LOOKAHEADKV: FAST AND ACCURATE KV CACHE EVICTION BY GLIMPSING INTO THE FUTURE WITHOUT GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer-based large language models (LLMs) rely on key–value (KV) caching to avoid redundant computation during autoregressive inference. While this mechanism greatly improves efficiency, the cache size grows linearly with the input sequence length, quickly becoming a bottleneck for long-context tasks. Existing solutions mitigate this problem by evicting prompt KV that are deemed unimportant, guided by estimated importance scores. Notably, a recent line of work improves eviction quality by "glimpsing into the future", in which a low-cost draft generator first produces a surrogate response that mimics the target model's true response, which is subsequently used to estimate the importance scores of cached KV. In this paper, we propose LookaheadKV, a lightweight eviction framework that leverages the strength of surrogate future response without the need for costly draft generation. LookaheadKV augments transformer layers with parameter-efficient modules trained to predict true importance scores with high accuracy. Our design ensures negligible runtime overhead comparable to existing inexpensive heuristics, while achieving accuracy superior to more costly approximation methods. Extensive experiments on long-context understanding benchmarks, across a wide range of models, demonstrate that our method not only outperforms recent competitive baselines in long-context understanding tasks by 25%, but also reduces the eviction cost by up to $14.5\times$, leading to significantly faster time-to-first-token.

1 INTRODUCTION

Long context length of Large Language Models (LLMs) is becoming increasingly critical for many emerging applications: processing long documents (Bai et al., 2024; Wang et al., 2024a; Hsieh et al., 2024), repository-level code understanding and generation (Luo et al., 2024; Liu et al., 2024; Jimenez et al., 2024), in-context learning (Li et al., 2025), extension to long multi-modal inputs such as video (Wang et al., 2024b), etc. However, a central challenge in enabling these applications is that the key-value (KV) cache size grows linearly in sequence length, which rapidly becomes a bottleneck for inference, restricting scalable deployment of such applications on both mobile devices and the cloud. For example, even for moderate-sized models, such as LLaMA3.1–70B (Dubey et al., 2024) in half-precision, storing a single 64K-token sequence already takes up 40GB of memory, while scaling to 512K tokens requires 160GB, exceeding the memory capacity of high-end consumer hardware.

A growing line of work addresses this challenge by identifying salient tokens to achieve effective KV cache eviction without loss of performance (Li et al., 2024; Cai et al., 2024; Galim et al., 2025; Wang et al., 2025; Zhang et al., 2023). Early methods often rely on simple heuristics, in which token importance is estimated based on the self-attention scores of the input tokens. SnapKV (Li et al., 2024), for instance, leverages the attention weights between the suffix of the input and the preceding context to estimate the importance of each prompt token. However, investigations in recent studies (SpecKV (Galim et al., 2025), LAQ (Wang et al., 2025)) reveal that leveraging the model's response, rather than the input suffix, can greatly improve the eviction quality. Furthermore, they show that

Source code to reproduce our results is available, released in supplementary.

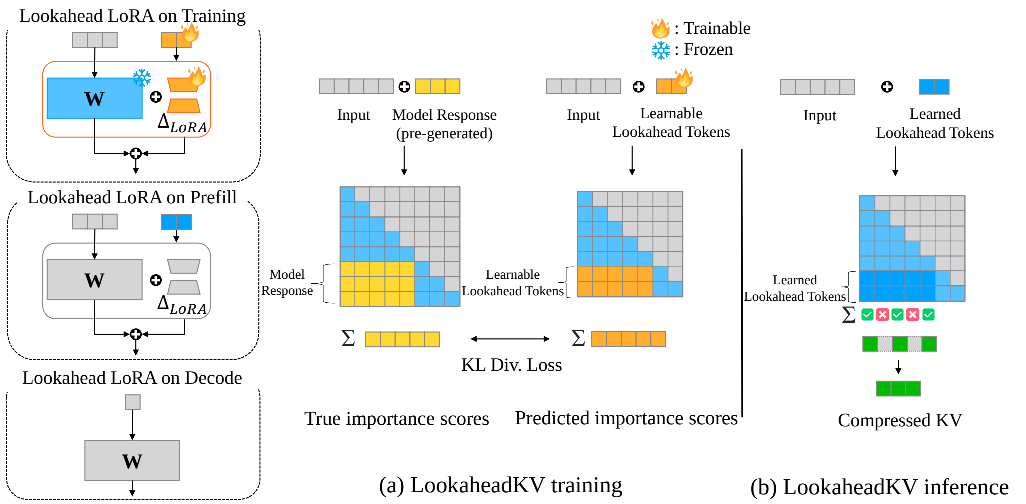


Figure 1: (a) Overview of LookaheadKV training (b) Overview of LookaheadKV inference.

a low-cost generated draft response, which closely approximates the true response, can serve as a powerful proxy for accurately estimating the importance scores. For example, SpecKV employs a smaller auxiliary model to produce draft tokens to approximate the target model's response, while Lookahead Q-Cache (LAQ) first applies a cheap KV eviction scheme to the target model, such as SnapKV, to obtain draft tokens, which in turn are used to approximate true importance scores.

While these draft-based methods substantially improve eviction quality, they often struggle with a fundamental trade-off between performance and efficiency, due to the need for costly draft token generation. Figure 2 presents the trade-off between accuracy and overhead of different approaches using the QASPER benchmark (Dasigi et al., 2021) and LLaMA3.1-8B-Instruct (Dubey et al., 2024) with a cache budget size of 128. While cheaper approaches like SnapKV are fast, inducing minimal overhead, they suffer a severe performance degradation under highly constrained budget settings. On the other hand, LAQ (Wang et al., 2025), a draft-based approach, shows impressive results even in extremely limited budget settings. However, it incurs a prohibitive computational overhead by generating an extra draft response, which limits its practicality in latency-sensitive applications such as mobile devices.

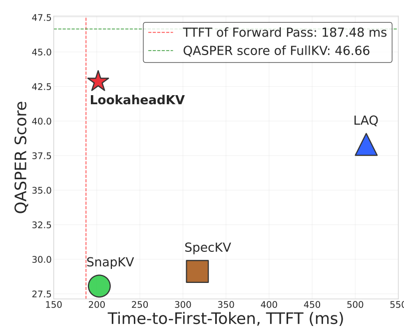


Figure 2: Accuracy-overhead Trade-off across KV cache eviction methods.

To overcome this limitation, we introduce LookaheadKV, a novel KV cache eviction method that augments LLMs with parameter-efficient modules, capable of accurately predicting future attention patterns, without the need for costly draft token generation. As shown in Figure 2, our method effectively overcomes the accuracy-overhead trade-off, achieving minimal performance loss with negligible overhead. LookaheaedKV, as depicted in Figure 1, our method employs a set of learnable special tokens, together with *Lookahead LoRA* modules, novel low-rank adapters that selectively activate for the special tokens, to produce queries that can reliably estimate token-importance scores. By fine-tuning these modules to predict the true importance scores, LookaheadKV effectively minimizes the quality loss incurred by KV cache eviction with marginal inference overhead.

To rigorously assess the effectiveness of LookaheadKV, we evaluate it on a diverse set of long-context benchmarks (Bai et al., 2024; Hsieh et al., 2024; Ye et al., 2025) across multiple models of varying sizes (Dubey et al., 2024; Yang et al., 2025). Experimental results consistently demonstrate that LookaheadKV outperforms strong baselines across multiple budgets and context lengths while incurring significantly less eviction latency.

To summarize, our contributions are as follows:

- We propose LookaheadKV, to the best of our knowledge, the first KV cache eviction framework that employs learnable lookahead tokens and special LoRA modules to accurately predict the importance scores from the model's true response without generating costly approximate response.
- Through extensive experiments, we demonstrate that the proposed approach is effective and robust across different models and context lengths, and especially under low-budget settings, making our method particularly useful in resource-constrained environments.
- By conducting a rigorous analysis of eviction latency, both theoretically and empirically, we demonstrate that our method incurs negligible eviction overhead of less than 2.16% at 32K context length, while being $14.5\times$ faster than draft-based methods.

2 Background

The primary objective of the KV cache eviction methods considered in this work, including our proposed approach, is to accurately estimate the importance score of individual key-value pairs of prompt tokens using attention weights, in order to guide the eviction process. In the following section, we formally define the problem of KV cache eviction and briefly discuss how prior methods have approached it.

KV cache eviction using importance scores. Let $X = \{x_1, ..., x_{n_{\text{in}}}\}$ be an input token sequence (e.g., a user instruction, part of a code snippet, etc.) and $Y = \{y_1, ..., y_{n_{\text{out}}}\}$ the model's generated response to X. For a given layer and attention head in an LLM, the attention scores of the complete sequence are given by:

$$\mathbf{Q} = \begin{bmatrix}\mathbf{X}\\ \mathbf{Y}\end{bmatrix}\mathbf{W}_q \qquad \mathbf{K} = \begin{bmatrix}\mathbf{X}\\ \mathbf{Y}\end{bmatrix}\mathbf{W}_k \qquad \mathbf{A} \;=\; \text{Softmax}\left(\frac{\mathbf{Q}\,\mathbf{K}^\top}{\sqrt{d}}\right), \tag{1}$$

where $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_{n_{\text{in}}}]^\top \in \mathbb{R}^{n_{\text{in}} \times d}$ and $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_{n_{\text{out}}}]^\top \in \mathbb{R}^{n_{\text{out}} \times d}$ are the hidden states of the input prompt and model-generated response, respectively. For better readability, we omit the layer and head index. We define the ground-truth importance scores $\mathbf{s}_{\text{GT}} = [s_1, ..., s_{n_{\text{in}}}]$ of the KV cache as the average cross-attention scores between the queries of $\mathbf{Y}$ and the keys of $\mathbf{X}$, i.e., $s_j = \frac{1}{n_{\text{out}}} \sum_{i=n_{\text{in}}+1}^{n_{\text{in}}+n_{\text{out}}} \mathbf{A}_{i,j}$. Intuitively, these scores quantify the relative contribution of each prompt token's key–value pair to the model's response generation. Based on these scores, the pruned KV cache can be obtained by retaining a subset of (e.g., TopK) important KV pairs to minimize the attention output perturbation, such that:

$$\text{Attn}(x, \text{KV}_{\text{orig}}) \approx \text{Attn}(x, \text{KV}_{\text{GT}}), \tag{2}$$

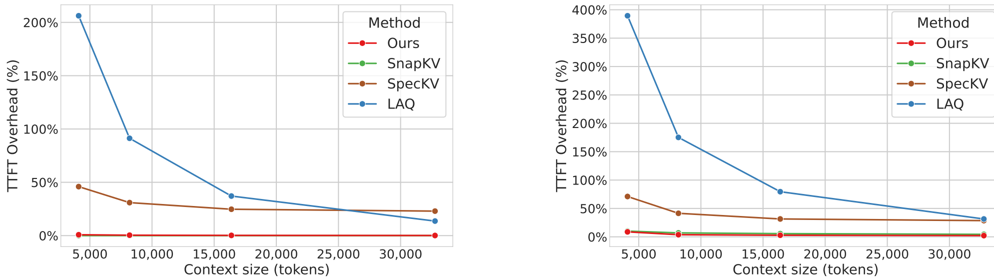
where KV_{orig} and KV_{GT} are the original and evicted KV cache using the ground-truth importance scores, respectively.

However, since the model's true future response is unknown during the prefill phase, such scores cannot be computed directly. Consequently, prior methods resorted to constructing a surrogate response sequence $\tilde{\mathbf{Y}} = [\tilde{y}_1, ..., \tilde{y}_{n_{\text{window}}}]^\top \in \mathbb{R}^{n_{\text{window}} \times d}$ to approximate the model's (partial) future response and predict the attention pattern:

$$\tilde{\mathbf{Q}} = \begin{bmatrix}\mathbf{X}\\ \tilde{\mathbf{Y}}\end{bmatrix}\mathbf{W}_q \qquad \tilde{\mathbf{K}} = \begin{bmatrix}\mathbf{X}\\ \tilde{\mathbf{Y}}\end{bmatrix}\mathbf{W}_k \qquad \tilde{\mathbf{A}} \;=\; \text{Softmax}\left(\frac{\tilde{\mathbf{Q}}\,\tilde{\mathbf{K}}^\top}{\sqrt{d}}\right), \tag{3}$$

resulting in the estimated importance score vector $\mathbf{s}_{\text{approx}} = [\tilde{s}_1, ..., \tilde{s}_{n_{\text{in}}}]$, whose entries are computed as $\tilde{s}_j = \frac{1}{n_{\text{window}}} \sum_{i=n_{\text{in}}+1}^{n_{\text{in}}+n_{\text{window}}} \tilde{\mathbf{A}}_{i,j}$. In short, these methods aim to obtain the estimated score vector whose ranking is similar to that of the ground-truth, such that the overlap between the retained KV pairs and KV_{GT} is high. Various approaches have been suggested to approximate the future response for effective KV cache eviction.

SnapKV. SnapKV (Li et al., 2024) proposes to use the suffix of input prompt to compute the estimate of the true future importance scores. Because SnapKV requires only marginal extra computation to



(a) Theoretical latency overhead

(b) Actual latency overhead

Figure 3: Time-to-First-Token (TTFT) latency overhead ratio across context lengths. Similar to SnapKV, LookaheadKV introduces negligible TTFT overhead across all tested context lengths; draft-based methods (LAQ, SpecKV) incur substantial latency, especially for shorter contexts.

perform eviction, as it uses attention weights that are already computed during the prefill forward pass, it has widely been adopted as a cheap and effective heuristic for KV cache eviction.

SpecKV and LAQ. Recently, several methods have proposed to use a low-cost generator to generate a (partial) approximate response first, and subsequently use it to estimate the true future importance scores. Notably, SpecKV (Galim et al., 2025) employs a smaller LLM to first generate a draft response, while Lookahead Q-Cache (LAQ) (Wang et al., 2025) first applies SnapKV to the target model to generate a draft response, which is in turn used to approximate the future importance.

These draft-based methods have consistently shown superior performance compared to cheaper heuristics (Li et al., 2024), demonstrating the effectiveness of employing surrogate future response, i.e., by "glimpsing into the future". However, the extra draft generation step still incurs substantial additional compute, resulting in significant increase in latency, as shown in Figure 3. In summary, existing methods face a clear trade-off: inexpensive heuristics are fast but less accurate, whereas draft-based techniques improve performance at the cost of increased inference time.

3 Proposed Method: LookaheadKV

To overcome the challenge of fast and accurate importance prediction, we introduce LookaheadKV, a framework that augments the LLM with a set of lightweight learnable modules which are optimized to predict ground-truth importance scores and guide the eviction process. LookaheadKV achieves the best of both worlds: **1)** it eliminates the need for generating a draft response for each query, resulting in significantly faster KV cache eviction, and **2)** it employs learned special tokens that serve as approximate future response for importance estimation, leveraging the strength of draft-based methods. The following section (and Figure 1) presents the detailed workflow of LookaheadKV.

3.1 Main Components

Learnable Lookahead Tokens. LookaheadKV initially performs KV cache eviction using a set of learnable special tokens during the pre-fill phase, and subsequently decodes auto-regressively with the retained KV cache. Specifically, for given input prompt tokens X, LookaheadKV appends a sequence of trainable "soft" lookahead tokens $L = \{l_1, ..., l_{n_{\text{lookahead}}}\}$ whose queries in the attention layers are used to estimate the attention pattern of the true model response. In essence, these tokens are trained to compress the attention information of the true response to serve as the "observation window" in the eviction phase. These tokens are new tokens added to the vocabulary, each with an embedding of shape $\mathbb{R} \in d$ that is randomly initialized. Note that the lookahead tokens are used during the prefill stage only for eviction, and adds zero additional overhead for decoding.

Lookahead LoRA. To enhance the quality of estimation, we introduce *Lookahead LoRA*, a novel low-rank adapter module that only activates for the lookahead tokens. While predicting attention patterns across all layers and heads using only the lookahead tokens is challenging due to their limited capacity, Lookahead LoRA provides complementary performance gains by allowing these tokens to learn richer representations, enabling their queries to more accurately predict token importance.

The selective activation mechanism of the LoRA modules ensures that the outputs of normal input tokens are unchanged, preserving the original model behavior. Moreover, since the original model weights remain unaltered, LookaheadKV modules can be selectively enabled or disabled depending on the particular requirements of a given application, thereby broadening the method's applicability.

Combining the modules together, LookaheadKV computes the queries and keys of the complete sequence as follows:

$$\mathbf{Q}_{\text{LKV}} = \begin{bmatrix} \mathbf{X} \\ \mathbf{L} \end{bmatrix} \mathbf{W}_q + \begin{bmatrix} \mathbf{0} \\ \mathbf{L} \end{bmatrix} \Delta\mathbf{W}_q \qquad \mathbf{K}_{\text{LKV}} = \begin{bmatrix} \mathbf{X} \\ \mathbf{L} \end{bmatrix} \mathbf{W}_k + \begin{bmatrix} \mathbf{0} \\ \mathbf{L} \end{bmatrix} \Delta\mathbf{W}_k, \tag{4}$$

where $\mathbf{L} \in \mathbb{R}^{n_{\text{lookahead}} \times d}$ denotes the hidden states of the lookahead embeddings, and $\Delta\mathbf{W}_q$, $\Delta\mathbf{W}_k$ are the Lookahead LoRA modules for query and key projections. Similar to prior methods (Li et al., 2024), we use the attention matrix $\mathbf{A}_{\text{LKV}} = \text{Softmax}(\frac{\mathbf{Q}_{\text{LKV}} \mathbf{K}_{\text{LKV}}^\top}{\sqrt{d}})$, to estimate the importance score $\tilde{s}_j = \frac{1}{n_{\text{lookahead}}} \sum_{i=n_{\text{in}}+1}^{n_{\text{in}}+n_{\text{lookahead}}} \mathbf{A}_{\text{LKV}\ i,j}$, and retain Top-K KV pairs with the highest importance scores.

3.2 LookaheadKV Training

We train LookaheadKV modules to compress the attention pattern of the true future response, using the model-generated responses as target. Specifically, given a data pair (X, Y), one iteration of LookaheadKV training consists of the following steps:

1. **GT Forward Pass.** For each layer $l = 1, ..., L$ and head $h = 1, ..., H$, the ground-truth importance scores $\mathbf{s}_{\text{GT}}^{l,h}$ between the input prompt X and model-generated response Y are computed.
2. **Lookahead Forward Pass.** Similarly, for each layer l and head h, we obtain the importance score estimates $\mathbf{s}_{\text{LKV}}^{l,h}$ between the input prompt X and the lookahead tokens L.
3. **Loss Computation.** We first normalize all score vectors so that they sum to 1, and compute the average KL divergence loss between the GT and LookaheadKV importance scores across all heads and layers:

$$\mathcal{L} = \frac{1}{L}\frac{1}{H}\sum_{l}^{L}\sum_{h}^{H} \text{KL}\big(\text{Norm}(\mathbf{s}_{\text{GT}}^{l,h}) \parallel \text{Norm}(\mathbf{s}_{\text{LKV}}^{l,h}) \big). \tag{5}$$

The loss is backpropagated to update the weights of the lookahead embeddings and Lookahead LoRA modules, while all other LLM layers remain frozen. The pseudo-code for LookaheadKV training and eviction is given in Algorithm 1 and Algorithm 2.

Training Objective. We want to ultimately optimize the similarity of the ranking between the two importance score vectors, such that we obtain TopK indices identical to those from ground-truth importance scores. Inspired from works on distilling attention scores (Wang et al., 2020; Izacard & Grave, 2021), we minimize the KL divergence between these normalized attention scores. As our attentions scores are normalized, this KL divergence is equivalent to the popular ListNet (Cao et al., 2007) ranking loss, with ϕ of ListNet as identity instead of $\exp$. We note that, while we employ normalized average attention scores as the importance metric throughout the experiments, any suitably defined token-wise importance metric can be used for training, provided that it can be computed for each token.

Lookahead LoRA Overhead. In principle, one can apply Lookahead LoRA to only a subset of the linear layers to tradeoff accuracy and latency. However, even when Lookahead LoRA is applied to every linear layer, there is a minor increase ($>1.3\%$) in latency compared to not using Lookahead LoRA at all (see Table 5 for ablation results), while significantly boosting performance compared to not using LoRA. Consequently, we train LookaheadKV with LoRA modules applied to all linear layers.

To avoid materializing the full attention score matrix, we use FlashAttention (Dao et al., 2022) in the forward pass, coupled with eager attention for importance score computation and loss backpropagation, as detailed in Section C.

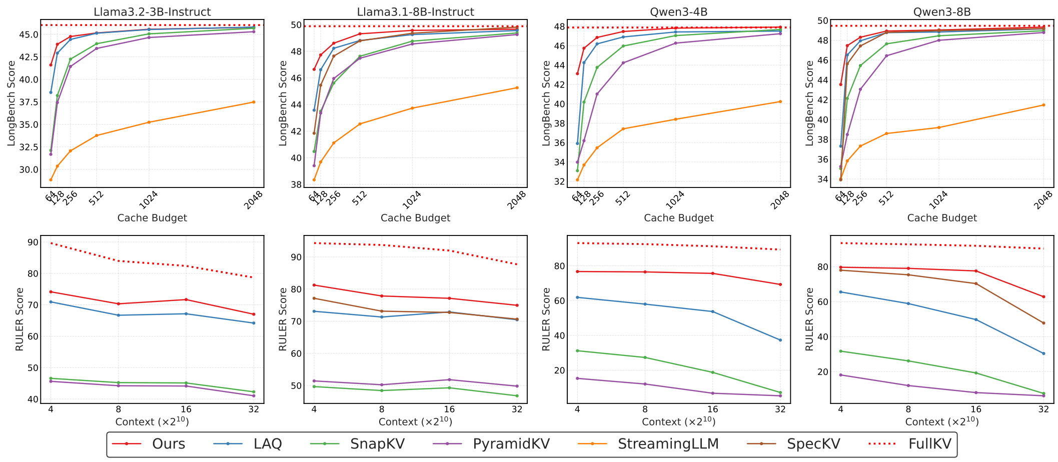


Figure 4: Top row: Average LongBench results across multiple budgets and models. Bottom row: Average RULER results across varying context lengths with a fixed budget of 128. Across all tested models, budgets and context lengths, LookaheadKV consistently demonstrates superior performance.

4 EXPERIMENTS

4.1 TRAINING

Dataset. To encourage the model to learn from diverse attention patterns, we curate training samples of varying lengths and sources, comprising of both instruction-following datasets as well as pretraining texts. We collect 50K samples from the long_sft subset of the ChatQA2 (Xu et al., 2025) dataset, 20K samples from the Tulu (Lambert et al., 2025) instruction-following dataset, 7K samples from the Stack (Kocetkov et al., 2023), and 9K few-shot completion data samples that we create based on the training splits of the MetaMath, ARC, and HellaSwag datasets, originally curated in Pal et al. (2024). For instruction-following data, we remove the last assistant response, and use the target model to obtain the (X, Y) pairs of input prompt and model response. For pretraining documents, we first truncate the text at random positions to obtain X, and use the target model to complete the sequence to obtain Y. We limit the maximum input sequence length to 16K, and generate all training responses using greedy decoding and max generation length of 512.

Training Details. We apply LookaheadKV on two widely used open-source architectures, LLaMA (Dubey et al., 2024) and Qwen (Yang et al., 2025), covering three model sizes each: LLaMA3.2-1B, LLaMA3.2-3B, LLaMA3.1-8B, Qwen3-1.7B, Qwen3-4B, and Qwen3-8B. For all models, we set the lookahead size $n_{\text{lookahead}} = 32$, and apply LoRA to all projection and feed-forward modules ($\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, $\mathbf{W}_o$, $\mathbf{W}_{up}$, $\mathbf{W}_{down}$, and $\mathbf{W}_{gate}$) with rank $r = 8$ and scaling factor $\alpha = 32$. This configuration introduces less than 0.5% additional trainable parameters across all models, as summarized in Table 1. Full hyperparameter settings are provided in Table 16.

Table 1: Additional trainable parameters introduced by LookaheadKV.

Model	Trainable Params	
	Params	% of Model
LLaMA3.2-1B	5.4M	0.44
LLaMA3.2-3B	11.9M	0.37
LLaMA3.1-8B	20.6M	0.26
Qwen3-1.7B	8.5M	0.49
Qwen3-4B	16.2M	0.40
Qwen3-8B	21.5M	0.26

4.2 EVALUATION SETUP

We evaluate our method on two popular long-context benchmarks: LongBench (Bai et al., 2024) and RULER (Hsieh et al., 2024). LongBench is a multi-task benchmark that comprehensively assesses long-context understanding across diverse tasks, such as question answering, summarization, few-shot learning, and code completion. We report results on the 16 English tasks, and use the average score as the main metric. RULER is another multi-task synthetic benchmark, primarily comprising 13 Needle-in-a-Haystack-style subtasks. Each sample can be constructed at varying sequence

Table 2: MT-Bench evaluation results. Bold and underlined values indicate best and second best scores, respectively.

Model	Budget	PyramidKV	SnapKV	StreamingLLM	SpecKV	LAQ	LookaheadKV
				FullKV score: 5.72			
LLaMA3.2-1B	64	4.64	4.70	4.54	N/A	$\underline{5.03}$	**5.21**
	128	5.10	5.39	4.94	N/A	$\underline{5.45}$	**5.60**
	256	5.49	**5.67**	5.37	N/A	$\underline{5.64}$	5.62
	512	$\underline{5.73}$	5.71	5.68	N/A	$\underline{5.73}$	**5.79**
				FullKV score: 7.35			
LLaMA3.2-3B	64	6.30	6.28	5.96	$\underline{6.52}$	6.48	**6.87**
	128	6.93	$\underline{7.03}$	6.42	7.02	6.93	**7.26**
	256	7.19	$\underline{7.30}$	7.20	7.28	**7.43**	$\underline{7.30}$
	512	**7.46**	7.24	7.24	$\underline{7.34}$	7.27	7.15
				FullKV score: 7.77			
LLaMA3.1-8B	64	6.85	6.80	6.17	6.77	$\underline{7.1}$	**7.26**
	128	7.39	7.50	6.84	7.34	$\underline{7.54}$	**7.63**
	256	$\underline{7.76}$	7.72	7.41	7.84	7.72	**7.92**
	512	7.82	7.78	7.73	**7.89**	7.85	$\underline{7.86}$
				FullKV score: 7.19			
Qwen3-1.7B	64	5.81	5.95	5.83	N/A	$\underline{6.19}$	**6.70**
	128	6.38	6.65	6.16	N/A	$\underline{6.91}$	**7.12**
	256	6.90	6.94	6.91	N/A	$\underline{7.02}$	**7.20**
	512	7.09	7.03	7.08	N/A	$\underline{7.20}$	**7.29**
				FullKV score: 8.02			
Qwen3-4B	64	6.85	6.60	6.24	7.05	7.06	**7.69**
	128	7.55	7.71	7.24	7.78	7.70	**8.12**
	256	7.90	**8.20**	7.87	8.11	$\underline{8.12}$	8.06
	512	**8.15**	$\underline{8.12}$	8.00	8.02	8.06	8.08
				FullKV score: 8.48			
Qwen3-8B	64	7.33	7.26	6.81	$\underline{7.69}$	7.58	**8.04**
	128	7.85	7.94	7.64	7.97	$\underline{8.24}$	**8.41**
	256	8.42	8.43	8.34	8.45	**8.56**	$\underline{8.51}$
	512	8.43	8.36	8.44	8.50	**8.63**	$\underline{8.53}$

lengths, allowing systematic evaluation of scaling behavior. Similar to LongBench, we use average score as the main metric, and report the results at 4K, 8K, 16K and 32K context lengths.

Baselines. We compare our method against popular KV-cache eviction methods: **1) SnapKV** (Li et al., 2024), **2) PyramidKV** (Cai et al., 2024), and **3) StreamingLLM** (Xiao et al., 2024). Additionally, we include stronger, more recent baselines that involve costly approximate future response generation, including **4) Lookahead Q-Cache** (LAQ) (Wang et al., 2025), and for 8B-scale models, **5) SpecKV** (Galim et al., 2025). In all experiments, Llama3.2-1B-Instruct and Qwen3-1.7B are used as draft models for Llama3.1-8B-Instruct and Qwen3-8B, respectively. We follow the standard eviction configuration settings for all baseline methods, which we detail in Section F

4.3 Performance Results

LongBench evaluation. Figure 4 shows the average LongBench scores of LookaheadKV and baselines, across cache budget settings ranging from 64 to 2048. Our method consistently demonstrates superior performance across all models and all budgets tested, demonstrating the effectiveness and robustness of our approach. Overall, results show that expensive draft-based methods, e.g., LAQ and SpecKV, outperform cheaper baselines, corroborating that employing approximate future response for importance estimation is effective. Nevertheless, our method significantly outperforms the draft-based approaches, especially at lower budget settings, highlighting that learning to estimate future importance is crucial for performance preservation. Due to space limitation, we report the results of 1B-scale models in Section E.

RULER evaluation. We report the RULER evaluation results of all methods with a fixed budget of 128 in Figure 4 (1B-scale results are provided in Section E). LookaheadKV consistently outperforms other baseline approaches here as well, maintaining strong performance across all evaluated context lengths. Further, note that while we limit the maximum training sequence length of LookaheadKV to 16K, our method generalizes to longer context length of 32K. We conduct additional experiments on the impact of training context length in Section 5.4.

Table 3: Theoretical and empirical cost analysis of LLaMA3.1-8B at $C = 128$.

		Theoretical Cost				Empirical Cost	
Context Length	Method	Compute (TFLOPs)	Memory Traffic (GB)	TTFT (ms)	TTFT Overhead (ms)	TTFT (ms)	TTFT Overhead (ms)
8192	Forward Pass Only	136	13	257	N/A	291	N/A
	LookaheadKV (ours)	137	13	258	1.03	302	11
	SnapKV	136	13	257	0.01	311	20
	SpecKV	159	81	337	79.53	411	121
	LAQ	137	445	492	234.59	800	509
32768	Forward Pass Only	928	13	1754	N/A	1760	N/A
	LookaheadKV (ours)	929	13	1755	1.74	1798	38
	SnapKV	928	13	1754	0.01	1838	78
	SpecKV	1115	106	2156	402.80	2263	503
	LAQ	930	451	1993	239.26	2314	554

Long-Form output evaluation. We further evaluate LookaheadKV on the **HTML to TSV** task from LongProc (Ye et al., 2025), which involves extracting structured information from long HTML documents and converting it into TSV format. This benchmark tests not only the model's ability to process long-context inputs, but also its capacity to generate long-form outputs. We assess LookaheadKV and baseline methods under two input–output settings: 12K–0.5K and 23K–2K tokens, both at a fixed cache budget ratio of 30%.

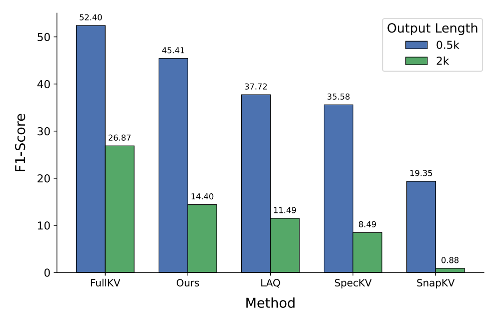


Figure 5: HTML-to-TSV evaluation results using LLaMA3.1-8B.

Figure 5 presents the results on the HTML to TSV task. Across both sequence-length configurations, LookaheadKV consistently outperforms prior approaches. We hypothesize that LookaheadKV, which learns to predict the attention pattern of the entire future response, is particularly superior in long-form generation tasks compared to draft-based methods that rely only on partial future response as the observation window.

MT-Bench evaluation. To test our method under multi-turn conversation setting, we evaluated LookaheadKV and baselines on MT-Bench (Zheng et al., 2023), a benchmark covering diverse domains, e.g., writing, coding, and math. The generated responses are evaluated using Qwen3-235B-A22B as the LLM judge. The results in Table 2 indicate that LookaheadKV is either on par or superior across all models and budgets tested. Similar to other evaluation results, we observe that our method is especially favorable at lower budget settings (e.g., $C = [64, 128]$). On the other hand, the performance gap quickly narrows at higher budgets since the context length of MT-Bench samples are relatively short compared to benchmarks targeting long-context capabilities.

5 Analysis

5.1 Efficiency Comparison

To assess the efficiency of our method against the baselines, we measure the Time-To-First-Token (TTFT) of LLaMA3.1-8B across multiple context lengths using their official implementations, with the exception of LAQ which we re-implement since it does not have an official implementation. Furthermore, since the latency of a method can vary significantly depending on the implementation, we conduct rigorous analysis and derive the theoretical latency for each method, based on the analytical model proposed in Davies et al. (2025). We discuss further details in Section B.

Table 3 presents the results of the TTFT analysis for 8K and 32K context lengths (see Table 15 for 4K and 16K results). Overall, we observe that draft-based methods incur significant overhead,

Table 4: Average LongBench performance at different temperature settings on LLaMA3.1-8B, with $C = 128$. LookaheadKV outperforms baselines across all tested temperature settings.

Method	FullKV	SnapKV	SpecKV	LAQ	LookaheadKV
Greedy	49.88	43.50	45.45	46.61	**47.72**
$T = 0.2$	49.58 (-0.60%)	43.29 (-0.48%)	44.99 (-1.01%)	46.73 (+0.26%)	**47.75** (+0.06%)
$T = 0.8$	47.82 (-4.13%)	41.39 (-4.85%)	43.43 (-4.44%)	45.27 (-2.87%)	**45.81** (-4.00%)

Table 5: 2D ablation across lookahead sizes and trainable modules, on LLaMA3.2-1B. Average LongBench scores with cache budget of 64 and TTFT overhead are reported.

	$n_{\text{lookahead}} = \mathbf{4}$		$n_{\text{lookahead}} = \mathbf{8}$		$n_{\text{lookahead}} = \mathbf{16}$		$n_{\text{lookahead}} = \mathbf{32}$		$n_{\text{lookahead}} = \mathbf{64}$		$n_{\text{lookahead}} = \mathbf{128}$	
Module	**score**	**overhead(%)**	**score**	**overhead(%)**	**score**	**overhead(%)**	**score**	**overhead(%)**	**score**	**overhead(%)**	**score**	**overhead(%)**
`emb-only`	25.5	3.4	25.7	3.8	26.4	3.4	26.4	4.2	25.8	7.3	26.2	10.7
`QV`	26.5	3.7	26.4	4.1	26.9	4.0	26.9	4.4	26.7	7.7	27.1	10.7
`all`	26.6	4.2	27.0	4.2	27.0	4.7	27.1	5.0	27.1	8.5	27.0	11.0

either due to increased computation (SpecKV) or memory traffic (LAQ). On the contrary, LookaheadKV requires marginal additional cost across all tested context lengths, achieving 14.5 times faster eviction overhead compared to LAQ at 32K sequence length.

5.2 Effect of Stochastic Decoding

To analyze the effect of stochastic generation on LookaheadKV's performance, we evaluate our method using two temperature settings: $[0.2, 0.8]$. Results in Table 4 show that LookaheadKV maintains superior performance over all other baselines across all temperature settings. Further, we observe that performance degradation at high temepraturе setting (3-4% at $T = 0.8$) is consistent across all methods, including FullKV, indicating that stochasticity in inference affects all approaches similarly. We further discuss the interplay between stochastic decoding for training data generation and LookaheadKV performance in Section E.3

5.3 Ablation on Trainable Modules

We study the impact of lookahead size $n_{\text{lookahead}}$ and LoRA placement through a 2D ablation across six lookahead sizes (4, 8, 16, 32, 64, 128) and three configurations: `emb-only` (No LoRA applied), `QV` (LoRA applied to Q and V), and `all` (LoRA applied to all linear layers). The results incidcate that both larger lookahead windows and broader LoRA coverage generally improve average LongBench performance. However, performance gains saturate at $n_{\text{lookahead}} = 32$; increasing the lookahead size beyond this point yields diminishing returns while incurring a noticeable increase in inference overhead. On the other hand, applying Lookahead LoRA to all layers results in relatively minor rise in TTFT while significantly improving the performance across all lookahead sizes. Based on this analysis, we set $n_{\text{lookahead}} = 32$ and apply LoRA to all linear modules in our main experiments.

5.4 Robustness to Training Context Length

Transformer-based language models trained with fixed context lengths often struggle to generalize beyond their training window. Similarly, one may raise concern about the context length generalization of our method. To examine this effect, we apply LookaheadKV training to LLaMA-3B with limited training context lengths of 2K, 4K, and 8K, and evaluate on RULER (Figure 6). We observe that while longer training context lengths yield better performance as expected, training on shorter contexts still remains effective with relatively minor degradation in performance, demonstrating that our method generalizes robustly to unseen sequence lengths.

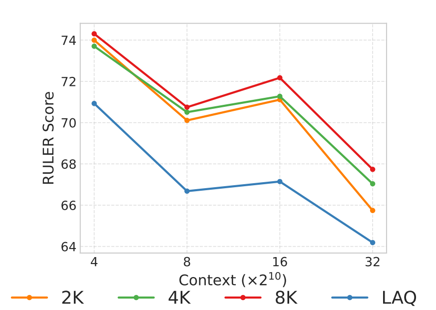


Figure 6: RULER evaluation on LookaheadKV trained with shorter contexts.

6 RELATED WORK

KV Cache Eviction. Early analyses revealed that attention scores tend to be sparse (Zhang et al., 2023), implying that only a small subset of KV entries substantially contributes to the attention output. Subsequent work showed that the importance of these tokens remains stable throughout generation, i.e., tokens deemed important early on tend to stay important (Liu et al., 2023). These observations motivated a range of eviction methods aimed at discarding unimportant KV entries while preserving model performance. A representative method is H2O (Heavy-Hitter Oracle) (Zhang et al., 2023), which proposes an eviction policy that considers the historical importance of tokens based on attention weights. NACL (Chen et al., 2024) performs eviction in a chunk-wise fashion, computing token importance locally within each chunk.

Prefill KV Cache Eviction. Another line of work, which we discuss extensively in our paper, focuses on eviction of prefill KV-cache. SnapKV (Li et al., 2024) introduced the notion of an "observation window" consisting of the suffix of the input prompt, which is used to predict important tokens to keep for subsequent response generation. Further, SpecKV (Galim et al., 2025) proposed to generate an approximate response with a smaller model and use the resulting tokens as a more reliable observation window for future importance prediction. Similarly, Lookahead Q-Cache (Wang et al., 2025) first applies a cheap eviction method, such as SnapKV, to obtain a partial low-cost draft response, then re-evicts KV entries based on the importance scores derived from the draft. KV-zip (Kim et al., 2025) adopts a query-agnostic strategy by inserting a repeated prompt and measuring which KV entries are essential for accurately reconstructing the input. Orthogonal to these approaches, several works proposed to allocate non-uniform budgets for each layer (Cai et al., 2024) and head (Feng et al., 2024) to further improve performance.

Prompt Tuning for Task Adaptation. Another line of work closely related to ours is parameter-efficient finetuning through learned prompts. Prompt Tuning (Lester et al., 2021) inserts a sequence of continuous, learnable embeddings into the frozen LLM for downstream task adaptation, while Prefix-Tuning (Li & Liang, 2021) extends this idea by pre-pending learned vectors across multiple layers. Further, P-Tuning v2 (Liu et al., 2022) demonstrated that prompt-based adaptation scales well across a wide range of model sizes. Unlike conventional prompt-tuning methods that aim to improve task performance, our work leverages learned prompts to predict internal model statistics, thereby enhancing computational efficiency rather than accuracy.

Training objectives similar to ours have been used in distillation (Wang et al., 2020), or in ranking/retrieval (Cao et al., 2007; Izacard & Grave, 2021). Some contemporaneous works (Greenewald et al., 2025; Peng et al., 2025; Samragh et al., 2025) also propose LoRA modules that selectively activate only for some tokens.

7 CONCLUSION AND LIMITATION

We introduce LookaheadKV, a trainable prefill-time KV cache eviction framework that accurately predicts token importance without relying on draft generation. The method augments a frozen LLM with a small set of learnable lookahead tokens and Lookahead LoRA modules that activate only on these tokens. Trained to match ground-truth importance distributions across layers and heads, LookaheadKV achieves performance superior to costly draft generation-based approaches while adding negligible inference overhead. Empirically, across LLaMA and Qwen model families and multiple long-context benchmarks, our approach consistently outperforms training-free heuristics and draft-based baselines, especially in low-budget regimes and long-from output tasks, while introducing less than 0.5% additional parameters and incurring only a marginal increase in prefill latency.

Due to limited compute resources, we were unable to conduct experiments on larger-sized models. However, experimental results indicate that LookaheadKV improves both performance and latency of KV cache eviction across a variety of model sizes. Further LookaheadKV currently focuses on the prefill KV cache eviction; extending LookaheadKV to also perform decoding-stage eviction remains an interesting future work.

8 Reproducibility Statement

Our source code is released in supplementary to reproduce our results, and pseudo-code is also provided in Section A. Section G provides links to datasets and evaluation benchmarks used, and Section 4.1 describes the pre-processing steps on the data.

References

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

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *ArXiv preprint*, abs/2406.02069, 2024. URL https://arxiv.org/abs/2406.02069.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pp. 129–136, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937933. doi: 10.1145/1273496.1273513. URL https://doi.org/10.1145/1273496.1273513.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. NACL: A general and effective KV cache eviction framework for LLM at inference time. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7913–7926, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.428. URL https://aclanthology.org/2024.acl-long.428/.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365/.

Michael Davies, Neal Crago, Karthikeyan Sankaralingam, and Christos Kozyrakis. Efficient llm inference: Bandwidth, compute, synchronization, and capacity are all you need, 2025. URL https://arxiv.org/abs/2507.14397.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

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

Kevin Galim, Ethan Ewer, Wonjun Kang, Minjae Lee, Hyung Il Koo, and Kangwook Lee. Draft-based approximate inference for llms. *ArXiv preprint*, abs/2506.08373, 2025. URL https://arxiv.org/abs/2506.08373.

Kristjan Greenewald, Luis Lastras, Thomas Parnell, Vraj Shah, Lucian Popa, Giulio Zizzo, Chulaka Gunasekara, Ambrish Rawat, and David Cox. Activated lora: Fine-tuned llms for intrinsics, 2025. URL https://arxiv.org/abs/2504.12397.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Gautier Izacard and Edouard Grave. Distilling knowledge from reader to retriever for question answering. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=NTEz-6wysdb.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025. URL https://openreview.net/forum?id=gcqzyyF654.

Denis Kocetkov, Raymond Li, Loubna Ben allal, Jia LI, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro Von Werra, and Harm de Vries. The stack: 3 TB of permissively licensed source code. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=pxpbTdUEpD.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243. URL https://aclanthology.org/2021.emnlp-main.243.

Cheng Li. Llm-analysis: Latency and memory analysis of transformer models for training and inference. https://github.com/cli99/llm-analysis, 2023.

Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context LLMs struggle with long in-context learning. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=Cw2xlg0e46.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th*

Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 4582–4597, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: LLM knows what you are looking for before generation. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Tianyang Liu, Canwen Xu, and Julian J. McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=pPjZIOuQuF.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 61–68, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.8. URL https://aclanthology.org/2022.acl-short.8.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/a452a7c6c463e4ae8fbdc614c6e983e6-Abstract-Conference.html.

Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. RepoAgent: An LLM-powered open-source framework for repository-level code documentation generation. In Delia Irazu Hernandez Farias, Tom Hope, and Manling Li (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 436–464, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-demo.46. URL https://aclanthology.org/2024.emnlp-demo.46/.

Arka Pal, Deep Karkhanis, Samuel Dooley, Manley Roberts, Siddartha Naidu, and Colin White. Smaug: Fixing failure modes of preference optimisation with dpo-positive. *ArXiv preprint*, abs/2402.13228, 2024. URL https://arxiv.org/abs/2402.13228.

Yuqi Peng, Lingtao Zheng, Yufeng Yang, Yi Huang, Mingfu Yan, Jianzhuang Liu, and Shifeng Chen. Tara: Token-aware lora for composable personalization in diffusion models, 2025. URL https://arxiv.org/abs/2508.08812.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential, 2025. URL https://arxiv.org/abs/2507.11851.

Minzheng Wang, Longze Chen, Fu Cheng, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, Yunshui Li, Min Yang, Fei Huang, and Yongbin Li. Leave no document behind: Benchmarking long-context LLMs with extended multi-doc QA. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5627–5646, Miami, Florida, USA, 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.322. URL https://aclanthology.org/2024.emnlp-main.322/.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5776–5788. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Xiaohan Wang, Yuhui Zhang, Orr Zohar, and Serena Yeung-Levy. Videoagent: Long-form video understanding with large language model as agent. In *European Conference on Computer Vision*, pp. 58–76. Springer, 2024b.

Yixuan Wang, Shiyu Ji, Yijun Liu, Yuzhuang Xu, Yang Xu, Qingfu Zhu, and Wanxiang Che. Lookahead q-cache: Achieving more consistent kv cache eviction via pseudo query. *ArXiv preprint*, abs/2505.20334, 2025. URL https://arxiv.org/abs/2505.20334.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Peng Xu, Wei Ping, Xianchao Wu, Chejian Xu, Zihan Liu, Mohammad Shoeybi, and Bryan Catanzaro. Chatqa 2: Bridging the gap to proprietary llms in long context and RAG capabilities. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=cPD2hU35x3.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Xi Ye, Fangcong Yin, Yinghui He, Joie Zhang, Howard Yen, Tianyu Gao, Greg Durrett, and Danqi Chen. Longproc: Benchmarking long-context language models on long procedural generation. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ruWC5LIMSo.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2O: heavy-hitter oracle for efficient generative inference of large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

A PSEUDO-CODE

The pseudocode for LookaheadKV training and eviction is described in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 LookaheadKV Training

Require: dataset $\mathcal{D}$ of input-response pairs
`scores` $\leftarrow$ [] ▷ GT importance scores
`estimates` $\leftarrow$ [] ▷ score estimates using LookaheadKV
for each training sample (X, Y) in dataset D **do**
    for each layer l **do** ▷ GT pass
        for each head h in layer l **do**
            $S \leftarrow$ GT importance score for head (l, h)
            `scores.append`(S)
        end for
    end for
    for each layer l **do** ▷ lookahead pass
        for each head h in layer l **do**
            $\hat{S} \leftarrow$ importance scores using lookahead embeddings for head (l, h)
            `estimates.append`$(\hat{S})$
        end for
    end for
    $L \leftarrow 0$ ▷ compute loss
    for all $(S, \hat{S})$ in `scores`, `estimates` **do**
        $L \leftarrow L + \mathrm{KL}\big(\mathrm{Norm}(S) \,\|\, \mathrm{Norm}(\hat{S})\big)$
    end for
    $L \leftarrow \frac{L}{|\texttt{scores}|}$
    L.backward()
end for

Algorithm 2 LookaheadKV Eviction

Require: Input prompt $X = \{x_1, \ldots, x_{n_{\text{in}}}\}$
Require: cache budget k
Append learned lookahead tokens to input and compute the sequence embeddings $\hat{\mathbf{X}} = [\mathbf{X}\ \mathbf{L}]^\top$
▷ shape: $(n_{\text{in}} + n_{\text{lookahead}}) \times d$
Perform a prefill forward pass with $\hat{\mathbf{X}}$:
for each layer l **do**
    for each head h **do**
        $\mathbf{A} \leftarrow \mathrm{Softmax}\Big(\frac{QK^\top}{\sqrt{d}}\Big)$ ▷ shape: $(n_{\text{in}} + n_{\text{lookahead}}) \times (n_{\text{in}} + n_{\text{lookahead}})$
        $\hat{\mathbf{A}} \leftarrow \mathbf{A}[n_{\text{in}} :,\ : n_{\text{in}}]$ ▷ attention between lookahead tokens and input prompt
        $\mathbf{s} \leftarrow \mathrm{MeanReduce}(\hat{\mathbf{A}})$ ▷ score vector, shape: $1 \times n_{\text{in}}$
        $\mathcal{I} \leftarrow \mathrm{TopK}(\mathbf{s},\ k)$ ▷ select Top-k indices
        $K^{\text{kept}} \leftarrow K[\mathcal{I}]$
        $V^{\text{kept}} \leftarrow V[\mathcal{I}]$
        Cache $(K^{\text{kept}},\ V^{\text{kept}})$ ▷ evict unimportant KV pairs and cache retained pairs
        Compute attention output for MLP layer
    end for
    Compute MLP output for next layer
end for
return

B Theoretical Estimation Details

This section details our methodology for theoretically estimation the Time-to-First-Token (TTFT) latency for various KV cache eviction algorithms. Our analysis is based on the analytical model for FLOPs and memory traffic proposed by Davies et al. (2025). To align configurations of theoretical estimates with them of actual measurements, we simulate the execution of LLaMA3.1-8B on a single NVIDIA H100 80GB GPU with a batch size of 1, assuming all weights and activations are in half-precision. We set KV cache budget size of 128, lookahead size as 32, and window size as 32. We only consider tensor operations which are dominant parts of the computations. To provide estimates that closely reflect real-world performance, our calculations incorporate practical hardware utilization by assuming a flops efficiency of 0.7 and a memory efficiency of 0.9, as described in Li (2023).

To isolate the specific overhead introduced by each eviction algorithm, we first establish a baseline by calculating the theoretical latency of a single forward pass. The TTFT overhead for each eviction method is then determined by subtracting this baseline forward pass latency from the method's total estimated TTFT. For LAQ, the total latency is calculated by summing the costs of its three consecutive steps—the first eviction, low-cost generation of pseudo response, and the second eviction. Similarly, the total latency of SpecKV is estimated by aggregating the latencies of its draft prefill, draft decode, and target model eviction phases. A comprehensive implementation of the code to derive theoretical estimates of all baselines is available in the Supplementary Materials.

C Implementation Optimization

Efficient attention implementations such as FlashAttention (Dao et al., 2022) do not materialize the full attention score matrix, but is required in our setting to compute importance scores and enable gradient backpropagation. A possible solution is to compute the complete attention matrix using native PyTorch (i.e., eager attention), but this quickly leads to an out-of-memory error as the matrix size grows quadratically with the sequence length, which is incompatible with our training setting (upto 16K sequence length). Fortunately, for our objective, we only require the cross-attention scores between the generated response and the entire input sequence, and the response length is typically much shorter than the input prompt.

Leveraging this observation, we adopt the following approach: for the attention layers' forward computation, we use flash attention, while for the importance score computation and loss backpropagation, we employ eager attention to only compute the partial attention score matrix with the queries of model response and keys of the entire sequence. This reduces the memory requirement of eager attention from $\mathcal{O}((|X|+|Y|)^2)$ to $\mathcal{O}(|X|\cdot|Y|+|Y|^2)$, where $|X|$ and $|Y|$ denote the lengths of the input prompt and model response, respectively, with $|X| \gg |Y|$.

D NEED FOR DATA GENERATION

One of the requirements of LookaheadKV training is that the target model's generated responses must be available as training data. However, generating these responses from the model can sometimes be costly, e.g., when applying LookaheadKV across multiple models. Hence, to assess whether this requirement of can be relaxed, we evaluate an alternative setting where training uses the responses from the source datasets instead of model-generated outputs.

We observe in Figure 7 that this substitution leads to a relatively minor drop in average LongBench performance in lower-budget regimes. We hypothesize that if the attention distribution of the model-generated responses and that of the source dataset responses are moderately similar, our method can still successfully learn to accurately predict the importance scores. Overall, these results suggest that, in scenarios where training data generation is impractical, using source responses provides a viable and effective alternative.

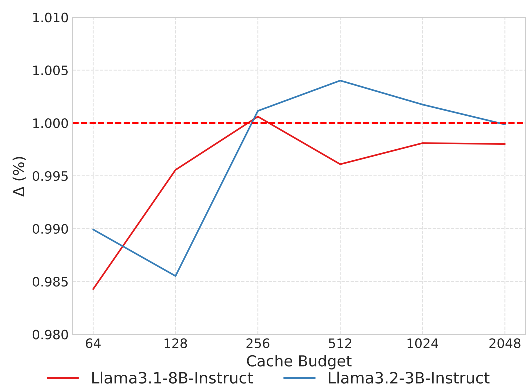


Figure 7: Performance ratio of training using model-generated data vs. source data.

E Additional Results

In this section, we provide additional experimental results excluded from the main text due to page limitation.

E.1 RULER Evaluation on Longer Contexts

To explore the capability of LookaheadKV on longer contexts, we evaluate our method on RULER at 64k and 128k context lengths using LLaMA3.1-8B-Instruct with a cache budget of 128. We sample 50 random examples per task from the RULER benchmark. As shown in Table 6, LookaheadKV achieves the best performance at these context lengths as well, showing that the effectiveness of our method scales to even longer context lengths.

Table 6: RULER evaluation results on longer context lengths using Llama3.1-8B-Instruct at $C = 128$.

Context Length	FullKV	LookaheadKV	SnapKV	SpecKV	LAQ
64k	88.48	**71.00**	36.15	65.08	64.73
128k	77.98	**55.83**	27.64	53.16	50.91

E.2 Effect of combining suffix window

To test the effect of incorporating suffix window, as proposed in SnapKV (Li et al., 2024), we augment LookaheadKV by also including queries of the last 32 prompt tokens for importance score estimation. As shown in Table 7, we observe a slight drop in performance when SnapKV importance scores are included. The degraded performance when averaging LookaheadKV importance scores with SnapKV scores, compared to using LookaheadKV scores alone, indicates that the importance predicted by our method is superior to SnapKV.

Table 7: Average LongBench scores using LookaheadKV window only and LookaheadKV + SnapKV-style suffix window, evaluated using LLaMA3.2-1B-Instruct with $C = 64$.

FullKV	LookaheadKV	LookaheadKV + Suffix Window
32.01	**29.10**	28.52 (-1.99%)

E.3 Discussion of Generation Stochasticity in LookaheadKV Training

For LookaheadKV training, various stochastic decoding methods may be employed to generate training data. One may hypothesize that the attention matrices induced by responses generated with higher stocasticity may diverge significantly from those induced by greedy responses, potentially limiting the generalizability of LookaheadKV modules trained exclusively on greedy responses to stochastic inference scenarios. To investigate this, we quantify the similarity between importance score vectors induced by greedy responses and those generated under varying temperature settings.

Table 8 presents recall@512 and Kendall rank correlation coefficients comparing importance scores induced by greedy decoding against stochastic decoding at multiple temperatures using LLaMA3.1-8B. The scores are averaged over 30 randomly selected samples from our training data, across all layers and heads. Even at relatively high temerature ($T = 0.8$), we observe strong persistence of attention patterns. Notably, the deviation is smaller than that induced by resposnes of a speculative model (Llama3.2-1B, equivalent to the SpecKV setting). This indicates that the ground-truth importance scores induced by stochastically generated responses are highly similar to the scores induced by greedy responses, which in turn indicates that greedy-generated training data provides sufficiently robust learning signals for stochastic settings.

Table 8: Importance score similarity with stochastic response using various temperatures vs. greedy response on LLaMA3.1-8B. LLaMA3.2-1B presents the similarity of importance scores using greedy response generated with LLaMA3.2-1B vs. LLaMA3.1-8B.

Generation Method	$T = 0.2$	$T = 0.4$	$T = 0.6$	$T = 0.8$	LLaMA3.2-1B
Recall@512 (%)	95.06	93.73	91.40	91.37	88.66
Kendall's Tau	91.44	88.63	84.61	84.79	80.05

E.4 Results on LongBench

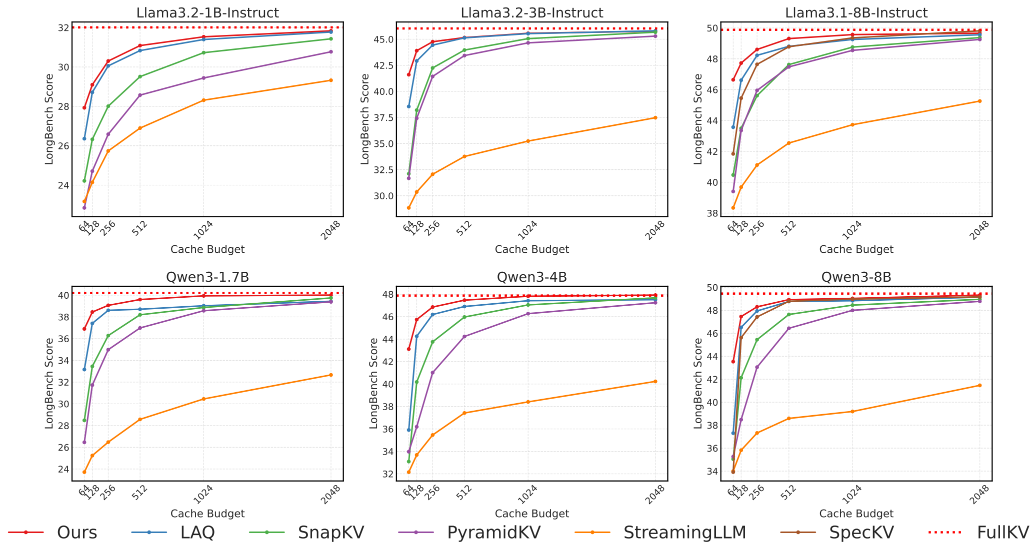


Figure 8: Full Longbench results across multiple cache budgets. 1B-scale results are included.

Table 9: LongBench evaluation results for Llama3.2-1B

LLMs	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	Lcc	RB-P	
FullKV	19.24	15.96	42.47	35.53	29.42	19.87	28.34	22.18	25.64	64.00	80.85	38.83	3.00	4.78	38.63	43.35	32.01
KV Cache Size = 64																	
StreamingLLM	14.42	12.30	24.02	24.04	24.05	9.35	13.56	19.23	13.61	35.50	68.94	29.09	4.00	3.71	37.61	37.33	23.17
SnapKV	14.24	12.18	30.53	27.30	25.66	12.44	14.27	19.27	12.37	35.00	73.57	29.29	2.00	4.49	38.01	36.84	24.22
PyramidKV	13.33	11.36	26.05	25.34	24.01	10.57	13.64	19.49	11.96	35.00	69.95	27.75	1.50	4.33	35.25	36.01	22.85
LAQ	17.21	12.76	37.30	30.27	27.36	14.22	16.42	20.09	14.28	39.50	76.02	31.19	3.00	4.58	39.15	38.37	26.36
LookaheadKV	17.69	13.30	40.80	33.66	29.80	16.95	18.76	20.65	18.97	45.50	80.12	34.77	2.50	3.22	33.69	36.47	27.93
KV Cache Size = 128																	
StreamingLLM	14.84	12.36	24.67	25.48	23.86	8.73	14.71	19.58	15.50	38.00	71.61	31.82	3.50	3.79	38.81	39.03	24.14
SnapKV	15.74	12.59	35.87	29.77	26.43	14.17	16.17	20.35	16.47	36.50	78.04	31.84	3.50	4.57	39.03	40.15	26.32
PyramidKV	14.84	12.10	33.63	27.73	23.95	11.77	15.27	19.79	13.99	35.50	74.17	30.50	1.50	4.71	37.90	37.99	24.71
LAQ	18.63	13.65	41.78	34.75	29.59	16.57	18.89	20.88	19.19	44.00	79.29	34.89	2.54	4.25	39.43	41.06	28.71
LookaheadKV	17.38	14.92	41.39	35.46	29.22	17.47	20.13	20.78	21.24	51.50	80.27	36.19	3.00	4.17	34.75	37.68	29.10
KV Cache Size = 256																	
StreamingLLM	14.74	12.39	25.28	26.71	23.87	8.88	16.99	19.67	17.97	44.50	74.85	35.96	3.50	3.90	40.37	42.19	25.74
SnapKV	16.59	13.78	38.80	32.54	28.11	16.55	18.55	20.00	19.69	41.50	79.31	33.70	4.00	4.58	39.15	41.26	28.01
PyramidKV	15.11	13.08	37.31	32.03	25.36	12.60	16.91	20.10	17.78	40.50	76.61	32.34	3.50	4.65	37.13	40.32	26.58
LAQ	18.31	14.64	41.83	35.34	29.61	17.27	20.68	21.21	21.37	52.00	79.62	36.99	4.04	4.17	40.44	43.37	30.06
LookaheadKV	18.23	14.70	40.25	36.52	30.45	18.50	21.75	20.91	22.46	57.50	80.09	38.05	4.00	4.50	36.19	40.73	30.30
KV Cache Size = 512																	
StreamingLLM	14.62	12.86	26.37	27.03	24.19	9.96	19.02	19.61	20.99	52.50	76.92	36.50	2.54	3.64	40.76	42.83	26.90
SnapKV	17.58	14.17	40.91	34.57	29.19	16.74	20.32	20.78	22.10	52.50	80.29	34.65	3.00	4.58	38.20	42.59	29.51
PyramidKV	16.55	13.48	39.67	32.62	28.38	15.59	18.50	20.87	20.54	48.50	79.30	34.43	4.00	4.50	38.79	41.42	28.57
LAQ	18.45	15.26	41.93	34.75	30.44	17.63	22.39	21.45	23.11	57.50	79.04	37.81	4.00	4.17	40.76	44.64	30.83
LookaheadKV	18.32	14.87	41.62	36.05	30.10	18.77	23.06	21.49	23.57	63.50	80.40	38.73	3.00	4.75	37.12	42.04	31.09
KV Cache Size = 1024																	
StreamingLLM	15.07	13.49	29.51	28.66	25.17	11.51	20.86	19.85	23.62	57.00	79.36	37.59	4.00	3.64	39.54	44.15	28.31
SnapKV	18.12	14.66	40.52	35.01	29.85	18.86	22.38	21.12	24.16	60.50	81.16	35.61	3.00	4.62	38.58	43.47	30.73
PyramidKV	15.94	14.15	39.34	34.16	28.37	16.68	20.27	21.11	23.14	56.00	79.93	35.36	1.50	4.58	38.48	42.06	29.44
LAQ	19.18	15.23	41.55	34.43	30.52	18.35	23.96	21.47	24.51	61.50	79.78	38.53	3.00	4.42	40.60	45.30	31.40
LookaheadKV	18.51	15.41	41.49	35.41	29.74	19.29	24.93	21.19	24.58	64.00	81.07	39.06	3.50	4.80	37.99	43.55	31.53
KV Cache Size = 2048																	
StreamingLLM	17.10	14.71	31.30	31.33	26.60	11.20	22.94	20.21	24.89	59.00	79.81	38.02	4.04	4.00	39.49	44.55	29.32
SnapKV	17.73	15.74	42.03	36.12	29.48	19.34	24.30	21.75	25.22	62.50	80.90	38.22	3.00	4.75	38.28	43.52	31.43
PyramidKV	18.83	14.50	41.40	35.75	28.89	17.40	22.05	21.14	24.97	60.50	80.62	37.33	2.50	4.50	38.51	43.46	30.77
LAQ	19.03	15.61	40.93	34.10	30.23	18.99	25.75	21.55	25.49	64.50	79.73	38.66	3.50	4.33	40.35	45.65	31.78
LookaheadKV	18.18	16.08	42.13	35.45	30.13	19.89	26.34	21.23	25.63	64.00	80.90	39.52	3.00	4.70	38.06	44.13	31.84

Table 10: LongBench evaluation results for Qwen3-1.7B

LLMs	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	Lcc	RB-P	
FullKV	18.94	24.78	46.15	39.15	33.11	19.03	30.33	23.06	25.18	74.00	85.21	42.87	0.00	94.50	46.82	40.24	40.21
KV Cache Size = 64																	
StreamingLLM	12.25	16.83	20.38	22.62	24.61	7.26	11.43	18.96	12.12	39.00	66.48	35.14	0.00	12.50	42.95	36.90	23.71
SnapKV	13.01	17.50	30.47	28.14	25.68	8.63	11.89	19.77	11.03	37.50	78.59	37.62	0.00	54.50	44.42	36.92	28.48
PyramidKV	13.20	16.31	27.19	23.83	25.80	6.95	11.23	19.12	10.72	37.00	73.99	35.99	0.00	43.00	43.01	35.97	26.46
LAQ	16.64	16.99	43.66	34.19	31.41	14.00	16.09	20.94	13.76	48.00	83.63	38.89	0.00	70.75	42.51	39.08	33.16
LookaheadKV	19.12	21.73	44.25	37.18	33.01	15.98	21.42	22.53	19.73	56.50	85.56	41.72	0.00	89.00	44.58	38.05	36.90
KV Cache Size = 128																	
StreamingLLM	13.58	16.57	22.67	22.16	24.16	7.54	13.01	19.26	14.80	44.50	70.96	37.72	0.00	12.50	46.07	38.30	25.24
SnapKV	15.94	18.73	38.02	33.35	28.02	11.74	15.40	20.80	15.71	47.00	81.65	38.77	0.00	86.00	45.47	38.55	33.45
PyramidKV	14.13	18.14	35.49	30.97	26.99	10.53	14.27	20.50	14.29	45.50	78.72	38.06	0.00	78.00	44.05	37.95	31.72
LAQ	18.38	21.08	45.04	38.04	33.52	15.36	20.17	22.70	18.78	62.50	85.21	41.79	0.14	92.00	42.94	40.84	37.41
LookaheadKV	19.46	23.27	44.59	37.81	33.82	17.97	23.71	23.12	21.70	65.50	85.56	42.38	0.12	92.75	44.80	38.61	38.45
KV Cache Size = 256																	
StreamingLLM	13.41	17.66	22.58	23.72	23.88	7.65	16.15	19.24	17.97	47.50	76.22	40.17	0.00	13.50	46.27	37.64	26.47
SnapKV	17.52	20.72	39.73	34.12	30.25	14.94	19.06	21.99	19.43	61.00	83.82	38.53	0.00	94.50	44.98	39.87	36.28
PyramidKV	17.07	19.74	38.19	33.18	29.09	13.95	17.92	21.26	17.67	56.00	81.47	39.26	0.00	91.50	45.19	38.23	34.98
LAQ	18.74	21.71	45.85	37.93	33.55	16.16	22.63	23.05	21.51	70.00	85.21	42.73	0.17	95.00	41.83	41.63	38.61
LookaheadKV	19.60	24.30	45.69	38.81	34.02	17.91	25.51	23.11	23.15	70.00	85.37	42.16	0.12	91.00	45.29	38.98	39.06
KV Cache Size = 512																	
StreamingLLM	13.92	18.40	25.10	24.91	24.77	7.67	20.38	19.63	20.78	61.00	81.80	40.38	0.00	12.50	47.28	38.62	28.57
SnapKV	19.32	22.22	44.07	36.25	30.04	15.66	22.20	22.15	21.73	70.50	84.81	40.54	0.14	94.50	46.75	40.26	38.20
PyramidKV	17.95	20.69	41.43	36.22	29.68	14.96	20.39	21.52	20.01	66.00	84.65	40.16	0.14	93.50	45.61	38.83	36.98
LAQ	16.99	22.67	46.97	38.10	33.62	16.38	24.58	23.39	22.99	71.50	84.71	42.25	0.00	94.00	40.85	40.30	38.71
LookaheadKV	19.04	24.66	44.68	39.04	33.66	17.64	27.46	23.31	24.17	73.00	85.37	42.87	0.17	94.00	45.27	39.25	39.60
KV Cache Size = 1024																	
StreamingLLM	15.32	18.63	26.90	27.88	26.44	8.47	23.56	20.34	23.82	65.50	84.00	41.69	0.00	18.00	46.93	39.71	30.45
SnapKV	18.68	24.05	44.25	38.57	30.72	16.63	24.97	22.28	23.62	71.50	85.26	40.43	0.50	95.00	46.39	39.03	38.87
PyramidKV	18.76	23.44	43.47	36.96	29.98	16.21	23.08	22.21	22.68	72.50	84.76	40.19	0.50	95.50	46.89	39.98	38.57
LAQ	16.76	22.60	45.45	38.65	33.44	17.15	26.68	23.32	24.11	73.00	84.71	43.11	0.00	95.00	39.42	41.02	39.03
LookaheadKV	19.20	25.15	44.48	39.09	32.85	18.39	29.18	23.24	25.14	73.00	84.84	43.19	0.17	95.00	46.24	39.86	39.94
KV Cache Size = 2048																	
StreamingLLM	16.32	22.12	29.96	31.34	28.13	9.37	26.32	20.96	24.58	68.50	84.59	42.84	0.00	31.00	46.63	39.92	32.66
SnapKV	19.44	24.89	45.18	38.97	32.65	17.32	27.89	22.60	24.81	72.50	85.21	42.37	0.17	95.00	46.50	40.43	39.75
PyramidKV	19.52	24.01	44.53	39.48	31.94	16.37	26.16	22.68	24.68	72.00	84.84	41.79	0.50	95.00	46.57	40.12	39.39
LAQ	16.42	22.65	45.81	38.66	33.44	17.38	28.79	23.41	25.14	73.00	84.71	42.94	0.00	95.00	43.38	40.30	39.44
LookaheadKV	19.08	25.15	45.04	38.79	33.00	17.85	29.86	23.00	25.26	73.50	85.21	43.06	0.17	94.00	46.62	40.53	40.01

Table 11: LongBench evaluation results for Llama3.2-3B

LLMs	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	Lcc	RB-P	
FullKV	27.45	43.30	54.45	55.63	43.43	31.61	32.24	24.61	25.00	73.00	88.76	43.65	0.75	96.50	64.29	61.39	47.88
KV Cache Size = 64																	
StreamingLLM	12.46	23.96	25.93	38.56	33.40	19.47	13.74	19.71	13.04	39.50	75.48	34.33	0.50	64.50	51.42	48.46	32.15
SnapKV	15.28	25.03	31.61	40.00	34.95	18.83	12.88	19.78	12.49	40.50	75.62	33.69	1.00	69.00	51.48	47.38	33.10
PyramidKV	15.50	24.84	34.33	40.70	35.07	19.39	13.48	19.85	13.03	41.50	76.69	33.95	1.50	73.00	52.98	47.77	33.97
LAQ	16.55	30.74	46.21	40.58	38.10	18.35	14.96	20.74	14.48	43.50	71.25	34.40	1.50	81.25	53.45	48.45	35.91
LookaheadKV	20.49	37.99	51.37	54.71	42.30	30.90	22.10	22.98	18.71	58.50	88.85	39.71	1.00	92.00	55.89	52.33	43.11
KV Cache Size = 128																	
StreamingLLM	15.69	23.56	26.02	38.03	32.38	18.86	14.79	19.67	15.20	45.50	78.32	37.32	0.50	65.50	55.83	51.76	33.68
SnapKV	19.56	29.48	43.55	49.81	37.95	25.99	16.27	21.17	16.16	49.50	86.31	38.10	1.50	95.00	58.75	53.78	40.18
PyramidKV	15.88	27.32	38.51	40.25	33.56	20.45	14.72	20.88	14.65	45.00	76.75	35.06	1.50	89.50	55.58	49.49	36.19
LAQ	21.44	37.82	53.26	54.98	42.75	32.08	20.44	23.69	18.86	60.50	87.55	40.48	2.50	93.00	60.85	57.87	44.25
LookaheadKV	25.17	40.13	52.28	55.10	43.47	31.38	24.83	24.46	21.57	67.00	88.85	41.37	1.00	96.50	61.00	57.77	45.74
KV Cache Size = 256																	
StreamingLLM	15.66	25.74	28.99	37.34	32.47	19.02	17.65	20.12	18.02	49.50	81.97	38.87	1.00	66.50	59.45	55.05	35.46
SnapKV	24.64	35.80	47.67	54.45	40.78	29.60	19.84	22.75	19.68	60.00	87.64	39.46	1.00	96.00	62.57	58.24	43.76
PyramidKV	18.31	31.30	44.14	51.08	36.87	25.14	18.68	22.04	17.78	56.50	85.53	38.77	1.50	95.50	59.16	53.92	41.01
LAQ	26.88	40.94	53.82	55.76	43.22	31.53	23.34	24.03	21.57	68.50	87.72	41.61	2.00	93.50	62.60	62.03	46.19
LookaheadKV	26.25	41.08	53.03	55.21	43.28	31.99	27.13	25.09	23.46	71.50	88.76	41.89	1.00	96.50	63.42	60.09	46.86
KV Cache Size = 512																	
StreamingLLM	18.02	27.64	30.11	39.03	33.32	20.70	21.47	20.39	21.96	60.50	85.45	40.27	0.50	59.50	62.54	57.33	37.42
SnapKV	25.27	39.10	51.45	54.22	42.21	32.86	23.30	23.53	22.33	70.00	88.76	40.24	1.00	96.50	64.28	60.45	45.97
PyramidKV	21.93	34.53	49.40	53.99	40.38	30.21	21.87	22.72	20.77	67.00	88.24	40.05	1.00	96.50	61.47	57.70	44.24
LAQ	26.50	42.56	53.88	55.24	43.25	32.14	25.92	24.46	23.42	73.00	87.72	42.94	1.50	93.50	62.99	61.47	46.91
LookaheadKV	26.86	41.97	53.10	55.59	43.97	32.09	29.57	25.35	24.61	72.00	88.76	42.85	1.50	96.50	63.83	60.96	47.47
KV Cache Size = 1024																	
StreamingLLM	20.48	30.08	32.30	42.20	34.23	20.65	24.81	20.84	24.19	64.50	87.39	40.95	1.00	47.00	64.74	59.17	38.41
SnapKV	25.91	41.72	52.26	56.50	43.15	32.08	26.69	24.53	24.02	71.50	88.76	41.77	1.00	96.50	64.46	61.91	47.05
PyramidKV	25.81	39.40	51.89	53.26	42.26	32.08	25.11	23.72	23.61	70.00	88.76	41.10	1.00	96.50	63.93	61.88	46.27
LAQ	27.40	43.93	54.30	55.95	43.62	31.66	28.18	25.16	24.64	73.00	87.77	43.33	1.75	93.50	62.54	62.00	47.42
LookaheadKV	27.47	42.45	53.70	55.64	43.85	32.40	30.68	24.98	25.21	73.00	88.76	42.96	1.00	96.50	64.61	61.89	47.82
KV Cache Size = 2048																	
StreamingLLM	20.87	34.01	36.39	44.11	37.06	21.93	28.06	21.64	25.16	67.50	88.39	41.55	0.50	52.00	63.58	60.98	40.23
SnapKV	26.80	43.04	53.50	55.54	44.01	33.33	29.49	24.64	24.86	73.00	88.76	41.94	1.25	96.50	64.10	62.08	47.68
PyramidKV	25.74	42.42	53.91	55.34	43.12	33.06	27.70	24.21	24.74	72.00	88.76	41.54	1.25	96.50	63.81	61.78	47.24
LAQ	27.21	43.52	53.62	55.67	43.89	31.73	30.42	24.93	25.04	73.00	87.72	43.77	1.50	93.25	63.02	61.92	47.51
LookaheadKV	27.48	42.86	53.71	55.31	43.82	32.42	31.79	24.75	25.33	73.00	88.76	43.34	1.25	96.50	64.18	62.23	47.92

Table 12: LongBench evaluation results for Qwen3-4B

LLMs	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	Lcc	RB-P	
FullKV	27.45	43.30	54.45	55.63	43.43	31.61	32.24	24.61	25.00	73.00	88.76	43.65	0.75	96.50	64.29	61.39	47.88
KV Cache Size = 64																	
StreamingLLM	12.46	23.96	25.93	38.56	33.40	19.47	13.74	19.71	13.04	39.50	75.48	34.33	0.50	64.50	51.42	48.46	32.15
SnapKV	15.28	25.03	31.61	40.00	34.95	18.83	12.88	19.78	12.49	40.50	75.62	33.69	1.00	69.00	51.48	47.38	33.10
PyramidKV	15.50	24.84	34.33	40.70	35.07	19.39	13.48	19.85	13.03	41.50	76.69	33.95	1.50	73.00	52.98	47.77	33.97
LAQ	16.55	30.74	46.21	40.58	38.10	18.35	14.96	20.74	14.48	43.50	71.25	34.40	1.50	81.25	53.45	48.45	35.91
LookaheadKV	20.49	37.99	51.37	54.71	42.30	30.90	22.10	22.98	18.71	58.50	88.85	39.71	1.00	92.00	55.89	52.33	43.11
KV Cache Size = 128																	
StreamingLLM	15.69	23.56	26.02	38.03	32.38	18.86	14.79	19.67	15.20	45.50	78.32	37.32	0.50	65.50	55.83	51.76	33.68
SnapKV	19.56	29.48	43.55	49.81	37.95	25.99	16.27	21.17	16.16	49.50	86.31	38.10	1.50	95.00	58.75	53.78	40.18
PyramidKV	15.88	27.32	38.51	40.25	33.56	20.45	14.72	20.88	14.65	45.00	76.75	35.06	1.50	89.50	55.58	49.49	36.19
LAQ	21.44	37.82	53.26	54.98	42.75	32.08	20.44	23.69	18.86	60.50	87.55	40.48	2.50	93.00	60.85	57.87	44.25
LookaheadKV	25.17	40.13	52.28	55.10	43.47	31.38	24.83	24.46	21.57	67.00	88.85	41.37	1.00	96.50	61.00	57.77	45.74
KV Cache Size = 256																	
StreamingLLM	15.66	25.74	28.99	37.34	32.47	19.02	17.65	20.12	18.02	49.50	81.97	38.87	1.00	66.50	59.45	55.05	35.46
SnapKV	24.64	35.80	47.67	54.45	40.78	29.60	19.84	22.75	19.68	60.00	87.64	39.46	1.00	96.00	62.57	58.24	43.76
PyramidKV	18.31	31.30	44.14	51.08	36.87	25.14	18.68	22.04	17.78	56.50	85.53	38.77	1.50	95.50	59.16	53.92	41.01
LAQ	26.88	40.94	53.82	55.76	43.22	31.53	23.34	24.03	21.57	68.50	87.72	41.61	2.00	93.50	62.60	62.03	46.19
LookaheadKV	26.25	41.08	53.03	55.21	43.28	31.99	27.13	25.09	23.46	71.50	88.76	41.89	1.00	96.50	63.42	60.09	46.85
KV Cache Size = 512																	
StreamingLLM	18.02	27.64	30.11	39.03	33.32	20.70	21.47	20.39	21.96	60.50	85.45	40.27	0.50	59.50	62.54	57.33	37.42
SnapKV	25.27	39.10	51.45	54.22	42.21	32.86	23.30	23.53	22.33	70.00	88.76	40.24	1.00	96.50	64.28	60.45	45.97
PyramidKV	21.93	34.53	49.40	53.99	40.38	30.21	21.87	22.72	20.77	67.00	88.24	40.05	1.00	96.50	61.47	57.70	44.24
LAQ	26.50	42.56	53.88	55.24	43.25	32.14	25.92	24.46	23.42	73.00	87.72	42.94	1.50	93.50	62.99	61.47	46.91
LookaheadKV	26.86	41.97	53.10	55.59	43.97	32.09	29.57	25.35	24.61	72.00	88.76	42.85	1.50	96.50	63.83	60.96	47.47
KV Cache Size = 1024																	
StreamingLLM	20.48	30.08	32.30	42.20	34.23	20.65	24.81	20.84	24.19	64.50	87.39	40.95	1.00	47.00	64.74	59.17	38.41
SnapKV	25.91	41.72	52.26	56.50	43.15	32.08	26.69	24.53	24.02	71.50	88.76	41.77	1.00	96.50	64.46	61.91	47.05
PyramidKV	25.81	39.40	51.89	53.26	42.26	32.08	25.11	23.72	23.61	70.00	88.76	41.10	1.00	96.50	63.93	61.88	46.27
LAQ	27.40	43.93	54.30	55.95	43.62	31.66	28.18	25.16	24.64	73.00	87.77	43.33	1.75	93.50	62.54	62.00	47.42
LookaheadKV	27.47	42.45	53.70	55.64	43.85	32.40	30.68	24.98	25.21	73.00	88.76	42.96	1.00	96.50	64.61	61.89	47.82
KV Cache Size = 2048																	
StreamingLLM	20.87	34.01	36.39	44.11	37.06	21.93	28.06	21.64	25.16	67.50	88.39	41.55	0.50	52.00	63.58	60.98	40.23
SnapKV	26.80	43.04	53.50	55.54	44.01	33.33	29.49	24.64	24.86	73.00	88.76	41.94	1.25	96.50	64.10	62.08	47.68
PyramidKV	25.74	42.42	53.91	55.34	43.12	33.06	27.70	24.21	24.74	72.00	88.76	41.54	1.25	96.50	63.81	61.78	47.24
LAQ	27.21	43.52	53.62	55.67	43.89	31.73	30.42	24.93	25.04	73.00	87.72	43.77	1.50	93.25	63.02	61.92	47.51
LookaheadKV	27.48	42.86	53.71	55.31	43.82	32.42	31.79	24.75	25.33	73.00	88.76	43.34	1.25	96.50	64.18	62.23	47.92

Table 13: LongBench evaluation results for Llama3.1-8B

LLMs	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	Lcc	RB-P	
FullKV	31.63	46.66	56.93	58.10	48.50	31.57	34.46	25.28	26.98	72.50	91.65	43.79	6.64	99.50	65.12	58.78	49.88
KV Cache Size = 64																	
StreamingLLM	25.75	21.75	31.22	49.09	42.11	23.98	17.29	20.99	16.04	38.50	82.81	34.50	7.50	99.50	54.27	48.14	38.34
SnapKV	27.37	24.99	41.77	54.27	45.27	27.52	16.75	21.73	16.32	39.00	86.32	36.58	7.50	98.50	55.57	48.07	40.47
PyramidKV	24.25	22.87	41.03	53.07	43.55	26.36	16.46	21.52	15.61	38.50	81.95	36.68	7.50	99.50	54.40	47.20	39.40
LAQ	27.62	33.71	52.35	55.85	48.92	28.06	19.74	23.19	18.90	46.00	88.29	40.62	6.83	100.00	55.55	51.49	43.57
SpecKV	24.87	26.57	51.22	55.29	46.57	25.42	19.78	22.29	19.20	33.50	85.12	39.14	8.50	97.00	57.78	57.19	41.84
LookaheadKV	30.62	41.46	55.77	56.42	48.56	30.30	23.54	24.08	21.23	60.50	91.62	42.56	7.50	99.50	58.74	53.86	46.64
KV Cache Size = 128																	
StreamingLLM	24.95	21.50	32.56	50.67	42.89	24.31	18.49	21.25	18.19	40.50	85.57	38.28	7.50	99.50	59.03	49.72	39.68
SnapKV	29.13	28.06	51.23	56.79	45.30	27.81	19.99	23.03	19.73	46.00	89.72	40.44	7.50	99.50	59.50	52.19	43.50
PyramidKV	27.70	28.86	52.00	56.76	46.11	28.13	19.86	22.81	20.03	44.50	88.41	39.73	7.50	99.50	59.84	51.96	43.36
LAQ	30.48	38.31	55.73	57.50	49.13	29.67	22.42	24.20	21.59	60.50	92.09	41.04	7.25	99.50	60.54	55.83	46.61
SpecKV	29.22	29.12	54.05	56.54	46.30	29.90	22.65	23.18	21.25	52.00	90.02	42.14	8.83	99.50	61.11	61.38	45.45
LookaheadKV	31.32	42.85	56.78	57.04	47.44	30.82	25.18	24.33	23.09	65.50	92.24	42.96	7.50	99.50	61.75	55.29	47.72
KV Cache Size = 256																	
StreamingLLM	25.96	24.08	33.73	50.56	42.61	23.49	20.86	21.60	20.64	46.00	87.50	41.09	7.50	99.50	61.19	51.53	41.12
SnapKV	27.96	34.49	55.07	57.40	46.57	29.50	22.49	23.51	22.42	54.00	91.10	40.61	7.33	99.50	62.48	55.36	45.61
PyramidKV	28.09	36.64	55.86	57.68	46.28	29.56	22.23	23.86	22.53	56.50	91.56	41.23	7.33	99.50	62.47	53.92	45.95
LAQ	31.03	43.97	55.93	57.78	49.42	30.42	24.48	24.60	23.29	68.00	92.20	42.61	7.08	100.00	62.70	58.09	48.23
SpecKV	28.66	36.19	57.26	58.17	48.51	30.85	24.83	24.60	23.32	61.00	91.16	42.46	8.33	99.50	64.21	63.18	47.64
LookaheadKV	31.96	44.01	56.80	57.99	47.41	31.46	27.26	24.56	24.59	69.00	92.55	42.93	7.33	100.00	62.81	57.02	48.61
KV Cache Size = 512																	
StreamingLLM	27.20	26.66	34.51	50.04	42.70	23.35	23.33	21.35	23.51	57.50	87.68	41.87	7.50	97.50	62.34	53.63	42.54
SnapKV	30.08	41.24	56.84	56.92	47.75	29.67	24.58	24.47	24.23	64.00	92.35	41.38	7.17	99.50	64.72	57.12	47.63
PyramidKV	29.50	40.46	56.47	57.30	47.55	30.34	24.26	24.46	24.00	66.50	91.32	41.64	7.20	99.50	63.65	55.49	47.48
LAQ	31.64	45.55	55.21	57.73	49.60	30.99	26.67	24.79	24.85	71.00	92.33	43.06	6.92	100.00	62.16	58.45	48.81
SpecKV	31.12	43.77	57.22	57.51	49.32	31.06	26.34	24.61	24.90	65.00	92.13	43.32	7.00	100.00	65.31	61.89	48.78
LookaheadKV	31.39	44.92	57.56	58.56	47.72	30.82	29.24	24.82	25.83	72.50	91.92	43.39	7.08	100.00	64.87	58.36	49.31
KV Cache Size = 1024																	
StreamingLLM	27.23	30.80	36.64	50.59	43.26	23.45	25.73	21.67	25.49	63.50	88.84	42.56	7.50	93.50	63.15	55.73	43.73
SnapKV	29.64	44.60	57.30	57.62	48.31	31.18	27.57	24.17	25.84	69.50	92.04	42.78	7.08	99.50	64.57	58.46	48.76
PyramidKV	30.79	44.91	56.65	58.13	48.17	30.56	26.65	24.53	25.88	68.00	91.78	42.20	6.83	99.50	64.41	57.77	48.55
LAQ	31.63	45.63	55.02	57.70	50.27	31.28	28.82	25.10	26.18	72.50	92.33	43.31	6.50	100.00	62.75	59.04	49.25
SpecKV	31.59	45.44	57.98	57.51	49.16	31.95	28.67	24.95	25.77	67.50	92.23	43.94	6.00	99.50	65.21	62.30	49.36
LookaheadKV	31.14	46.04	57.77	58.22	48.43	30.72	30.75	25.31	26.66	72.50	91.92	43.39	7.08	100.00	64.87	58.36	49.57
KV Cache Size = 2048																	
StreamingLLM	28.53	37.02	39.90	51.22	45.83	23.69	28.41	21.91	26.50	67.50	90.98	42.53	7.25	90.50	64.88	57.52	45.26
SnapKV	31.22	46.14	56.94	58.12	48.21	31.74	30.24	24.81	26.78	71.50	91.49	43.16	6.38	99.50	64.98	58.80	49.38
PyramidKV	31.37	46.01	56.61	58.02	48.21	31.50	29.73	24.70	26.57	71.50	91.65	42.83	6.64	99.50	64.94	58.32	49.26
LAQ	31.30	45.69	55.62	57.61	49.91	31.33	30.96	25.51	26.77	72.50	92.33	43.54	6.83	100.00	63.77	59.28	49.56
SpecKV	31.88	46.64	57.39	57.97	48.80	32.72	30.96	25.38	26.82	71.00	91.48	43.65	5.88	99.50	65.79	61.16	49.81
LookaheadKV	31.01	46.37	57.24	58.15	48.31	31.12	32.56	25.22	27.07	72.50	91.48	43.56	6.38	99.50	64.96	59.13	49.66

Table 14: LongBench evaluation results for Qwen3-8B

LLMs	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	Pre	Lcc	RB-P	
FullKV	26.04	47.76	53.33	59.23	43.37	36.05	33.66	24.05	24.79	71.50	90.21	44.43	2.00	100.00	69.39	65.57	49.46
KV Cache Size = 64																	
StreamingLLM	16.62	25.37	25.56	40.57	33.95	18.48	13.97	18.98	12.40	39.50	80.65	35.10	1.50	69.00	59.25	53.13	34.00
SnapKV	15.87	28.01	35.93	41.97	33.93	21.23	14.24	19.01	12.20	42.00	80.85	32.86	3.50	69.00	58.23	51.93	35.05
PyramidKV	15.45	27.42	36.67	42.79	34.00	19.69	14.67	18.95	12.89	43.00	80.62	33.89	2.00	71.50	58.34	52.26	35.26
LAQ	16.22	32.11	45.02	42.35	37.53	21.07	15.71	19.56	13.47	44.00	76.64	35.15	3.50	83.50	59.06	52.04	37.31
SpecKV	15.30	27.73	44.79	37.60	36.56	12.94	16.55	19.59	14.67	32.00	58.10	32.81	4.50	74.50	60.22	54.75	33.91
LookaheadKV	22.11	43.13	51.85	59.01	42.50	34.34	22.66	21.61	18.49	64.50	88.75	39.48	1.50	67.75	62.78	56.07	43.53
KV Cache Size = 128																	
StreamingLLM	17.65	26.69	28.40	41.05	33.46	20.82	15.72	19.15	15.14	43.00	82.57	38.44	1.50	70.00	62.86	56.69	35.82
SnapKV	19.14	32.65	45.99	54.81	38.95	26.59	17.66	20.83	16.04	49.50	87.10	38.90	3.50	99.50	64.62	58.29	42.13
PyramidKV	15.57	30.19	41.84	46.01	35.73	19.57	16.51	19.67	14.86	47.00	83.51	35.56	2.50	92.00	62.14	53.07	38.48
LAQ	22.74	42.15	53.55	57.89	42.84	36.74	21.33	22.25	18.34	64.50	89.55	40.93	3.00	100.00	66.74	61.70	46.52
SpecKV	23.03	37.14	53.58	56.77	42.24	31.82	21.33	22.86	19.04	60.00	88.31	41.50	3.50	100.00	66.82	61.96	45.62
LookaheadKV	26.06	44.30	53.24	58.78	42.79	35.89	25.29	22.95	21.13	66.50	88.95	41.64	3.50	99.50	65.95	62.88	47.46
KV Cache Size = 256																	
StreamingLLM	18.18	28.53	28.52	42.81	33.58	21.34	18.63	19.20	17.76	48.00	85.58	40.08	1.00	69.00	65.50	59.41	37.32
SnapKV	23.03	38.32	51.04	57.36	40.67	32.82	21.51	21.89	18.97	59.50	89.46	41.06	2.00	100.00	67.62	61.88	45.45
PyramidKV	18.47	34.87	47.44	55.68	37.89	26.67	20.43	20.92	17.43	58.50	85.20	38.98	3.50	100.00	65.51	57.32	43.05
LAQ	26.00	45.44	53.84	57.00	43.53	36.62	24.22	23.38	20.38	70.00	89.05	42.47	3.00	100.00	68.17	64.03	47.95
SpecKV	22.58	41.09	53.89	59.85	42.42	34.50	24.53	23.64	21.25	68.00	88.13	43.12	3.00	100.00	68.39	64.40	47.42
LookaheadKV	25.88	45.40	52.68	58.47	44.05	36.13	27.77	23.71	22.88	69.00	89.05	43.32	2.00	100.00	67.83	64.71	48.31
KV Cache Size = 512																	
StreamingLLM	18.94	30.86	30.21	43.89	33.26	22.51	22.24	19.62	21.16	58.50	87.48	41.11	2.00	57.00	67.06	61.59	38.59
SnapKV	24.63	43.72	51.96	58.37	42.36	34.04	25.03	22.55	21.66	69.00	89.53	42.06	3.00	100.00	69.37	64.96	47.64
PyramidKV	23.12	40.52	51.43	57.57	40.89	32.85	23.85	21.92	19.70	68.50	89.55	40.89	3.00	100.00	67.47	61.73	46.44
LAQ	27.34	46.98	53.70	57.31	43.35	37.64	26.93	23.67	22.19	72.50	88.96	43.81	3.00	100.00	68.23	64.71	48.77
SpecKV	24.22	45.65	54.34	60.53	43.85	35.26	27.21	24.04	22.53	70.50	90.20	43.71	3.50	100.00	69.25	65.85	48.79
LookaheadKV	25.33	46.49	52.04	59.32	43.09	36.92	29.56	23.80	24.01	71.50	90.21	44.20	2.00	100.00	68.88	65.58	48.93
KV Cache Size = 1024																	
StreamingLLM	21.25	32.82	31.44	45.94	34.38	23.34	25.73	20.25	23.50	62.00	88.71	41.18	0.50	44.00	68.39	63.65	39.19
SnapKV	24.26	46.13	52.48	58.52	42.66	36.89	28.39	23.61	23.33	69.00	89.55	43.13	2.00	100.00	69.05	66.27	48.45
PyramidKV	23.77	42.89	53.01	58.86	42.32	35.47	27.32	23.07	22.72	71.00	89.95	42.56	2.00	100.0	68.81	64.25	48.00
LAQ	26.11	47.27	53.45	57.01	43.52	37.26	29.50	23.88	23.47	71.50	89.63	44.00	2.00	100.00	67.94	64.83	48.84
SpecKV	24.98	46.56	54.07	59.04	43.37	34.12	29.32	24.18	23.68	71.00	90.11	44.56	3.00	100.00	69.09	66.53	48.98
LookaheadKV	25.36	47.23	52.56	59.30	43.25	36.39	31.65	23.72	24.61	71.00	90.21	44.69	0.50	100.00	68.93	65.22	49.04
KV Cache Size = 2048																	
StreamingLLM	21.73	38.54	38.02	47.96	36.78	25.53	28.69	21.47	24.11	65.00	90.30	42.85	1.00	48.50	68.37	64.63	41.47
SnapKV	25.55	47.52	53.20	58.73	42.70	36.08	30.64	23.78	24.40	71.50	90.21	43.27	1.10	100.00	69.33	65.36	48.96
PyramidKV	25.47	46.69	53.21	58.41	42.90	36.61	29.41	23.61	24.18	71.50	90.05	42.90	1.10	100.00	69.28	65.21	48.78
LAQ	24.94	47.22	53.71	57.72	43.45	37.40	31.24	24.03	24.51	72.50	90.13	44.54	2.00	100.00	68.29	64.68	49.15
SpecKV	24.86	47.00	53.80	61.43	43.74	34.94	31.38	23.98	24.57	70.50	91.11	44.46	0.00	100.00	69.12	65.87	49.17
LookaheadKV	26.76	48.01	52.92	59.43	43.20	36.21	32.64	23.93	24.93	71.00	90.21	44.74	1.00	100.00	69.23	65.01	49.33

E.5 Results on RULER

We report the RULER results across all six models tested, with cache budget settings at 64 (Figure 9) and 128 (Figure 10).

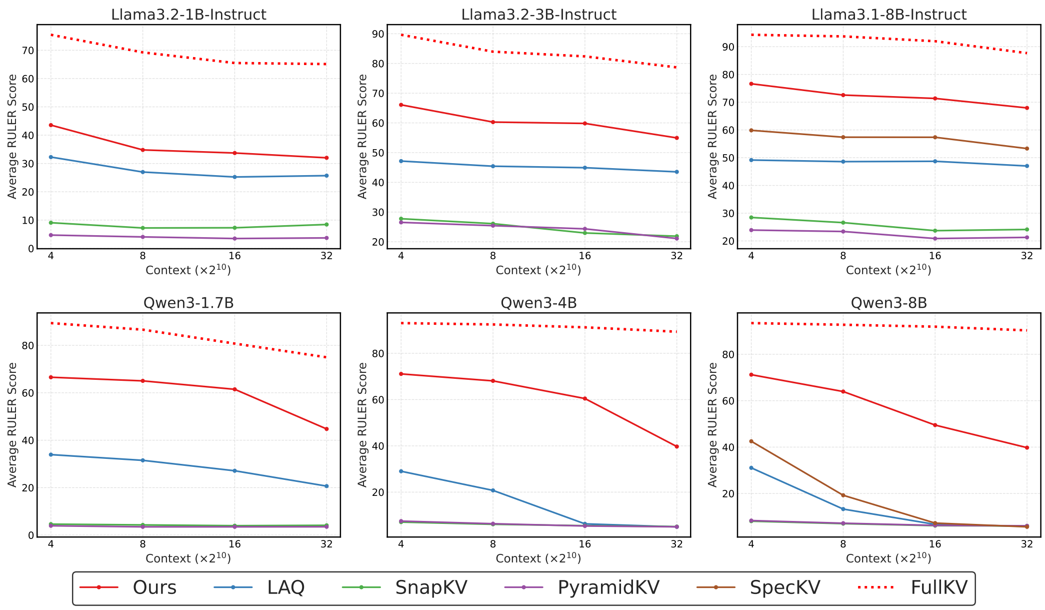


Figure 9: Full RULER results across context lengths (budget = 64)

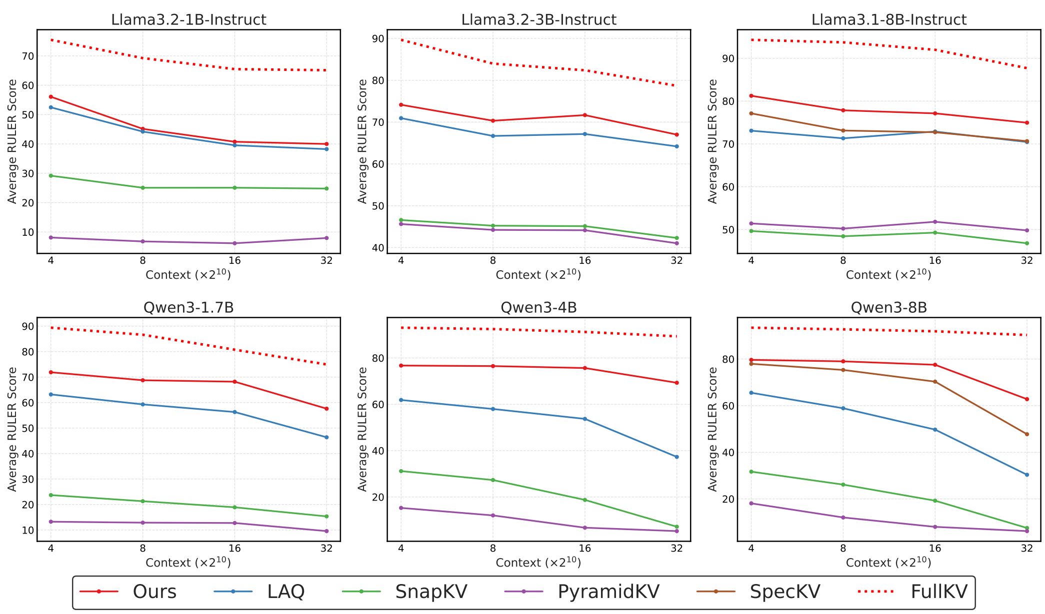


Figure 10: Full RULER results across context lengths (budget = 128)

E.6 ADDITIONAL EFFICIENCY ANALYSIS

We show the full results of the latency analysis that were omitted in the main paper due to space limitation in this section. Note that the empirical TTFT overheads for some methods can be larger than theoretical estimations. These are probably due to a combination of measurement noise and inefficient implementation of these methods in KVCache-Factory or their official implementation. Better implementations may reduce these overheads significantly, more in line with the theoretical cost.

Table 15: Theoretical and Practical Analysis across various context lengths and methods.

		Theoretical Cost				Empirical Cost	
Context Length	**Method**	**Compute (TFLOPs)**	**Memory Traffic (GB)**	**TTFT (ms)**	**TTFT Overhead (ms)**	**TTFT (ms)**	**TTFT Overhead (ms)**
4096	Forward Pass Only	60	13	113	N/A	130	N/A
	LookaheadKV (ours)	60	13	114	0.92	141	11.38
	SnapKV	60	13	113	0.01	143	13.14
	SpecKV	70	77	165	52.10	223	92.42
	LAQ	61	444	347	233.81	637	506.58
8192	Forward Pass Only	136	13	257	N/A	291	N/A
	LookaheadKV (ours)	137	13	258	1.03	302	10.88
	SnapKV	136	13	257	0.01	311	20.17
	SpecKV	159	81	337	79.53	411	120.51
	LAQ	137	445	492	234.59	800	509.38
16384	Forward Pass Only	336	13	635	N/A	658	N/A
	LookaheadKV (ours)	337	13	636	1.27	677	18.50
	SnapKV	336	13	635	0.01	695	37.12
	SpecKV	398	89	792	157.05	866	207.31
	LAQ	337	447	871	236.15	1182	523.54
32768	Forward Pass Only	928	13	1754	N/A	1760	N/A
	LookaheadKV (ours)	929	13	1755	1.74	1798	38.04
	SnapKV	928	13	1754	0.01	1838	77.67
	SpecKV	1115	106	2156	402.80	2263	502.87
	LAQ	930	451	1993	239.26	2314	553.68

F Hyper-Parameters

Training hyper-parameters. Learning rate was searched for Llama and Qwen model family among $[5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}]$. The final hyper-parameters for all experiments are shown in Table 16.

Table 16: Training hyperparameters.

Parameters	Values
Optimizer	Adam
β_1, β_2	$0.9, 0.95$
Effective Batch Size	32
Drop-out (p)	0.0
Max Sequence Length	16384 (prompt length) + 512 (response length)
Train Iters	7600
Learning rate	1×10^{-3} (for Llama), 2×10^{-4} (for Qwen)
Schedule	Cosine
Warmup steps	2%
Min LR	0.0
Gradient clipping	1.0

Eviction hyper-parameters. We use the implementations in KVCache-Factory or their official implementations (SpecKV) for all baseline methods, except for LAQ which we re-implement ourselves. Following prior works (Li et al., 2024; Cai et al., 2024; Galim et al., 2025), we use standard configuration settings for all baseline methods, including an observation window size of 32, maxpooling kernel size of 7, and mean reduction for GQA compatibility (Feng et al., 2024). For LookaheadKV we use the same settings, except we do not use window size, as our method does not train with the suffix window for prediction. Further, since our lookahead size $n_{\text{lookahead}}$ is 32, we set the maximum generation limit of LAQ and SpecKV to 32 tokens so that the methods can be compared using the same number of draft tokens.

G Datasets, Benchmarks, and Software

Software Our source code is available in the supplementary, and our implementation is built on KVCache-Factory.

Training Dataset Our training dataset mixture consist of random samples from publicly available datasets: 50K long_sft subset of ChatQA2-Long-SFT-data, 20K subset of tulu-3-sft-olmo-2-mixture, 7K samples from The Stack, and 3K samples from MetaMathFewshot, HellaSwag_DPO_Fewshot, and ARC_DPO_Fewshot, respectively.

Evaluation Benchmarks We used LongBench dataset as fetched and processed by KVCache-Factory, see HF Dataset for the official source. For RULER, we used RULER Github. For LongProc, we used LongProc Github.

H LLM Usage

LLM assistants were used to refine the wording of selected sentences, while the majority of the text was written by human. All LLM-generated text was carefully inspected to ensure that it contained no harmful or controversial content. Additionally, we used LLMs to help in finding some of the related literature discussed in the paper.