TAKE: Task-Aware Chunked KV Cache Eviction for Efficient Long-Context LLM Prefill

Anonymous ACL submission

Abstract

Memory bottlenecks severely constrain the deployment of large language models (LLMs) in long-context scenarios. While existing global pruning methods reduce memory usage during the decoding phase, they largely ignore out-of-memory errors in the prefill stage before generation begins. To address this, we present **Task-Aware Chunked KV Cache Eviction (TAKE)**, a training-free framework for memory-efficient long-context prefill. TAKE refines the prefill process by partitioning long sequences into chunks and performing incremental KV eviction, thereby avoiding the computational overhead of full-sequence processing. Crucially, to mitigate the semantic discontinuity introduced by chunk-wise processing, we introduce accumulated task-aware probe tokens and a delayed eviction strategy to reduce information loss. Extensive evaluations show that TAKE achieves the highest efficiency, slashing peak memory usage to 59.53% of the baseline while accelerating Time-To-First-Token (TTFT) by over 60% under 128k-token inference. Our code is available at Source Code of TAKE.

1 Introduction

Large Language Models (LLMs) have shown strong capabilities in processing long-context inputs (Jin et al., 2025; Wang et al., 2024; Li et al., 2024a), with recent models (Team et al., 2026; Meta AI, 2025; Qwen Team, 2025) supporting context windows beyond 128k tokens. However, such long-context capabilities remain difficult to deploy in practice due to the substantial memory cost of key-value (KV) caches. In particular, the prefill stage must instantiate KV states for all input tokens before generation starts, causing peak memory consumption to scale with sequence length and often leading to out-of-memory (OOM) failures on resource-constrained devices.

Some approaches design efficient architectures, such as MLA (Liu et al., 2024a) and MoBA (Lu et al., 2025), to compress the KV cache. Although these techniques achieve effective performance, they require custom model architectures and pretraining, limiting their applicability to existing LLMs. Another line of inference acceleration methods relies on training-free KV eviction (Feng et al., 2025; Park, 2025). However, a critical "Prefill Paradox" remains unresolved: global KV eviction methods often need to first compute the KV cache for the entire input sequence before pruning it. These methods do not fundamentally address the memory bottleneck of the prefill stage.

To address the above limitations, we present Task-aware Chunked KV Cache Eviction (TAKE), a training-free framework that decouples memory usage from sequence length for efficient long-context prefill. As illustrated in Figure 1, unlike global pruning methods that prune after full KV computation, TAKE performs eviction during incremental chunked computation. We partition the sequence and interleave attention computation with immediate eviction, ensuring the peak memory footprint never exceeds a fixed size. The main technical challenge is how to preserve task-relevant information while evicting KV entries before seeing the entire sequence. First, we design an *accumulated task-aware probe* mechanism. The probe token is inserted into the current chunk and attends to both the newly processed tokens and the retained cache, allowing it to collect task-relevant saliency signals during chunked prefill. Second, we introduce a *delayed eviction* strategy. Instead of pruning shallow-layer KV caches immediately, TAKE postpones shallow-layer eviction until more reliable saliency patterns emerge in deeper layers.

We evaluate TAKE on diverse long-context benchmarks, including Needle-in-a-Haystack (NIAH) and LongBench, across different models. TAKE consistently reduces prefill memory usage and first-token latency, while maintaining strong performance across tasks. Without quantization

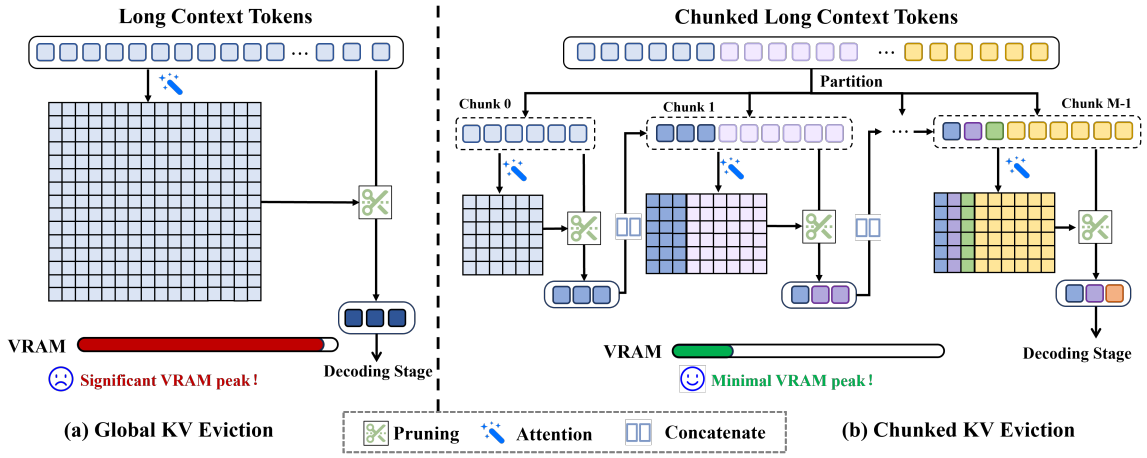


Figure 1: Comparisons of global KV eviction and chunked KV eviction strategies at prefill stage.

or offloading, TAKE enables stable 128k-token inference within 24GB VRAM budget, where existing methods fail due to OOM in prefill stage. We summarize our contributions as follows:

- We propose TAKE, a training-free chunked KV eviction framework that interleaves prefill computation with incremental eviction, thereby bounding peak memory by the chunk size and enabling constant-memory prefill.
- We design accumulated task-aware probe tokens and a delayed eviction strategy to mitigate semantic discontinuity in chunked prefill and improve the reliability of KV selection.
- Extensive experiments validate that TAKE outperforms the baseline methods in resource consumption and inference speed without compromising long-context performance.

2 Related Work

Research on KV cache eviction in long-context LLM inference encompasses methods ranging from simple sliding-window to sophisticated eviction (Adnan et al., 2024; Dong et al., 2024). StreamingLLM (Xiao et al., 2023) retains initial and recent tokens, but risks discarding crucial distant information. H2O (Zhang et al., 2023) preserves tokens with high accumulated attention scores over time, while SnapKV (Li et al., 2024b) and MorphKV (Ghadia et al., 2025) leverage attention distributions of recent queries. ChunkKV (Liu et al., 2025) treats contiguous semantic chunks rather than individual tokens as basic eviction units, and KVzip (Kim et al., 2025) quantifies KV pair importance by their contribution to context reconstruction for query-agnostic eviction. Locret (Huang et al., 2024) learns retaining heads for KV eviction and integrates them with chunked prefill. InfiniPot (Kim et al., 2024) continually compresses past context into fixed-size memory for training-free long-context inference. KeyDiff (Park, 2025) identifies important tokens via cosine similarity of the key vector, and DefensiveKV (Feng et al., 2026) challenges the prevalent stability assumption that underlies most eviction methods and proposes worst-case risk-aware aggregation.

Some studies exploit the heterogeneity across layers and heads. PyramidInfer (Yang et al., 2024) and CAKE (Qin et al., 2025) adapt KV budgets per layer based on how attention patterns evolve with depth. GemFilter (Shi et al., 2024) and FastKV (Jo et al., 2025) select critical tokens at early layers and propagate only their hidden states deeper. AdaKV (Feng et al., 2025) allocates per-head KV budgets via top-k selection, while FastGen (Ge et al., 2023) and RazorAttention (Tang et al., 2025) assign head-specific compression policies.

Some KV cache eviction methods incorporate structural and semantic adaptivity. SepLLM (Chen et al., 2025) aggregates segment information through separator tokens, and ClusterKV (Liu et al., 2024b) compresses KV to semantic cluster centers. RocketKV (Behnam et al., 2025) employs a fine-grained compression strategy across both sequence and head dimensions. TRIM-KV (Bui et al., 2025) learns per-token retention gates via lightweight MLPs, and Expected Attention (Devoto et al., 2025) estimates KV importance from the distribution of future queries.

3 Chunked KV Eviction

3.1 Chunked Prefill

Chunked prefill (Agrawal et al., 2023) is the prerequisite and original motivation for the effectiveness of chunk-wise KV eviction. Chunked prefill optimizes memory and throughput by partitioning long input sequences into fixed-size chunks of length Z. These chunks are processed incrementally, while the KV cache is updated sequentially via concatenation to form a cumulative history. Notably, all intermediate representations, including both KV and layer-wise hidden states, produced under chunked prefill are identical to those from a standard prefill stage (see in the Appendix A for a proof). Therefore, chunked prefill is equivalent to the global prefill while enabling memory-efficient long-context inference, without compromising accuracy.

3.2 Analysis of chunked KV eviction

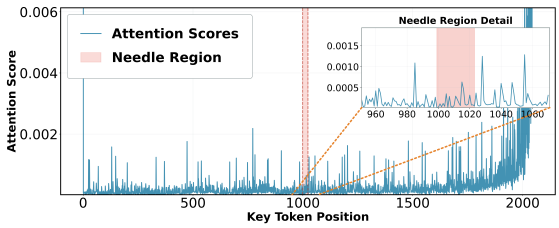
Chunked KV eviction refers to a method that interleaves KV cache eviction within the chunked prefill process. Specifically, it immediately prunes the KV cache after each chunk's computation, thereby maintaining a stable KV cache size throughout the prefill stage and achieving inference acceleration. However, there are several limitations of chunked KV eviction that degrade model performance.

3.2.1 Semantic Discontinuity in Local Chunks

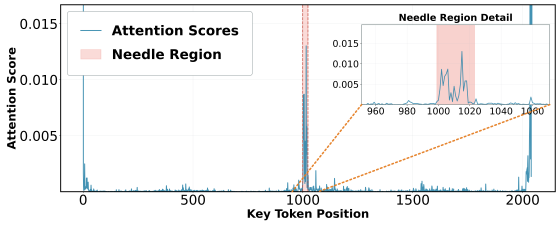
In global pruning, the model looks at the entire history to identify the critical tokens. However, in chunk-wise eviction, the attention computation is confined to the current chunk and the compressed summary of previous chunks. This fragmentation weakens global coherence, as each chunk makes optimal pruning decisions locally without full contextual information. Consequently, critical information from distant contexts may be discarded prematurely if its relevance is not immediately apparent within the local chunk's limited scope.

3.2.2 Attention Bias in Chunks

As shown in Figure 2, the model's attention distribution exhibits a strong tendency to concentrate on the initial and the terminal region of the sequence. In the context of chunk-wise KV eviction, this inherent bias leads the model to assign elevated importance scores to temporally adjacent tokens, as well as to the earliest head tokens in the sequence. Consequently, the eviction mechanism becomes biased toward retaining the current chunk's KV pairs. It therefore aggressively discards the historical tokens from preceding chunks, leading to a rapid loss of global context and long-range information.



(a) Attention scores of last token at Layer 0.



(b) Attention scores of last token at Layer 15.

Figure 2: Attention scores of different layers on NIAH.

3.2.3 Low Attention Sparsity in Shallow Layers

Effective KV eviction relies on the assumption of attention sparsity, that only a few tokens are critical. However, shallow layers typically function as information gatherers rather than discriminators. As visualized in Figure 2, early layers (e.g., Layer 0) exhibit a high entropy, nearly uniform attention distribution, assigning comparable weights to almost all tokens to construct initial semantic representations. In contrast, deeper layers (e.g., Layer 15) present clear sparsity, focusing sharply on specific target tokens. When forcing a limited budget on dense attention layers, the eviction mechanism is compelled to make premature selection decisions based on indistinguishable signals, often discarding semantically vital but not yet highlighted tokens.

4 Method

4.1 Overview of TAKE

To address these challenges, TAKE integrates two complementary mechanisms: accumulated task-aware probe and delayed eviction to improve the accuracy of identifying important tokens and reduce information decay during chunked prefill.

Figure 3 provides an overview of the chunked KV eviction in TAKE. Given an input sequence of length N, we partition it into M non-overlapping chunks $\{C_0, C_1, \ldots, C_{M-1}\}$ with the size at most

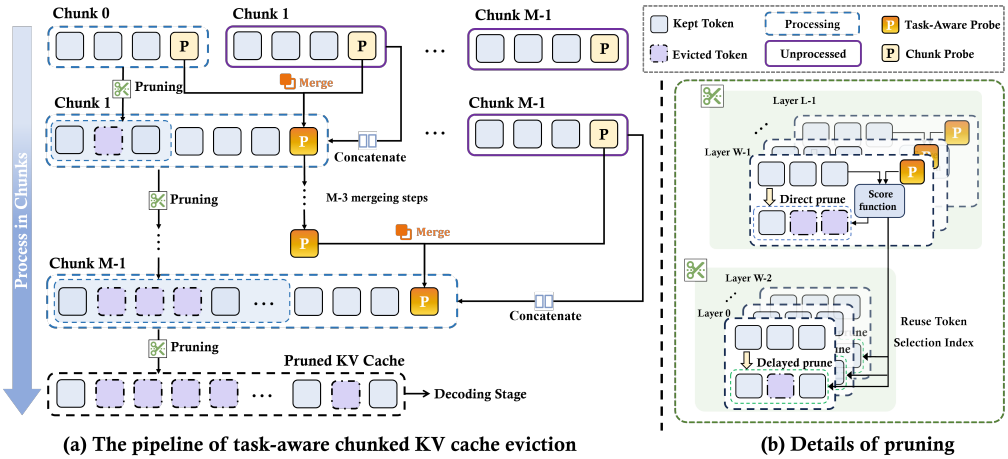


Figure 3: Overview of the proposed TAKE method. The left shows iterative chunk pruning with probe fusion to produce a final KV cache of size B. The right shows pruning within a chunk, where layer $W-1$ directs pruning for all prior layers' cache, while subsequent layers are pruned directly.

Z. Each chunk appends p task-related probe tokens P, which propagate task-relevant semantics and facilitate key-token identification. We perform a complete forward propagation of L decoder layers and KV cache eviction within each chunk. For the i-th chunk at layer ℓ, we first merge its KV states $(K_{i,\ell}^{\text{chunk}}, V_{i,\ell}^{\text{chunk}})$ with the pruned summary $(\widetilde{K}_{i-1,\ell}, \widetilde{V}_{i-1,\ell})$ from the previous chunk:

$$K_{i,\ell} = \text{Concat}(\widetilde{K}_{i-1,\ell}, K_{i,\ell}^{\text{chunk}}), \quad (1)$$

$$V_{i,\ell} = \text{Concat}(\widetilde{V}_{i-1,\ell}, V_{i,\ell}^{\text{chunk}}). \quad (2)$$

Secondly, the output of the i-th chunk's attention operation at layer ℓ is computed with $Q_{i,\ell}^{\text{chunk}}$, the query states of C_i at layer ℓ. At last, the previous chunk's task-aware probe $Q_{i-1,\ell}^{\text{aprobe}}$ is merged with the current chunk's probe and input into the score function as shown in Figure 3(b). The probe identifies important tokens among the current KV cache $(K_{i,\ell}, V_{i,\ell})$ and keeps them in memory.

A delayed eviction strategy is employed to further reduce information loss. It defers eviction for the first $W-1$ layers, and once pruning is enabled at layer $W-1$, the selected indices $\mathcal{I}_{i,W-1}$ of tokens are retrospectively applied to prune KV caches in all preceding layers. The pruning is formulated as:

$$(\widetilde{K}_{i,\ell}, \widetilde{V}_{i,\ell}) = \begin{cases} \text{Prune}(\mathcal{I}_{i,W-1}, K_{i,\ell}, V_{i,\ell}), & \ell \leq W-1, \\ \text{Prune}(\mathcal{I}_{i,\ell}, K_{i,\ell}, V_{i,\ell}), & \text{otherwise}, \end{cases} \quad (3)$$

$$\mathcal{I}_{i,\ell} = \text{Score}(Q_{i,\ell}^{\text{aprobe}}, K_{i,\ell}, k_{i,\ell}), \quad (4)$$

where W is the number of initial warm-up layers. A relaxed KV cache budget B_{warmup} is applied before the last chunk to avoid excessive pruning. $\mathcal{I}_{i,\ell}$ is the selection index of top-$k_{i,\ell}$ tokens based on a score function and $k_{i,\ell}$ is the budget for KV cache of the i-th chunk at layer ℓ. The pruned cache $(\widetilde{K}_{i,0\cdots L-1}, \widetilde{V}_{i,0\cdots L-1})$ is propagated forward to support attention computation in the next chunk C_{i+1}, where further KV eviction continues. The details of task-aware probe and delayed eviction are presented in Sec. 4.2 and Sec. 4.3. Once the final chunk C_{M-1} is processed, the pruned cache is retained for auto-regressive decoding.

4.2 Accumulated Task-aware Probe

The accumulated task-aware probe mechanism is designed to address the semantic discontinuity in Sec. 3.2.1. It utilizes the task instructions to establish foundational semantic features that serve as global anchors across chunks. To mitigate attention bias illustrated in Sec 3.2.2, we further propose fusing the probe features incrementally along with chunk-wise iteration. This strategy progressively incorporates historical semantics, thereby increasing the weight of past information within the probe's representation to counteract local overfitting. We maintain the accumulated probe $Q_{i,\ell}^{\text{aprobe}}$ via an exponential moving average (EMA):

$$Q_{i,\ell}^{\text{aprobe}} = \alpha\, Q_{i-1,\ell}^{\text{aprobe}} + (1-\alpha)\, Q_{i,\ell}^{\text{probe}}, \alpha \in [0,1], \quad (5)$$

where $Q_{i,\ell}^{\text{aprobe}}$ is the query states of the accumulated probe to evaluate the token importance and α is a hyperparameter to balance semantics informa-

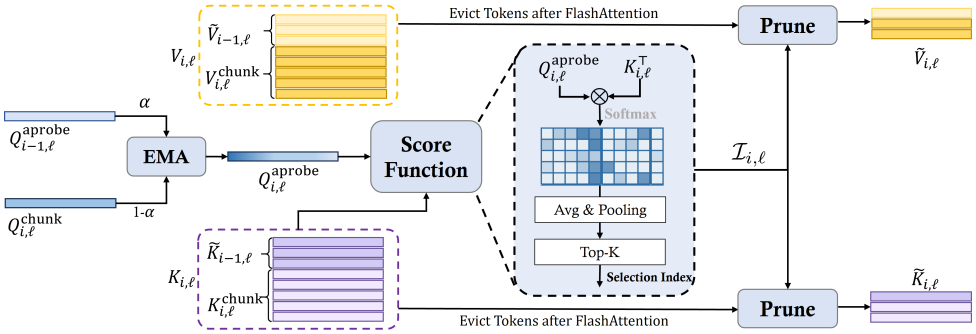


Figure 4: Probe-based KV eviction in TAKE.

tion between previous chunks and new chunks.

Crucially, due to causal masking, these probe tokens function strictly as evaluators for context tokens, ensuring that the integrity of the original context is preserved. Detailed illustration of KV eviction is shown in Figure 4 and formalized as the following equations:

$$A_{i,\ell} = \text{Softmax}\left(\frac{Q_{i,\ell}^{\text{aprobe}}(K_{i,\ell})^{\top}}{\sqrt{d}}\right) \in \mathbb{R}^{b\times h\times p\times(Z+k_{i-1,\ell})}, \tag{6}$$

$$S_{i,\ell} = \text{AvgPool}\left(\frac{1}{p}\sum_{u=1}^{p} A_{i,\ell}[:,:,u,:];\ \text{dim}=2\right) \in \mathbb{R}^{b\times h\times(Z+k_{i-1,\ell})}, \tag{7}$$

$$\mathcal{I}_{i,\ell} = \text{Top-}k_{i,\ell}(S_{i,\ell};\ \text{dim}=2)\,, \tag{8}$$

$$(\widetilde{K}_{i,\ell}, \widetilde{V}_{i,\ell}) = \text{Gather}\left((K_{i,\ell}, V_{i,\ell}), \mathcal{I}_{i,\ell}\right). \tag{9}$$

TAKE computes the matrix inner product between $Q_{i,\ell}^{\text{aprobe}}$ and $K_{i,\ell}^{T}$, and then applies softmax, producing a tensor $A_{i,\ell} \in \mathbb{R}^{b\times h\times p\times(Z+k_{i-1,\ell})}$. Here, b denotes batch size, h denotes the number of attention heads, and d is the hidden state dimension. We average $A_{i,\ell}$ along the probe length dimension, and then apply mean pooling over the last dimension to form token importance scores $S_{i,\ell}$. Finally, the collection of selected token index $\mathcal{I}_{i,\ell}$ is obtained through sorting $S_{i,\ell}$, based on which $(\widetilde{K}_{i,\ell}, \widetilde{V}_{i,\ell})$ are gathered from $(K_{i,\ell}, V_{i,\ell})$.

For stability, TAKE also employs a structural protection mechanism: the initial tokens and the local tokens are assigned effectively infinite importance and are thus never pruned. To preserve the functional diversity of multi-head attention, there is no aggregation applied across the head dimension.

4.3 Delayed Eviction

The observation in Sec. 3.2.3 regarding dense attention in shallow layers highlights a critical vulnerability in chunk-wise eviction. In contrast to global pruning, where tokens are ranked within a global scope, chunk-wise methods make irrevocable selection decisions at each step. This localized pruning leads to early-stage information loss propagating forward, preventing deeper layers from accessing essential context and causing a cumulative performance decline. To address this limitation, the TAKE framework implements a delayed KV eviction strategy along two distinct dimensions: the chunk-level and the layer-level.

Chunk-level delayed pruning For the initial W shallow layers, a relaxed budget B_{warmup} is applied, while deeper layers are pruned to the final target budget B due to attention sparsity. At the last chunk, shallow-layer caches are reduced from B_{warmup} to the target size B. Chunk-level delayed pruning makes shallow layers to integrate cross-chunk evidence before eviction, yielding semantic representations while ensuring decoding starts with a consistent cache size.

Layer-level delayed pruning Since shallow layers mainly handle syntactic and semantic parsing rather than token discrimination selection, TAKE synchronizes pruning decisions from a deeper key layer. When pruning is first enabled at layer W, the token indices selected at this layer are used to update the KV caches of all preceding layers simultaneously. This key-layer decision, global synchronization strategy ensures that shallow layer caches inherit reliable importance decisions without requiring separate scoring and selection, thereby reducing computation overhead and preserving early task-relevant semantic features.

Table 1: LongBench results of LLaMA-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3 with $128L$ and $512L$ KV budgets. The best results are **bolded**.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Coding		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep.	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RB-P	
LLaMA-3.1-8B-Instruct, *KV Budget = Full*															
Full KV	30.21	45.53	55.01	56.01	46.65	31.28	35.13	25.28	27.25	73.00	91.64	43.80	63.38	56.64	48.63
LLaMA-3.1-8B-Instruct, *KV Budget = 128L*															
SnapKV	27.23	30.96	50.62	**53.04**	43.13	28.66	22.70	**23.16**	21.90	62.00	91.19	40.37	58.37	49.09	43.03
AdaKV	**27.71**	30.13	50.27	51.65	40.87	29.21	**23.42**	23.08	21.72	**67.50**	91.10	40.54	58.87	50.15	43.30
FastKV	27.49	28.05	**51.56**	52.12	44.54	28.78	22.12	23.04	20.77	59.50	91.06	40.79	59.03	49.16	42.71
CAKE	27.41	32.01	49.58	52.99	45.76	28.32	21.85	23.15	21.56	47.50	89.61	**40.90**	57.92	49.83	42.03
TAKE	27.39	**35.36**	51.46	52.94	**46.69**	**29.47**	22.00	22.25	**26.19**	59.00	**91.29**	39.06	**61.33**	**52.76**	**44.09**
LLaMA-3.1-8B-Instruct, *KV Budget = 512L*															
SnapKV	30.04	41.14	53.83	54.77	45.12	**31.20**	**27.08**	24.04	25.00	70.50	91.90	42.34	61.34	52.24	46.47
AdaKV	30.09	40.20	54.14	54.27	44.73	30.77	26.84	23.98	24.17	70.50	91.90	41.74	60.63	53.02	46.21
FastKV	29.96	40.88	54.55	54.78	46.19	30.76	26.63	24.05	24.29	**72.50**	91.37	**42.52**	62.14	52.65	46.66
CAKE	**31.82**	42.99	51.65	54.37	**46.89**	30.73	26.36	**24.94**	25.27	63.50	91.54	**42.52**	62.31	54.30	46.37
TAKE	30.45	**43.22**	**55.20**	**56.99**	46.77	30.18	26.44	23.77	**26.60**	70.50	**92.48**	41.84	**63.25**	**56.67**	**47.45**
Mistral-7B-Instruct-v0.3, *KV Budget = Full*															
Full KV	29.20	38.27	50.09	51.20	36.49	26.67	34.21	25.79	26.43	76.00	88.59	47.46	59.37	60.52	46.45
Mistral-7B-Instruct-v0.3, *KV Budget = 128L*															
SnapKV	23.77	28.60	45.64	44.60	29.85	21.92	22.66	22.56	21.32	47.50	87.94	43.32	54.67	51.75	39.00
AdaKV	24.54	**29.42**	**45.65**	42.40	28.66	22.18	**22.89**	**23.36**	21.53	**65.50**	88.89	**44.55**	53.89	**53.23**	40.18
FastKV	**24.97**	26.68	45.46	43.75	30.01	21.06	22.43	22.33	20.37	63.50	**89.24**	42.50	54.37	51.80	39.89
CAKE	22.31	29.15	43.51	44.51	30.36	**22.85**	21.56	20.47	18.96	47.00	88.60	39.36	44.96	46.19	37.13
TAKE	23.25	28.56	43.64	**48.80**	**34.32**	20.92	22.06	21.86	**25.63**	56.50	89.16	43.19	**58.40**	51.97	**40.43**
Mistral-7B-Instruct-v0.3, *KV Budget = 512L*															
SnapKV	26.03	34.82	**49.74**	49.25	34.11	23.88	**27.23**	24.33	24.59	74.00	**89.82**	45.94	58.13	58.49	44.31
AdaKV	**27.14**	35.10	48.65	48.98	34.30	23.80	25.76	**24.38**	24.27	**75.00**	89.14	45.87	58.18	**59.49**	44.29
FastKV	25.94	33.20	49.03	48.37	34.42	**25.43**	26.63	23.88	24.10	71.50	89.27	**46.49**	58.72	58.63	43.97
CAKE	25.82	34.24	46.93	45.34	31.71	23.38	25.81	21.47	22.41	68.00	88.46	41.27	49.04	49.67	40.96
TAKE	26.65	**35.61**	49.49	**50.11**	**37.42**	22.99	26.18	24.09	**25.98**	73.50	89.54	45.12	**59.86**	58.18	**44.62**

5 Experiments

5.1 Experimental Settings

LLMs & Evaluation Tasks Experiments are conducted on two LLMs with different sizes: Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Mistral-7B-Instruct-v0.3 (Jiang et al., 2023). We evaluate our method on two long-context benchmarks: NIAH (Kamradt, 2023) and LongBench (Bai et al., 2023). NIAH embeds sparse key information into long and distracting texts. LongBench is a multitask suite for long-context evaluation, covering extended-context QA, summarization, few-shot learning, and coding tasks.

Baseline Methods. We compare TAKE against Full-KV (no eviction), SnapKV (Li et al., 2024b) (importance-based token retention), AdaKV (Feng et al., 2025) (per-head adaptive budgets), FastKV (Jo et al., 2025) (joint hidden state and KV pruning), and CAKE (Qin et al., 2025) (layer-adaptive cascading eviction).

Implementation Details TAKE is built upon the HuggingFace Transformers library, ensuring compatibility with FlashAttention-2 (Dao, 2024) backends. For hyperparameter settings, we apply delayed pruning to the initial 50% of the model layers and B_{warmup} is decided by context length. The EMA factor α in the accumulated probe is set to 0.32 for NIAH and 0.2 respectively for LongBench based on task properties. Details in Appendix D.

5.2 Evaluation on NIAH and LongBench

Needle-in-a-Haystack As shown in Figure 5, we compare TAKE to baseline methods on the NIAH benchmark with $512L$ KV budget. The green denotes 100% accuracy while red denotes 0%. The results show that TAKE consistently outperforms baselines on NIAH, achieving stronger recall of sparse information. In Appendix C.1, additional comparison results with different KV budgets are presented. Interestingly, TAKE even surpasses standard full-KV inference by filtering irrelevant content. These results demonstrate that reducing the KV cache size improves LLM performance when processing input sequences with excessive redundant information, further highlighting the effectiveness of task-aware eviction.

LongBench Table 1 reports results of LongBench on LLaMA-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3 with $128L$ and $512L$ KV budgets. With a KV budget of $512L$ on LLama-3.1-8B-Instruct, TAKE achieves an average score of 47.45, outperforming SnapKV by +0.98, AdaKV by +1.24, FastKV by +0.79, and CAKE by +1.08. TAKE consistently outperforms baseline pruning methods under limited KV budgets (see more

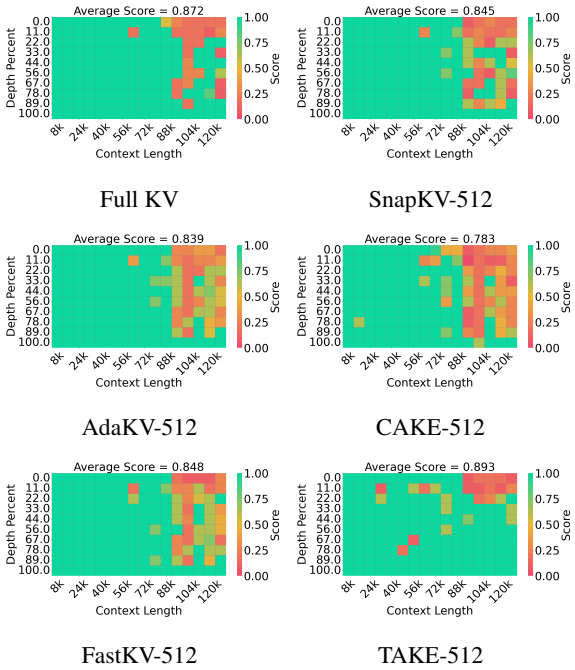
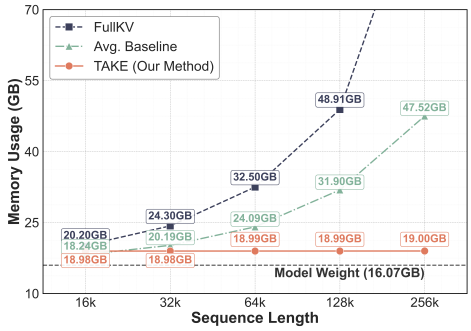
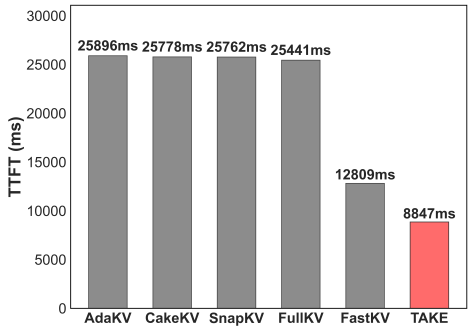


Figure 5: NIAH results of LLaMA-3.1-8B-Instruct with $512L$ KV budget.



(a) Memory usage of different context length. The Avg. Baseline is the average of SnapKV, AdaKV and FastKV methods. The baselines' memory consumption is so close (within 0.2GB) that averaging them does not hide any outlier. It simply serves as a clean, representative proxy.



(b) TTFT of 128k-token inference.

Figure 6: Memory usage and TTFT comparisons.

results in Appendix C.2), especially in multi-document question answering and coding tasks, where the effective integration of long-range dependencies and retention of key tokens are crucial. It greatly demonstrates the robustness and advancement of TAKE with various KV budgets.

5.3 Evaluation on Memory Usage and TTFT

Since eviction is interleaved with computation, the memory footprint at layer ℓ is bounded by $Z + k_{i,\ell}$ tokens, mitigating the VRAM peak. The memory usage and TTFT are evaluated on an H800 GPU by averaging results over multiple runs. Figure 6(a) shows that for 128k-token inference, full-KV and other global KV eviction methods approximately demand around 48.91GB and 31.90GB of GPU memory. By contrast, TAKE only requires 18.99GB due to its chunked prefill design, limiting the peak memory usage. This reduction translates to a 91.11% decrease in KV cache and activation memory compared to full-KV inference, and an 81.55% decrease compared to baseline methods.

In terms of inference speed, TAKE also demonstrates substantial gains. As illustrated in Figure 6(b), TAKE cuts prefill latency by over 60% compared to the baseline and by 30.9% compared to FastKV. Global pruning methods retain full-sequence attention before pruning, resulting in TTFT that matches or exceeds native full-KV due to added computation overhead. FastKV reduces hidden-state size but still computes attention over the entire sequence in shallow layers.

These memory and latency improvements arise from chunked eviction, which eliminates excessive padding and confines attention in subsequent chunks to pruned summaries rather than the full prior context. We further estimate floating-point operations (FLOPs) in Appendix B. For 128k-token inference, TAKE requires only 7.1% of the FLOPs of full-KV. Although chunking introduces extra computations, the overall reduction outweighs these overheads, resulting in a significant decrease in inference cost.

5.4 Ablation Study

Effect of chunk size. We investigate the sensitivity of TAKE to the chunk size Z regarding performance, memory footprint, and latency. As shown in Table 2, the choice of Z has a negligible impact on model performance, indicating that our accumulated probe mechanism is robust to the granularity of context partitioning. However, Z plays a crit-

ical role in resource consumption. Increasing Z from 2,048 to 8,192 results in a distinct rise in peak VRAM usage from 17.55GB to 20.32GB, as larger chunks require buffering more intermediate activations. Consequently, smaller chunk sizes are preferable to mitigate OOM risks.

Table 2: Impact of chunk size on efficiency and latency.

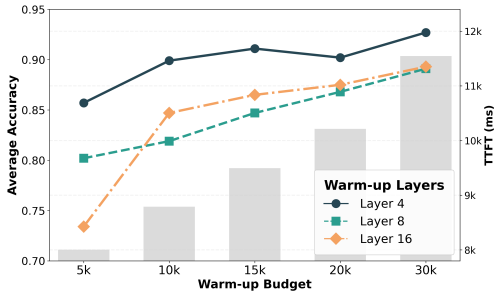
Chunk Size	Peak VRAM	Latency	Avg. Score
2048	17.55 GB	27.8 s	47.11
4096	18.22 GB	28.1 s	47.39
8192	20.32 GB	28.6 s	47.23

Effect of task-aware probe and delayed eviction. As shown in Table 3, we evaluate the effectiveness of the accumulated task-aware probe and delayed eviction on LLama-3.1-8B under the $512L$ KV budget. There are four settings: only chunk-wise eviction (chunk-only), chunk-wise delayed eviction without probe (w/o probe), chunk-wise eviction with probe (w/o delay), and chunk-wise delayed eviction with probe (TAKE). We report NIAH score of 64K-token sequences to avoid the saturation effects of short contexts and the collapse effects of extreme lengths. The accuracy of w/o delayed eviction on the NIAH reaches 80.00%, a 69.10% gain over the chunk-only methods. On LongBench datasets, the w/o delayed eviction method achieves a 19.43 improvement compared with chunk-only eviction. The results of chunk-wise delayed eviction without probe are the same as the results with only chunk-wise eviction on the two benchmarks. It confirms the inefficacy of delayed eviction when used in isolation and the cooperative efficacy of the two strategies in the TAKE.

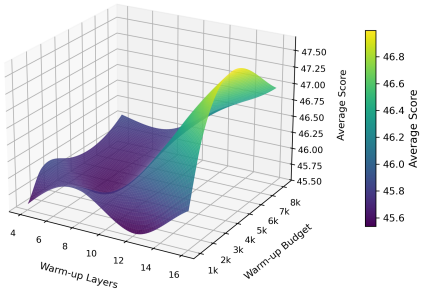
Table 3: Ablation study on LLama-3.1-8B under $512L$ KV Budget. The best results are **bolded**.

Configuration	NIAH Accuracy (64K-token)	LongBench (Macro Avg.)
Chunk-only	10.90%	26.46
w/o Probe	10.90%	26.46
w/o Delay	80.00%	45.89
TAKE (Ours)	**91.8%**	**47.45**

Effect of warm-up parameters. The number of warm-up layers W and the warm-up budget B_{warmup}, are critical in the delayed pruning. We perform analysis of W and B_{warmup} on NIAH and LongBench on LLama-3.1-8B with $512L$ budget shown in Figure 7(a) and Figure 7(b). For the



(a) Average scores (line) on NIAH across various warm-up parameters and TTFT (bar) of 128k-token inference with 16 warm-up layers.



(b) Average scores on Longbench across various warm-up parameters.

Figure 7: Analysis of warm-up parameters.

NIAH passkey retrieval task, reducing W improves accuracy, as it allows the model to start filtering out irrelevant information at an earlier stage. For warm-up budget, increasing B_{warmup} improves performance across both benchmarks by retaining more information in shallow layers. This performance gain introduces more TTFT and GPU memory consumption, highlighting a clear trade-off between accuracy and computation cost.

6 Conclusion

In this paper, we present Task-Aware Chunked KV Cache Eviction (TAKE), a training-free, chunk-wise KV cache eviction method that balances memory efficiency and model performance for long-context LLMs inference. TAKE revolutionizes KV cache management by interleaving computation with pruning, dramatically lowering peak GPU memory usage. This approach moves beyond traditional methods that perform pruning only after full attention computation. Extensive experiments on two typical benchmarks demonstrate that TAKE achieves superior memory savings and inference latency, enabling stable prefill with arbitrary input length. We discuss limitations in Appendix E.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J 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.

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. 2023. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*.

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*.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. 2025. RocketKV: Accelerating long-context LLM inference via two-stage KV cache compression. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pages 3358–3392.

Ngoc Bui, Hieu Trung Nguyen, Arman Cohan, and Rex Ying. 2025. Make each token count: Towards improving long-context performance with KV cache eviction. *arXiv preprint arXiv:2505.09649*.

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. 2025. Sepllm: Accelerate large language models by compressing one segment into one separator. In *Forty-second International Conference on Machine Learning*.

Tri Dao. 2024. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*.

Alberto Devoto, Seanie Lee, Yutong He, Tianyu Cui, Marta Marchiori Manerba, Haebeom Cho, Minseop Park, Sangwoo Mo, Se-Young Yun, Francesco Fournier, and Cees G.M. Snoek. 2025. KV cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*.

Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. 2024. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. In *International Conference on Machine Learning*, pages 11437–11452. PMLR.

Yuan Feng, Haoyu Guo, Junlin Lv, S. Kevin Zhou, and Xike Xie. 2026. DefensiveKV: Taming the fragility of KV cache eviction in LLM inference. In *The Thirteenth International Conference on Learning Representations (ICLR)*.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2025. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

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*.

Ravi Ghadia, Avinash Kumar, Gaurav Jain, Prashant J Nair, and Poulami Das. 2025. Dialogue without limits: Constant-sized kv caches for extended response in llms. In *Forty-second International Conference on Machine Learning*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Yuxiang Huang, Binhang Yuan, Xu Han, Chaojun Xiao, and Zhiyuan Liu. 2024. Locret: Enhancing eviction in long-context llm inference with trained retaining heads on consumer-grade devices. *arXiv preprint arXiv:2410.01805*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Bowen Jin, Jinsung Yoon, Jiawei Han, and Sercan O Arik. 2025. Long-context llms meet rag: Overcoming challenges for long inputs in rag. In *The Thirteenth International Conference on Learning Representations*.

Dongwon Jo, Jiwon Song, Yulhwa Kim, and Jae-Joon Kim. 2025. Fastkv: Kv cache compression for fast long-context processing with token-selective propagation. *arXiv preprint arXiv:2502.01068*.

Greg Kamradt. 2023. Needle in a haystack-pressure testing llms. *GitHub repository*.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. 2025. KVzip: Query-agnostic KV cache compression with context reconstruction. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Minsoo Kim, Kyuhong Shim, Jungwook Choi, and Simyung Chang. 2024. Infinipot: Infinite context processing on memory-constrained llms. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16046–16060.

Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Shafiq Joty, Soujanya Poria, and Lidong Bing. 2024a. Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources. In *The Twelfth International Conference on Learning Representations*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024b. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, and 1 others. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. 2024b. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. *arXiv preprint arXiv:2412.03213*.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. 2025. ChunkKV: Semantic-preserving KV cache compression for efficient long-context LLM inference. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, and 1 others. 2025. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*.

Meta AI. 2025. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. *Meta AI Blog*.

Jiwoo Park. 2025. Keydiff: Key similarity-based KV cache eviction for long-context LLM inference in resource-constrained environments. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. 2025. CAKE: Cascading and adaptive KV cache eviction with layer preferences. In *The Thirteenth International Conference on Learning Representations*.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhenmei Shi, Yifei Ming, Xuan-Phi Nguyen, Yingyu Liang, and Shafiq Joty. 2024. Discovering the gems in early layers: Accelerating long-context llms with 1000x input token reduction. *arXiv preprint arXiv:2409.17422*.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. 2025. Razorattention: Efficient kv cache compression through retrieval heads. In *The Thirteenth International Conference on Learning Representations*.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, and 307 others. 2026. Kimi k2.5: Visual agentic intelligence. *Preprint*, arXiv:2602.02276.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *The Twelfth International Conference on Learning Representations*.

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. 2024. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*.

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.

A Proof of Equivalence Between Chunked Prefill and Original Prefill

Let $X = (x_0, \ldots, x_{N-1})$ be an input sequence of N tokens, and denote $h_j^{(l)}$ as the hidden state of token x_j at layer ℓ. The base case at layer $\ell = 0$ corresponds to the token embedding layer, where both chunked prefill and original prefill receive identical raw input embeddings, thus $h_j^{(0)}$ are the same.

Assuming as induction hypothesis that for layer $l-1$, both methods produce identical hidden states $h_j^{(l-1)}$ for all tokens, we prove that this holds true at layer l as well. In layer l, the query, key and value vectors (q_j, k_j, v_j) of the token x_j are computed using linear projections in $h_j^{(l-1)}$ with weight matrices W_Q, W_K, W_V. By the hypothesis, $h_j^{(l-1)}$ are the same for both methods, so the projections yield identical q_j, k_j, v_j.

The self-attention output O_j attends to token x_j and all preceding tokens, respecting the causal mask. In the global prefill mode, K and V are computed once for the full sequence; in chunked prefill, the previously computed KV caches store all preceding blocks' keys and values. The attention context for x_j in either mode is the same: $K_{1..j}$ and $V_{1..j}$. Thus, outputs O_j are identical. This equality relies on the linearity of projection and the additive property of concatenated KV vectors, ensuring perfect correspondence between chunked and full-sequence generated KV states.

Since attention outputs O_j and inputs $h_j^{(l-1)}$ are identical, subsequent operations within the layer, such as residual connections, layer normalization, and position-wise feedforward network, also produce identical outputs, leading to exactly matching hidden states $h_j^{(l)}$. By induction, chunked prefill and full prefill generate perfectly equivalent hidden states and KV caches across all layers, with no information loss.

B FLOPs Calculation

Full-KV Method In the full-KV attention method without chunking, the query, key, and value tensors are $Q, K, V \in \mathbb{R}^{b \times H \times N \times d}$, where b is batch size, H is number of attention heads, N is sequence length, and d is hidden dimension. The attention operation is

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V. \tag{10}$$

The floating-point operations (FLOPs) consist of

- $2bHN^2d$ for QK^T matrix multiplication,
- bHN^2 for scaling by $\frac{1}{\sqrt{d}}$,
- $3bHN^2$ for the softmax normalization,
- $2bHN^2d$ for multiplication with V.

Summing these yields the total FLOPs for L layers:

$$FLOPs_{\text{FullKV}} = 4bLHN^2(d+1). \tag{11}$$

TAKE Method TAKE partitions the input into M chunks each of size Z, with pruning budget B_{warmup} for half layers of model and B for remaining layers. Length of task-aware probe is p. The FLOPs include chunked attention and token scoring overhead.

Chunked attention FLOPs:

$$\begin{aligned} FLOPs_{\text{C}} = {} & 4b\frac{L}{2}H\left[MZ^2(d+1)\right. \\ & \left.+(M-1)(Z+B_{\text{warmup}})(Z+p)(d+1)\right] \\ & + 4b\frac{L}{2}H\left[MZ^2(d+1)\right. \\ & \left.+(M-1)(Z+B)(Z+p)(d+1)\right]. \end{aligned} \tag{12}$$

Token scoring FLOPs (performed on half the layers due to delayed pruning):

$$\begin{aligned} \text{FLOPs}_{\text{P}} = {} & \frac{L}{2}\Big[2bHpZ(d+1) \\ & + 2bHp(M-1)(Z+B)(d+1)\Big]. \end{aligned} \tag{13}$$

The total FLOPs for TAKE is:

$$\begin{aligned} FLOPs_{\text{TAKE}} &= FLOPs_{\text{C}} + FLOPs_{\text{P}} \\ &= bLH(d+1)\Bigg[4MZ^2 + 5MZp \\ &\quad + \frac{5p(M-1)(B+B_{\text{warmup}})}{2} \\ &\quad + 2Z(M-1)(B+B_{\text{warmup}})\Bigg] \end{aligned} \tag{14}$$

The FLOPs ratio is calculated by:

$$\begin{aligned} r &= \frac{FLOPs_{\text{TAKE}}}{FLOPs_{\text{FullKV}}} \\ &= \frac{1}{8N^2}\big(8MZ^2 + 10MZp \\ &\quad + 5p(M-1)(B+B_{\text{warmup}}) \\ &\quad + 4Z(M-1)(B+B_{\text{warmup}})\big). \end{aligned} \tag{15}$$

Given $N \approx MZ$ and $p \ll N$, simplifying yields:

$$
\begin{aligned}
r \approx & \frac{Z}{N} + \frac{5p}{4N} + \frac{5p(B + B_{\text{warmup}})}{8ZN} \\
& + \frac{B + B_{\text{warmup}}}{2N} - \frac{5p(B + B_{\text{warmup}})}{8N^2} \\
& - \frac{Z(B + B_{\text{warmup}})}{2N^2} \\
\approx & \frac{2Z + B + B_{\text{warmup}}}{2N} - \frac{Z(B + B_{\text{warmup}})}{2N^2}.
\end{aligned} \tag{16}
$$

Substituting experimental parameters $Z = 4096$,$B_{\text{warmup}} = 10240$,$B = 512$, $N = 128000$, we obtain

$$r \approx 7.10\%.$$

Thus, TAKE reduces computation to roughly 7.10% of the full-KV baseline, showing significant efficiency gains.

C Additional Experimental Results

C.1 Needle-in-a-Haystack

We evaluate long-context retrieval accuracy on LLaMA-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3 under KV budgets of $256L$, $512L$, and $1024L$. As shown in Figures 8–10, TAKE achieves the highest retrieval accuracy on LLaMA-3.1-8B-Instruct across all KV budgets. On Mistral-7B-Instruct-v0.3 (Figures 11–13), TAKE remains competitive in retrieval accuracy. Notably, TAKE matches or surpasses baseline methods in information retrieval while delivering substantial improvements in resource consumption and inference speed.

C.2 LongBench

We provide detailed LongBench results under different KV budgets for LLaMA-3.1-8B-Instruct (Table 4) and Mistral-7B-Instruct-v0.3 (Table 5). On LLaMA-3.1-8B-Instruct, TAKE achieves the highest average score across $128L$, $256L$, and $512L$ budgets, and remains competitive at $1024L$. On Mistral-7B-Instruct-v0.3, TAKE consistently attains the best average performance across all KV budgets.

D Detailed Implementation

D.1 Software Environment

All experiments are conducted on a single NVIDIA GPU H800 with CUDA 12.4. The core software stack includes Python 3.10.16, PyTorch 2.5.1 , Transformers 4.50.3, Accelerate 1.6.0, and FlashAttention 2.7.3. Other key dependencies are specified in our Source Code.

D.2 Modifications to Existing Libraries

We extend the official Transformers library by implementing custom Llama and Mistral model files. These files override the standard attention forward pass to integrate our compression mechanism: chunked processing, probe-based KV selection, and token merging. We preserve the original model configuration and generation APIs.

E Limitations

Since TAKE relies on the *accumulated task-aware probe* to identify critical tokens based on semantic relevance during chunked inference, it struggles to effectively retain information that lacks semantic association with the task instruction. Consequently, TAKE is less effective in scenarios where information retention relies purely on rote memorization of non-semantic patterns. We extended our evaluation to the RULER (Hsieh et al., 2024) benchmark and observed a performance degradation when the context length exceeds 8K tokens. This limitation stems from the inherent design of RULER's retrieval tasks, which often require the recall of random strings devoid of inherent semantic meaning. Developing more universal feature representations that can guide chunked pruning in the absence of strong semantic correlations remains a promising avenue for future research.

F LLM Statement

We used LLM for language polishing and grammar correction during manuscript preparation. All research contributions, including problem formulation, methodology design, implementation, experiments, data analysis, and scientific conclusions, were conceived, executed, and verified solely by the authors. This use conforms to the ACL Ethics Policy regarding authorship criteria.

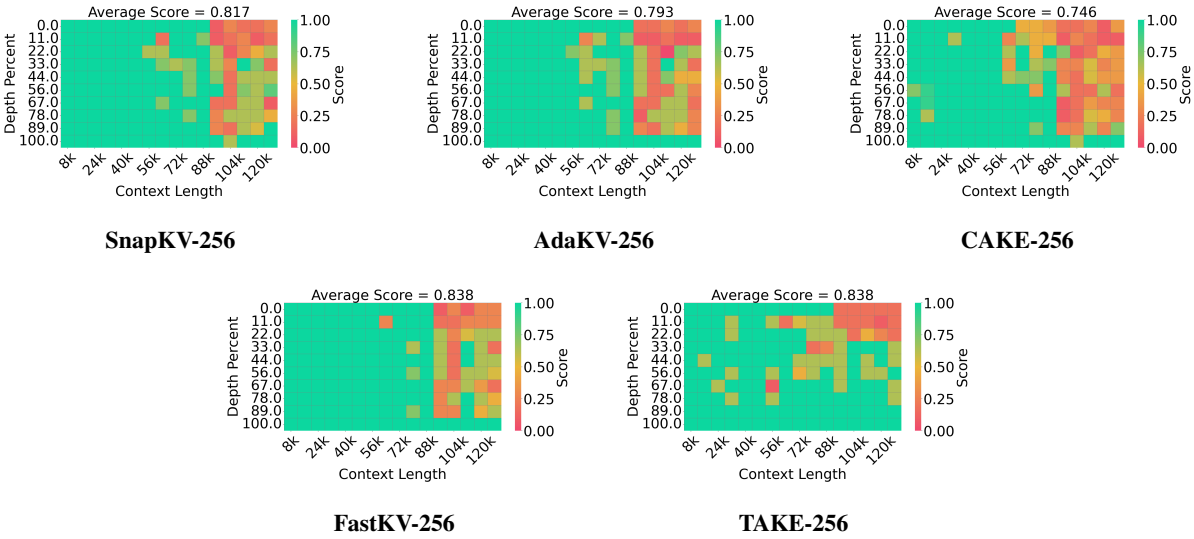


Figure 8: Needle-in-a-Haystack results of LLaMA-3.1-8B-Instruct with 256L KV budget.

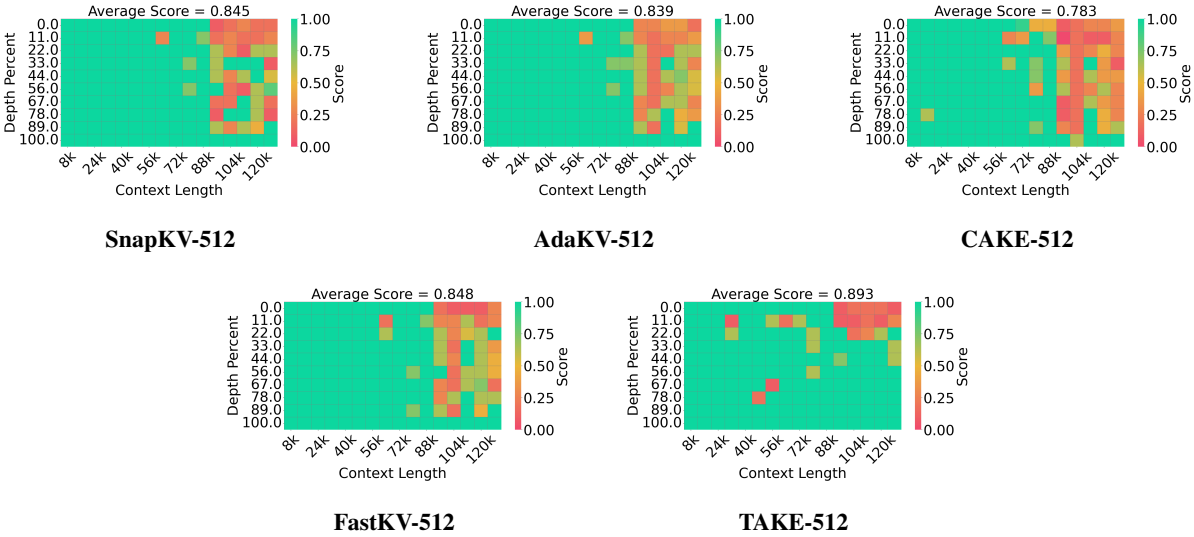


Figure 9: Needle-in-a-Haystack results of LLaMA-3.1-8B-Instruct with 512L KV budget.

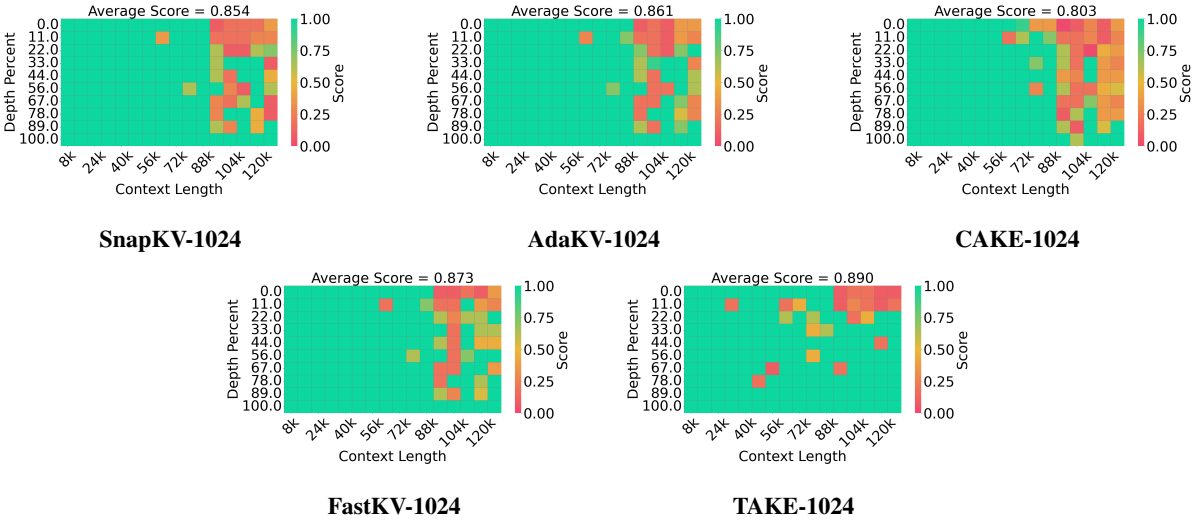


Figure 10: Needle-in-a-Haystack results of LLaMA-3.1-8B-Instruct with 1024L KV budget.

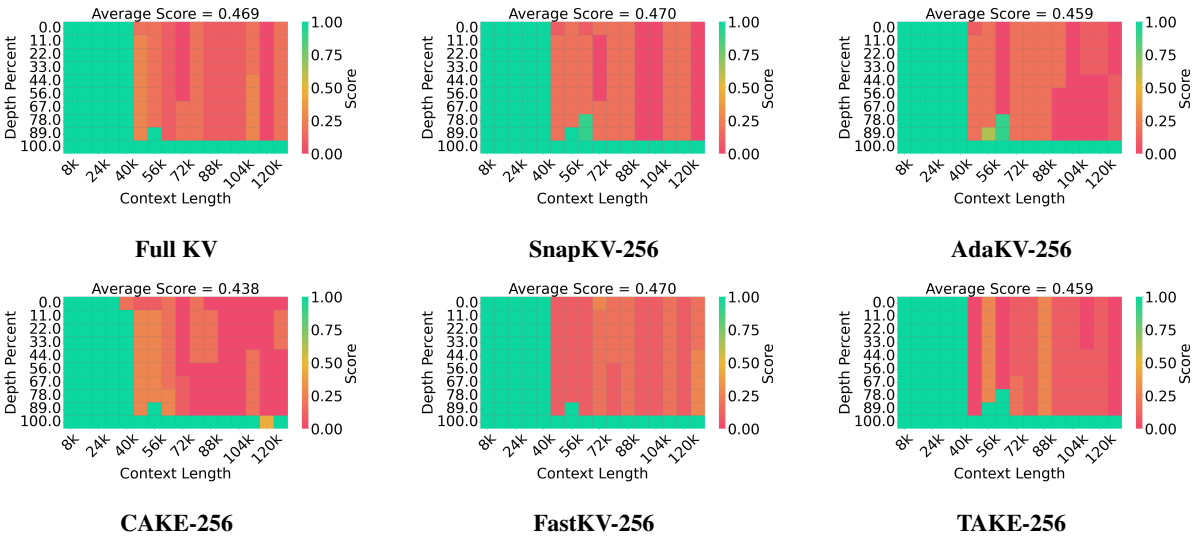


Figure 11: Needle-in-a-Haystack results of Mistral-7B-Instruct-v0.3 with 256L KV budget.

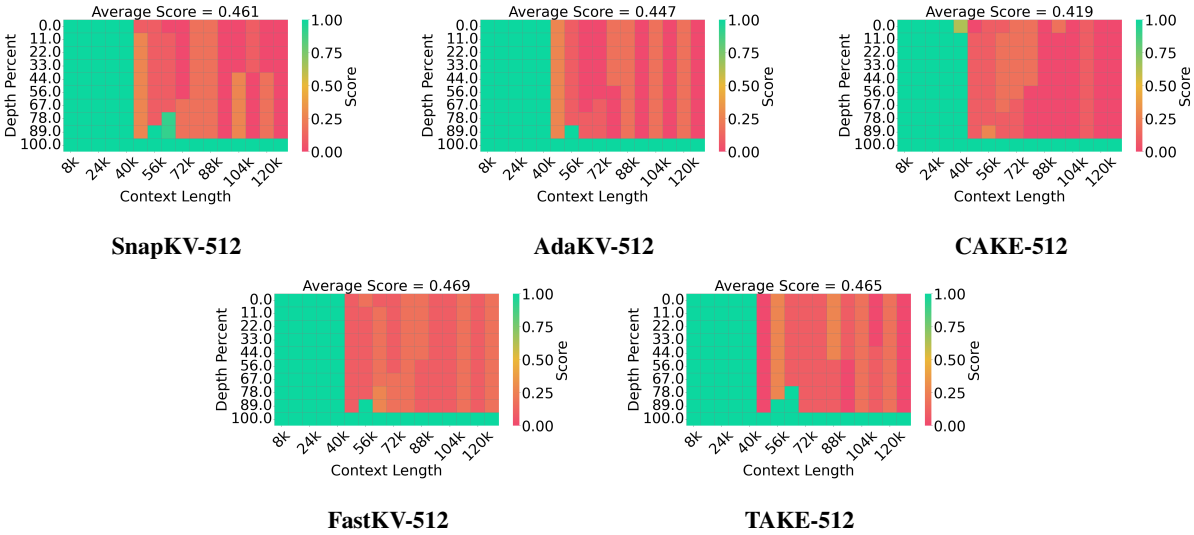


Figure 12: Needle-in-a-Haystack results of Mistral-7B-Instruct-v0.3 with 512L KV budget.

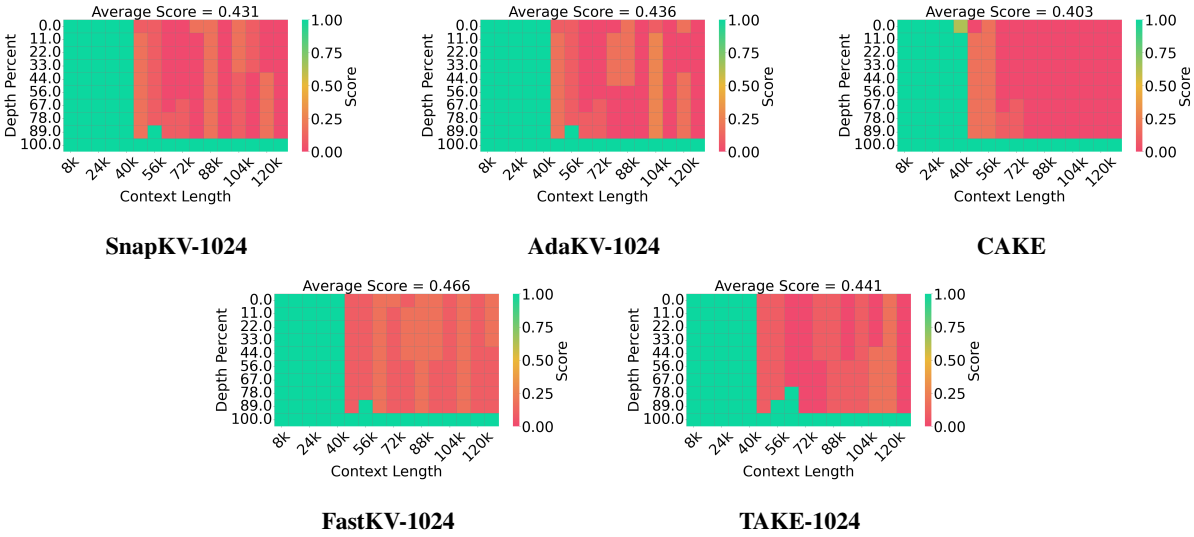


Figure 13: Needle-in-a-Haystack results of Mistral-7B-Instruct-v0.3 with 1024L KV budget.

Table 4: LongBench results comparison on LLaMA-3.1-8B-Instruct. The best results are **bolded**.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Coding		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep.	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RB-P	
LLaMA-3.1-8B-Instruct, *KV Budget = Full*															
Full KV	30.25	45.53	55.01	56.01	46.65	31.28	35.13	25.28	27.25	73.00	91.64	43.80	63.38	56.64	48.63
LLaMA-3.1-8B-Instruct, *KV Budget = 128*															
SnapKV	27.23	30.96	50.62	53.04	43.13	28.66	22.70	23.16	21.90	62.00	91.19	40.37	58.37	49.09	43.03
AdaKV	27.71	30.13	50.27	51.65	40.87	29.21	23.42	23.08	21.72	67.50	91.10	40.54	58.87	50.15	43.30
FastKV	27.49	28.05	51.56	52.12	44.54	28.78	22.12	23.04	20.77	59.50	91.06	40.79	59.03	49.16	42.71
CAKE	27.41	32.01	49.58	52.99	45.76	28.32	21.85	23.15	21.56	47.50	89.61	40.90	57.92	49.83	42.03
TAKE	27.39	35.36	51.46	52.94	46.69	29.47	22.00	22.25	26.19	59.00	91.29	39.06	61.33	52.76	**44.09**
LLaMA-3.1-8B-Instruct, *KV Budget = 256L*															
SnapKV	29.51	36.43	52.75	54.06	43.38	28.80	25.21	23.98	23.40	69.00	91.96	41.26	60.16	50.31	45.02
AdaKV	29.17	34.41	51.67	54.53	42.79	30.54	25.09	23.56	23.19	70.50	92.12	41.15	59.80	51.16	44.98
FastKV	29.70	35.74	53.85	54.45	45.54	29.95	24.59	23.63	23.10	67.00	91.55	41.59	60.98	50.52	45.16
CAKE	29.75	38.74	52.27	54.52	45.39	29.57	24.15	24.24	23.77	57.00	90.97	41.82	61.01	51.90	44.65
TAKE	29.36	39.85	53.84	55.01	44.65	25.25	24.03	22.92	26.31	63.50	91.81	41.42	62.03	54.44	**45.60**
LLaMA-3.1-8B-Instruct, *KV Budget = 512L*															
SnapKV	30.04	41.14	53.83	54.77	45.12	31.20	27.08	24.04	25.00	70.50	91.90	42.34	61.34	52.24	46.47
AdaKV	30.09	40.20	54.14	54.27	44.73	30.77	26.84	23.98	24.17	70.50	91.90	41.74	60.63	53.02	46.21
FastKV	29.96	40.88	54.55	54.78	46.19	30.76	26.63	24.05	24.29	72.50	91.37	42.52	62.14	52.65	46.66
CAKE	31.82	42.99	51.65	54.37	46.89	30.73	26.36	24.94	25.27	63.50	91.54	42.52	62.31	54.30	46.37
TAKE	30.45	43.22	55.20	56.99	46.77	30.18	26.44	23.77	26.60	70.50	92.48	41.84	63.25	56.67	**47.45**
LLaMA-3.1-8B-Instruct, *KV Budget = 1024L*															
SnapKV	31.23	42.52	53.96	55.48	45.43	31.50	29.54	24.78	26.17	70.50	91.73	42.52	62.25	54.72	47.31
AdaKV	31.00	44.35	54.58	55.86	45.39	31.96	28.77	24.92	26.04	71.00	91.48	42.32	61.88	54.75	47.45
FastKV	30.21	43.86	55.16	55.14	46.06	30.65	29.42	24.18	26.14	73.50	91.23	43.42	62.63	55.69	47.66
CAKE	30.88	44.95	52.38	55.49	46.99	30.82	28.68	24.91	26.39	69.00	91.94	42.60	62.65	56.89	47.47
TAKE	30.83	44.53	54.37	56.42	47.30	27.31	28.78	24.42	26.79	72.00	92.41	40.92	63.32	55.90	**47.52**

Table 5: LongBench results comparison on Mistral-7B-Instruct-v0.3. The best results are **bolded**.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Coding		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep.	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RB-P	
Mistral-7B-Instruct-v0.3, *KV Budget = Full*															
Full KV	29.20	38.27	50.09	51.20	36.49	26.67	34.21	25.79	26.43	76.00	88.59	47.46	59.37	60.52	46.45
Mistral-7B-Instruct-v0.3, *KV Budget = 128L*															
SnapKV	23.77	28.60	45.64	44.60	29.85	21.92	22.66	22.56	21.32	47.50	87.94	43.32	54.67	51.75	39.00
AdaKV	24.54	29.42	45.65	42.40	28.66	22.18	22.89	23.36	21.53	65.50	88.89	44.55	53.89	53.23	40.18
FastKV	24.97	26.68	45.46	43.75	30.01	21.06	22.43	22.33	20.37	63.50	89.24	42.50	54.37	51.80	39.89
CAKE	22.31	29.15	43.51	44.51	30.36	22.85	21.56	20.47	18.96	47.00	88.60	39.36	44.96	46.19	37.13
TAKE	23.25	28.56	43.64	48.80	34.32	20.92	22.06	21.86	25.63	56.50	89.16	43.19	58.40	51.97	**40.43**
Mistral-7B-Instruct-v0.3, *KV Budget = 256L*															
SnapKV	24.72	31.23	47.01	46.72	31.09	22.74	25.08	23.60	22.82	62.00	88.99	45.11	56.82	56.39	41.74
AdaKV	25.88	31.39	46.56	46.47	31.67	22.21	24.18	23.93	22.88	72.00	89.16	45.17	56.71	56.97	42.51
FastKV	24.75	29.61	49.27	48.33	31.91	24.13	24.45	22.80	22.60	70.00	89.17	43.65	56.69	55.05	42.31
CAKE	24.01	30.78	45.88	45.06	31.45	23.43	22.81	20.87	20.92	58.50	88.29	40.16	47.76	48.43	39.17
TAKE	24.82	32.47	48.26	50.05	34.64	22.33	24.18	22.81	25.85	68.50	89.34	44.90	59.37	55.64	**43.02**
Mistral-7B-Instruct-v0.3, *KV Budget = 512L*															
SnapKV	26.03	34.82	49.74	49.25	34.11	23.88	27.23	24.33	24.59	74.00	89.82	45.94	58.13	58.49	44.31
AdaKV	27.14	35.10	48.65	48.98	34.30	23.80	25.76	24.38	24.27	75.00	89.14	45.87	58.18	59.49	44.29
FastKV	25.94	33.20	49.03	48.37	34.42	25.43	26.63	23.88	24.10	71.50	89.27	46.49	58.72	58.63	43.97
CAKE	25.82	34.24	46.93	45.34	31.71	23.38	25.81	21.47	22.41	68.00	88.46	41.27	49.04	49.67	40.96
TAKE	26.65	35.61	49.49	50.11	37.42	22.99	26.18	24.09	25.98	73.50	89.54	45.12	59.86	58.18	**44.62**
Mistral-7B-Instruct-v0.3, *KV Budget = 1024L*															
SnapKV	28.31	36.57	48.21	50.57	35.02	24.71	29.22	24.86	26.29	75.00	88.99	46.08	58.76	60.24	45.20
AdaKV	26.91	36.41	48.78	50.02	35.76	24.85	27.43	24.66	25.55	75.50	89.61	46.92	58.93	60.07	45.10
FastKV	26.43	36.69	49.44	49.27	35.04	25.69	28.63	24.77	25.83	73.50	88.64	47.34	59.13	59.63	45.00
CAKE	26.09	36.34	48.11	45.97	32.39	23.49	27.56	21.45	24.03	72.50	88.61	42.71	51.06	51.25	42.25
TAKE	26.55	37.10	50.26	51.46	37.93	23.94	28.56	24.64	26.29	75.00	89.71	45.39	59.77	58.4	**45.36**