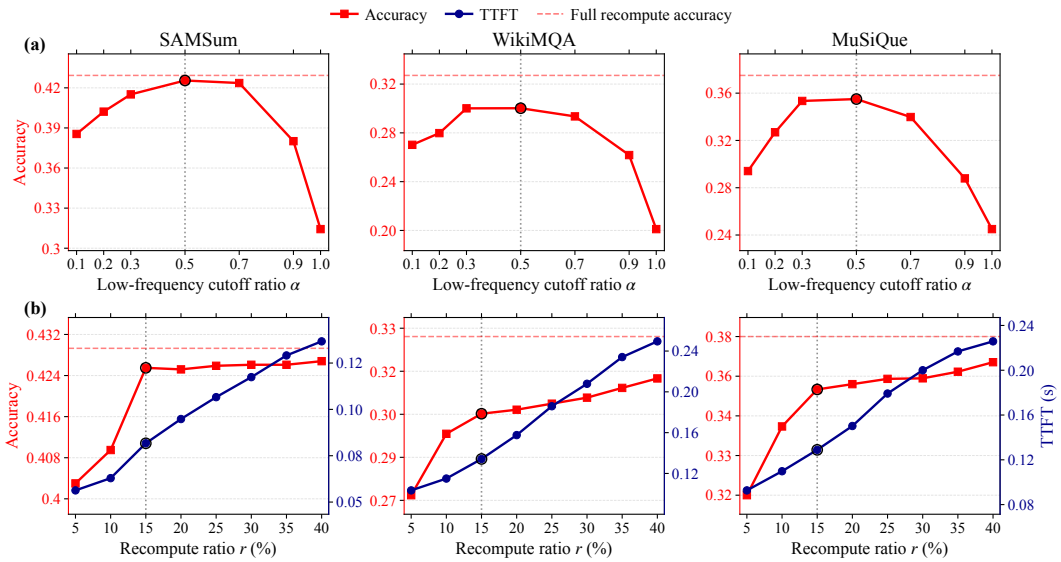

Figure 11: Sensitivity of model accuracy and TTFT to the low-frequency cutoff ratio α and recomputation ratio r, using Mistral v0.3-7B and a CPU memory KV Cache pool.

only selected positions incur online computation. Further increasing the recomputation ratio adds GPU work and gradually reduces the computational savings from reuse.

Reading data from the SSD is more expensive, shifting the balance between computation and transfer. Increasing the recomputation ratio reduces the complementary KV portion that must be read and moves more work to the GPU. The savings in read overhead can offset the additional computation cost. In this configuration, higher recomputation ratios yield lower cache reuse latency than full reuse.

The two sets of curves show that the benefits of cache reuse depend jointly on chunk length, GPU computation cost, and cache access cost. Low recomputation ratios suit fast memory, whereas slower storage benefits more from replacing data movement with computation. Accordingly, the scheduler in Section 4.3 profiles the target storage path and selects the recomputation ratio based on measured TTFT, applying the trade-off observed at chunk granularity to end-to-end request execution.

E.4 Effects of the Frequency Cutoff Ratio and Recomputation Ratio

The low-frequency cutoff ratio α determines which spectral components contribute to token scoring, while the recomputation ratio r determines how many positions are updated along that ranking. Using Mistral v0.3-7B and a CPU memory KV Cache pool, we vary these two parameters separately on SAMSum, WikiMQA, and MuSiQue. Figure 11(a) fixes $r = 15\%$ and measures task quality for different α values; Figure 11(b) fixes $\alpha = 0.5$ and measures both quality and TTFT. The quality of full recomputation serves as the reference line.

We first increase α from 0.1 to 1.0. Generation quality improves substantially on all three datasets as α increases to 0.3–0.5. CacheTune maintains strong quality within $\alpha \in [0.3, 0.7]$, with $\alpha = 0.5$ providing the most stable performance across tasks. Quality declines when α is further increased to 0.9 or 1.0. A small cutoff ratio may filter out spectral information useful for ranking positions, while retaining too many high-frequency components weakens the role of low-frequency structure in scoring. The intermediate range therefore provides better recovery.

The setting $\alpha = 1.0$ provides a useful comparison. All frequencies are retained, the inverse transform recovers the original KVs, and the importance scores approximately reduce to a ranking by the L2 norm of the original KV pairs. This setting underperforms $\alpha = 0.5$ on all three tasks, indicating that filtering the spectrum improves the selection of recomputation positions. Given its stable performance across tasks, we use $\alpha = 0.5$ as the default cutoff ratio.

We then fix $\alpha = 0.5$ and increase r from 5% to 40%. As more positions are updated in the current context, model quality generally improves and then gradually saturates, while TTFT continues to rise

Table 4: TTFT (s) as cache-loading optimizations are added progressively. Each row adds the indicated optimization to the preceding configuration; the final row is the full configuration.

Configuration	SAMSum	WikiMQA	MuSiQue
No cache-loading optimizations	0.179	0.289	0.281
+ Pipeline overlap and cross-layer prefetching	0.094	0.159	0.151
+ Sparse loading	0.086	0.140	0.137
+ Deferred RoPE (full configuration)	0.082	0.134	0.129

with the amount of online computation. A ratio of $r = 15\%$ recovers most of the quality loss caused by non-prefix reuse while maintaining a TTFT speedup of more than $4\times$ over full recomputation. This ratio therefore provides a favorable quality–latency trade-off in the CPU cache setting and is used as the default in the main experiments.

This result also determines the quality lower bound for the hardware search. On fast media such as GPU and CPU memory, the low cache access cost places the latency balance point from the cost model in Section 4.3 below 10%. The quality curves show that an excessively small recomputation budget leaves more pronounced quality loss, so we set $r_{\min}$ to 15%. For SSDs/HDDs, reading complementary KVs is more expensive, and increasing r can reduce external transfer. Appendix E.5 further reports the calibrated ratios and end-to-end latency in these settings.

E.5 Balancing Recomputation and Transfer on External Storage

External caching expands the storage space for reusable KVs but also introduces additional data access overhead. Using Mistral v0.3-7B, this section first compares end-to-end latency with GPU-resident and CPU-resident caches, then evaluates the three cache-loading optimizations introduced in Section 4.2 through cumulative ablation. We subsequently extend the cache pool to SSDs/HDDs to examine whether adjusting the recomputation ratio can alleviate slower cache reads.

GPU- and CPU-resident caches. In the GPU–CPU comparison, CacheTune fixes $r = 15\%$, and CacheTune, CacheBlend, CacheSlide, and prefix caching use the same medium within each group of experiments. CacheTune's GPU path uses the native cache reuse implementation, while its CPU path integrates sparse transfer, asynchronous prefetching, and deferred positional recovery into the same pipeline. Table 1 shows that the optimized CPU path retains the low-latency benefits on all three tasks; compared with CacheBlend, CacheSlide, and prefix caching under the same CPU cache setting, it reduces average TTFT by approximately 25.3%, 11.9%, and 74.4%, respectively. This result reflects the role of applying offline indices to cache loading. Knowing in advance which positions require recomputation, the system can skip reading these KVs and prefetch the complementary cache for subsequent layers while the current layer executes. By coordinating transfer, recomputation, and positional recovery through the pipeline, the system overlaps external cache access with computation wherever possible, allowing the capacity expansion provided by CPU caching to coexist with low TTFT.

Cache-loading ablation. To further distinguish the benefits of these optimizations, we fix $\alpha = 0.5$ and $r = 15\%$ under the same CPU cache configuration, using the same requests, cached chunks, and recomputation indices. Starting from a configuration without the cache-loading optimizations introduced in Section 4.2, we successively add pipeline overlap and cross-layer prefetching, sparse loading, and deferred RoPE. The final row corresponds to the full configuration with all three optimizations enabled.

Pipeline overlap and cross-layer prefetching provide the largest latency reduction. Adding this optimization reduces TTFT from 179/289/281 ms to 94/159/151 ms across the three datasets, a reduction of 45.0%–47.5%. Computation in the current layer proceeds alongside cache loading for subsequent layers, placing previously serialized loading work within the computation window and reducing forward-pass stalls while waiting for KVs.

Adding sparse loading further reduces TTFT to 86/140/137 ms, an additional reduction of 8.5%–11.9% relative to the preceding configuration. The offline indices allow the system to skip KVs that will be overwritten by recomputation and transfer only the reuse complement. At $r = 15\%$, the effective KV transfer volume is theoretically reduced by approximately 15%. Because the pipeline

already hides part of the transfer, this reduction in data volume does not translate proportionally into lower TTFT.

Finally, adding deferred RoPE reduces TTFT to 82/134/129 ms, a further reduction of 4.3%–5.8% relative to the preceding configuration. Global positional recovery for reused Keys is incorporated into the asynchronous execution path, reducing synchronization waits for positional correction in the main forward stream. This optimization changes when positional recovery is performed and provides a smaller benefit than the preceding two optimizations. With all three enabled, the full configuration reduces TTFT by 53.6%–54.2% compared with the configuration without cache-loading optimizations.

Calibration for SSD/HDD caches. We compare a fixed ratio of $r = 15\%$ with hardware-aware calibration. Figure 6 presents the TTFT of each method, and Table 5 summarizes the average speedups with fixed and calibrated ratios. With a fixed ratio of $r = 15\%$, the HDD and SSD configurations achieve TTFT speedups of $1.92\times$ and $2.05\times$, respectively, over full recomputation in their corresponding configurations. Hardware-aware search selects $r = 36.4\%$ for the HDD configuration and $r = 30.9\%$ for the SSD configuration, increasing the speedups to $2.36\times$ and $2.34\times$, respectively. Both configurations reduce cache reads by increasing recomputation, showing that assigning more work to the GPU can improve end-to-end latency when disk access is expensive.

Table 5: Comparison of fixed and calibrated recomputation ratios. Speedups are relative to full recomputation in each configuration.

	Fixed		Calibrated	
Cache pool	r	Speedup	r	Speedup
HDD	15%	1.92×	36.4%	2.36×
SSD	15%	2.05×	30.9%	2.34×

The selected ratios reflect the combined costs of computation and transfer in each configuration. A100/HDD and RTX 4090/SSD differ in GPU compute capability and storage bandwidth, and calibration seeks the balance between the two execution paths separately for each. The HDD configuration uses a higher ratio, moving more work away from the slower read path, whereas the SSD configuration retains more cache reuse. Compared with the 15% used in the CPU memory setting, these two results further show that the recomputation budget serves both quality recovery and the reduction of slow I/O. The calibration of r minimizes measured TTFT while keeping the empirical quality-preserving lower bound $r_{\min}$ fixed, without using reference answers. With the same model and hardware configuration, the calibrated ratio can be reused across tasks with similar workload characteristics without task-specific recalibration.

F Discussion and Limitations

Our experimental results show that CacheTune's advantage stems from the tight coordination of its three key components. Unlike prior approaches that rely on heuristic rules, attention relationships, or learned signals to determine recomputation targets online, CacheTune constructs a query-independent token-importance ranking for each KV Chunk during cache generation based on its intrinsic frequency-domain characteristics. This ranking is used not only to identify the critical KVs that should be recomputed under the global context, but also directly as a cache-loading plan, allowing the system to transfer only the complementary KVs that will be reused. In this way, CacheTune transforms the conventional full-cache loading followed by partial repair workflow into a complementary KV-construction process based on selective recomputation and sparse transfer, thereby reducing I/O overhead. Under this execution model, the recomputation ratio jointly determines GPU computation and data-transfer overhead across cache media. The hardware-aware scheduler therefore adapts the workload allocation between the recomputation and transfer paths according to the compute and I/O characteristics of the target storage tier.

CacheTune's cache-pool abstraction further has the potential to extend to emerging heterogeneous memory architectures. For example, with memory-interconnect technologies such as Compute Express Link (CXL) (Das Sharma et al., 2024; Seo et al., 2025), extended memory beyond the GPU and CPU can be organized as a more unified remote cache pool; CacheTune can still estimate the

effective transfer cost through hardware profiling and adaptively tune the recomputation–reuse ratio. CacheTune nevertheless has several limitations. When external-storage bandwidth is extremely low or the asynchronous overlap window is insufficient, the system may remain I/O-bound. In future work, we will further evaluate CacheTune under higher concurrency, CXL-based disaggregated memory, and more complex workloads.