Compress & Blend

Zihan Zheng
University of Illinois Urbana-Champaign
zihanz23@illinois.edu

Ziheng (Jack) Chen
University of Illinois Urbana-Champaign
zihengc2@illinois.edu

Abstract

Recent developments in retrieval-augmented generation (RAG) and reasoning-intensive prompting have emphasized the critical need to scale large language models (LLMs) along the sequence length dimension. As input sequences grow longer, the key-value (KV) cache incurs greater memory overhead, and the prefill stage becomes increasingly time-consuming. Several methods—such as CacheBlend, xKV, and H2O—have been proposed to address these challenges individually, but their interactions have not been thoroughly examined. To jointly leverage these techniques without compromising generation quality, this project develops a unified framework that applies and tunes them concurrently. Built on top of the vLLM serving platform, our system benchmarks the aforementioned methods, analyzes their compatibilities, and proposes a novel compress-and-blend workflow that integrates their respective advantages.

1 Introduction

The emerging Retrieval-Augmented Generation (RAG) and reasoning-intensive applications increasingly demand the processing of long input sequences, leading to significant performance challenges. Longer contexts result in higher memory footprints, increased compute overhead, and degraded system-level metrics such as time-to-first-token (TTFT) and inference throughput, due to the quadratic complexity of attention and the linear growth of KV cache storage with sequence length.

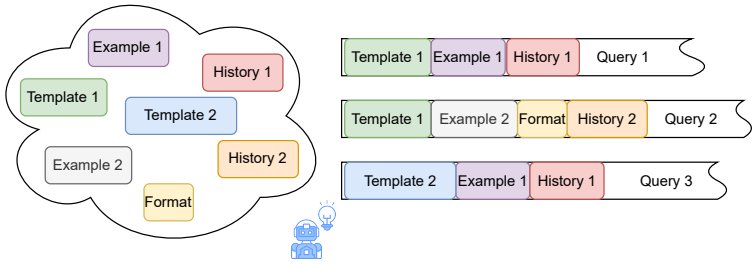

Figure 1: Illustration of RAG document reuse across queries. Queries may use multiple documents, share documents with other queries, and consume them in varying orders.

However, not all computation within long sequences is inherently necessary. As illustrated in Figure 1, queries in a RAG system often utilize multiple documents. These documents may also be shared across different queries and consumed in varying orders depending on the context. For example, in an agentic framework, a user request could require combining templates, examples, histories, and output formats to generate an appropriate response. These characteristics highlight the potential for reuse and further motivate the need for efficient KV cache management. During the serving of these

CS 498 Machine Learning System, Spring 2025.

requests, multiple retrieved document chunks are concatenated into a single long sequence before computing the KV cache, resulting in substantial prefill overhead. CacheBlend [Yao et al., 2025] addresses this inefficiency by storing the KV caches of individual chunks separately and merging them efficiently with only partial recomputation. This significantly reduces the compute cost of the prefill stage and improves TTFT, without compromising input coverage. However, it still requires maintaining full-size KV caches for all chunks, resulting in a high memory footprint that limits scalability, especially in multi-query or multi-document scenarios.

Another line of work reduces computation by dropping tokens deemed less important. H2O [Zhang et al., 2023] identifies and retains only the most impactful tokens—recent ones and a few "heavy hitters" with high attention scores—while discarding the rest. This reduces KV cache size and improves throughput, but still requires a full prefill pass and introduces additional overhead to evaluate token importance. StreamingLLM [Xiao et al., 2024] takes a more aggressive approach, keeping only the most recent tokens and a small set of fixed "attention sink" tokens. While this enables efficient streaming and low memory usage, it risks losing crucial contextual information, especially for tasks that depend on long-range dependencies.

Complementary to token dropping, KV cache compression methods like xKV [Chang et al., 2025] reduce memory usage by projecting key and value tensors into low-rank subspaces using techniques such as cross-layer SVD. This allows multiple layers to share a compact representation, significantly reducing KV size without retraining. However, despite their individual benefits, these techniques—whether merging KV caches, dropping tokens, or compressing representations—have largely been developed in isolation. Making them work together requires significant engineering effort, and their interactions are poorly understood. For example, dropping tokens may distort compression patterns, or merging may conflict with layer-wise factorization. This motivates the need for a unified framework that supports all three techniques in a coherent and compatible manner.

To address these challenges, we propose a unified scaffolding framework that enables joint optimization of KV cache recomputation, eviction, and compression. Our framework generalizes the recompute logic from CacheBlend, introduces a modular cache modifier for controlled eviction and compression, and supports systematic exploration of their combinations. Central to our design is a token categorization scheme that identifies intrinsic, relational, and dummy tokens—allowing fine-grained control over which tokens to recompute, retain, or discard. This principled approach facilitates more effective allocation of compute and memory, improving efficiency while preserving generation quality.

2 Background

2.1 KV Cache Eviction

Recent works have actively explored KV cache eviction strategies and streaming solutions for large language models (LLMs) to enable efficient and scalable inference. Notable approaches include StreamingLLM [Xiao et al., 2024], NACL [Chen et al., 2024], and H2O [Zhang et al., 2023].

StreamingLLM [Xiao et al., 2024] leverages the observation that LLMs naturally assign strong attention to initial tokens, which act as "attention sinks." During inference, it explicitly preserves a small number of these early-position tokens—typically the first 4—alongside a sliding window of the most recent tokens. This strategy requires no fine-tuning and is applied throughout autoregressive generation. By maintaining the key-value (KV) cache entries corresponding to these sink tokens, StreamingLLM stabilizes attention distributions and mitigates the degradation observed in naive window attention, which otherwise suffers from performance collapse when early tokens are evicted. The retained sink tokens serve no semantic role but anchor the attention distribution during generation.

H2O [Zhang et al., 2023] introduces a dynamic token selection strategy based on accumulated attention statistics. At each decoding step, it computes a local importance score for each token by summing its attention weights across heads and layers. Tokens with the highest cumulative attention—termed "heavy hitters"—are retained in the KV cache, along with a fixed number of most recent tokens to preserve local context. This greedy eviction policy is implemented online during generation and enables substantial cache size reduction with minimal degradation in output quality. However, because token importance is determined solely from local statistics, the method may become biased in long sequences or fail to retain task-relevant but low-attention tokens.

NACL [Chen et al., 2024] addresses the limitations of local attention-based eviction by combining global and randomized strategies. Unlike H2O, NACL performs a one-shot global eviction during the encoding phase, where it computes token importance scores using "proxy tokens"—typically those at the end of the prompt, such as questions in QA tasks—to evaluate the relevance of earlier tokens. The resulting importance scores are used to rank and select a subset of tokens for retention in the KV cache. During decoding, NACL applies lightweight randomized eviction, injecting diversity into token selection by randomly dropping tokens based on soft importance distributions with head- and layer-level perturbations. This hybrid scheme improves robustness by mitigating attention bias and ensures that diverse yet relevant tokens are maintained across different heads and layers during generation.

While all three methods provide efficient solutions to streaming and KV cache management, they face a shared challenge: preserving task-critical information becomes increasingly difficult under aggressive cache size constraints. Consequently, performance degradation becomes inevitable as the retained token set shrinks or as sequences grow extremely long. Despite their clear gains in throughput and memory efficiency, these methods highlight the fundamental trade-off between scalability and fidelity in long-context LLM deployment, underscoring the need for further research in adaptive, context-aware caching strategies.

2.2 KV Cache Concatenation and Recomputation

While reusing pre-computed KV caches offers the promise of reducing prefill latency, it turns out that leveraging multiple cached chunks is not as simple as concatenating them together.

As illustrated in Figure 2, at the first layer, concatenating pre-computed KV chunks appears straightforward. Each chunk's key and value vectors can be concatenated with proper handling of positional embeddings, allowing the queries to attend across all retrieved chunks. This setup largely preserves the semantics of the original input.

However, complications arise from the second layer onward. Starting with the first attention output, each token's representation has already incorporated cross-chunk information due to the softmax-weighted aggregation over all value vectors. The outputs from the attention module thus encode mixed information, combining signals from all chunks. When these mixed representations are projected to form the KV caches of subsequent layers, they are no longer cleanly separable by chunk. Instead, the new keys and values contain holistic information integrated from all input chunks. Therefore, naive concatenation becomes insufficient since concatenating KV caches from precomputed chunks in higher layers does not produce representations equivalent to those generated from processing the combined input directly.

Prior works on KV cache management, such as H2O and Attention Sink [Zhang et al., 2023], reveal that not all KV entries contribute equally to the attention computation. Many tokens contribute marginally and can even be safely evicted or downsampled without noticeably affecting model quality. Therefore, even if concatenating precomputed KV caches introduces inaccuracies, the degradation may be minor and acceptable, particularly for long contexts where exact attention patterns are less critical. Selective recomputation techniques, as seen in CacheBlend [Yao et al., 2025] and EPIC [Hu et al., 2025], can further mitigate any adverse effects by updating only the most sensitive parts of the KV cache.

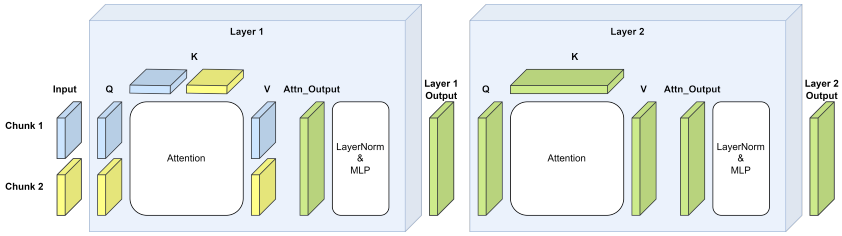

Figure 2: Illustration of KV cache concatenation across layers. In Layer 1, KV chunks can be concatenated with proper position handling. From Layer 2 onward, keys and values already contain integrated information due to cross-attention and cannot be simply concatenated.

CacheBlend addresses the limitations of simple prefix caching and full KV reuse in retrieval-augmented generation (RAG), particularly in scenarios where reused text chunks are no longer positioned at the start of the prompt. Naively concatenating precomputed KV caches leads to degraded performance, as the model fails to capture cross-chunk interactions. To mitigate this, CacheBlend introduces a selective recomputation strategy. It begins by performing full recomputation on the first layer to obtain updated attention maps, then compares these to the attention computed from the reused (precomputed) KV cache. By identifying the tokens that contribute most to the mismatch between the two, CacheBlend selects a small subset—typically around 10–16%—of the most impactful tokens. These tokens are then recomputed across all subsequent layers to update their KV entries, while the rest continue to use the original cached values. The final KV cache, now a hybrid of recomputed and reused entries, is passed into the decoding phase. This allows the model to recover critical cross-attention patterns without incurring the full cost of recomputing the entire prompt.

EPIC improves upon CacheBlend by simplifying the recomputation process. Instead of relying on attention comparisons to guide selection, EPIC uses a fixed heuristic based on the observation that initial tokens in each chunk tend to dominate attention distributions in later layers—a phenomenon known as the attention sink. EPIC statically selects the first k tokens of each chunk (with k typically small, such as 2, 4, or 8) and recomputes only these tokens across all layers. The remaining tokens use their precomputed KV values without modification. This eliminates the need for computing attention deviations and reduces recomputation complexity from CacheBlend's quadratic form to $O(kN)$, where $k \ll N$. While EPIC is more efficient and easier to implement, its static nature means it cannot adapt to token importance in context, which may lead to missed recomputation opportunities and a slight decrease in generation quality.

In both approaches, the selected tokens are recomputed before generation begins and inserted back into the KV cache. During autoregressive decoding, the model then attends over this modified KV cache—composed of both recomputed and reused entries—to produce the output. CacheBlend provides better accuracy due to its dynamic and context-aware selection but incurs more computation, while EPIC offers speed and simplicity at the cost of modest quality degradation.

Overall, these methods demonstrate two ends of the quality-performance tradeoff: one adaptive and fine-grained, the other static and efficient. Determining the optimal recomputation strategy and token selection granularity remains an open question, dependent on application workload and desired performance constraints.

2.3 KV Cache Compression

xKV tackles KV cache optimization via compression rather than recomputation, focusing on reducing memory footprint by exploiting redundancy across transformer layers. Its core insight is that dominant singular vectors of KV matrices are often well-aligned across layers.

To leverage this, xKV applies cross-layer SVD during the prefill phase. It groups KV matrices from several layers, performs SVD on the concatenated matrix, and factorizes it as:

$$[X_{\ell_1}, \ldots, X_{\ell_n}] \approx A[B_{\ell_1}, \ldots, B_{\ell_n}]$$

where A is a shared low-rank token basis, and each B_ℓ is a layer-specific projection. Only these compressed forms are stored, substantially reducing memory usage.

During decoding, xKV reconstructs KV entries on-the-fly using A and B_ℓ, introducing minimal overhead while preserving generation quality. It is orthogonal to reuse-based strategies like CacheBlend and EPIC, and can complement them by compressing reused or growing KV caches.

Experiments show up to 6.8$\times$ compression with negligible or even improved accuracy, making xKV a lightweight and practical addition to long-context LLM inference.

2.4 Summary and Open Questions

In summary, prior works naturally form two optimization axes: (i) Token Selection and Recomputation: determining which and how many tokens to recompute, trading off compute cost and generation quality. (ii) Token Eviction and Compression: managing unselected tokens to save memory, either by eviction or compression.

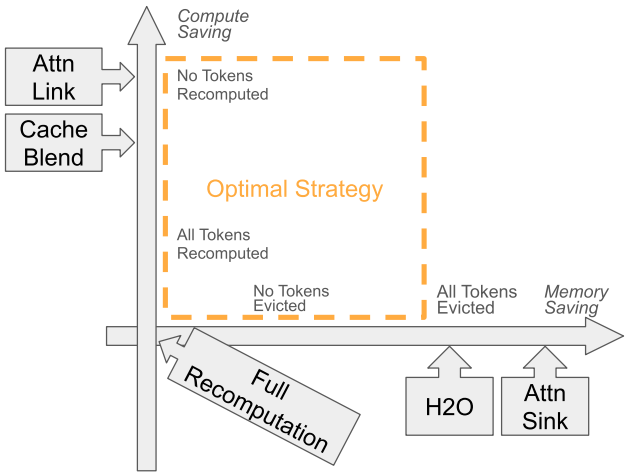

Figure 3: Design space of existing methods along compute saving (recomputation) and memory saving (eviction) axes. The optimal strategy should lie somewhere in between.

As shown in Figure 3, full recomputation lies at the origin of this design space, consuming maximum compute and memory. Moving right along the memory-saving axis, methods like H2O and Attention Sink aggressively evict low-importance tokens. Moving upwards along the compute-saving axis, methods like CacheBlend and EPIC selectively recompute a fraction of tokens, with EPIC being more aggressive by recomputing only the first few tokens of each chunk. Existing solutions generally operate along or near these axes.

However, we argue that the optimal strategy lies in between these two axes. Neither pure recomputation nor pure eviction offers the best trade-off alone. Instead, an ideal approach should judiciously combine selective recomputation with selective eviction or compression. This motivates the key research questions that our project seeks to address, given a RAG document with precomputed KV cache: (i) what is the minimum recomputation required to retain performance? (ii) What is the optimal token selection method to determine which tokens need recomputation? (iii) What is the optimal recompute budget, or number of tokens to recompute, for balancing latency and quality? (iv) How should unselected tokens be managed? Should they be evicted entirely or compressed into a more compact representation?

3 Methodology

3.1 Scaffolding Framework for Joint Optimization

To explore the design space of recomputation and eviction strategies, we develop a unified scaffolding framework that enables joint tuning of generation quality and system efficiency. As shown in Figure 4, the framework consists of two main components: the Recompute Engine and the KV Cache Modifier.

Recompute Engine We generalize and modularize the layerwise recomputation logic inspired by CacheBlend, exposing interfaces to easily implement and compare various token selection strategies (e.g., EPIC [Hu et al., 2025], CacheBlend [Yao et al., 2025]). This allows flexible control over how many and which tokens are recomputed per layer.

KV Cache Modifier To evaluate eviction and compression schemes, we introduce a KV cache modifier. This module provides an interface for manipulating the KV cache during generation by bridging the vLLM serving platform and the HuggingFace Transformers library. It enables both cache eviction and custom compression strategies. When interacting with the Transformers library, we focus solely on generation quality, as time-to-first-token (TTFT) and generation speed are not considered due to unoptimized KV cache management overhead.

Proposed Compress-and-Blend Workflow We observe that the recomputation method introduced in CacheBlend [Yao et al., 2025] effectively treats the KV cache of unselected tokens as ground truth and uses them to partially correct the next layer's attention scores. This implies that dropping any unselected tokens may degrade accuracy, as their information is lost. Motivated by this insight, the compress-and-blend workflow retains all tokens during the prefill recomputation stage and defers token dropping to the decode stage. Furthermore, this observation also informs the token categorization introduced in the next section.

Overall, this scaffolding serves as a flexible platform for systematic exploration of recomputation and eviction trade-offs in KV cache optimization.

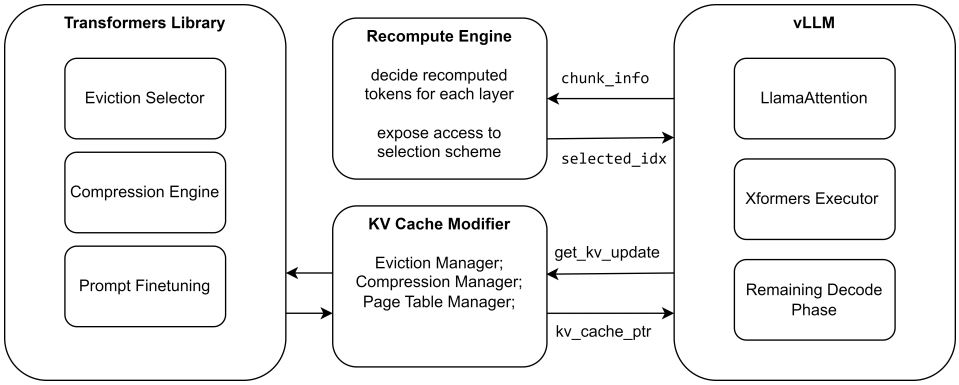

Figure 4: Scaffolding framework design. It decouples recomputation and eviction to enable separate evaluation of accuracy and performance.

3.2 Token Categorization

To systematically manipulate KV cache entries for recomputation and eviction, we introduce a categorization scheme that captures the diverse behaviors of tokens across layers and contexts. The motivation is simple: recomputation and eviction are only viable because tokens do not contribute equally to generation quality. However, existing methods largely ignore or simplify these differences.

Through observation and analysis, we identify three distinct categories of tokens:

Intrinsic Tokens These tokens are essential and *intrinsic* to their respective data chunks. Although their KV values do not change significantly after recomputation, they tend to hold large values and play critical roles in maintaining chunk-local semantics. As such, they must be preserved, and are not suitable for eviction or aggressive recompression.

Relational Tokens These tokens exhibit small self-attention within the chunk but become important due to strong *cross-attention* with tokens from other chunks. As a result, they have high deviation when using precomputed KV caches and almost always require recomputation. Relational tokens often drive inter-chunk interactions and directly impact generation correctness.

Dummy Tokens These tokens maintain consistently small values and limited attention across both self and cross interactions. As a result, they are likely candidates for eviction or compression, contributing little to final output quality if removed or downsampled.

Importantly, token behavior is dynamic and layer-dependent. A token classified as dummy in early layers may become relational later due to accumulated cross-chunk interactions. Existing eviction-oriented methods such as H2O [Zhang et al., 2023], which rely heavily on accumulated attention scores, may miss this subtlety; some tokens with low scores may still become high-deviation after recomputation.

Moreover, certain templates and prompt tokens may exhibit relational behavior—carrying little intrinsic importance themselves but potentially attracting cross-attention from other tokens. These nuanced patterns are difficult to capture with simple heuristics, suggesting the potential benefit of more sophisticated token selection schemes.

In summary, this token categorization serves as a guiding principle for designing more sophisticated recomputation and eviction strategies. By understanding token roles, we can better allocate compute and memory resources to maximize efficiency while retaining generation quality.

4 Result

4.1 Reproducing Attention Sink

To validate and better understand the attention sink phenomenon, we reproduced the analysis on Mistral-7B [Jiang et al., 2023]. Specifically, we extracted attention maps from two representative layers (Layer 0 and Layer 10) and two heads (Head 0 and Head 10), as shown in Figure 5. This allows us to observe the behavior across both layers and heads.

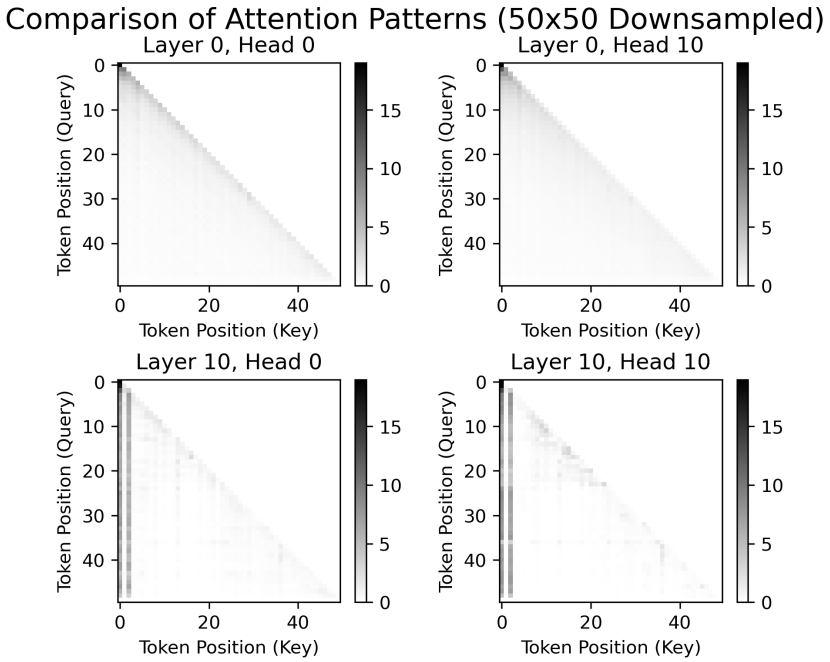

Figure 5: Reproducing attention sink on Mistral-7B. The sink behavior emerges at later layers and differs across attention heads.

Observations The reproduced attention patterns confirm two key insights. First, sink behavior tends to emerge more strongly at later layers. While Layer 0 exhibits relatively smooth and local attention distributions, Layer 10 reveals the presence of distinct tokens that absorb disproportionate attention, forming clear sink patterns. Second, this behavior is not uniform across heads. For example, Head 0 maintains a more evenly distributed attention pattern, whereas Head 10 displays stronger signs of sink formation, with certain tokens dominating attention. These differences suggest that the emergence of sink behavior is both depth- and head-dependent.

Implications These findings have important implications for KV cache management. Since sink behavior is more prominent in deeper layers and varies across heads, this opens up the possibility for more fine-grained compression and eviction strategies that are aware of layer and head characteristics, rather than relying on uniform global policies. Moreover, the context-dependent nature of sink formation indicates that adaptive schemes—those that adjust token selection based on current attention dynamics—may offer better trade-offs between memory efficiency and generation quality. Such insights further support the motivation for our proposed layer-wise and context-aware KV cache manipulation framework.

In summary, our reproduction results justify the need for layer-wise and context-aware KV cache manipulation, supporting the direction taken by our proposed framework and token categorization methodology.

4.2 Reproducing CacheBlend and EPIC with Our Framework

As a second validation, we demonstrate that our proposed framework can seamlessly support various recomputation strategies. In this experiment, we showcase CacheBlend and EPIC (with different hyperparameters) to verify compatibility and compare their behaviors.

Table 1: Testing dataset and baseline outputs from full recomputation.

ID	Question	Full Recompute (Baseline)
1	Where did the performer of song I'll Say It graduate from?	Carnegie Mellon University.
2	Which film has the director who was born first, Hell Up In Harlem or The Soviet Story?	The Soviet Story (older)
3	Who was born first, Cipriano Castro or Damir Nikšić?	Cipriano Castro was born first.
4	Which song came out first, Joel The Lump Of Coal or Jugband Blues?	Jugband Blues came out first.
5	What is the place of birth of Ratna Malla's father?	Bulacan, Philippines.
6	Who is Archibald Acheson, 4th Earl Of Gosford's paternal grandfather?	Arthur Acheson, 1st Earl of
7	Where did the performer of song Fantasy (George Michael Song) die?	At home in Goring-on-Thames
8	Do director of film Betrayal (1932 Film) and director of film The Godsend (Film) share the same nationality?	Yes, both are French.
9	Which film whose director was born first, The Abduction Club or Wooden Crosses?	Wooden Crosses (1932)
10	Which film came out earlier, Above Rubies or The Magic Aster?	Above Rubies came out earlier.

Since this project is in its early stages, we have not yet conducted full-scale benchmark evaluations. However, initial case studies have already yielded meaningful insights. All experiments in this work are conducted on a dataset of 10 queries provided by CacheBlend. Each query is paired with 5-6 context passages, simulating a RAG setup. This structure presents a challenging testbed for evaluating semantic accuracy, as the model must synthesize answers from multiple relevant sources. All subsequent experiments are based on this dataset. Table 1 lists the questions and the baseline responses generated using full KV cache recomputation over the provided context chunks. Table 2 shows qualitative comparisons across different recomputation strategies.

Table 2: Example outputs from different recomputation schemes and their Time To First Token (TTFT). Incorrect results are highlighted in red, and differences are shown in blue.

ID	Full Recompute (100%)	CacheBlend (16%)	EPIC (sink=50)	EPIC (sink=2)
1	Carnegie Mellon University. TTFT: 0.25s	Carnegie Mellon University. TTFT: 0.074s	Graduated from Carnegie Mellon. TTFT: 0.058s	Graduated from Carnegie Mellon. TTFT: 0.07s
5	Bulacan, Philippines. TTFT: 0.25s	Bulacan, Philippines. TTFT: 0.066s	Bulacan, Philippines. TTFT: 0.047s	Unknown, mentioned as Manna TTFT: 0.03s
6	Arthur Acheson, 1st Earl of TTFT: 0.26s	Arthur Acheson, 1st Earl of TTFT: 0.067s	Archibald Stewart, Earl of Mar. TTFT: 0.047s	Archibald Acheson, 2nd TTFT: 0.03s
9	Wooden Crosses (1932) TTFT: 0.24s	Wooden Crosses (1932) TTFT: 0.065s	Wooden Crosses (1932) TTFT: 0.045s	Wooden Crosses (earlier birth) TTFT: 0.03s

Observations We observe that while EPIC achieves lower TTFT across the board—reaching nearly 2× speedup over CacheBlend—its accuracy deteriorates in some cases. Incorrect or hallucinated tokens appear, especially at lower sink sizes (e.g., “earlier birth" and “Unknown, mentioned as Manna"). CacheBlend, on the other hand, strikes a more balanced tradeoff: achieving around **30% of the TTFT compared to full recomputation**, while preserving correct answers in almost all shown examples.

Implications These results suggest that while aggressive recomputation reduction (as in EPIC) is beneficial for latency, it can harm generation quality. Therefore, finer-grained and context-aware recomputation policies may be required for better trade-offs, which our framework is well suited to support.

In summary, our early results validate that the framework flexibly supports a variety of schemes and provides a valuable tool to explore and analyze trade-offs between speed and quality.

4.3 Examining the Token Selection Mechanism in CacheBlend

This subsection evaluates alternative token selection strategies within the CacheBlend framework. We analyze how selection based on K vs. V projections, selection at deeper transformer layers, and choosing mid-ranked rather than top-ranked tokens impact output quality and latency. These experiments highlight the trade-offs and limitations of current heuristics, offering insights for more robust token selection in recomputation-based acceleration.

4.3.1 Selecting Importance from *K*

In the original CacheBlend method, important tokens are selected based on the deviation of the value (V) projections relative to their fully recomputed counterparts. However, since both the key (K) and value (V) components contribute to the attention mechanism, we explored an alternative strategy: selecting important tokens based on deviations in K.

The results of this comparison are summarized in Table 3. Among all configurations, the "no selection" approach is the fastest, requiring only about 10% of the computation time compared to full recomputation. CacheBlend, which selects and recomputes the top 16% of tokens based on V deviation, uses roughly 40% of the total time. Interestingly, selecting tokens based on K deviation does not always align with the outputs produced under the "no recomputation" setting, and in some cases, produces different results. This highlights that K-based importance may not be fully interchangeable with V-based selection in preserving semantic output fidelity.

Table 3: Example outputs from different recomputation schemes and their Time To First Token (TTFT). Incorrect results are highlighted in red, and differences are shown in blue.

ID	Full Recompute (100%)	Imp-from-V (baseline)	Imp-from-K	No Selection
5	Bulacan, Philippines. TTFT: 0.255s	Bulacan, Philippines. TTFT: 0.067s	Bulacan, Philippines TTFT: 0.067s	Unknown, mentioned as "a village in bul TTFT: 0.030s
6	Arthur Acheson, 1st Earl of TTFT: 0.256s	Arthur Acheson, 1st Earl of TTFT: 0.067s	King Robert II of Scotland. TTFT: 0.068s	Archibald Acheson, 2nd TTFT: 0.030s
9	Wooden Crosses (1932) TTFT: 0.242s	Wooden Crosses (1932) TTFT: 0.066s	Fernando Ayala (The Abduction Club) TTFT: 0.066s	Wooden Crosses (older) TTFT: 0.029s

4.3.2 Selecting Important Tokens at Later Layers

In the original CacheBlend approach, important tokens are identified at the first transformer layer by comparing the value (V) projections against their fully recomputed counterparts and selecting those with the highest L2 norm differences. In this experiment, we modified the strategy by deferring token selection to later layers—specifically, the second and third layers—while fully recomputing all earlier

layers, shown in Table 4. This ensures that the early-layer representations remain identical to the baseline but incurs additional computation, as reflected by the increased Time To First Token (TTFT) for deeper selection layers.

Surprisingly, this change did not improve output quality. When token selection is performed at the second layer, the output deviates from the baseline yet remains semantically correct. However, selecting at the third layer leads to significant semantic errors in the generated text. We hypothesize that this is due to the reduced distinctiveness of token importance at later layers. As discussed in 2.2, each token's representation in the attention module already incorporates cross-chunk information due to softmax-weighted aggregation over all value vectors. As a result, by the time these representations are projected into the KV cache of subsequent layers, they encode blended signals from all chunks. This mixing effect makes it difficult to isolate the importance of individual tokens, which may explain the degradation in accuracy when selection is delayed to later layers.

Table 4: Example outputs from different recomputation schemes and their Time To First Token (TTFT). Incorrect results are highlighted in red, and differences are shown in blue.

ID	Select 1st Layer (Baseline)	Select 2nd Layer	Select 3rd Layer
1	Carnegie Mellon University. TTFT: 0.074s	Carnegie Mellon University. TTFT: 0.083s	Graduated from West Berlin. TTFT: 0.157s
2	The Soviet Story was made first. TTFT: 0.065s	The Soviet Story was made later. TTFT: 0.070s	Hell Up In Harlem. (Lee Thompson TTFT: 0.144s
5	Bulacan, Philippines. TTFT: 0.067s	Bulacan, Philippines. TTFT: 0.073s	Ducal house of Bourbon. TTFT: 0.152s
6	Arthur Acheson, 1st Earl of TTFT: 0.067s	Arthur Acheson, 1st Earl of TTFT: 0.073s	King Robert II of Scotland. TTFT: 0.156s

4.3.3 Selecting Tokens with Lower Importance

Finally, we investigate the effect of selecting tokens with lower importance while keeping the total number of selected tokens fixed. As shown in Table 5, instead of selecting the top 16% of tokens based on the highest deviation in the value (V) projection, we shift the selection window downward to include tokens ranked 4%-20%, 8%-24%, 12%-28%, and 16%-32%, ensuring no overlap with the top-ranked tokens used in the baseline.

As expected, overall output quality degrades when less important tokens are selected. However, an interesting observation emerges in question 2: selecting tokens in the 8%-24% range leads to improved accuracy compared to the 4%-20% range. This suggests that highly influential tokens—those critical to the final answer—may not always reside in the top-most positions by L2 deviation. Instead, they can occasionally fall just outside the top 16%, highlighting the potential of alternative selection strategies that better capture semantically vital information.

4.4 EPIC as a Subset of CacheBlend

To better understand the relationship between EPIC and CacheBlend, we conducted an analysis to compare their token selection behaviors. Specifically, we fixed a single chunk and paired it with five different chunks as prefixes to generate diverse cross-attention patterns. We then observed which tokens in the fixed chunk were selected for recomputation under each method.

Observations As shown in Figure 6, several key trends emerge from the token selection patterns. First, we observe that sink tokens are almost always selected, regardless of the prefix context. These relational tokens tend to appear consistently across different configurations, aligning closely with EPIC's selection behavior. This suggests that EPIC's token selection can be interpreted as a subset of what CacheBlend identifies as important. Second, CacheBlend selects additional tokens beyond those chosen by EPIC. These extra tokens, which are not necessarily sink-related, may contribute to CacheBlend's ability to preserve generation quality, as evidenced in earlier experimental results. Finally, tokens located toward the end of the chunk are rarely selected under any prefix configuration.

Table 5: Example outputs from different layer selection strategies and their Time To First Token (TTFT). Incorrect results are highlighted in red, and differences are shown in blue.

ID	0% - 16% (Baseline)	4% - 20%	8% - 24%	12% - 28%	16% - 32%
2	The Soviet Story was made first. TTFT: 0.065s	The Soviet Story was made first. TTFT: 0.065s	Hell Up In Harlem (1973 TTFT: 0.065s	Hell Up In Harlem (older) TTFT: 0.065s	Hell Up In Harlem (older) TTFT: 0.065s
4	Jugband Blues came out first. TTFT: 0.068s	Joel The Lump Of Coal came out first. TTFT: 0.068s	Jugband Blues came out first. TTFT: 0.068s	Jugband Blues came first. TTFT: 0.068s	Jugband Blues came first. TTFT: 0.068s
5	Bulacan, Philippines. TTFT: 0.067s	Bulacan, Philippines. TTFT: 0.068s	Bulacan province. TTFT: 0.067s	Unknown, mentioned as "village" TTFT: 0.067s	Unknown, mentioned as "village" TTFT: 0.067s
6	Arthur Acheson, 1st Earl of TTFT: 0.067s	King Robert II of Scotland TTFT: 0.068s	Archibald Acheson, 2nd TTFT: 0.068s	Archibald Acheson, 2nd TTFT: 0.068s	Archibald Acheson, 2nd TTFT: 0.068s
9	Wooden Crosses (1932) TTFT: 0.066s	Wooden Crosses (1932) TTFT: 0.066s	Wooden Crosses was first released. TTFT: 0.066s	Wooden Crosses was made first. TTFT: 0.066s	Wooden Crosses (1960) TTFT: 0.066s

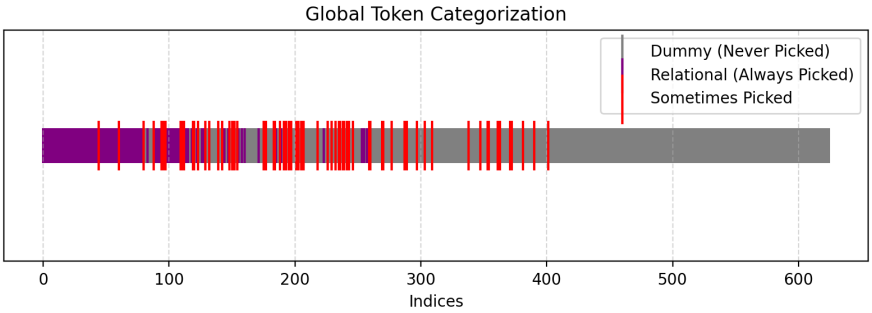

Figure 6: Token categorization across CacheBlend recomputations. “Relational” tokens are always picked, while “Dummy” tokens are never picked. Tokens in red are sometimes picked.

These appear to function as “dummy” tokens with minimal contribution to cross-chunk attention, and therefore present opportunities for compression or eviction in future designs.

Implications This comparative analysis reveals fundamental differences in how EPIC and CacheBlend prioritize tokens. While EPIC takes a narrow focus on sink tokens, CacheBlend appears more adaptive to context, selecting a richer set of positions that may be critical for generation fidelity. The clear distinction between consistently selected relational tokens and consistently ignored dummy tokens reinforces the value of fine-grained token categorization. These findings support the hypothesis that context-aware and layer-sensitive selection strategies are essential for optimizing both inference performance and output quality.

Overall, our results underscore the importance of token characterization in LLM inference and motivate the design of unified frameworks that jointly explore recomputation, compression, and eviction strategies.

5 Conclusion and Future Work

In this work, we presented a framework that enables efficient reuse of RAG documents through precomputed KV caches, with the potential to significantly reduce both compute and memory costs

during inference. By supporting techniques such as EPIC, CacheBlend, and Attention Sink within a unified evaluation and tuning framework, our solution opens up new opportunities for design space exploration across recomputation and compression strategies.

As illustrated in Figure 1, our approach targets the underexplored middle ground between fully recomputing KV caches and storing them entirely in memory. This space offers promising potential to balance compute savings and memory efficiency, especially in agentic and RAG-heavy workloads where document reuse is frequent.

Potential Impact Our framework significantly reduces compute and memory consumption by reusing precomputed KV caches, which is especially beneficial in memory-constrained or latency-sensitive environments. Additionally, it enables systematic design space exploration (DSE) for various recomputation and compression strategies, helping researchers and practitioners better understand trade-offs and fine-tune systems for real-world use cases.

Limitations At present, the implementation is model-specific and has only been evaluated on Llama-based models. As a result, the generalizability of our approach to other architectures remains an open question and warrants further investigation.

Work in Progress Ongoing efforts include developing offline optimization strategies to identify optimal KV cache compression schemes tailored to specific workloads. We are also investigating methods for dynamically adjusting sequence length across layers to further improve memory efficiency. In addition, we are studying the impact of various KV cache eviction policies on recomputation performance to guide the design of more robust reuse mechanisms.

In summary, this framework serves as a stepping stone towards more efficient and scalable RAG-based generation. We believe it provides valuable opportunities for advancing both system-level optimizations and algorithmic design in future LLM inference research.

6 Collaboration

All authors contributed to discussions throughout the project. The final presentation and report were joint efforts.

Zihan Zheng

- **Design**
  - Propose the *Compress-and-Blend* framework
  - Propose the architecture of the KV cache scaffolding
  - Identify token recompute behaviors for categorization
- **Implementation**
  - Reproduce attention sink analysis and EPIC method based on vLLM
  - Design and implement the KV cache modifier

Ziheng (Jack) Chen

- **Implementation**
  - Reproduce CacheBlend
  - Generated and compared results with various token selection methods in CacheBlend
  - Ran experiments to categorize tokens during recomputation

References

C.-C. Chang, C.-Y. Lin, Y. Akhauri, W.-C. Lin, K.-C. Wu, L. Ceze, and M. S. Abdelfattah. xkv: Cross-layer svd for kv-cache compression, 2025. URL https://arxiv.org/abs/2503.18893.

Y. Chen, G. Wang, J. Shang, S. Cui, Z. Zhang, T. Liu, S. Wang, Y. Sun, D. Yu, and H. Wu. NACL: A general and effective KV cache eviction framework for LLM at inference time. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7913–7926, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.428. URL https://aclanthology.org/2024.acl-long.428/.

J. Hu, W. Huang, H. Wang, W. Wang, T. Hu, Q. Zhang, H. Feng, X. Chen, Y. Shan, and T. Xie. Epic: Efficient position-independent context caching for serving large language models, 2025. URL https://arxiv.org/abs/2410.15332.

A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

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

J. Yao, H. Li, Y. Liu, S. Ray, Y. Cheng, Q. Zhang, K. Du, S. Lu, and J. Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 94–109, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. Wang, and B. Chen. H2o: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.