Lethe: Layer- and Time-Adaptive KV Cache Pruning for Reasoning-Intensive LLM Serving

Hui Zeng[1, 2], Daming Zhao[3], Pengfei Yang[1*], WenXuan Hou[1], Tianyang Zheng[1], Hui Li[1], Weiye Ji[1], Jidong Zhai[3]

[1]Xidian University, Xi'an, China
[2]Zhongguancun Academy, Beijing, China
[3]Tsinghua University, Beijing, China
hzeng@stu.xidian.edu.cn, cszdm@mail.tsinghua.edu.cn, pfyang@xidian.edu.cn

Abstract

Generative reasoning with large language models (LLMs) often involves long decoding sequences, leading to substantial memory and latency overheads from accumulating key-value (KV) caches. While existing KV compression methods primarily focus on reducing prefill memory from long input sequences, they fall short in addressing the dynamic and layer-sensitive nature of long-form generation, which is central to reasoning tasks. We propose **Lethe**, a dynamic KV cache management framework that introduces adaptivity along both the spatial and temporal dimensions of decoding. Along the spatial dimension, Lethe performs *layerwise sparsity-aware allocation*, assigning token pruning budgets to each transformer layer based on estimated attention redundancy. Along the temporal dimension, Lethe conducts *multi-round token pruning* during generation, driven by a *Recency-Aware Selective Retention* (RASR) mechanism. RASR extends traditional recency-based heuristics by also considering token relevance derived from evolving attention patterns, enabling informed decisions about which tokens to retain or evict. Empirical results demonstrate that Lethe achieves a favorable balance between efficiency and generation quality across diverse models and tasks, increases throughput by up to 2.56×.

Extended version — https://arxiv.org/abs/2511.06029

Introduction

Large Language Models have demonstrated impressive capabilities across a wide range of natural language understanding and generation tasks(Zhao et al. 2023). Numerous studies have been proposed to optimize both the training and inference processes of LLMs, aiming to improve their efficiency and scalability(Du et al. 2025). During inference, these models rely heavily on the *Key-Value (KV) Cache* mechanism to store intermediate representations—specifically the key and value tensors from each transformer layer—in order to enable efficient autoregressive generation(Vaswani et al. 2017). However, in many generation-intensive scenarios, especially those involving *Chain-of-Thought (CoT)*(Wei et al. 2022; Zhang et al. 2022) prompting, the memory footprint of the KV cache grows rapidly with each decoding step. Unlike fixed-length prompts used in instruction tuning or retrieval-augmented generation, CoT-style generation introduces a unique challenge: tokens are not statically provided but are dynamically generated across multiple intermediate reasoning steps, leading to unpredictable and often lengthy sequences during inference.

This dynamic growth exacerbates the memory bottleneck in long-context scenarios, making existing optimizations—largely designed for compressing static prompts—insufficient. In particular, strategies that focus exclusively on reducing the prompt's contribution to KV cache size fail to account for the cumulative and evolving nature of the cache during generation. Moreover, not all intermediate tokens contribute equally to final answers; overemphasis on historically high-attention tokens can mislead later predictions. This motivates the need for a principled mechanism to manage memory *dynamically* during the decoding process.

There exist numerous methods(Zhang et al. 2023; Cai et al. 2024; Hooper et al. 2024) and benchmarks(Zhang et al. 2024; Bai et al. 2023; An et al. 2023) aimed at addressing the challenges posed by long-context processing and evaluating the effectiveness of proposed solutions. While many approaches have been developed to reduce memory pressure from long contexts, they often rely on static assumptions that limit their applicability in dynamic generation scenarios. Specifically, these methods typically focus on optimizing the initial input alone and perform a one-time truncation or pruning after the prefill phase. Such strategies overlook two critical aspects in real-world chain-of-thought generation tasks:

First, there is significant **layerwise heterogeneity** in attention usage. While prior work such as PyramidKV (Yang et al. 2024) assumes monotonic patterns to allocate token budgets, we find that token importance varies non-monotonically across layers in reasoning tasks, making static allocation suboptimal. Second, **temporal inconsistency** in token relevance presents an equally important challenge. During autoregressive generation, attention distributions evolve with each decoding step, and the contextual importance of tokens fluctuates dynamically. Tokens that were once crucial may quickly become obsolete, while others gain importance as reasoning progresses. Static compression or

*Corresponding author.

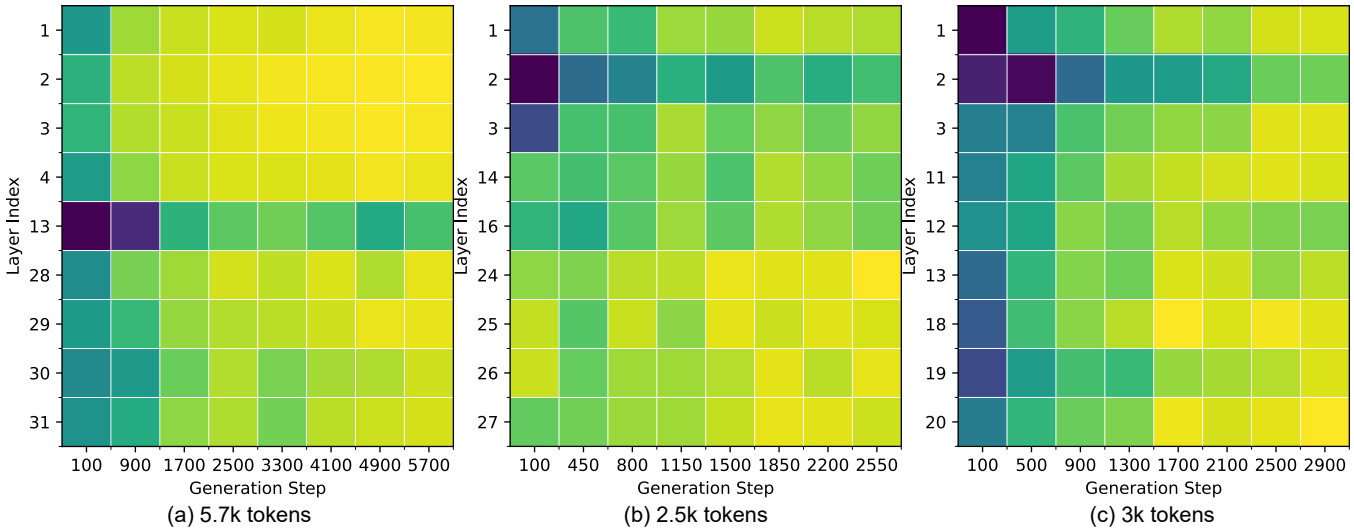

Figure 1: Layerwise attention sparsity heatmaps over decoding steps for prompts from Math500 and MMLU using DeepSeek-R1-Distill-LLaMA-8B and Qwen-7B. Warmer colors indicate higher sparsity. Variations across time and layers motivate spatial-temporal adaptive cache management.

one-time pruning fails to accommodate such shifts.

Therefore, a temporal adaptation mechanism is necessary—one that continuously re-evaluates token utility as the generation unfolds. To address these challenges, we introduce **Lethe**, a memory-efficient inference framework that dynamically prunes the KV cache along both spatial and temporal dimensions. Lethe introduces two key techniques: (1) a layerwise sparsity estimator to allocate token budgets based on runtime attention patterns, enabling adaptive per-layer pruning; and (2) **Recency-Aware Selective Retention (RASR)**, a mechanism that incrementally prunes tokens based on both attention history and recent activity, supporting multi-round pruning during decoding.

The combination of these two strategies allows Lethe to significantly reduce memory overhead while preserving, and in some cases even improving, inference accuracy and throughput.

This paper makes the following key contributions:

- We reveal the layerwise and temporal variation in KV cache utility during transformer inference, showing that different layers contribute unevenly to output semantics and that relevance shifts over time.
- We propose **Lethe**, a memory-efficient inference framework that jointly addresses spatial and temporal inefficiencies. In the spatial dimension, Lethe employs a runtime **sparsity estimator** to adaptively allocate token budgets across transformer layers. In the temporal dimension, Lethe introduces a **Recency-Aware Selective Retention (RASR)** strategy, which performs iterative cache pruning based on a combination of attention history and recent usage, allowing context-sensitive memory reduction without compromising semantic fidelity.
- We evaluate Lethe across multiple models and CoT-style tasks. Lethe achieves up to **2.56×** higher throughput and **91.7%** reduction in KV cache memory usage compared to full caching, while maintaining or surpassing the accuracy of state-of-the-art baselines such as H2O, StreamingLLM, and PyramidKV.

Related Work

Efficient Attention and System Optimizations

To address the scalability limits of self-attention in autoregressive LLMs, prior work has explored both algorithmic and system-level optimizations.

On the algorithmic side, methods like Multi-Query Attention (MQA) (Shazeer 2019), Grouped-Query Attention (GQA) (Ainslie et al. 2023), and Multi-Head Latent Attention (MLA) (Liu et al. 2024a) reduce redundancy by sharing KV projections or introducing latent representations. Other approaches approximate attention using kernel methods (Choromanski et al. 2020; Peng et al. 2021), low-rank projections (Wang et al. 2020), or block-sparsity (Child et al. 2019; Zaheer et al. 2020).

System-level solutions like FlashAttention (Dao et al. 2022; Dao 2023; Shah et al. 2024), PagedAttention (Kwon et al. 2023), and InfiniGen (Lee et al. 2024) improve efficiency via fused kernels, memory paging, or KV prefetching. However, most of these methods retain all cached tokens, limiting scalability under long-context generation.

In contrast, Lethe dynamically prunes KV entries based on attention sparsity and temporal utility, enabling finer-grained memory control during decoding.

Quantization of KV Cache

Quantization offers a complementary way to reduce KV cache size by compressing stored representations (Lin et al. 2024; Liu et al. 2024b; Hooper et al. 2024; Tao, Yu, and

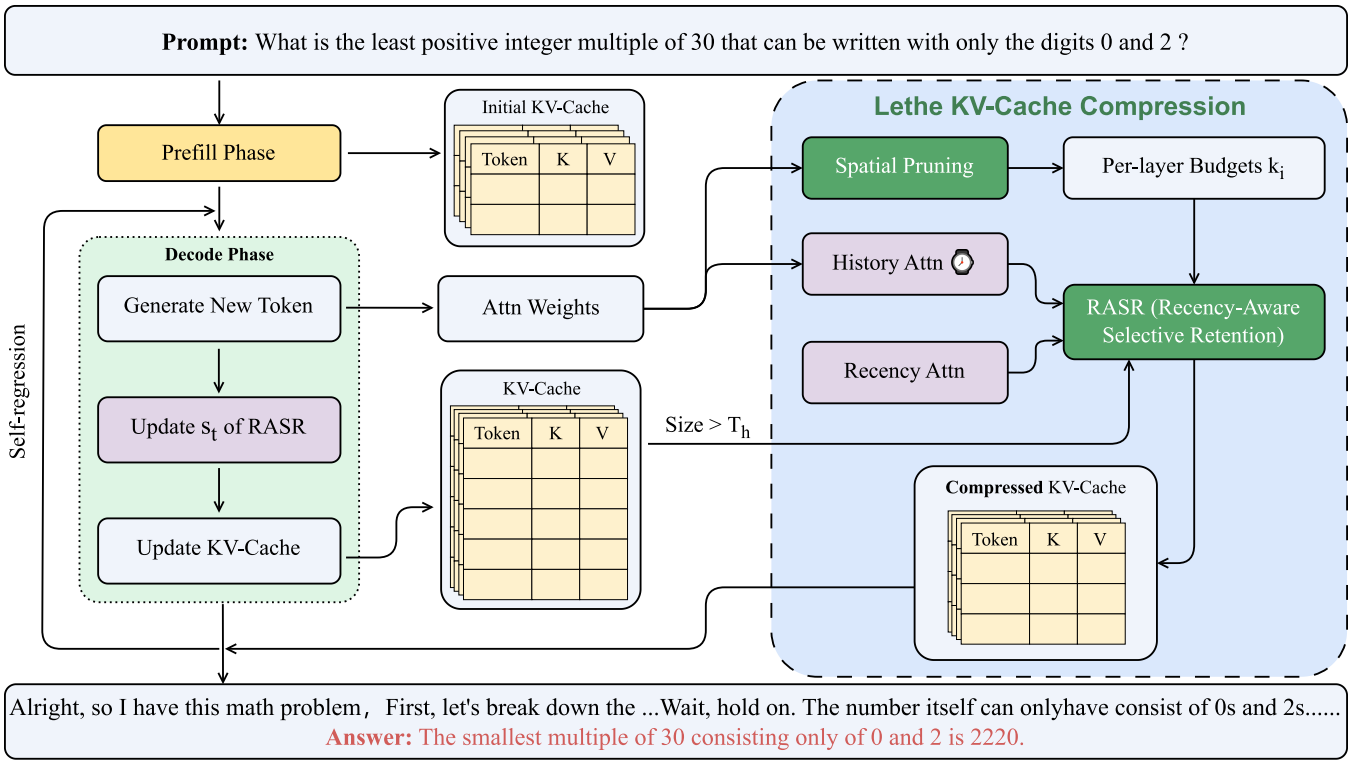

Figure 2: System overview of Lethe. During decoding, Lethe prunes the growing KV cache using layerwise sparsity estimation and attention-based ranking, reducing memory and computation without sacrificing quality.

Zhou 2024; Duanmu et al. 2024; Yue et al. 2024; Tao et al. 2025).

KIVI (Liu et al. 2024b) and KVQuant (Hooper et al. 2024) apply adaptive granularity across tokens and channels. FlexGen (Sheng et al. 2023) combines 4-bit quantization with offloading for constrained setups. AsymKV (Tao, Yu, and Zhou 2024) and SKVQ (Duanmu et al. 2024) exploit asymmetric precision and structured grouping, while CocktailQuant (Tao et al. 2025) mixes bitwidths across layers or regions.

While these methods reduce storage per token, Lethe focuses on reducing the number of tokens themselves. The two are complementary: Lethe can be layered on top of quantized caches for compounded memory savings.

Cache Eviction Strategies

Evicting unimportant tokens has emerged as a key strategy for memory-efficient inference. One-shot methods such as H2O (Zhang et al. 2023), SNAP-KV (Li et al. 2024), and Scissorhands (Liu et al. 2023) apply heuristics or saliency-based rules. Attention-guided methods like DMC (Nawrot et al. 2024), FastGen (Ge et al. 2023), and Quest (Tang et al. 2024) analyze attention scores or query activations.

Recent approaches consider structural trends: PyramidKV (Cai et al. 2024) exploits layerwise attention shifts; Keyformer (Adnan et al. 2024) keeps only high-impact keys; and StreamingLLM (Xiao et al. 2023) retains early attention sinks for long contexts.

Lethe differs by jointly modeling spatial and temporal dynamics. It allocates layer-specific budgets using sparsity estimates and applies recency-aware pruning during decoding. This unified, adaptive strategy better supports reasoning-intensive tasks where token utility evolves non-uniformly across layers and time.

Empirical Observations on Attention Sparsity

Attention Sparsity as a Basis for Fine-Grained Cache Management

Prior work (Zhang et al. 2023) shows that attention weights correlate with token utility in generation, suggesting that not all tokens are equally important. This motivates selectively retaining high-importance tokens to reduce KV cache memory without sacrificing performance.

To formalize this, we quantify *attention sparsity*—the degree to which attention is concentrated on a few tokens. A sparse distribution implies only a small subset of tokens are critical at each step.

We adopt the **Hoyer sparsity metric** (Hoyer 2004), a continuous, scale-invariant measure suitable for evaluating attention patterns across layers and models. Let $\mathbf{a} \in \mathbb{R}^n$ be a non-negative attention score vector. Its Hoyer sparsity is defined as:

$$\text{Sparsity}(\mathbf{a}) = \frac{\sqrt{n} - \frac{\|\mathbf{a}\|_1}{\|\mathbf{a}\|_2}}{\sqrt{n} - 1} \tag{1}$$

This yields values in $[0, 1]$, with higher values indicating

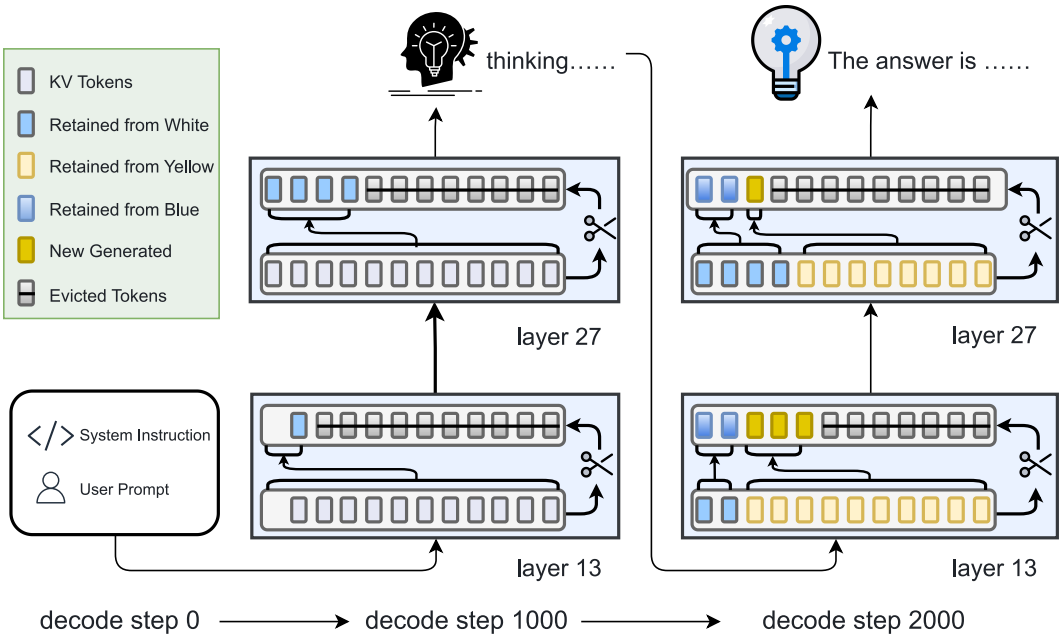

Figure 3: Visualization of Lethe's dual-dimensional KV cache pruning. Lethe prunes adaptively across layers and steps, reducing memory and computation in long-context decoding.

more peaked (i.e., selective) attention. Attention sparsity thus provides a principled basis for fine-grained KV cache management.

Layerwise Variability in Attention Sparsity

Attention sparsity is not uniform across layers. Allocating KV cache equally across layers can be inefficient: sparse layers may be over-provisioned, while dense ones suffer from under-allocation.

While PyramidKV (Cai et al. 2024; Yang et al. 2024) assumes a pyramidal sparsity trend—lower layers being dense and upper layers sparse—our analysis reveals this structure does not hold in reasoning models used with CoT prompts.

Figure 1(a–c) shows layerwise sparsity heatmaps for three Math500 prompts. For LLaMA-8B (a), both early and late layers are sparse, while mid-layers are dense—contradicting the pyramid assumption. Qwen-7B (b, c) shows varied trends: in (b), sparsity increases steadily; in (c), it fluctuates unpredictably. These results highlight prompt-specific and non-monotonic sparsity patterns.

Such variability indicates that fixed, layer-agnostic cache allocation strategies—uniform or pyramidal—are inherently suboptimal, particularly for complex reasoning tasks. An adaptive strategy is needed to match the diverse and dynamic sparsity across layers.

Temporal Dynamics in Attention Patterns During Decoding

Attention sparsity also evolves over decoding steps. Unlike standard text generation, CoT-style reasoning involves multi-step chains of thought, making the model's attention patterns temporally dynamic.

Figure 1(c) shows that early layers begin with low sparsity but become increasingly selective, while others fluctuate (e.g., layer 13 in (a)). This temporal evolution implies that fixed per-layer budgets fail to reflect shifting token importance.

In summary, both spatial (layerwise) and temporal (stepwise) variations in attention sparsity demand a dynamic, adaptive approach to KV cache management for efficient and high-quality inference.

Lethe

System Overview

Lethe is an inference-time memory management framework for large language models, designed to reduce memory and computation by dynamically pruning the key-value (KV) cache. As shown in Figure 2, Lethe operates during autoregressive decoding: after an initial prefill stage, the KV cache grows with each generated token. To avoid memory overflow, Lethe monitors cache size and triggers pruning once a configurable threshold is exceeded. Guided by attention sparsity, Lethe selectively evicts low-utility tokens on a per-layer basis, preserving model quality while improving efficiency.

Spatial Key-Value Cache Pruning via Layerwise Sparsity Estimation

Lethe introduces spatially adaptive KV pruning, allocating memory differently across transformer layers based on measured sparsity. Unlike fixed-budget or preallocated schemes, Lethe adapts retention dynamically, improving utilization in deep models where attention varies significantly across layers.

Algorithm 1: SEGMENTED ATTENTION-BASED TOKEN SHRINKING (LETHE)

Require: Attention score vector $s \in \mathbb{R}^K$ from layer l,
Number of segments D,
Threshold parameter $\tau > 1$,
Sink length s_{len},
Recent window size r,
Initial eviction threshold L_{evict}
Ensure: Token retention indices $\mathcal{T} \subseteq \{0, 1, \ldots, K-1\}$,
Updated eviction threshold L_{evict}

```
top_values, top_indices ← TopK(s, K)
cut_points ← { ⌊(K·d)/D⌋ | d = 1, ..., D−1 }
breakpoint ← −1
for each c ∈ cut_points do
    v_head ← top_values[0]
    v_cut ← top_values[c]
    if v_head / v_cut ≤ τ then
        breakpoint ← c
        break
    end if
end for
sink_indices ← {0, 1, ..., s_len − 1}
recent_indices ← {K − r, ..., K − 1}
if breakpoint ≥ 0 then
    salient_indices ← top_indices[: breakpoint]
    L_evict ← max(L_evict, breakpoint + r)
else
    L_evict ← 2 · L_evict
end if
return T, L_evict
```

Token importance is computed by aggregating attention weights across heads and batches. Let $\mathcal{A}^{(l)} \in \mathbb{R}^{B \times H_Q \times Q \times K}$ be the raw attention tensor; Lethe collapses it as:

$$\mathbf{s}^{(l)} = \sum_{b=1}^{B} \sum_{h=1}^{H_Q} \sum_{q=1}^{Q} \mathcal{A}^{(l)}_{b,h,q,:} \tag{2}$$

This head-invariant scoring supports architectures like GQA/MQA by avoiding duplicated key expansion:

$$\tilde{\mathbf{K}} = \text{repeat}(\mathbf{K}, \text{repeats} = H_Q / H_{KV}, \text{axis} = 1) \tag{3}$$

Tokens are sorted by $\mathbf{s}^{(l)}$ and divided into D segments. Lethe identifies the first segment where attention drops sharply:

$$\frac{\mathbf{v}_{\text{top}}[0]}{\mathbf{v}_{\text{top}}[k^*]} \leq \tau \tag{4}$$

The selected top-k tokens, combined with recent and sink tokens, are retained. If no breakpoint is found, Lethe conservatively delays pruning. Algorithm 1 formalizes this.

Temporal Memory Management with Recency-Aware Selective Retention (RASR)

To address the temporal growth of key-value (KV) caches in autoregressive decoding, Lethe introduces **Recency-Aware Selective Retention** (RASR), which prunes outdated or low-utility tokens based on attention history. Unlike conventional heuristics (e.g., LRU, LFU), RASR integrates both *recency* and *significance* derived from attention dynamics.

At each decoding step t, Lethe maintains a score vector $\mathbf{s}_t \in \mathbb{R}^{L_t}$ for all cached tokens:

$$\mathbf{s}_t = \gamma \cdot \mathbf{s}_{t-1} + \sum_{h=1}^{H} \sum_{i=1}^{q} \sum_{j=1}^{k} \mathbf{A}_h^{(t)}(i, j), \tag{5}$$

where $\gamma \in (0, 1)$ controls decay, and $\mathbf{A}_h^{(t)}(i, j)$ denotes attention weight from head h to token j.

Tokens are periodically ranked by a combination of $\mathbf{s}_t$ and their age. Those below a dynamic threshold are evicted, except for recent and boundary tokens. This allows Lethe to gradually forget stale context while preserving salient history.

RASR complements spatial pruning (Section) by operating along the time axis. Together, they enable lightweight, adaptive, and repeated memory reduction with minimal overhead. Empirical results show RASR reduces memory and improves output quality in long-context inference.

An Illustration of Lethe's Layerwise and Temporal KV Cache Pruning

Figure 3 illustrates Lethe's layer- and time-aware pruning. At step 1000, Lethe prunes differently across layers (e.g., layer 13 vs. 27). By step 2000, retained tokens include both recent outputs and previously retained entries. Gray tokens denote evictions; yellow and blue indicate retained tokens from current and prior steps. Lethe reduces attention length from thousands to hundreds of tokens without quality loss, enabling efficient long-context decoding.

Experiment

Experimental Setup

We evaluate Lethe from two perspectives: **accuracy preservation** and **inference efficiency**. These aspects are examined across a range of model sizes and practical scenarios to assess the generality and robustness of the proposed method.

Our experiments are conducted on four reasoning-oriented language models from the DeepSeek-R1-Distill family: Qwen-7B, Qwen-32B, LLaMA-8B, and LLaMA-70B. All models are tested on NVIDIA A100 80GB GPUs, with 3-way model parallelism applied for the 70B variant. Due to hardware limitations, the full 671B model is omitted; its officially released distilled variants are used instead as proxies.

We adopt two challenging benchmarks: **Math500** (Lightman et al. 2023), a set of complex math problems, and a subset of **MMLU** (Hendrycks et al. 2020) covering 8 diverse subjects. Together, they test both symbolic reasoning and factual understanding. Lethe is compared against four baselines: (i) *FullKV*, which retains the full history of tokens without pruning; (ii) *H2O* (Zhang et al. 2023), a top-k attention-based retention method; (iii) *StreamingLLM* (Xiao et al. 2023), which uses a fixed-size sliding window for cache management; and (iv) *PyramidKV* (Cai et al. 2024),

Method	MMLU								math500
	ab. algebra	anatomy	astronomy	bus. ethics	clin. knowl.	coll. biology	coll. chem.	coll. cs	
DeepSeek-R1-Distill-Qwen-7B									
FullKV	85.00	51.85	72.37	64.00	61.89	60.42	58.00	78.00	86.40
H2o	81.00	42.96	72.37	62.00	**63.40**	57.64	52.00	63.00	68.00
StreamingLLM	86.00	45.19	**77.63**	65.00	63.02	58.33	57.00	66.00	67.60
PyramidKV	84.00	**51.11**	75.00	62.00	60.38	63.89	49.00	64.00	58.40
Lethe(ours)	**87.00**	48.89	73.68	**66.00**	63.02	**65.28**	**61.00**	**74.00**	**85.40**
DeepSeek-R1-Distill-Qwen-32B									
FullKV	92.00	74.81	94.08	83.00	87.17	97.92	65.00	92.00	88.00
H2o	92.00	79.25	90.13	77.00	**87.92**	**95.14**	65.00	81.00	69.40
StreamingLLM	93.00	80.74	93.42	82.00	86.42	94.44	**69.00**	86.00	74.00
PyramidKV	79.00	80.00	**94.74**	81.00	**87.92**	92.36	51.00	75.00	72.60
Lethe(ours)	**94.00**	**81.48**	92.11	**84.00**	87.17	**95.14**	64.00	**87.00**	**80.00**
DeepSeek-R1-Distill-Llama-8B									
FullKV	79.00	60.74	76.32	68.00	72.45	78.47	56.00	68.00	80.80
H2o	**74.00**	60.00	67.11	72.00	72.45	70.83	52.00	61.00	65.40
StreamingLLM	73.00	59.26	70.39	69.00	72.45	77.08	50.00	**66.00**	34.40
PyramidKV	50.00	58.52	69.74	**73.00**	**73.21**	74.31	41.00	58.00	43.00
Lethe(ours)	71.00	**60.74**	**75.00**	72.00	**73.21**	**79.17**	**54.00**	63.00	**77.80**
DeepSeek-R1-Distill-Llama-70B									
FullKV	93.00	85.93	92.11	83.00	88.30	95.83	73.00	93.00	88.80
H2o	92.00	84.44	92.76	**87.00**	89.81	95.14	**70.00**	82.00	80.60
StreamingLLM	**94.00**	85.19	92.76	82.00	89.05	**95.83**	67.00	89.00	75.20
PyramidKV	83.00	84.44	91.45	82.00	89.43	94.44	62.00	78.00	77.80
Lethe(ours)	87.00	**87.41**	**93.42**	85.00	**90.19**	95.14	**70.00**	**90.00**	**82.00**

Table 1: Performance comparison of Lethe with PyramidKV, H2O, StreamingLLM, and FullKV on math500 and MMLU. Abbreviations: *abs. algebra* (abstract algebra), *anat.* (anatomy), *astron.* (astronomy), *bus. ethics* (business ethics), *clin. knowl.* (clinical knowledge), *coll. biol.* (college biology), *coll. chem.* (college chemistry), *coll. CS* (college computer science).

which applies a static layerwise allocation strategy. All baselines are re-implemented within a unified framework to ensure consistency. *SnapKV* (Li et al. 2024) is excluded, as its design centers around long-form input prefill, making it unsuitable for incremental decoding with intermediate reasoning.

Our evaluation covers both single-batch and multi-batch decoding scenarios. In the former, we vary token lengths from 1.5k to 20k to simulate long-form generation. In the latter, we scale the batch size from 1 to 32 to reflect concurrent serving workloads. Key performance metrics include decoding latency, peak memory usage, and token throughput.

Ablation studies are conducted to analyze the influence of Lethe's two main hyperparameters: `sparse_ratio`, which determines the threshold for sparsity-based token pruning, and `recent_ratio`, which specifies the fraction of most recent tokens preserved irrespective of sparsity. These experiments help to characterize the trade-offs between memory reduction and generation quality.

Accuracy Preservation under Cache Compression

We evaluate Lethe's ability to retain model accuracy under aggressive KV cache pruning. Results are summarized in Table 1.

Lethe consistently preserves or closely matches the accuracy of FullKV across models and tasks. On the challenging Math500 benchmark, Lethe maintains strong performance: under Qwen-7B, it achieves 85.4% vs. 86.4% (FullKV), while H2O and StreamingLLM degrade to 67.2% and 65.8%, respectively. Similar trends hold for Qwen-32B and LLaMA-8B, where Lethe outperforms other baselines by significant margins (e.g., $+17.3\%$ over StreamingLLM on LLaMA-8B).

On MMLU, Lethe exhibits stable accuracy across diverse subjects. Notably, under Qwen-32B, Lethe slightly surpasses FullKV on categories such as *abstract algebra* and *business ethics*, suggesting that moderate cache pruning may remove noisy context and benefit generalization. StreamingLLM performs poorly in subjects requiring long-range context, such as *philosophy* and *clinical knowledge*, while Lethe maintains strong and balanced accuracy.

Compared to PyramidKV, Lethe avoids the drop in performance caused by static per-layer pruning. For instance, under LLaMA-70B on Math500, PyramidKV suffers a 7.9% drop relative to FullKV, while Lethe preserves nearly full accuracy (88.1% vs. 88.7%). Overall, Lethe strikes a robust balance between compression and accuracy.

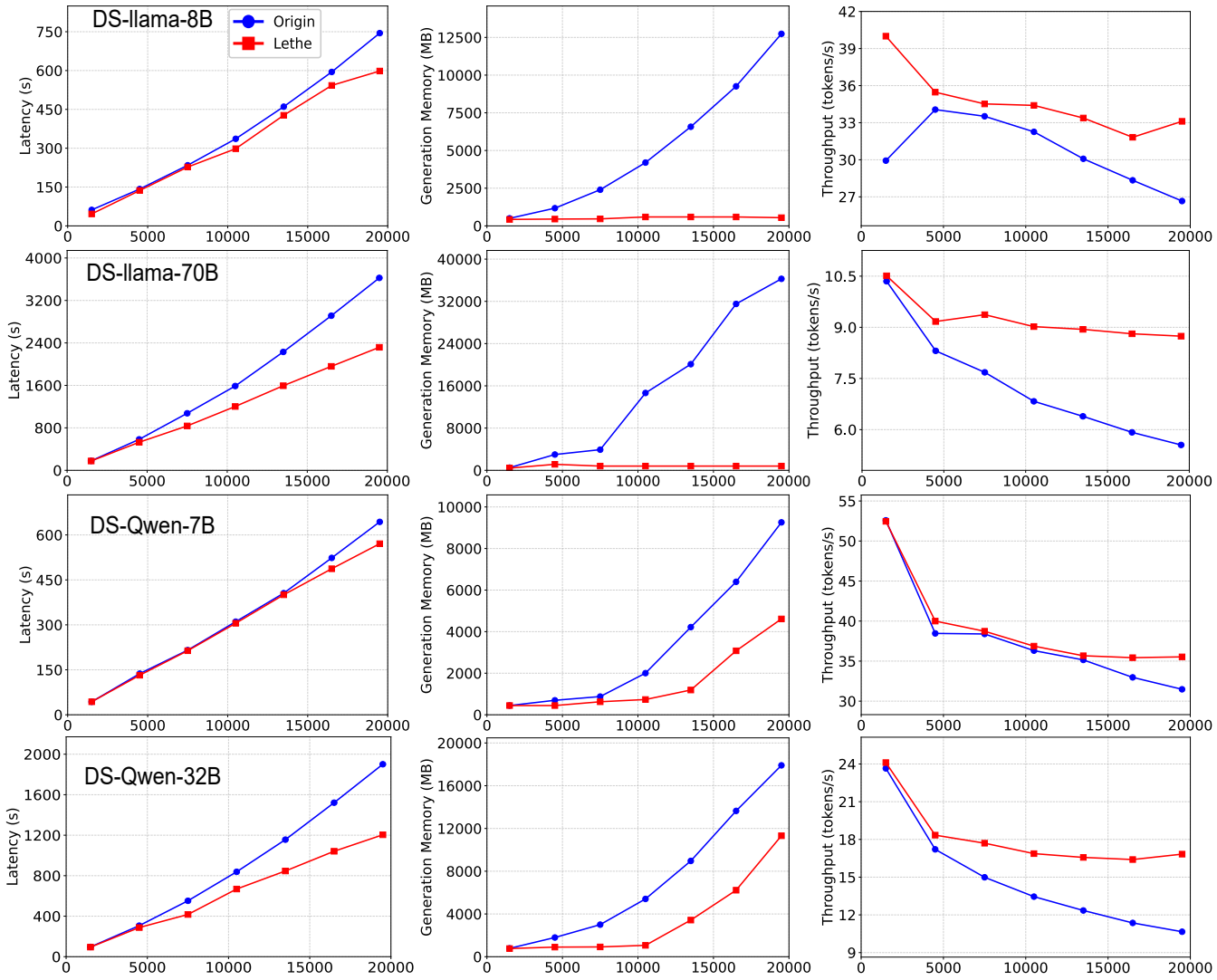

Figure 4: Latency, generation memory and throughput performance versus generated tokens comparing full KV cache (blue) and Lethe (red)

Inference Efficiency: Latency, Memory and Throughput Gains

To assess the inference performance of our proposed method, *Lethe*, we systematically compare it with the original baseline across four representative large language models: DeepSeek-R1-Distill-Qwen-7B, DeepSeek-R1-Distill-Qwen-32B, DeepSeek-R1-Distill-LLaMA-8B, and DeepSeek-R1-Distill-LLaMA-70B. All experiments were conducted on NVIDIA A100 GPUs, with the 70B model evaluated using three A100 GPUs in parallel. The generation memory is computed as the peak GPU memory usage minus the memory immediately after model loading.

Batch-Level Performance Lethe significantly reduces memory usage across all batch sizes and prevents out-of-memory (OOM) errors that occur with FullKV at batch size 32. Throughput is consistently improved; for example, on LLaMA-70B at batch size 32, Lethe enables inference (89.5 tok/s), while FullKV fails.

Token-Level Scalability Figures 4 show Lethe's latency, memory, and throughput trends over longer token lengths. Lethe reduces latency by 20–40% and achieves a stable memory footprint even at 20k tokens. For LLaMA-70B, memory usage plateaus at 800MB post-6k tokens, compared to 36GB+ for FullKV.

Ablation Study

We fix `sparse_ratio`=400 and `recent_ratio`=0.3 unless otherwise noted. Increasing `sparse_ratio` improves accuracy but with diminishing returns beyond 400. Lower values harm performance due to over-pruning.

Similarly, `recent_ratio`=0.3 yields a strong balance between coherence and compression. Higher values retain

Method	batch size				
	1	4	8	16	32
DeepSeek-R1-Distill-Qwen-7B					
FullKV	1552	10308	66504	66370	OOM
Lethe(ours)	1162	6372	40680	66602	66624
DeepSeek-R1-Distill-Qwen-32B					
FullKV	1751	17373	18633	OOM	OOM
Lethe(ours)	1551	18617	18639	18659	OOM
DeepSeek-R1-Distill-Llama-8B					
FullKV	1555	21062	65096	65811	OOM
Lethe(ours)	879	2107	5383	18643	65589
DeepSeek-R1-Distill-Llama-70B					
FullKV	2096	29640	36610	36694	OOM
Lethe(ours)	896	5110	18124	36472	36602

Table 2: Per-GPU generation memory (MB) across models and batch sizes. 70B uses 3× GPUs due to tensor parallelism. Lethe reduces memory vs. FullKV and avoids OOM.

Method	batch size				
	1	4	8	16	32
DeepSeek-R1-Distill-Qwen-7B					
FullKV	33.1	104.1	117.7	111.0	OOM
Lethe(ours)	34.6	111.7	177.3	193.3	245.0
DeepSeek-R1-Distill-Qwen-32B					
FullKV	15.2	39.1	52.9	OOM	OOM
Lethe(ours)	15.6	55.3	85.4	91.8	OOM
DeepSeek-R1-Distill-Llama-8B					
FullKV	30.1	90.9	115.6	123.4	OOM
Lethe(ours)	31.4	124.2	217.8	316.4	385.7
DeepSeek-R1-Distill-Llama-70B					
FullKV	8.3	24.1	28.7	29.4	OOM
Lethe(ours)	8.5	28.7	47.1	68.7	89.5

Table 3: Throughput (tokens/s) across models and batch sizes. Lethe consistently outperforms FullKV, especially at larger batches. "OOM" denotes out-of-memory.

unnecessary tokens, while smaller ones risk breaking contextual flow.

Conclusion

In this paper, we proposed **Lethe**, an adaptive KV cache compression framework to improve the inference efficiency of large language models. Unlike prior work that targets static prompts, Lethe focuses on the overlooked overhead from intermediate tokens in multi-step reasoning tasks. Empirical analysis reveals both *spatial heterogeneity* and *temporal dynamics* in attention patterns during decoding. To address this, Lethe employs layer-wise sparsity estimation for adaptive per-layer cache allocation and an LRU-inspired mechanism for dynamic token pruning. Experiments on Math500 and MMLU show that Lethe maintains accuracy while achieving up to **2.56**× inference speedup, highlighting its potential for efficient and scalable LLM deployment.

Acknowledgments

This work was supported in part by the Shaanxi Key Technology R&D Program under Grant 2024GX-ZDCYL-02-15 and the Natural Science Funds for Distinguished Young Scholar of Shaanxi under Grant 2025JC-JCQN-079. This work was also supported by the Zhongguancun Academy under the Internal Research Grant No. C20250501.

References

Adnan, M.; Arunkumar, A.; Jain, G.; Nair, P. J.; Soloveychik, I.; and Kamath, P. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6: 114–127.

Ainslie, J.; Lee-Thorp, J.; De Jong, M.; Zemlyanskiy, Y.; Lebrón, F.; and Sanghai, S. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

An, C.; Gong, S.; Zhong, M.; Zhao, X.; Li, M.; Zhang, J.; Kong, L.; and Qiu, X. 2023. L-eval: Instituting standardized evaluation for long context language models. *arXiv preprint arXiv:2307.11088*.

Bai, Y.; Lv, X.; Zhang, J.; Lyu, H.; Tang, J.; Huang, Z.; Du, Z.; Liu, X.; Zeng, A.; Hou, L.; et al. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Cai, Z.; Zhang, Y.; Gao, B.; Liu, Y.; Liu, T.; Lu, K.; Xiong, W.; Dong, Y.; Chang, B.; Hu, J.; et al. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Child, R.; Gray, S.; Radford, A.; and Sutskever, I. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Choromanski, K.; Likhosherstov, V.; Dohan, D.; Song, X.; Gane, A.; Sarlos, T.; Hawkins, P.; Davis, J.; Mohiuddin, A.; Kaiser, L.; et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Dao, T. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Dao, T.; Fu, D.; Ermon, S.; Rudra, A.; and Ré, C. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359.

Du, J.; Wei, Y.; Ye, S.; Jiang, J.; Chen, X.; Huang, D.; and Lu, Y. 2025. Co-Designing Transformer Architectures for Distributed Inference With Low Communication. *IEEE Transactions on Parallel and Distributed Systems*, 36(4): 717–730.

Duanmu, H.; Yuan, Z.; Li, X.; Duan, J.; Zhang, X.; and Lin, D. 2024. Skvq: Sliding-window key and value cache quantization for large language models. *arXiv preprint arXiv:2405.06219*.

Ge, S.; Zhang, Y.; Liu, L.; Zhang, M.; Han, J.; and Gao, J. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Hendrycks, D.; Burns, C.; Basart, S.; Critch, A.; Li, J.; Song, D.; and Steinhardt, J. 2020. Aligning ai with shared human values. *arXiv preprint arXiv:2008.02275*.

Hooper, C.; Kim, S.; Mohammadzadeh, H.; Mahoney, M. W.; Shao, Y. S.; Keutzer, K.; and Gholami, A. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37: 1270–1303.

Hoyer, P. O. 2004. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(Nov): 1457–1469.

Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J.; Zhang, H.; and Stoica, I. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 611–626.

Lee, W.; Lee, J.; Seo, J.; and Sim, J. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 155–172. Santa Clara, CA: USENIX Association. ISBN 978-1-939133-40-3.

Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; and Chen, D. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37: 22947–22970.

Lightman, H.; Kosaraju, V.; Burda, Y.; Edwards, H.; Baker, B.; Lee, T.; Leike, J.; Schulman, J.; Sutskever, I.; and Cobbe, K. 2023. Let's Verify Step by Step. *arXiv preprint arXiv:2305.20050*.

Lin, Y.; Tang, H.; Yang, S.; Zhang, Z.; Xiao, G.; Gan, C.; and Han, S. 2024. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *arXiv preprint arXiv:2405.04532*.

Liu, A.; Feng, B.; Wang, B.; Wang, B.; Liu, B.; Zhao, C.; Dengr, C.; Ruan, C.; Dai, D.; Guo, D.; et al. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Liu, Z.; Desai, A.; Liao, F.; Wang, W.; Xie, V.; Xu, Z.; Kyrillidis, A.; and Shrivastava, A. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36: 52342–52364.

Liu, Z.; Yuan, J.; Jin, H.; Zhong, S.; Xu, Z.; Braverman, V.; Chen, B.; and Hu, X. 2024b. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.

Nawrot, P.; Łańcucki, A.; Chochowski, M.; Tarjan, D.; and Ponti, E. M. 2024. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*.

Peng, H.; Pappas, N.; Yogatama, D.; Schwartz, R.; Smith, N. A.; and Kong, L. 2021. Random feature attention. *arXiv preprint arXiv:2103.02143*.

Shah, J.; Bikshandi, G.; Zhang, Y.; Thakkar, V.; Ramani, P.; and Dao, T. 2024. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37: 68658–68685.

Shazeer, N. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Sheng, Y.; Zheng, L.; Yuan, B.; Li, Z.; Ryabinin, M.; Chen, B.; Liang, P.; Re, C.; Stoica, I.; and Zhang, C. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, 31094–31116. PMLR.

Tang, J.; Zhao, Y.; Zhu, K.; Xiao, G.; Kasikci, B.; and Han, S. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*.

Tao, Q.; Yu, W.; and Zhou, J. 2024. Asymkv: Enabling 1-bit quantization of kv cache with layer-wise asymmetric quantization configurations. *arXiv preprint arXiv:2410.13212*.

Tao, W.; Zhang, B.; Qu, X.; Wan, J.; and Wang, J. 2025. Cocktail: Chunk-Adaptive Mixed-Precision Quantization for Long-Context LLM Inference. *arXiv preprint arXiv:2503.23294*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, S.; Li, B. Z.; Khabsa, M.; Fang, H.; and Ma, H. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Xiao, G.; Tian, Y.; Chen, B.; Han, S.; and Lewis, M. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Yang, D.; Han, X.; Gao, Y.; Hu, Y.; Zhang, S.; and Zhao, H. 2024. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*.

Yue, Y.; Yuan, Z.; Duanmu, H.; Zhou, S.; Wu, J.; and Nie, L. 2024. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*.

Zaheer, M.; Guruganesh, G.; Dubey, K. A.; Ainslie, J.; Alberti, C.; Ontanon, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33: 17283–17297.

Zhang, X.; Chen, Y.; Hu, S.; Xu, Z.; Chen, J.; Hao, M. K.; Han, X.; Thai, Z. L.; Wang, S.; Liu, Z.; et al. 2024. ınfty-Bench: Extending Long Context Evaluation Beyond 100K Tokens. In *ACL (1)*.

Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; et al. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36: 34661–34710.

Zhang, Z.; Zhang, A.; Li, M.; and Smola, A. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).