PagedEviction: Structured Block-wise KV Cache Pruning for Efficient Large Language Model Inference

Krishna Teja Chitty-Venkata[1]*[$], **Jie Ye*** [2], **Siddhisanket Raskar** [3], **Xian-He Sun**[2], **Anthony Kougkas**[2], **Murali Emani**[1], **Venkatram Vishwanath**[1], **Bogdan Nicolae**[1]
[1]Argonne National Laboratory, Lemont, IL, USA
[2]Illinois Institute of Technology, Chicago, IL, USA
[3]Pacific Northwest National Laboratory, Richland, WA, USA
schittyvenkata@anl.gov,jye20@hawk.iit.edu,s.raskar@pnnl.gov, sun@iit.edu, akougkas@iit.edu, memani@anl.gov, venkat@anl.gov, bnicolae@anl.gov
*Equal Contribution, [$] Now at Red Hat, Inc.

Abstract

KV caching significantly improves the efficiency of Large Language Model (LLM) inference by storing attention states from previously processed tokens, enabling faster generation of subsequent tokens. However, as sequence length increases, the KV cache quickly becomes a major memory bottleneck. To address this, we propose PagedEviction, a novel fine-grained, structured KV cache pruning strategy that enhances the memory efficiency of vLLM's PagedAttention. Unlike existing approaches that rely on attention-based token importance or evict tokens across different vLLM pages, PagedEviction introduces an efficient block-wise eviction algorithm tailored for paged memory layouts. Our method integrates seamlessly with PagedAttention without requiring any modifications to its CUDA attention kernels. We evaluate PagedEviction across Llama-3.1-8B-Instruct, Llama-3.2-1B-Instruct, and Llama-3.2-3B-Instruct models on the LongBench benchmark suite, demonstrating improved memory usage with better accuracy than baselines on long context tasks.

1 Introduction

Large Language Models (LLMs) have revolutionized the field of NLP and are capable of understanding and generating human-like text. These models, trained on vast amounts of data, are capable of performing a wide range of language and summarizing tasks. The state-of-the-art (SOTA) LLMs are exploding to large sizes, include GPT (Brown et al., 2020), LLaMA (Touvron et al., 2023), and DeepSeek (Dai et al., 2024; Guo et al., 2025).

LLM inference is an iterative process that involves incremental computations, reusing a significant portion of previous attention results, which are stored in the form of KV Cache. However, this comes at a high memory cost: the KV cache grows linearly with sequence length and can even exceed the memory used by the model weights. For instance, caching keys/values for tens of thousands of tokens may consume more GPU memory, severely limiting LLM inference throughput. Attention mechanisms tend to focus disproportionately on a few critical tokens, while many other tokens contribute very little to the output, suggesting not all past tokens are equally important for generating the next tokens. This insight has led to the development of several KV cache compression algorithms that evict less important tokens from the cache.

Several KV Cache compression methods have been proposed to manage KV cache growth without retraining the model. For example, StreamingLLM (Xiao et al., 2023) focuses on retaining recent tokens or specific combinations of initial and recent tokens to sustain performance over longer contexts. Dynamic policies like H2O (Zhang et al., 2023) utilize runtime information, such as attention weights, to estimate token importance. However, most of these eviction techniques focus on the trade-off between the compression achieved by eviction and the accuracy loss while ignoring the practical aspects of GPU memory management. While these methods can effectively minimize the GPU memory, they often require custom modifications to the LLM serving frameworks or storing attention weights.

PagedAttention (Kwon et al., 2023) is a memory management technique designed to address the inefficiencies in serving LLMs by optimizing how the KV cache is stored and accessed during inference. Allocating contiguous memory space for the KV cache leads to substantial memory waste due to both internal and external fragmentation as memory chunks are over-reserved and scattered gaps prevent efficient reuse, especially when different requests generate variable sequence lengths. PagedAttention solves this issue by partitioning the KV cache into small, fixed-size blocks/pages rather than storing in a single contiguous chunk, each sequence's KV cache is divided into many equally

Findings of the Association for Computational Linguistics: EACL 2026, pages 3207–3218

sized blocks, where the pages are only allocated on demand, reducing wasted memory. It is a widely used technique in production-ready runtimes such as vLLM to mitigate memory fragmentation.

The current KV Cache eviction methods, which rely on attention scores ($Q*K^T$), cannot be integrated into PagedAttention, as FlashAttention never returns the attention score during the inference process. Also, the existing compression methods do not consider the paged structure in the vLLM framework while evicting tokens. They evict tokens across different pages, leading to a memory fragmentation issue for which PagedAttention was developed. Without synergy between token eviction and PagedAttention, the benefits of both approaches are limited. To address this challenge, we develop PagedEviction, a structured block-wise token eviction strategy optimized for a PagedAttention strategy. The high-level goal of PagedEviction is to limit KV cache by evicting entire blocks of tokens, so that memory usage (and computation per token) remains low, while the model's accuracy on long-context tasks remains essentially unchanged from using a full cache. Unlike attention-score-based methods, PagedEviction does not require storing the attention weights during the forward pass. Our method relies on Key and Value tensors to reduce the KV Cache and requires no changes to the CUDA attention kernels. PagedEviction can free entire blocks of KV memory at once, avoiding the fragmentation or overhead that token-level evictions might introduce.

Contributions. Our contributions are as follows:

- **PagedEviction: Block-Aligned KV Cache Compression:** We introduce a structured block-wise KV cache eviction algorithm that aligns with vLLM's block-based memory. Our method computes token or block importance using a proxy for attention scores, avoiding the need to store attention, which is not possible with FlashAttention. PagedEviction operates in both prefill and decode phases. During prefill, we evict tokens based on per-token importance before block partitioning. During decode, we evict ab entire block only after the most recent block becomes full, reducing fragmentation and per-step eviction overhead.
- **High Accuracy Under Tight Budgets:** On LongBench datasets, PagedEviction outperforms other attention-free baselines. For example, on GovReport with LLaMA-3.2-1B, it achieves a ROUGE score of 24.5 at a 1024-token budget, outperforming StreamingLLM (21.0) and KeyDiff (21.2) by 15–20%. On MultiNews with LLaMA-3.2-3B, we achieve 23.6 ROUGE, outperforming Inverse Key L2-Norm by 1.1 points.
- **Hardware Benefits:** At a cache budget of 1024, we achieve up to 3020 tokens/sec on LLaMA-1B, a **37%** improvement over Full Cache (2200 tokens/sec), and a **39%** improvement over Inverse Key L2-Norm (2170 tokens/sec). PagedEviction reduces latency by approximately **10–12%** across 1B, 3B, and 8B models and scales robustly with larger model sizes. Its block-level evictions avoid frequent cache updates, making it more scalable under batch inference.

2 Background and Related Work

2.1 Self-Attention

Given the input tensor $\mathbf{X}$, the multi-head attention mechanism performs multiple attention operations in parallel. It computes three projections: Query ($\mathbf{Q}$), Key ($\mathbf{K}$), and Value ($\mathbf{V}$) matrices through three linear layers by multiplying $\mathbf{X}$ with $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$. The attention activation is calculated using the scaled dot-product attention. This process is repeated H times, each with different learned projections $\mathbf{W}_Q^{(i)}, \mathbf{W}_K^{(i)}, \mathbf{W}_V^{(i)}$ for each head h. The outputs from different heads are concatenated and projected back to the original dimension d using a final learned matrix $\mathbf{W}_O \in \mathbb{R}^{hd_k \times d}$.

2.2 KV Cache

During the autoregressive LLM inference, the tokens are generated sequentially. Without a caching mechanism, the Key ($\mathbf{K}$) and Value ($\mathbf{V}$) states are computed in every step for all the previously predicted tokens. The KV Cache addresses this inefficiency by saving Key and Value activations for each token. Instead of recalculating these activations, the model retrieves the cached $\mathbf{K}$ and $\mathbf{V}$ activations and concatenates with the current token. Although KV caching can significantly reduce computational costs by avoiding redundant calculations, storing the cached values for every token in the sequence incurs substantial memory, which grows linearly with the sequence length and batch sizes.

2.3 KV Cache Eviction

The KV Cache Eviction methods identify less important tokens in $K[1:t-1]$ and $V[1:t-1]$

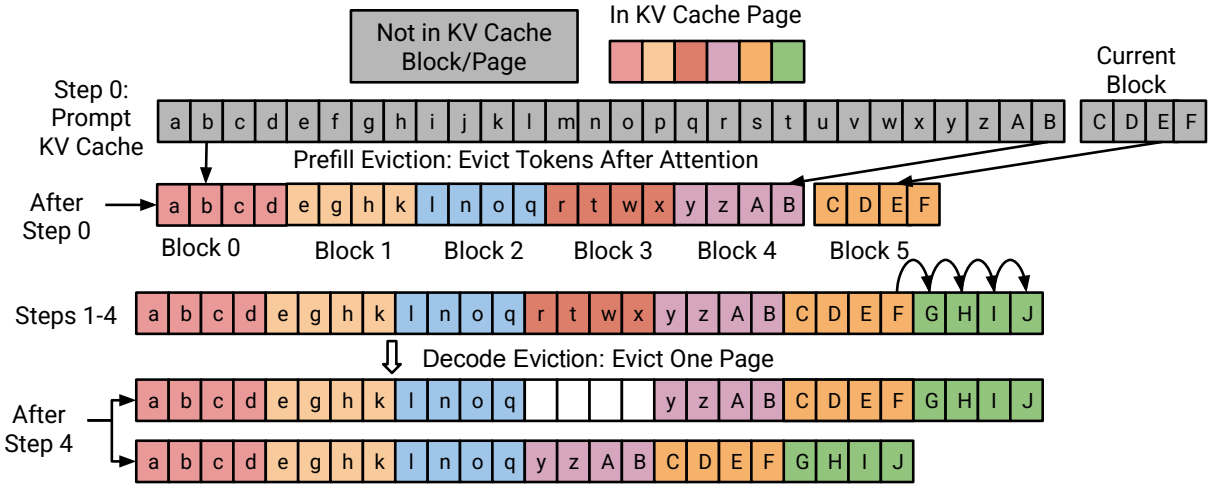


Figure 1: Illustration of the PagedEviction technique with a block size (B) and a cache budget (C). During the *prefill* stage, after the initial set of Key and Value tokens is generated, a subset of tokens is evicted until the cache budget is reached. In *decode* phase, one block of tokens is evicted once the recent block becomes full.

by developing a function fun_{kv} that identifies redundant tokens. The attention operation ($\text{Att}(\mathbf{Q}, \mathbf{K}', \mathbf{V}')$) is performed on non-evicted Key and Value tokens. Several strategies have been developed to evict less important tokens based on their importance and relevance. H2O (Zhang et al., 2024c) uses cumulative normalized attention scores to retain high-impact tokens while preserving recent tokens due to their strong correlations with current tokens. StreamingLLM (Xiao et al., 2023) maintains a limited number of initial KV pairs to ensure model performance remains stable. Keyformer (Adnan et al., 2024) addresses the issue of token removal by introducing regularization techniques to smooth and approximate the original softmax probability distributions, mitigating distortions caused by token eviction. PyramidInfer (Yang et al., 2024a) dynamically adjusts KV cache size layer-wise based on redundancy. However, these methods depend on attention scores for eviction, requiring CUDA kernel modifications to track them.

3 Challenges and Limitations of SOTA

In this section, we outline the key limitations of existing KV cache eviction methods and present our proposed solutions to address them.

Limitation 1: KV Cache Memory Organization. vLLM structures the KV cache into blocks, each storing a fixed number of key-value tokens per attention head. However, existing eviction methods do not consider this block structure and often evict different number of tokens across different blocks. This leads to block fragmentation, disrupting the uniformity of the memory layout. Such fragmentation reduces memory allocation efficiency for future requests, as blocks no longer maintain consistent occupancy.

Our Solution. We introduce a block-aware structured eviction mechanism (Figure 1) that preserves the native block alignment in vLLM after eviction. This approach maintains structural consistency, reduces fragmentation, and improves memory reuse efficiency.

Limitation 2: Block-Aware Eviction. Most existing methods have been designed for HuggingFace-based implementations, where KV cache tokens are stored contiguously in memory. This simplifies global token importance comparisons and merging of retained tokens. However, in vLLM, global comparisons are computationally prohibitive due to its non-contiguous block-based storage. Evaluating importance across all tokens during decoding introduces significant overhead, especially for long sequences.

Our Solution. We restrict eviction to block-level granularity during decoding by computing a single score per block rather than per token. This eliminates the need to move tokens across blocks and allows for efficient full-block eviction without modifying the underlying attention kernel. Our method retains consistent block sizes throughout inference and enhances memory reuse by preserving the structural layout of evicted blocks.

Limitation 3: Dependence on Attention Scores. Methods like H2O (Zhang et al., 2023) rely on cumulative attention scores to identify unimportant tokens. However, optimized attention kernels such as FlashAttention do not return at-

tention scores. Accessing or maintaining attention scores during inference requires integration with schedulers or host memory, which is decoupled from the CUDA kernel in vLLM. This dependency introduces substantial runtime and memory overhead, undermining inference efficiency.

Our Solution. We avoid reliance on attention scores by using a proxy importance metric derived solely from the static key and value states already stored in the KV cache. This proxy is computed on-the-fly without modifying the attention kernel or maintaining additional memory, ensuring compatibility with optimized inference paths.

Limitation 4: Per-Step Token Eviction. Eviction strategies such as StreamingLLM (Xiao et al., 2023) and H2O (Zhang et al., 2024c) perform eviction at every decoding step, requiring constant updates to the KV cache table across all layers. This results in significant per-step latency and reduces throughput, especially under large batch inference.

Our Solution. We adopt a coarse-grained eviction strategy wherein entire blocks are evicted only when the current block is full. This reduces the frequency of eviction operations and avoids the creation of partially filled blocks, which are incompatible with vLLM. Our approach simplifies eviction logic and reduces runtime overhead while maintaining efficient cache utilization.

4 PagedEviction

Our PagedEviction method is divided into two different components: Prefill KV cache eviction and Decode KV Cache eviction. This design choice is to align with the implementation in the vLLM framework. Figure 1 summarizes our PagedEviction algorithm in both prefill and decode phases.

4.1 Token Importance

vLLM's attention kernel, implemented in CUDA, is separate from its KV Cache memory management. Methods like H2O (Zhang et al., 2024c) require significant changes to the attention kernel to maintain a running sum of attention scores per head and per layer across different blocks. Therefore, we need a cost-effective method to evaluate the token importance without adding complexity and additional memory. The value activation matrix (V) represents the feature embeddings of each token that will be selectively combined to create the contextualized output. The Value tensor (V) holds the actual information to extract relevant pieces of information from the hidden states. This allows LLM to focus on specific aspects of the input that are most important in the current context. Also, previous work by Devoto et al. (Devoto et al., 2024) observed a unique correlation between the Key tensor and attention weights. The L2-norm of each Key token is inversely proportional to the cumulative attention score of each token. Therefore, we compute the score/importance (S) of each token or block using the ratio of the L2 norm of the Value token to that of the Key token ($\|V_i\|_2 \div \|K_i\|_2$). This requires us to fetch token importance directly from the static KV cache, eliminating the need to rely on the cumulative attention score.

Algorithm 1 Token/Block Importance

Require: Key K, Value V, Page Size B, Evict M

if M $=$ `token` **then**
  for each token i **do**
    $S_i \leftarrow \|V_i\|_2 \,/\, \|K_i\|_2$
  end for
else if M $=$ `block` **then**
  Divide K & V into blocks of size B
  for each block j **do**
    Compute block score S_j as the average:
    $S_j \leftarrow \frac{1}{B} \sum_{i \in \text{block } j} (\|V_i\|_2 \,/\, \|K_i\|_2)$
  end for
end if

4.2 PagedEviction: Prefill Phase

During the prefill phase, the prompt tokens are processed in a single forward pass through multiple layers, where Q, K, and V matrices are generated in each layer. In each layer, the contiguous Key and Value tensors are written to the corresponding KV cache blocks. We implement token eviction before the Key and Value states are divided into different pages. This approach is important because performing the eviction after storing tokens across different blocks would require significant memory reordering and movement between blocks. Therefore, during the prefill phase, PagedEviction performs token-level KV cache compression by computing the importance of each token and evicting the least important ones to reach a fixed cache budget (B). As shown in Algorithm 2, the process begins with a forward pass over the prompt to generate the initial KV Cache. Following the self-attention operation, PagedEviction computes a per-token importance score $S_i = \|V_i\|_2 / \|K_i\|_2$,

where tokens with lower scores are considered less critical to future predictions. We evict E tokens (E = L - C) from the input length (L). This simple yet effective scoring mechanism enables PagedEviction to compress the KV cache before decoding begins, reducing memory usage while preserving important KV states.

Algorithm 2 PagedEviction: Prefill Eviction

Require: Input Hidden State I, Cache Budget C and Block/Page Size B

Initialization: No KV cache exists initially
Process the prompt to generate KV cache
for One Forward Pass **do**
    $Q = Q_w(I), K = K_w(I), V = V_w(I)$
    Attn = Self-Attention(Q, K, V)
    O = Out$_{proj}$(Attn)
    Compute Importance of token i (S_i)
    S_i = $||V_i||_2 \div ||K_i||_2$
    L = Sequence Length of Input I
    Evict Tokens (E) = L - C
    Evict E Tokens in Key and Value Cache
    K, V = Evict(K, E), Evict(V, E)
    Divide K and V into different pages
end for

4.3 PagedEviction: Decode Phase

During the decode phase, PagedEviction evicts one KV cache page/block after the recent block is completely full. As shown in Algorithm 3 and illustrated in Figure 1, after each newly generated block of tokens (i.e., when the current sequence length L is a multiple of the block size B), the algorithm evaluates all existing pages in the cache. Each page is assigned an importance score based on the aggregated token-level scores within that page, which is the mean of the $\|V_i\|_2/\|K_i\|_2$ ratio across all tokens i in the block. The page with the lowest importance score is evicted from the cache, and the internal KV cache block table is updated accordingly. This strategy ensures that PagedEviction continuously frees memory in a structured, block-wise fashion, enabling long-context inference. We evict a block only when the last block becomes full. This strategy ensures that we maximize space utilization, as evicting a single token does not free the entire block, making partial eviction inefficient. Additionally, triggering eviction only when a new block is needed reduces the overall frequency of eviction operations. This process of generating new tokens and periodically evicting less important ones continues until either the end-of-sequence (EOS) token is reached or the maximum number of new tokens is generated. In this way, we preserve the most significant information for ongoing token generation while maintaining the block structure in vLLM's PagedAttention. Our proposed eviction policy offers several advantages over random token eviction strategies. We eliminate the need for extensive token rearrangement across all blocks during each eviction step as we evict a fixed number of tokens equal to the original block size. Also, the attention kernel can operate without modifications, as all blocks maintain a uniform size throughout the process, and our method is compatible with Flash Attention.

Algorithm 3 PagedEviction: Decode Eviction

Require: Input Hidden State I, Block/Page Size B, Cache Budget C and KV Cache KV$_{Cache}$

while EOS or max new tokens **do**
    L = Current Sequence Length
    if L % B == 0 **then**
        N = Number of KV Cache Pages
        Compute Score of Page i (S_i)
        S_i = $||V_i||_2 \div ||K_i||_2$
        Evict One page with lowest score S_i
        K, V = Evict_page(K, V)
        Update the KV Cache Block Table
    end if
    Attn = PagedAttention(Q, K, V)
    O = Out$_{proj}$(Attn)
    Continue generating new tokens
end while

5 Evaluation, Results and Discussion

5.1 Experimental Setup

We evaluate our method under cache budgets of 256, 512, 1024, 2048, 4096. For all experiments, we use a page size of 16, which has been shown to be optimal for vLLM (Kwon et al., 2023). Nonetheless, our proposed PagedEviction algorithm is compatible with any page size and cache budget. We implement our method on top of vLLM version 0.9.0. We use several datasets from the LongBench benchmark (Bai et al., 2023), including HotpotQA (Yang et al., 2018), Qasper (Dasigi et al., 2021), GovReport (Huang et al., 2021), MultiNews (Fabbri et al., 2019), and MultiFieldQA. These datasets have long input context and long outputs. We exclude simple QA tasks like 2WikiMultihopQA (Ho

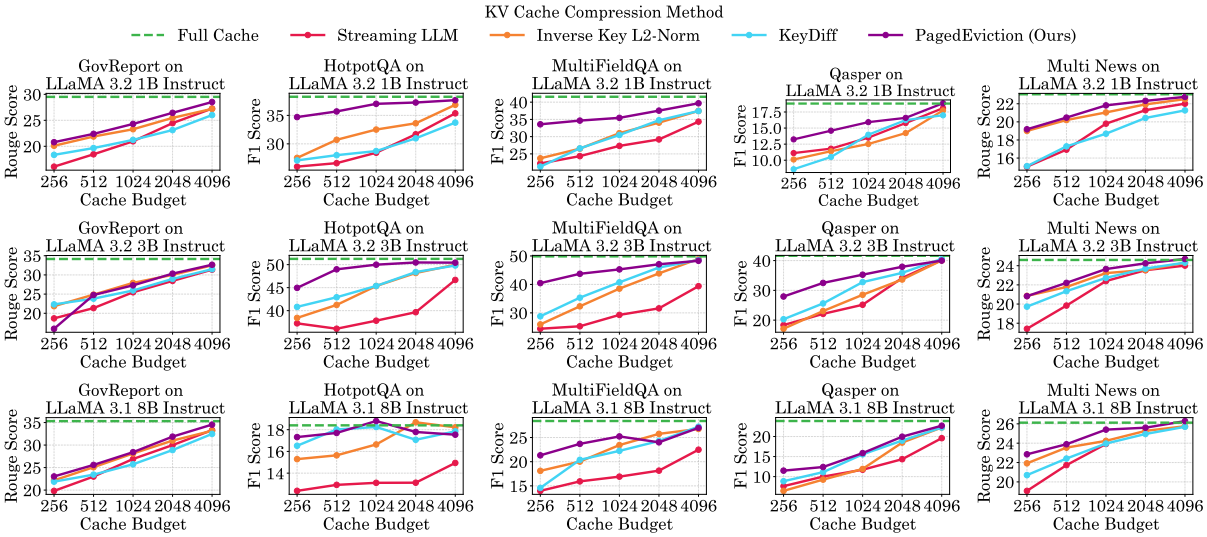


Figure 2: Accuracy vs Cache Budget of 1B, 3B, and 8B models on GovReport, HotpotQA, MultiFieldQA, Qasper and Multi News Datasets. PagedEviction consistently achieves higher or comparable accuracy relative to other attention-free baselines, especially under low cache budgets (e.g., 256–1024), and closely approaches Full Cache performance at high budgets (2048–4096)

et al., 2020), whose output is one or two tokens. All experiments are conducted on NVIDIA A100 GPUs with 40GB memory. For throughput evaluation, we use synthetic inputs with input sequence length of 1024 tokens, output sequence length of 8192 tokens, and 64 concurrent requests per batch. We report throughput (total number of input + output tokens processed per second) and Time per Output Token (TPOT). We evaluate our method on three LLMs of different scales, which are Meta-Llama-3.2-1B-Instruct, Meta-Llama-3.2-3B-Instruct, and Meta-Llama-3.1-8B-Instruct (Meta, 2024) to demonstrate generality across model sizes.

5.2 Baseline Methods

We evaluate our proposed PagedEviction algorithm against strong baselines that do not rely on attention scores to estimate token importance. To the best of our knowledge, the most relevant such baselines include: Full Cache (no eviction), StreamingLLM (Xiao et al., 2023), Inverse Key L2-Norm (Devoto et al., 2024), and KeyDiff (Park et al., 2025). These methods estimate token importance using key or value states, without modifying the attention score. StreamingLLM maintains a fixed-size KV cache consisting of a small prefix of initial tokens (e.g., the first 4 tokens) that are retained as "attention sinks", combined with a sliding window of the most recent tokens (e.g., the last 1024 tokens). Inverse Key L2-Norm prunes the cache by evicting tokens with high L2 norms of their key vectors KeyDiff emphasizes key vector diversity by evicting tokens whose keys are redundant, i.e., highly similar to others, thereby preserving a more diverse and informative KV cache. We categorize attention-free KV cache eviction methods into structured and unstructured approaches. Structured methods, such as our PagedEviction and StreamingLLM, evict tokens within a single block or remove entire blocks together. In contrast, unstructured methods like Inverse Key L2-Norm and KeyDiff operate at the token level, evicting tokens across the entire sequence based on individual importance scores. All of these baselines evict tokens at a fine-grained token level across different pages during each decoding step. Importantly, none of them require changes to the underlying CUDA attention kernel. Since our primary objective is to retain compatibility with vLLM and avoid modifying the core attention implementation, we restrict our evaluation to baselines that can be integrated within the vLLM runtime. Other methods, such as H2O (Zhang et al., 2024c), require kernel modifications that hinder deployment efficiency and fall outside the scope of our work. Thus, for a fair and practical comparison, we compare PagedEviction against the three non-trivial but vLLM-friendly methods.

5.3 LongBench Results

Figure 2 compares our PagedEviction with several attention-free KV cache compression methods

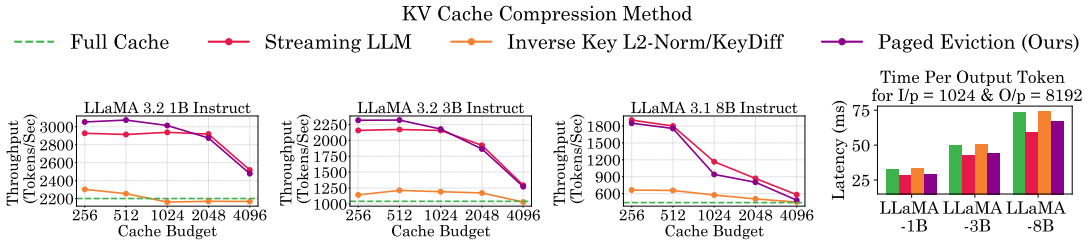
across multiple datasets and three LLaMA variants (1B, 3B and 8B) across varying KV cache budgets from 256 to 4096. Our PagedEviction method (highlighted in purple) demonstrates remarkably consistent performance across several settings, maintaining scores competitive with the full cache baseline while requiring substantially less memory footprint. The StreamingLLM approach shows moderate compression effectiveness, particularly excelling at high cache budgets, which aligns with its design principle of maintaining attention sinks for stable long-context processing. The Inverse Key L2-Norm method exhibits variable performance, with notable degradation on certain tasks like Qasper, suggesting that low-norm key embeddings may not universally correlate with attention importance across all model architectures and tasks. The KeyDiff method demonstrates steady but modest compression gains, reflecting its similarity-based eviction strategy that preserves geometrically distinctive keys. These results underscore the critical importance of task-specific and model-aware selection of KV cache compression techniques, as no single method universally dominates across all evaluation scenarios.

Across all models and datasets, PagedEviction consistently achieves superior performance under tight cache budgets, demonstrating its ability to retain high accuracy while significantly reducing memory usage. For instance, our PagedEviction on LLaMA-3.2-1B-Instruct on the GovReport dataset (long context summarization) consistently outperforms all baselines. At a cache budget of 1024, PagedEviction achieves a score of ∼24.5, which is ∼15–20% higher than StreamingLLM (∼21) and KeyDiff (∼21.2). At the cache budget of 4096, it reaches ∼29.5, closely matching the full-cache score 30, demonstrating its effectiveness across all memory constraints. On the MultiNews dataset using the LLaMA-3.2-3B-Instruct model, PagedEviction achieves a ROUGE score of approximately ∼23.6 at a cache budget of 1024, outperforming Inverse Key L2-Norm (∼22.5) and StreamingLLM (∼22.0) by roughly 5–9%. At the budget of 4096, our method nearly matches the full-cache performance (∼24.5), demonstrating both high compression efficiency and minimal accuracy degradation. This highlights the effectiveness and generalizability of our block-wise eviction policy across tasks and models. Our main contribution is the block-wise KV cache eviction mechanism, which can potentially serve as a basis for further optimizations. Techniques such as layer-wise budget allocation (Yang et al., 2024a) and quantized KV caching (Dong et al., 2024) can be built on top of our approach to further enhance performance.

5.4 vLLM Throughput and Latency

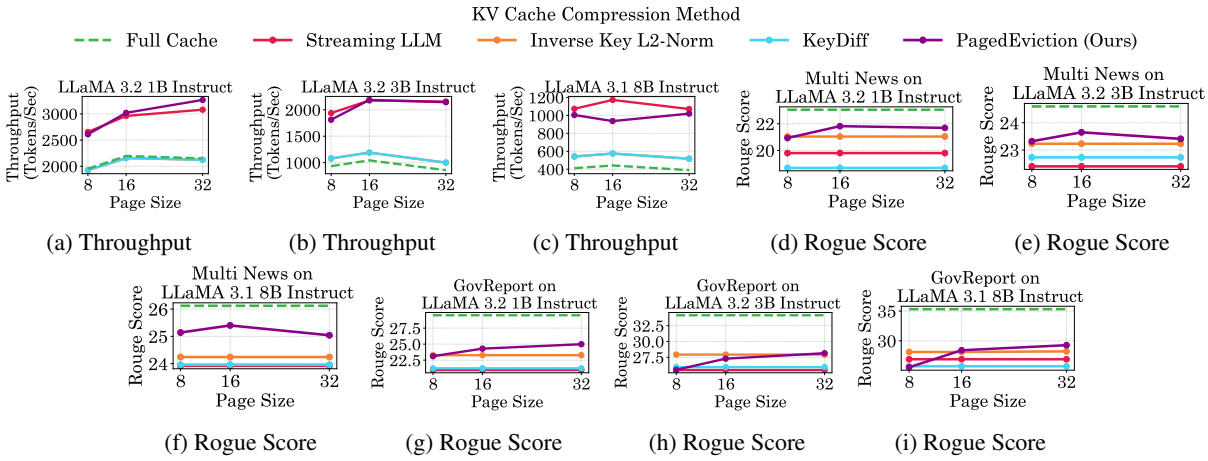
Figures 3 (a), (b), and (c) illustrates the throughput results for the LLaMA-1B, LLaMA-3B, and LLaMA-8B models, respectively. PagedEviction consistently achieves better throughput than Inverse Key L2-Norm and KeyDiff while almost matching StreamingLLM across all cache budgets and models. For instance, at a cache budget of 1024, PagedEviction on LLaMA-1B model delivers ∼3020 tokens/sec, while StreamingLLM and Inverse Key L2-Norm/KeyDiff trail achieves only ∼2920 and ∼2170 tokens/sec, respectively, showing a 4.1% and 39% throughput improvement. PagedEviction outperforms Full Cache baseline (green dashed line at ∼2200 tokens/sec) by 37% at cache budget 1024, while almost retaining the accuracy. Overall, these results validate that PagedEviction offers both higher efficiency and robustness across a wide range of memory budgets. StreamingLLM is straightforward to integrate into vLLM, as it simply requires zeroing out the last token and evicting the oldest block once it is full. However, it incurs overhead by evicting one token per decoding step and updating the KV cache block table at every step, which can be computationally expensive. In contrast, our proposed PagedEviction method performs eviction at fixed intervals, significantly reducing overhead. As a result, PagedEviction achieves comparable throughput to StreamingLLM despite some extra computation of calculating L2-Norms. On the other hand, unstructured eviction methods operate across the entire sequence and only evict a block once all its tokens have been individually evicted. This requires frequent checks across all blocks and prevents block-level eviction, offering little to no speedup during the decode phase. More visualizations of StreamingLLM and Unstructured Eviction (Inverse Key L2-Norm) and depicted in Appendix A (Figures 5 and 6).

Figure 3 (d) depicts the time per output token across LLaMA models under cache budget of 1024. Our *Paged Eviction* method consistently reduces latency compared to the Full Cache baseline, achieving reductions of approximately 12%, 10%, and 11% for LLaMA-1B, 3B, and 8B models, respectively. Compared to StreamingLLM, Paged



(a) LLaMA-1B (b) LLaMA-3B (c) LLaMA-8B (d) Time Per Output Token

Figure 3: (a,b,c) Cache Budget vs Throughput of Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct and Llama-3.1-8B-Instruct models and (d) Time Per Output Token of 1B, 3B and 8B models



(a) Throughput (b) Throughput (c) Throughput (d) Rogue Score (e) Rogue Score

(f) Rogue Score (g) Rogue Score (h) Rogue Score (i) Rogue Score

Figure 4: Throughput (a–c) and ROUGE Score (d–i) across different LLaMA models (1B, 3B, 8B) and datasets (MultiNews, GovReport) for various KV cache compression methods and page sizes. PagedEviction consistently delivers high throughput—up to 3.1× over Full Cache while maintaining near-optimal accuracy with less than 3–5% degradation. It outperforms Inverse Key L2-Norm and KeyDiff, which suffer from substantial accuracy drops.

Eviction yields similar or slightly lower latency, indicating its efficiency despite less frequent evictions. The latency under Paged Eviction scales sublinearly with model size, highlighting its effectiveness in managing memory.

5.5 Ablation Study: Varying Page Sizes

We conduct an ablation study by varying the page size and evaluating its impact on both accuracy and throughput. In Figure 4, we compare throughput and model accuracy across two datasets for different page sizes under different KV cache compression methods. Across all settings, we consistently achieve a strong balance between throughput and accuracy. For instance, PagedEviction improves over Full Cache throughput by up to 3.1× and closely matches StreamingLLM, especially at page sizes 16 and 32. In terms of ROUGE score (d–i), PagedEviction yields less than 3–5% degradation from Full Cache, outperforming other attention-free baselines. While KeyDiff and Inverse Key L2-Norm show significantly lower accuracy (20% drop in (d)), we maintain robustness across both datasets and model scales, demonstrating its effectiveness on different page sizes.

6 Conclusion

We propose PagedEviction, a structured block-wise KV cache eviction method for vLLM's PagedAttention. Unlike existing methods that rely on attention scores or per-token comparisons, PagedEviction leverages a block-wise importance based on Key and Value states. This design preserves the structural integrity of memory blocks while minimizing runtime overhead. Our method delivers high compression efficiency with minimal degradation in accuracy, achieving within 0.5-1.5 ROUGE points of full-cache performance at tight budgets (1024 tokens), while significantly improving throughput by up to 3.1× over Full Cache. Across LLaMA-1B, 3B, and 8B models, PagedEviction consistently yields 10–12% lower latency, 15–20% better accuracy than other methods like StreamingLLM.

7 Limitations

While we demonstrate the effectiveness of our proposed approach across various datasets and vLLM, our methodology has certain limitations.Our L2-norm calculation, block eviction and updating the block table are currently implemented in Python, leading to suboptimal speedup. These components could be further optimized by creating a custom kernel for KV cache pruning to improve efficiency.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, and 1 others. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A simple and effective l_2 norm-based strategy for kv cache compression. *arXiv preprint arXiv:2406.11430*.

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. 2024. Qaq: Quality adaptive quantization for llm kv cache. *arXiv preprint arXiv:2403.04643*.

Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. 2019. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2024. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. 2024. Zipcache: Accurate and efficient kv cache quantization with salient token identification. *arXiv preprint arXiv:2405.14256*.

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

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*.

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

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024a. Minicache: Kv cache compression in depth dimension for large language models. *arXiv preprint arXiv:2405.14366*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2024b. Scissorhands: Exploiting the persistence of importance

hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024c. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.

Meta. 2024. Llama-3.1-8b-instruct. https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct.

Junyoung Park, Dalton Jones, Matthew J Morse, Raghavv Goel, Mingu Lee, and Chris Lott. 2025. Keydiff: Key similarity-based kv cache eviction for long-context llm inference in resource-constrained environments. *arXiv preprint arXiv:2504.15364*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, and Mi Zhang. 2024. D2o: Dynamic discriminative operations for efficient generative inference of large language models. *arXiv preprint arXiv:2406.13035*.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024a. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. 2024b. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yichi Zhang, Bofei Gao, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, Wen Xiao, and 1 others. 2024a. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024b. Cam: Cache merging for memory-efficient llms inference. In *Forty-first international conference on machine learning*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2024c. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209.

A Appendix

A.1 StreamingLLM

Figure 5 illustrates the StreamingLLM eviction strategy. Initially, prompt tokens are processed, where only the earliest tokens (used as attention sinks) are retained, and the remaining prompt tokens are evicted based on recency. The KV cache is then organized into fixed-size blocks. In each decoding step, one token is evicted to make room for a new token, simulating a sliding window behavior. During decoding, one token is evicted per step. The oldest block is completely evicted only when all tokens are evicted from the block.

A.2 Unstructured Eviction

Figure 6 demonstrates the behavior of unstructured eviction methods such as Inverse Key L2-Norm or KeyDiff. Unlike structured strategies, unstructured eviction selects individual tokens for removal based on token-level importance scores, without regard for block structure. During prefill, tokens with lower importance are pruned. During decoding, the lowest-scoring token is removed in every step, which causes token-level fragmentation across blocks. This prevents entire blocks from being evicted efficiently, resulting in suboptimal

memory utilization and requiring constant cache management.

B Related Work - KV Cache Optimization

B.1 KV Cache Compression

Various strategies have been developed to optimize KV cache memory by evicting less important tokens based on importance and relevance. H2O (Zhang et al., 2024c) employs accumulative normalized attention scores to retain high-impact tokens, known as Heavy Hitters, while ensuring recent tokens are preserved due to their strong correlations with current tokens. StreamingLLM (Xiao et al., 2023) highlights the critical role of preserving key-value pairs from initial sequence tokens to maintain model performance. Keyformer (Adnan et al., 2024) addresses distortions in softmax probability distributions caused by token removal by introducing regularization techniques to smooth and approximate the original distribution. FastGen (Ge et al., 2023) adopts a hybrid strategy, selecting token retention policies during prompt encoding (e.g., keeping special, punctuation, recent, or attention-weighted tokens) and applying these during decoding. SnapKV (Li et al., 2024) simplifies this by focusing solely on retrieving tokens based on importance scores, emphasizing that only a subset of prompt tokens is crucial for response generation. Scissorhands (Liu et al., 2024b) leverages the temporal significance of historically important tokens, preserving repetitive attention patterns through selective retention.

B.2 KV Cache Merge

Several methods have been proposed to optimize KV cache storage by leveraging token similarity for merging. MiniCache (Liu et al., 2024a) identifies high angular similarity in KV caches of middle-to-deep layers and merges the Key and Value pairs of adjacent similar layers into a shared representation. D2O (Wan et al., 2024) merges the Key or Value of evicted tokens with retained tokens based on cosine similarity. KVMerger (Wang et al., 2024) clusters consecutive tokens with high cosine similarity to group contextually relevant tokens for merging. CaM (Zhang et al., 2024b) uses attention scores to merge Keys or Values of multiple evicted tokens with retained tokens, producing a final merged representation.

B.3 Budget Allocation

Many papers have explored hierarchical and adaptive strategies for KV cache management to optimize memory allocation and maintain model performance. PyramidInfer (Yang et al., 2024a) adopts a layer-wise approach, assigning more weight to recent tokens and using a decay ratio to reduce KV cache lengths in deeper layers, forming a pyramid structure. It also dynamically updates significant tokens during decoding based on attention values. PyramidKV (Zhang et al., 2024a) employs a similar pyramid-shaped memory allocation strategy, allocating larger cache capacities to lower layers with uniform attention distributions and progressively reducing capacity in upper layers where attention is more concentrated on specific tokens. AdaKV (Feng et al., 2024) introduces head-specific memory allocation by leveraging distinct attention patterns across heads, optimizing cache distribution within a layer-wise budget using an L1 loss bound to preserve multi-head attention outputs.

B.4 KV Cache Quantization

There have been several quantization strategies proposed to optimize KV cache compression in LLMs while minimizing performance degradation. QAQ (Dong et al., 2024) employs separate quantization for key and value caches, using an attention window to predict future attention scores and avoid over-compression of crucial tokens. GEAR (Kang et al., 2024) integrates uniform quantization, low-rank approximation for residuals, and sparse matrices to handle outlier errors. KVQuant (Hooper et al., 2024) addresses outliers by quantizing keys per channel before Rotary Positional Embedding (RoPE) and values per token, while retaining the first token in full precision to preserve performance. KIVI (Liu et al., 2024c) adopts mixed strategies by quantizing keys per channel and values per token, keeping recent KV pairs in full precision due to their critical role in token generation. MiKV (Yang et al., 2024b) uses mixed precision based on token importance, storing less significant KV pairs at a lower precision. ZipCache (He et al., 2024) focuses on accurately computing token importance for efficient compression, while Atom (Zhao et al., 2024) applies fine-grained group quantization, handling outliers with higher precision and compressing normal channels to INT4 for maximum efficiency.

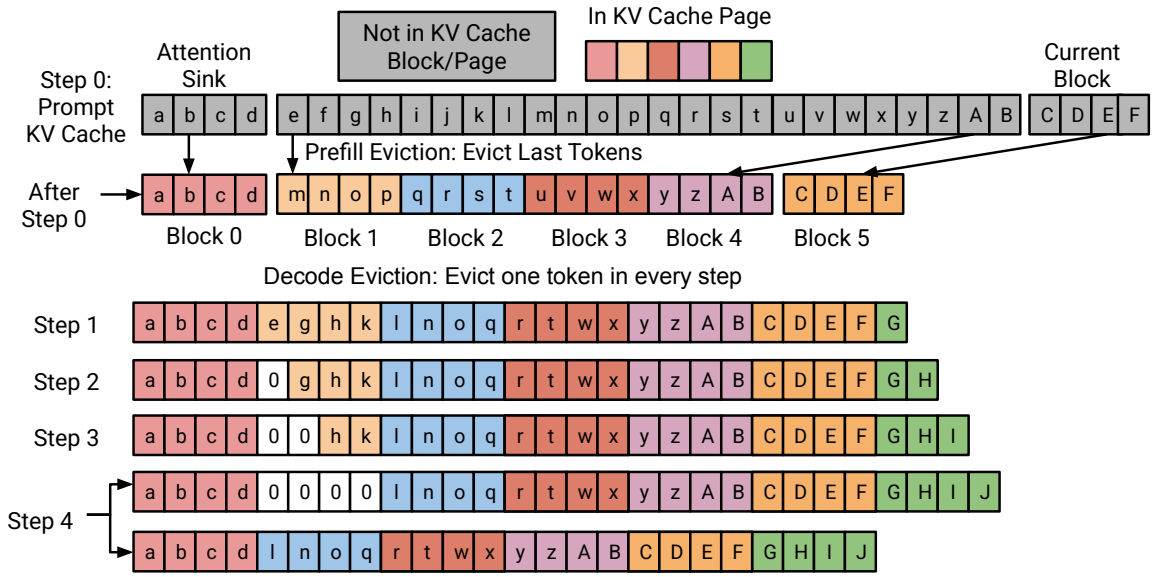


Figure 5: Illustration of the StreamingLLM KV cache eviction strategy. During prefill (Step 0), the last tokens are evicted to fit new blocks while preserving the first few tokens as attention sinks. During decoding, one token is evicted per step. The oldest block is completely evicted only when all tokens are evicted from the block. This approach ensures continuous streaming by maintaining a sliding window over the KV cache.

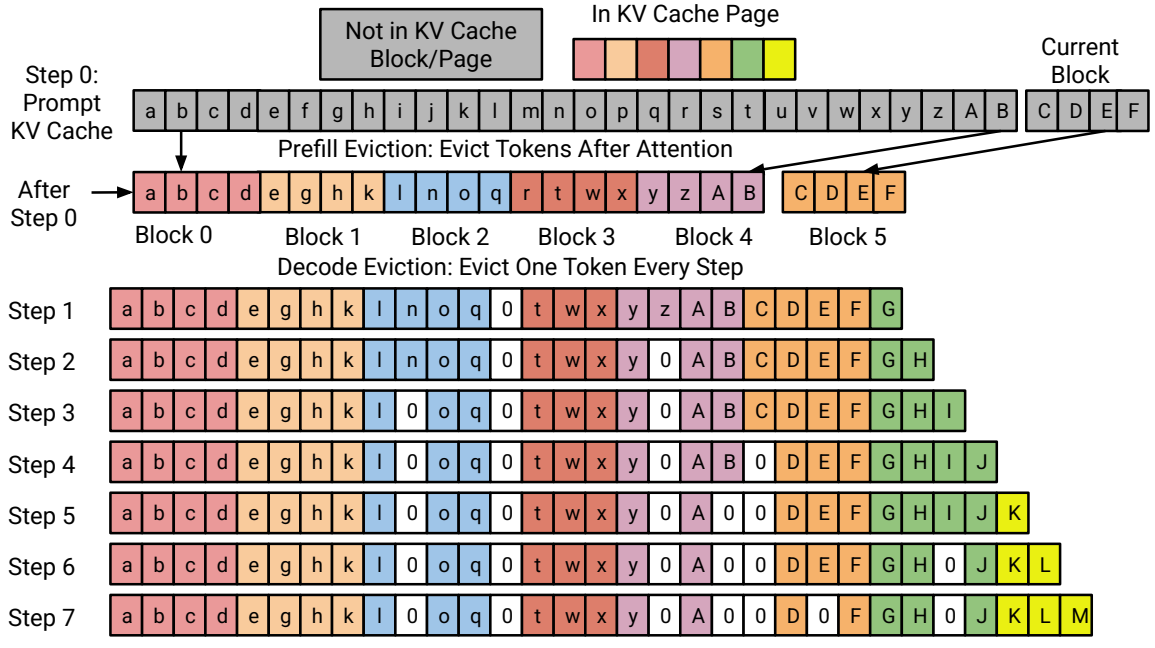


Figure 6: Illustration of Unstructured KV Cache Eviction using token-wise importance metrics such as Inverse Key L2-Norm or KeyDiff. During prefill (Step 0), low-importance tokens are evicted until we reach the cache budget. In the decode phase, one token is evicted per step based on its importance score, regardless of its position or block alignment. This leads to fragmented block occupancy, reducing the potential for full block eviction and resulting in inefficient memory usage.