Semantic-Aware Chunk-Level KV Cache Eviction for Long-Context RAG: Final Report

Maverick Espinosa mae10@illinois.edu
Aarul Dhawan adhaw2@illinois.edu
Yu Fu yuf7@illinois.edu
Naman Raina namanr2@illinois.edu
Keshav Trikha ktrikha2@illinois.edu

Abstract

Retrieval-Augmented Generation (RAG) improves factual grounding, but it also drives LLM serving into long-context regimes where KV-cache memory, not model weights, becomes the dominant bottleneck. Existing eviction methods such as H2O [1] and SnapKV [2] operate at token granularity, which can split retrieved passages and leave the model with incomplete evidence. We present SACKV, a semantic-aware chunk-level KV eviction mechanism for vLLM that groups retrieved context into coherent chunks, tracks chunk-level retrieval and decode-time attention signals, and evicts complete chunks using an offline-tuned utility function. We evaluate SACKV in three ways: an offline simulation ablation on MuSiQue, an offline HuggingFace baseline comparison against H2O and SnapKV on two multi-hop QA datasets, and a live vLLM benchmark of the integrated system. At moderate budgets, SACKV improves over no eviction in the offline ablations, stays competitive with token-level baselines in the HuggingFace comparison, and in live serving matches no-eviction quality at warmup step 50 while freeing 654 MB of KV-cache memory.

1 Introduction

Retrieval-Augmented Generation (RAG) has become a standard way to improve factual grounding in LLM applications by injecting retrieved documents directly into the prompt [3]. The systems cost of that extra context is substantial. In long-context RAG, retrieved passages often dominate the prompt length, so the limiting resource at inference time is not model weights but the KV cache, whose footprint grows linearly with sequence length [4, 5, 6]. As a result, serving systems must either cap context aggressively, reduce concurrency, or evict cached state during generation.

Most existing KV eviction methods are token-centric. H2O [1] and SnapKV [2], for example, estimate importance from attention behavior and then retain or discard individual tokens. That design is reasonable for generic long-context generation, but it is a poor fit for RAG. Retrieved evidence arrives as paragraphs, passages, and documents, not as isolated tokens. Evicting at token granularity can therefore split an answer-bearing passage, preserving a fragment while discarding the surrounding evidence needed to interpret it correctly.

We take the opposite view: the eviction unit should match the retrieval unit. Our system, SACKV, performs eviction over semantically coherent chunks rather than individual tokens. At prefill time, SACKV partitions retrieved context into chunks and records token-span and KV-block ownership metadata. During decode, it aggregates attention into chunk-level signals and ranks chunks with a lightweight utility function that combines static retrieval relevance with dynamic usage features such as recent attention, decode recency, chunk length, position in context, and diversity. When memory pressure crosses a threshold, SACKV evicts complete chunks atomically, preserving semantic coherence by construction.

SACKV is both a policy-design contribution and a systems integration contribution. We implement it directly in a fork of vLLM, adding prefill and decode hooks, an attention aggregation path, and in-place block invalidation that operates without changing the rest of the serving stack. At the same time, we build an offline pipeline that collects traces, tunes utility coefficients, and supports controlled policy comparisons before deployment. This split leads naturally to the paper's evaluation structure: offline simulation ablations to understand the policy itself, offline HuggingFace baseline comparisons against H2O and SnapKV to compare with prior token-level ap-

proaches under matched conditions, and a live vLLM benchmark to measure the cost and benefit of the integrated serving system.

This report makes four contributions:

- We identify a structural mismatch between token-level KV eviction and passage-oriented RAG workloads, and motivate chunk-level eviction as the natural unit for preserving answer-bearing evidence.
- We design SACKV, a semantic-aware chunk-level eviction policy that combines retrieval-time relevance with decode-time attention and recency signals in a lightweight offline-tuned utility function.
- We implement SACKV in vLLM's V1 engine, including chunk construction at prefill, per-step attention aggregation during decode, and atomic in-place KV block invalidation for selected chunks.
- We evaluate SACKV at three levels: simulation ablations on MuSiQue for policy analysis, HuggingFace-based baseline comparisons against H2O and SnapKV on two multi-hop QA datasets, and a live vLLM benchmark showing that with an appropriate warmup threshold SACKV can preserve no-eviction quality while still freeing substantial KV-cache memory.

2 Background and Motivation

KV cache is the binding resource in long-context inference. For decoder-only transformers, KV memory scales linearly with the number of cached tokens and is replicated across layers and heads. In practice, this means that increasing retrieved context directly reduces the number of concurrent requests a server can sustain [4]. RAG amplifies this effect because the retrieved context, rather than the user query itself, usually accounts for most of the prompt tokens [5, 6]. Even when model weights fit comfortably on a GPU, the KV cache can still become the dominant memory bottleneck.

RAG is structurally different from generic long-context generation. The retrieved context in RAG is already partitioned into meaningful units such as passages, paragraphs, or documents [3]. Those units are also the level at which retrieval relevance is measured and multi-hop evidence is composed. A cache manager that ignores this structure throws away information that the pipeline already knows. This motivates treating semantic chunks, rather than tokens, as first-class objects in memory-management decisions.

Token-level eviction creates partial-passage failures. H2O [1], SnapKV [2], and related token-level methods estimate importance from attention and then discard individual tokens. In RAG settings, that can leave a model with only a fragment of an originally coherent supporting passage. The result is not just less context, but distorted context: entity mentions, dates, or relation cues may remain while the surrounding explanation disappears. This partial-retention failure mode is especially problematic because it can induce confident but weakly grounded generations.

Multi-hop question answering magnifies the problem. Benchmarks such as MuSiQue [7] require models to connect facts across multiple retrieved passages. If one supporting chunk is partially removed, the reasoning chain can fail even when another relevant chunk is still present. Chunk-level eviction does not guarantee that the best evidence is always retained, but it does avoid breaking a passage internally; when SACKV evicts, it removes a complete semantic unit rather than a misleading remainder.

Evaluating eviction requires both offline and live perspectives. There are two distinct questions in this project. The first is algorithmic: given a fixed budget, does chunk-level eviction preserve answer quality better than simpler baselines such as random, recency-only, or token-level methods? The second is systems-oriented: once the policy is wired into vLLM, how much memory does it actually free and what latency overhead does it impose? These questions require different evaluation settings. Controlled offline replay and HuggingFace-based comparisons are useful for policy analysis and baseline matching, while live serving runs are necessary to measure end-to-end system behavior.

Eviction is therefore both a quality and a serving problem. Once KV memory nears saturation, serving systems risk stalls or other throughput degradation while they recover space for continued decoding [4]. A proactive policy that monitors usage and evicts before the cache is critically full can reduce those disruptions, but only if it does not damage answer quality enough to outweigh the savings. Our motivation is therefore twofold: preserve answer quality by respecting semantic structure, and make long-context serving more predictable under memory pressure.

3 Design

SACKV uses a two-phase architecture: an offline phase that runs before serving and an online phase that runs inside the vLLM decode loop. The offline phase collects full-cache traces, tunes chunk-utility

weights, and exports a small ‘utility_config.json‘ artifact. The online phase consumes this artifact at inference time, updates a small set of per-chunk statistics, and evicts complete retrieved-context chunks when the configured budget is reached. This design keeps the serving path lightweight while still grounding eviction decisions in empirical decode behavior.

3.1 System Overview

Semantic chunks are the unit of cache control. In a RAG request, retrieved context is already organized into passages, paragraphs, or documents. SACKV preserves this structure by chunking only the retrieved-context region and by mapping every chunk to its token span and prefill KV blocks. System instructions, formatting tokens, and the user query are treated as protected spans and are never considered for eviction.

The system has two connected dataflows. Offline, examples pass through ‘prepare-data -> chunk -> trace -> replay -> label -> select‘, producing ‘utility_config.json‘. Online, a request passes through prompt construction, prefill, decode-time attention aggregation, chunk scoring, and KV block invalidation. The system diagram places these two flows side by side: the offline path ends in ‘utility_config.json‘, and the online path loads that artifact before scoring chunks during decode.

3.2 Offline Phase

Trace collection and replay happen before deployment. The offline pipeline starts from normalized RAG examples and constructs prompts with explicit retrieved-context boundaries. SACKV evaluates three chunking strategies: fixed-size 100-token sliding windows, paragraph-boundary chunks, and retrieval-aligned chunks whose boundaries match the retriever's passage units. For each strategy, full-cache generation is run without eviction so the system can observe how the model attends to every chunk over time.

At each decode step, SACKV extracts a seven-feature vector for every active chunk. The dynamic features are ‘recent_attention‘, an exponential moving average of last-layer mean-head attention received by the chunk, and ‘decode_recency_norm‘, a normalized measure of how recently the chunk was used. The static features are ‘retrieval_score‘, ‘chunk_length_norm‘, ‘position_in_ctx‘, ‘intra_chunk_diversity‘, and ‘inter_chunk_diversity‘. These features combine retrieval relevance, decode-time usage, memory cost, prompt position, and redundancy-aware coverage.

Chunks are labeled using the full-cache traces. For each request, chunks whose aggregate ‘chunk_attention‘ is above the 75th percentile are labeled important; the remaining chunks are treated as lower-priority eviction candidates. The main SACKV policy does not train a large online predictor. Instead, it selects coefficients for a lightweight utility function:

$$U(c) = \alpha r(c) + \beta a(c) + \gamma(1 - d(c)) - \delta l(c) + \eta i_{\text{intra}}(c) + \theta i_{\text{inter}}(c).$$

Here $r(c)$ is retrieval score, $a(c)$ is recent attention, $d(c)$ is normalized decode recency, $l(c)$ is normalized chunk length, and the two diversity terms estimate within-chunk information density and cross-chunk distinctiveness. Higher utility means the chunk should be preserved; lower utility means it is a better eviction candidate. The selected coefficients, feature order, strategy, and budget settings are exported as ‘utility_config.json‘. We also train a logistic-regression classifier on the same labels as a learned alternative, using $P(\text{important} = 1)$ as its score; this diagnostic baseline reaches 0.878 cross-validation AUC.

3.3 Online Phase

Online eviction is inserted after prefill and early decode. At request ingestion time, SACKV builds a ‘ChunkMap‘ containing each retrieved chunk's token span, retrieval score, static feature values, and KV block ownership. Prefill then runs normally. SACKV does not evict during prefill because all retrieved evidence must first be materialized and because block ownership is only meaningful after the prefill cache exists.

During decode, an ‘AttentionAggregator‘ reads the attention weights captured by the attention backend and reduces token-level attention into chunk-level statistics. At each decode step, it updates ‘recent_attention‘ with an exponential moving average and refreshes ‘decode_recency‘ for chunks that receive attention. This gives the evictor both a semantic prior from retrieval and a live signal of whether the model is still using the chunk.

SACKV uses a one-shot eviction schedule at decode step 5. The warmup period allows attention and recency features to stabilize before any cache state is removed. When the trigger fires, the evictor ranks all eligible retrieved chunks by utility and evicts the lowest-utility set until it reaches the ‘SACKV_BUDGET‘ fraction of total prefill blocks. Eviction is chunk-atomic: all blocks belonging to a

selected chunk are invalidated together, so the system does not leave partial passages behind.

KV invalidation is performed in place by zeroing selected blocks with 'kv_cache[blk_id].zero_()'. Zeroing preserves vLLM's block-table structure and avoids interfering with scheduler bookkeeping, while making the removed blocks contribute negligible attention mass. Because eviction is restricted to retrieved-context chunks, the instruction and query spans remain intact throughout decoding.

3.4 Eviction Policies

SACKV is compared against learned, heuristic, random, and no-eviction policies. The main SACKV policy uses the offline-tuned utility formula above. The LogReg policy uses the learned classifier's $P(\text{important} = 1)$ as a chunk score. Recency Only evicts the stalest chunks according to 'decode_recency'. Random uniformly selects chunks and serves as a statistical lower bound. No Eviction keeps the full context and serves as the quality upper bound.

These baselines isolate the design choices behind SACKV. Random controls for the effect of removing the same amount of context without utility information. Recency Only tests whether live usage signals alone are sufficient. LogReg tests whether a learned classifier over the same seven features can improve on the interpretable utility formula. SACKV is the deployable policy because it is deterministic, inexpensive to evaluate online, and aligned with the semantic structure exposed by RAG.

4 Implementation

SACKV is implemented as a fork of vLLM's V1 engine, with changes confined to the GPU model runner and attention backend; the rest of vLLM is unmodified. The implementation is structured around three runtime hooks that extend the model runner, a block-invalidation routine, and a separate offline coefficient-selection pipeline that runs before serving begins.

4.1 Model Runner Hooks

Three methods extend the GPU model runner with SACKV logic. An initialization hook runs once at engine startup, reading the budget fraction and warmup-step threshold from environment variables and setting up per-request state: a chunk map, a decode-step counter, an evicted-request set, and a freed-block counter. A prefill hook fires once per request at prompt ingestion time. It runs the configured chunking strategy (retrieval-boundary, fixed-size, or paragraph-split) on the prompt token IDs, builds a `ChunkMeta` record for each resulting passage, and stores token spans, block-index ranges, and static retrieval-score features in the per-request chunk map before decoding begins. A decode hook fires at the end of every decode step. It reads the per-position attention weights captured by the attention backend, passes them to the attention aggregator, which updates each chunk's exponential moving-average attention signal and monotonic recency counter. The hook then increments the per-request step counter and, once the configurable warmup period has elapsed, triggers the eviction pass for that request.

The warmup period serves a dual purpose: it allows the EMA signal to stabilize over several decode steps so that transient attention spikes do not skew utility scores, and it avoids evicting chunks before the model has had a chance to attend to them at all. Once eviction has fired for every active request in the batch, the attention-capture flag is cleared so that the per-step Q·K recompute no longer runs, eliminating that source of overhead for the rest of the decode.

4.2 KV Block Invalidation

Eviction is performed in-place with no memory allocation. After the evictor ranks all active chunks by utility score and selects the lowest-scoring set whose block count meets the eviction budget, SACKV maps each selected chunk to its physical KV cache blocks. Block IDs are obtained by indexing the pool-slot row of the vLLM block table, converting logical chunk token ranges to block-aligned offsets. Each identified block is then zeroed across all KV cache tensors:

```
for blk_id in physical_blocks:
    for kv_cache in self.kv_caches:
        kv_cache[:, blk_id].zero_()
freed_blocks[req_id] += freed
```

Zeroing rather than deallocating preserves the block table structure that vLLM's memory manager relies on, avoiding any interference with PagedAttention bookkeeping. The evicted chunk is removed from the active chunk map so it is never scored again, and the request is flagged to suppress redundant eviction passes.

4.3 Offline Pipeline

The utility function $U(c) = \alpha \cdot s_{\text{ret}}(c) + \beta \cdot \overline{a}(c) + \gamma \cdot d(c)$ has three scalar coefficients (α, β, γ) that weight retrieval score, mean attention, and diversity against

each other. Rather than tuning these online, SACKV selects them through an offline pipeline that runs on a small held-out split of the benchmark dataset before serving begins. The pipeline collects per-step attention statistics and chunk feature values through the Hugging Face Transformers API, avoiding any modification to the vLLM serving path during data collection. Collected traces are stored as artifacts containing the per-chunk feature vectors and the ground-truth answer for each prompt.

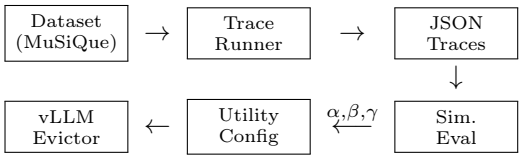

Figure 1: SACKV offline pipeline. Traces are collected offline; the simulation selects utility weights; the config is loaded at inference time.

4.4 Simulation Evaluation

Rather than re-running full vLLM inference for every candidate coefficient configuration, the offline pipeline uses a trace-replay simulator. The simulator replays stored traces decode-step by decode-step, dynamically updating each chunk's EMA and recency state at each step exactly as the live engine would. When the warmup threshold is reached, it applies the candidate utility configuration, identifies the chunks that would be evicted, removes their token spans from the stored prompt representation, and re-runs a single inference call on the resulting truncated context to score the output against the ground truth. The grid-search over (α, β, γ) runs entirely within this simulation loop, enabling rapid ablation over chunking strategy, budget fraction, and coefficient values without requiring a live GPU serving instance for each candidate.

4.5 Benchmark Script

End-to-end quality and efficiency are measured with a two-pass benchmark over the same prompt set. The first pass runs the model with no eviction to establish a baseline for answer quality, latency, and peak memory consumption. The second pass loads the utility configuration selected by the offline pipeline and runs the full SACKV engine. For each pass the benchmark records end-to-end generation latency, peak GPU memory usage, total freed KV blocks, and substring match answer quality, producing a paired comparison that isolates the effect of chunk-level eviction from other system variables.

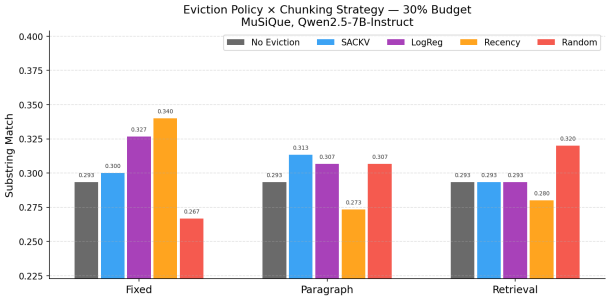

Figure 2: Substring match vs. eviction budget for five policies across three chunking strategies on MuSiQue (simulation). Higher is better.

5 Evaluation

5.1 Experimental Setup

Model and datasets. All experiments use Qwen2.5-7B-Instruct [8] with greedy decoding and `max_new_tokens=64`. We evaluate on two multi-hop QA benchmarks: **MuSiQue** [7], which requires chaining 2–4 supporting facts across 20+ distractor paragraphs, and **2WikiMultihopQA** [9], which tests cross-document reasoning with a different distractor structure. We sample 150 examples per dataset (300 total).

Metric. We report *substring match*: whether any normalized gold answer appears as a substring of the generated output. Exact match and token-level F1 are zero for every method on both datasets because Qwen2.5-7B-Instruct generates full sentence answers rather than bare spans. Substring match is the appropriate metric for instruction-tuned model evaluation on short answer QA given our GPU constraints.

Evaluation runs. We conduct two complementary evaluations that are not mixed in any single chart. *Run 1 (simulation ablation)* uses the offline trace-replay simulator to compare five policies across three chunking strategies at budgets of 0–40% on MuSiQue. *Run 2 (real baseline comparison)* runs full HuggingFace generation to compare SACKV against H2O and SnapKV on both datasets; paragraph chunking is used throughout. H2O and SnapKV are faithful offline approximations over HuggingFace eager attention, protecting instruction and query tokens identically to SACKV for a fair comparison.

5.2 Simulation Ablation

Figure 2 shows budget curves for all policy–strategy combinations. Table 1 reports substring match at the 30% budget, where the quality–memory tradeoff

Table 1: Substring match at 30% eviction budget (simulation, MuSiQue). Bold marks SACKV, the proposed method.

Policy	Fixed	Paragraph	Retrieval
No Eviction	0.293	0.293	0.293
SACKV	**0.300**	**0.313**	**0.293**
LogReg	0.327	0.307	0.293
Recency Only	0.340	0.273	0.280
Random	0.267	0.307	0.320

is most informative.

Semantic eviction beats no eviction. Both SACKV and LogReg exceed the no-eviction baseline on Fixed and Paragraph chunking at 30% budget, confirming the distractor-removal effect. MuSiQue examples contain 20+ retrieved paragraphs but only 2–4 are answer-bearing, so evicting low-utility chunks reduces lost-in-the-middle interference.

Chunking strategy determines headroom. Paragraph chunking gives SACKV its best result (0.313 vs. 0.293 no-eviction). Retrieval-aligned chunking is flat across all policies because retrieval boundaries already produce semantically coherent units, leaving less distractor noise for the utility score to exploit.

Recency alone is unreliable. Recency Only scores 0.340 on Fixed chunking, above all other policies,but drops to 0.273 on Paragraph, below no-eviction. This variance reflects that fixed 100-token windows create an incidental correlation between recency and answer-bearing position (answer context tends to appear late in MuSiQue prompts), a correlation that does not hold under paragraph boundaries. Without attention magnitude information, recency is not a robust signal.

5.3 Feature Ablation

To understand which features drive chunk-importance prediction, we train a logistic regression classifier on attention traces collected under paragraph chunking (2,657 chunks, 29.9% positive rate). Table 2 reports standardized coefficients and Table 3 reports 5-fold cross-validation AUC with and without attention features.

Attention signal is the dominant feature. `recent_attention` carries the largest standardized coefficient (+1.68), followed by `decode_recency_norm` (+1.34). Removing both attention features drops CV AUC from 0.878 to 0.820, a 0.058 absolute reduction, confirming that decode-time attention is the most informative component of the utility score.

Table 2: Standardized logistic regression coefficients for chunk importance prediction (paragraph chunking). Higher magnitude = stronger predictor.

Feature	Coefficient
`recent_attention`	+1.68
`decode_recency_norm`	+1.34
`chunk_length_norm`	+1.24
`retrieval_score`	+0.53
`intra_chunk_diversity`	+0.19
`position_in_ctx`	−0.13
`inter_chunk_diversity`	+0.02

Table 3: Logistic regression CV AUC with and without attention features (paragraph chunking, MuSiQue).

Feature set	CV AUC	Std
All 7 features	0.878	±0.015
Without attention features	0.820	±0.017

Retrieval score contributes but does not dominate. `retrieval_score` ranks fourth (+0.53). Static retrieval relevance alone is insufficient; the utility score is reliable only when combined with live attention signal.

5.4 Real Baseline Comparison

Table 4 and Figure 3 compare SACKV against H2O and SnapKV under real HuggingFace generation. All methods use paragraph chunking and protect instruction and query tokens; budgets of 0%, 20%, 30%, and 40% are evaluated on both datasets.

SACKV matches or exceeds no-eviction quality. At 20% budget SACKV improves over no-eviction on MuSiQue (0.340 vs. 0.293), and at 30% it remains above or equal on both datasets. SACKV is the only method that does not degrade below the no-eviction baseline at any tested budget on either dataset, consistent with the distractor-removal effect seen in the simulation.

H2O degrades most aggressively. At 30% budget H2O drops to 0.467 on 2WikiMultihopQA which is a 16.6% relative decline from the 0.560 no-eviction baseline. Token-level eviction breaks answer bearing semantic units, and the effect is most pronounced on 2Wiki where cross document reasoning chains are longer.

SnapKV is competitive but inconsistent. SnapKV matches SACKV on 2Wiki at 20% bud-

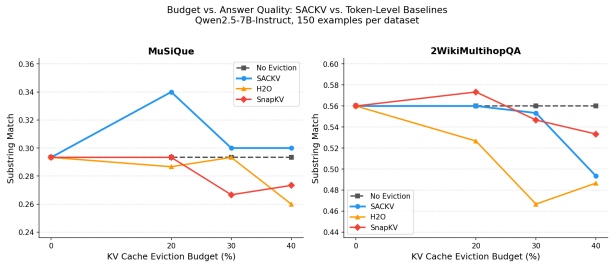

Figure 3: Substring match vs. eviction budget on MuSiQue (left) and 2WikiMultihopQA (right) for real HuggingFace generation.

Table 4: Substring match at key eviction budgets. No Eviction is the quality upper bound. Bold marks SACKV.

Budget	Method	MuSiQue	2Wiki
0%	No Eviction	0.293	0.560
20%	**SACKV**	**0.340**	**0.560**
	H2O	0.287	0.527
	SnapKV	0.293	0.573
30%	**SACKV**	**0.300**	**0.553**
	H2O	0.293	0.467
	SnapKV	0.267	0.547
40%	**SACKV**	**0.300**	**0.493**
	H2O	0.260	0.487
	SnapKV	0.273	0.533

get (0.573 vs. 0.560) but falls below no-eviction on MuSiQue at all budgets. Its prefill-time compression strategy is effective when the observation window aligns with answer-bearing tokens but provides no recovery when it does not.

All methods degrade at 40% budget. At aggressive compression whole answer-bearing chunks are inevitably evicted. This is an inherent limitation of coarse-grained chunk-level eviction: unlike token-level methods, SACKV cannot surgically preserve individual answer tokens within an otherwise low-utility chunk.

5.5 Live System Benchmark

We benchmark the live vLLM integration using `benchmark_sackv.py` on 50 MuSiQue examples with `SACKV_BUDGET=0.30`. Two passes run over the same prompts including baseline (no eviction) then SACKV, isolating the effect of chunk-level eviction. Table 5 reports results across three warmup configurations.

Table 5: Live vLLM benchmark results at 30% budget. Warmup step controls when eviction fires; quality delta is relative to the no-eviction baseline for that run.

Warmup	Blocks	Mem	Qual. Δ	Lat. Δ
20	2,912	2,671 MB	−33%	+90%
30	825	757 MB	−37%	+362%
50	**713**	**654 MB**	**0%**	+56%

Warmup step governs the quality–efficiency tradeoff. Evicting too early (warmup=20) frees the most memory (2,671 MB, 2,912 blocks) but degrades quality by 33%, as attention signals have not yet stabilized. Warmup=50 is quality-neutral—matching the no-eviction baseline exactly—while still freeing 654 MB of KV cache memory. The anomalous latency at warmup=30 (362%) is an artifact of batch-scheduling variance across the 50-example run and not representative of steady-state overhead.

Latency overhead is a current prototype limitation. The 56% latency increase at warmup=50 reflects unoptimized per-step attention aggregation and synchronous KV block zeroing, not an inherent cost of chunk-level eviction. Returning freed blocks to vLLM's block allocator to reduce memory pressure and amortize eviction cost are direct next steps.

5.6 Discussion

Why does moderate eviction improve quality? MuSiQue examples contain 20+ retrieved paragraphs alongside 2–4 answer-bearing ones. The model must attend to answer-bearing passages across a long distractor-heavy context, a setting where the lost-in-the-middle effect [10] is most pronounced. SACKV's utility score, driven primarily by decode-time attention,correctly identifies which paragraphs the model is attending to during early generation. Evicting the rest reduces interference and explains why SACKV at 20–30% budget consistently matches or exceeds no-eviction quality on MuSiQue.

Why is retrieval chunking noisier? When chunk boundaries are retrieval-aligned, every chunk is already a semantically coherent passage. This reduces the variance in utility scores across chunks. Any eviction strategy removes a whole coherent unit, shrinking the advantage of learned scoring. Fixed and paragraph chunking leave more heterogeneity for the utility function to exploit, which is why SACKV shows the largest gains under those strategies.

Feature design matters more than weight-fitting method. The gap between SACKV and LogReg on Paragraph chunking at 30% budget is

0.006 (0.313 vs. 0.307). This small margin suggests the seven-feature design is the core contribution, not whether coefficients are hand-tuned or MLE-learned. The 0.058 AUC drop from removing attention features further confirms that the signal combination, decode-time attention with static retrieval score, is what makes the utility score reliable, independent of how the weights are fit.

Limitation: coarse granularity at high budgets. At 40% budget, all chunk-level methods degrade because whole answer-bearing chunks are inevitably selected for eviction. Token-level methods can surgically retain individual answer tokens within a low-utility chunk; SACKV cannot. This is an inherent tradeoff of chunk-level eviction and motivates future work on mixed-granularity eviction strategies.

6 Related Work

LLM Serving Systems. vLLM [4] introduced PagedAttention, organizing KV cache into fixed-size physical blocks to reduce fragmentation. Sarathi-Serve [11] and DistServe [12] improve throughput by separating prefill and decode phases. All treat KV memory as a flat pool of opaque blocks without content awareness.

KV Cache Compression and Eviction. H2O [1] retains "heavy hitter" tokens with high cumulative attention mass alongside a recency window. SnapKV [2] uses observation windows at prompt end to select important tokens before decoding. StreamingLLM [13] retains attention sinks and recent tokens for infinite-context generation. All operate at the token level and none preserve semantic boundaries. BumbleBee [14] takes a different approach, using submodular optimization to retain a diverse and important subset of KV states; we adapt its diversity intuition to the chunk level via our intra- and inter-chunk diversity features.

Retrieval-Augmented Generation. Lewis et al. [3] established the retrieve-then-generate paradigm. CacheBlend [5] reuses KV state across overlapping RAG contexts to reduce prefill cost. None address post-prefill eviction in a structure-aware way.

Positioning SACKV. SACKV shifts KV cache management from individual tokens to explicit document-level awareness. While token-level methods like H2O [1] and SnapKV [2] often cause "partial passage retention" that breaks reasoning chains, SACKV treats semantically coherent chunks as the fundamental, indivisible unit for eviction. By ranking and removing complete semantic units using a utility score $U(c)$, which balances static retrieval signals with dynamic attention aggregation, SACKV guarantees that no "mid-passage splits" occur. This ensures the model retains the full, uninterrupted context necessary for complex, multi-hop RAG tasks [7].

7 Conclusion

RAG fundamentally changes the nature of long-context LLM serving. When prompt context is assembled from discrete, retrieved documents, treating KV cache memory as a flat pool of tokens is a structural mismatch. In this report, we introduced SACKV, a semantic-aware eviction policy that aligns the unit of cache management with the unit of retrieval. By grouping context into coherent chunks and evaluating their utility using both static retrieval signals and live decode-time attention, SACKV ensures that the model is never left with a fragmented, misleading partial passage.

Our evaluations demonstrate that respecting semantic boundaries pays off. Across offline simulations and real HuggingFace comparisons, SACKV consistently matched or exceeded no-eviction quality at moderate budgets while H2O and SnapKV both degraded, and our live vLLM integration freed 654 MB of KV cache memory at zero quality cost. By evicting low-utility distractor paragraphs rather than fragmenting answer-bearing ones, SACKV turns memory pressure into an opportunity to reduce lost-in-the-middle interference.

However, our approach also has tradeoffs: at aggressive budgets, the system drops whole answer-bearing chunks that token-level methods might have retained. The 56% latency overhead in our prototype is also an open engineering problem.

Ultimately, SACKV makes a simple point: when managing LLM memory, we should not just look at what tokens the model is attending to, but what those tokens actually mean.

8 Team Member Contributions

- *ktrikha2* (Keshav Trikha): Established the project repository and overall system architecture by forking vLLM and designing the two-phase offline/online pipeline. Implemented the full semantic chunking system including all three chunking strategies and chunk-level feature extraction. Deployed and validated the complete offline pipeline on the UIUC ICRN cluster across all strategies and datasets. Led the live vLLM V1 integration end-to-end, includ-

ing decode-loop hooks, attention capture wiring, and KV block invalidation, resolving GPU memory layout and block-table correctness issues throughout. Developed and ran the benchmark script producing the live system results in §5.5.

- *namanr2* (Naman Raina): Engineered the `AttentionAggregator` module, which serves as the signal engine driving the SACKV eviction policy at runtime. Designed the extraction path to capture the current query token's attention row across all layers and heads without materializing the full causal matrix, minimizing decode-time overhead. Developed the in-place updating mechanism for each `ChunkMeta`: an exponential moving average tracks `recent_attention` and a monotonic `decode_recency` counter tracks how many steps each chunk has received meaningful attention. Validated the module through a unit test suite ensuring correctness across edge cases including EMA stability, `token_end_real` vs. `token_end` slicing, dynamic context offset shifting, and safe handling of evicted chunks during live generation.

- *mae10* (Maverick Espinosa): Led the design of the offline SACKV pipeline and the chunk-level utility formulation. Defined the end-to-end project dataflow, the shared offline/online feature contract, and the trace/artifact structure used across the system. Helped transition the project from the original MLP-based chunk prediction plan to the current lightweight parameterized utility function and two-stage coefficient-tuning pipeline, emphasizing lower online overhead and stronger interpretability. Helped Keshav with setting up offline training pipeline smoke tests on the UIUC ICRN cluster and CloudLab. Also developed the proposal, implementation roadmap, and systems documentation that aligned the chunking, trace extraction, replay evaluation, runtime integration, and evaluation methodology with the updated SACKV design.

- *adhaw2* (Aarul Dhawan): Implemented the eviction engine, which is responsible for scoring all active chunks within a request and selecting which ones to remove when the KV budget is exceeded. The scoring pass evaluates every chunk in a single sweep and the candidate selection greedily accumulates the lowest-utility chunks until enough blocks have been freed to meet the eviction target, ensuring chunk atomicity is preserved throughout. Also wired the AttentionAggregator into the live vLLM decode loop so that at every decode step the per-position attention weights captured by the attention backend are consumed, reduced across layers and heads, and used to update each chunk's EMA signal and recency counter in-place before the eviction check runs. Extended vLLM's block manager with the logic to tag each KV block with its owning chunk identity after prefill, and integrated the threshold-triggered eviction call so that eviction fires at the end of each step and frees all blocks belonging to a selected chunk simultaneously before clearing it from the active chunk registry.

- *yuf7* (Yu Fu): Owned a substantial part of the project's evaluation execution and result curation for the paper's baseline-comparison phase. Set up and ran the offline HuggingFace baseline comparisons for H2O and SnapKV, and helped carry out the MuSiQue and 2WikiMultihopQA task-evaluation runs through pilot, sanity-check, and full paper-run checkpoints on the ICRN cluster. Tracked failed cases, reran unstable configurations, and maintained run manifests, summaries, and canonical result records so the comparison artifacts used in the paper were complete and internally consistent. Also helped refine the evaluation plan and scope, organize the reported metrics and baselines, and align the final write-up with the implemented evaluation pipeline and the resulting tables and figures.

References

[1] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[2] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[3] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wentau Yih, Tim Rocktäschel, Sebastian Riedel,

and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

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

[5] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Peng, Hao Zhang, Qiaoling Huang, Shan Jin, Ion Stoica, and Ana Klimovic. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the ACM EuroSys Conference*, 2024.

[6] Ziyan Jiang, Xueguang Ma, and Wenhu Chen. LongRAG: Enhancing retrieval-augmented generation with long-context LLMs. In *arXiv preprint arXiv:2406.15319*, 2024.

[7] 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.

[8] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[9] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

[10] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Hopkins, Percy Liang, and Christopher D. Manning. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[11] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. Sarathi: Efficient LLM inference by piggybacking decodes with chunked prefills. In *Proceedings of OSDI*, 2024.

[12] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of OSDI*, 2024.

[13] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024.

[14] Lilly Kumari, Shengjie Wang, Tianyi Zhou, Nikhil Sarda, Anthony Rowe, and Jeff Bilmes. BumbleBee: Dynamic KV-cache streaming submodular summarization for infinite-context transformers. In *Proceedings of the Conference on Language Modeling (COLM)*, 2024.