

LeanK: Learnable K Cache Channel Pruning for Efficient Decoding

Yike Zhang^{1*}, Zhiyuan He², Huiqiang Jiang², Chengruidong Zhang²,
Yuqing Yang², Jianyong Wang¹, Lili Qiu²

¹Tsinghua University, ²Microsoft Research

zhangyik21@mails.tsinghua.edu.cn, zhiyuhe@microsoft.com

Abstract

Large language models (LLMs) enable long-context tasks but face efficiency challenges due to the growing key-value (KV) cache. We propose LeanK, a learning-based method that prunes unimportant key (K) cache channels by leveraging static channel sparsity. With a novel two-stage training process, LeanK learns channel-wise static mask that could satisfy specific sparsity ratio and hardware alignment requirement. LeanK reduces GPU memory and accelerates decoding without sacrificing accuracy. Experiments demonstrate up to 70% K cache and 16%-18% V cache memory reduction. Custom decoding kernel enables 1.3x speedup for attention computation. We also provide insights into model channels and attention heads during long-context inference by analyzing the learned importance distribution. Our code is available at <https://aka.ms/LeanK>.

1 Introduction

Large language models (LLMs) have advanced to support long-context tasks such as document understanding (Li et al., 2024a), multi-turn dialogues (Yi et al., 2024), repository-level code completion (Jimenez et al., 2024), and complex reasoning (Zhou et al., 2025). However, efficient inference under these long-context settings remains challenging due to the growing size of the key-value (KV) cache, which not only significantly increases GPU memory usage but also repeatedly stresses GPU memory bandwidth during token generation, leading to slower inference speeds (Zhang et al., 2023; Tang et al., 2024; Liu et al., 2024b).

Existing efforts to optimize the KV cache include: (1) Eviction, which discards cache of less important tokens (Li et al., 2024b; Zhang et al., 2023) or cache in less important attention heads (Xiao et al., 2024b,a); (2) Selection, which retains the full KV cache but selectively reads relevant

entries during inference (Tang et al., 2024; Chen et al., 2025; Liu et al., 2024a); and (3) Quantization, which compresses the KV cache using compressed data types (Liu et al., 2024b). Despite the effectiveness of these methods, they typically assume that all channels in the key (K) cache are equally necessary when the final attention score is calculated, which limits their efficiency optimization potential.

We identify a unique and largely underexplored opportunity for optimizing the K cache by leveraging the sparsity in its channel dimension. Specifically, we observe: (1) Previous studies suggest that RoPE influences the feature encoded in each dimension of K (Barbero et al., 2025). Dimensions associated with high frequencies tend to be less stable for text retrieval, revealing a potential opportunity for pruning. (2) Besides, we find that the importance of K cache channels tends to be static and can be determined offline. This static sparsity can offer consistent speedups during on-line inference. (3) K channel sparsity is orthogonal to existing approaches, and can be combined with them for further acceleration.

Based on our observation, we propose LeanK, a learning-based method for pruning the channel dimension of the K cache to enable efficient long-context decoding. LeanK learns static sparsity through a double-stage process. In the first stage, it estimates the global importance of each K channel. In the second stage, it learns a sparse channel mask that adheres to a target sparsity ratio and is optimized for hardware efficiency. At inference time, LeanK prunes the K cache channels based on the learned static sparsity, significantly reducing GPU memory usage and improving decoding speed.

We conduct extensive experiments on two recent long-context LLMs, Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen2.5-7B-Instruct (Qwen et al., 2025), across three different benchmarks. Results show that LeanK enables approximately 70% GPU memory reduction in K cache

*Work during internship at Microsoft.

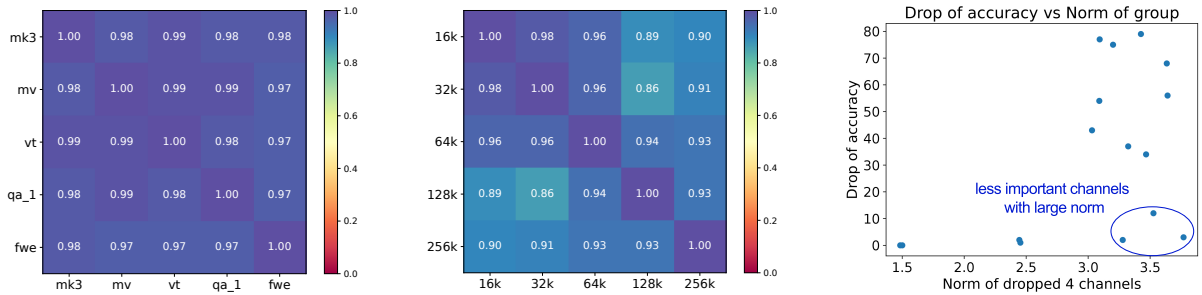


Figure 1: **Left** and **Middle**: K channel sparsity remains static across different RULER tasks and sequence lengths. **Right**: There exist channels with relatively large norm but has limited impact on end-to-end performance.

and 16%–18% memory reduction in V cache size. Custom decoding kernel enables 1.3x speedup for attention computation while preserving model accuracy. Moreover, LeanK is highly compatible with existing KV cache optimization techniques. When combined with quantization methods such as KIVI (Liu et al., 2024b), LeanK improves the overall KV cache compression ratio from $5.3\times$ to $9.7\times$, substantially alleviating memory bottlenecks in long-context inference. Furthermore, we analyze the learned channel-wise importance distribution of K cache and gained insights into model’s behavior related to RoPE.

2 Motivation

The motivation of our method is based on the following three observations.

2.1 The use of RoPE introduces channel inefficiency in K.

In modern LLMs, RoPE is applied to both Q and K in Transformer attention. RoPE encodes positional information by assigning each pair of K dimensions a specific frequency. However, recent studies show that high-frequency dimensions tend to be unstable and contribute little to long-context inference (Hong et al., 2024; Barbero et al., 2025). These findings indicate that many K channels are underutilized during long-context inference, presenting an opportunity for effective pruning.

2.2 Sparsity in K channels tends to be static.

We assess the staticity of important K channels by computing Pearson correlation coefficient (Sedgwick, 2012) between channel norm distribution of different input sequences.¹ As shown in Figure 1, channel norm distribution on Llama-3.1-8B-Instruct exhibits consistently high Pearson cor-

¹Detailed methods of computing channel norm distribution is provided in Appendix A, which roughly assesses each channel’s importance through its norm.

relation coefficients across five diverse RULER tasks (Hsieh et al., 2024) and multiple sequence lengths, suggesting an inherent staticity in channel importance.

In contrast, ThinK (Xu et al., 2025) assumes dynamic sparsity in K channels, estimating each channel’s importance dynamically via $Q_{window}K^T$ during inference, where Q_{window} corresponds to several recent context tokens. Instead, we test a simpler static, norm-based pruning approach. We derive a static pruning mask based on the average norm of each channel across 100 input sequences from RULER 64K NIAH_multikey3 task, and apply this mask universally across all tasks. Results in Table 1 show that the static method achieves comparable performance to ThinK.

	Method	Pruning Ratio	Acc
Llama-3.1-8B-Instruct	Original	-	84.38
	ThinK (Dynamic norm)	60%	80.56
	Static norm	60%	81.57

Table 1: RULER 64K performance of static norm-based pattern and dynamic norm-based method (ThinK). Detailed results and analysis are in Appendix I.

2.3 Some channels exhibit large magnitudes but limited impact.

Furthermore, We conduct experiments by removing subsequent groups of every 4 channels from Llama-3.1-8B-Instruct model and evaluate the resulting performance degradation on RULER 64K NIAH_multikey3 task. Relationship between average norm of each group and corresponding performance drop is presented in Figure 1. There exist channels with large norm but have little impact on model performance (marked by the blue circle). Relying solely on magnitude to decide channel importance may overlook the heterogeneity between different decoder layers and attention heads and miss some pruning opportunities.

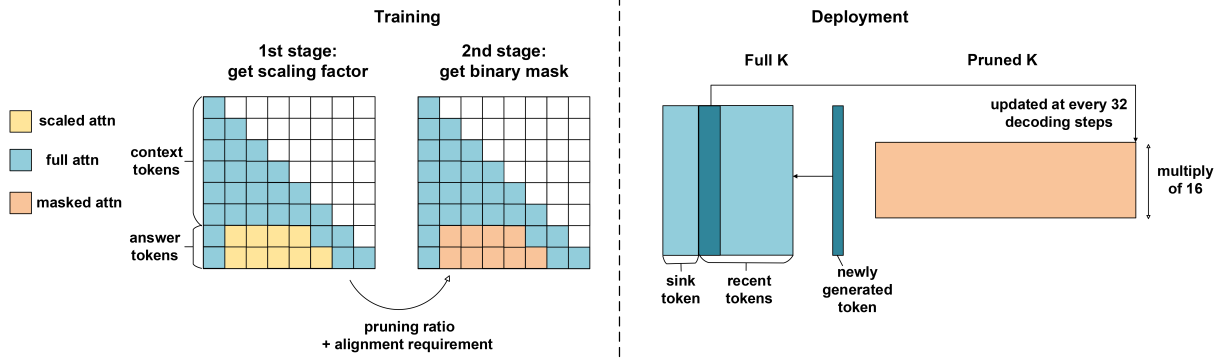


Figure 2: Overall demonstration of LeanK’s double-stage training and deployment method.

3 Method

Based on our observations, we present **LeanK**, a learning-based method that exploits channel sparsity in K cache to accelerate long-context decoding. Our goal is to learn a binary mask for pruning K channels according to a predefined pruning ratio, using a **double-stage process**. In the first stage, we learn a **continuous scaling factor** representing the global importance of each K channel. In the second stage, we convert the learned scaling factor into a **binary mask** suitable for deployment.

3.1 Training stage 1

Consider a transformer model with L layers, where each layer has either Multi-Head Attention (MHA) or Grouped Query Attention (GQA) containing n heads (or groups), each with a dimension of d . We introduce a scaling factor $\alpha \in \mathbb{R}^{L \times n \times d}$, initialized with all elements set to 1. The value of α represents the global importance score of each channel and will be learned in the first stage.

Suppose the input sequences for learning are represented as $\mathbf{X} = [\mathbf{X}_{\text{ctx}}; \mathbf{X}_{\text{ans}}]$, where the semi-colon denotes concatenation. Each sequence consists of context tokens \mathbf{X}_{ctx} and answer tokens \mathbf{X}_{ans} . Since the decoding phase is the primary focus of our work, the training loss is computed only based on the answer part. Initially, standard full attention is applied, from which we obtain the hidden states corresponding to the answer tokens from the last layer, denoted by $\mathbf{H}_{\text{full}} \in \mathbb{R}^{N_{\text{ans}} \times n \times d}$, where N_{ans} is the number of answer tokens.

Then, we apply a specialized *scaled attention* based on α . As shown in Figure 2, the attention corresponding to \mathbf{X}_{ctx} in each head is still full attention, formulated as:

$$\mathbf{A}_{\text{ctx}} = \text{softmax}(\mathbf{Q}_{\text{ctx}} \mathbf{K}_{\text{ctx}}^T \odot \mathbf{M}_{\text{causal}}) \mathbf{V}$$

For \mathbf{X}_{ans} , we scale the attention logit as:

$$\begin{aligned} L_{\text{ans}} = & \mathbf{Q}_{\text{ans}} \mathbf{K}^T \odot \mathbf{M}_{s+1} \\ & + \mathbf{Q}_{\text{ans}} (\mathbf{K} \text{diag}(\alpha_{i,j}))^T \odot \mathbf{M}_{\text{mid}} \end{aligned} \quad (1)$$

where \mathbf{M}_{s+1} is a binary mask preserving only the sink and sliding window attention, while \mathbf{M}_{mid} preserves only the middle attention region (excluding sink and sliding windows), as illustrated in Figure 2. The scaling factor $\alpha_{i,j} \in \mathbb{R}^d$ corresponds to layer i , head j , and scales each channel dimension of the key matrix $\mathbf{K} \in \mathbb{R}^{N \times d}$ (N denotes the number of input tokens), producing $\mathbf{K} \text{diag}(\alpha_{i,j})$. Due to masks \mathbf{M}_{s+1} and \mathbf{M}_{mid} , scaling affects only the middle-region attention logits, keeping the sink and sliding window attention intact, as they are more critical and consume constant GPU memory regardless of sequence length (Xiao et al., 2024b). In contrast, the middle attention grows with the number of tokens, making it the primary target for pruning. Then, the scaled attention logits for the answer tokens are computed as:

$$\mathbf{A}_{\text{ans}} = \text{softmax}(L_{\text{ans}} \odot \mathbf{M}_{\text{causal}}) \mathbf{V}$$

In each attention head, the attention maps for context and answer tokens are concatenated as $\mathbf{A} = [\mathbf{A}_{\text{ctx}}; \mathbf{A}_{\text{ans}}]$. We obtain the hidden states from the final layer corresponding to the answer tokens, denoted as $\mathbf{H}_{\text{scaled}} \in \mathbb{R}^{N_{\text{ans}} \times n \times d}$.

The training loss for the first stage is defined as:

$$\mathcal{L}_{1\text{st}} = \|\mathbf{H}_{\text{full}} - \mathbf{H}_{\text{scaled}}\|_2^2 + \lambda \|\alpha\|_1,$$

where the first term is an L2 distillation loss that encourages the scaled hidden states to remain close to the full ones, and the second term is an L1 regularization that promotes sparsity in the learned scaling factors α . The coefficient λ controls the trade-off between performance preservation and sparsity.

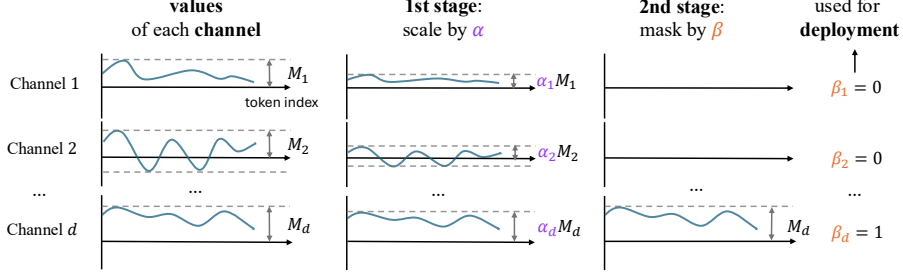


Figure 3: Visualization of the training objectives of the two training stages.

We train our model on two passkey retrieval tasks: (1) Dense retrieval, where \mathbf{X}_{ctx} consists of key-value pairs and the goal is to retrieve the value for a given key as \mathbf{X}_{ans} ; and (2) Multi-value retrieval, where \mathbf{X}_{ctx} includes distraction text and keys with multiple values, and the task is to retrieve all values for a given key. All keys and values are randomly generated. We select these tasks because the first is challenging and thus effective for preserving the model’s retrieval capabilities, while the second task involves generating relatively long answers, potentially improving the model’s long-term generation performance.

3.2 Training stage 2

The scaling factor $\alpha \in \mathbb{R}^{L \times n \times d}$ learned in the first stage encodes the importance of each channel. Our next goal is to derive a channel-wise binary mask $\beta \in \{0, 1\}^{L \times n \times d}$ that could directly be used for pruning, where $\beta_{i,j,k} = 0$ indicates that the k -th channel of the K cache in layer i and head j should be pruned. We require such a binary mask to satisfy two requirements: **(R1)** The desired **pruning ratio** $s\%$ should be specified before deployment, and **(R2)** The mask should be GPU-friendly, i.e., the number of remained channels for each head should satisfy the **alignment requirement** for efficient memory loading and computation (e.g. be a multiple of $r = 16$ or 32).

Training stage 1 does not take these two requirements into consideration, bringing up the necessity of a second training stage. In the second training stage we optimize a binary mask β generated by:

$$\beta = \text{Top}_{s\%,r}(\alpha),$$

where $\text{Top}_{s\%,r}(\cdot)$ is a two-step operator. First, $\text{Top}_{s\%}(\alpha)$ selects the top $s\%$ most important channels across all heads. Then, for each layer i and head j , let $n_{i,j}$ be the number of channels initially selected. We round $n_{i,j}$ to the nearest multiple of r , denoted by $n'_{i,j} = \lfloor \frac{n_{i,j}}{r} \rfloor \times r$, and keep the

top $n'_{i,j}$ entries from $\alpha_{i,j}$ to form the final mask: $\beta_{i,j} = \text{Top}_{n'_{i,j}}(\alpha_{i,j})$.

The scaled attention in the second stage is applied similarly as in the first stage, but with α replaced by the binary mask β in Equation 1. Since the purpose of this stage is not to induce additional sparsity but to preserve model performance under pruning, the training objective only includes the distillation loss that aligns the model performance:

$$\mathcal{L}_{2nd} = \|\mathbf{H}_{\text{full}} - \mathbf{H}_{\text{scaled}}\|_2^2.$$

Figure 3 illustrates the roles of the scaling factor α and binary mask β in the double-stage training process. **Both stages are crucial** for learning an effective pruning mask. Directly applying Top-K to α results in suboptimal performance as validated in Appendix E (relying on test-time to decide pruning ratio misaligns with **R1**), and might be unfavorable for GPU efficiency (violates **R2**). Conversely, skipping stage 1 and directly optimizing a binary mask is difficult, especially at high pruning ratios. In practice, we observe that such training approach often fails to converge.

3.3 Deployment

During deployment, after the prefilling stage, the key cache \mathbf{K} is pruned and partitioned into two parts, $\mathbf{K} = [\mathbf{K}_{\text{s+1}}; \mathbf{K}_{\text{prun}}]$, where $\mathbf{K}_{\text{s+1}}$ includes the full K cache for attention sink and local windows, and \mathbf{K}_{prun} is the cache pruned based on the binary mask β . As illustrated in Figure 2, during decoding, the oldest key vector in the local window is taken from $\mathbf{K}_{\text{s+1}}$, pruned using β , and appended to \mathbf{K}_{prun} ; the key of the newly generated token is then appended to $\mathbf{K}_{\text{s+1}}$. To reduce overhead, we perform this update every 32 tokens instead of every step. Attention computation at each decoding step is:

$$\mathbf{A} = \text{softmax}(\mathbf{q}\mathbf{K}_{\text{s+1}}^T + \mathbf{q}_{\text{prun}}\mathbf{K}_{\text{prun}}^T)\mathbf{V},$$

where \mathbf{q}_{prun} represents the query vector pruned in each attention head according to β . Notably, some

attention heads have all channels pruned, making \mathbf{K}_{prun} and its associated value cache unnecessary. For these heads, the attention calculation simplifies to:

$$\mathbf{A} = \text{softmax}(\mathbf{q}\mathbf{K}_{s+1}^T)\mathbf{V}_{s+1},$$

where \mathbf{V}_{s+1} corresponds exclusively to the sink and local window tokens, allowing additional memory savings in the V cache.

4 Experiments

4.1 Settings

LLMs. We experiment with two recent and widely-used LLMs, Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen2.5-7B-Instruct (Qwen et al., 2025), both supporting a 128K context window.

Baselines. We mainly compare with ThinK (Xu et al., 2025), which uses a dynamic, query-driven strategy to prune unimportant channels in each attention head. It estimates channel importance via query-key multiplication at the end of prefilling. To our knowledge, it is the only method targeting pruning along the K channel dimension as we do. We also compare our channel selection method with Double Sparsity (Yang et al., 2024), with results discussed in Appendix B.

Benchmarks. We evaluate our method on three long-context benchmarks: (1) LongBench (Bai et al., 2024), a realistic and diverse benchmark where we evaluate on all tasks, including QA, few-shot learning, code generation, summarization, and counting; (2) RULER (Hsieh et al., 2024), a challenging benchmark with tasks such as hay-in-the-stack, KV retrieval, variable tracking, and QA, evaluated at input lengths from 4K to 128K with 200 samples per task; (3) GSM-Infinite (Zhou et al., 2025), a benchmark designed to assess mathematical reasoning under long-generation settings, where we evaluate on Medium and Hard tasks with context lengths of 8K, 16K and 32K.

Implementation Details. We apply the training method to Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct. Since Qwen2.5 adopts Yarn (Peng et al., 2024) for context lengths beyond 32K, we train separate pruning masks for $\leq 32\text{K}$ and $> 32\text{K}$ contexts. Training token lengths are uniformly sampled from 16K–96K for Llama, 4K–32K for original Qwen, and 64K–96K for Qwen with Yarn. The double-stage training consists of 2000 steps in the first stage and 200 steps in the second, with the latter using half the learning rate for stability. Specifically, we use learning rates of 0.02/0.01 for Llama

and 0.04/0.02 for Qwen. We set $\lambda = 0.06$ for both models, use a local window size of 1024, an attention sink size of 128, and optimize scaling factors using the Adam optimizer (Kingma and Ba, 2015).

Methods	4K	8K	16K	32K	64K	128K	Avg.
<i>Llama-3.1-8B</i>	95.1	93.1	90.2	86.0	84.4	73.5	87.1
ThinK, 60%	89.5	81.7	79.7	81.8	80.6	69.8	80.5
ThinK, 70%	58.3	39.2	37.5	36.4	35.3	40.0	41.1
LeanK, 70%	95.3	93.4	88.8	85.8	84.1	73.2	86.8
<i>Qwen2.5-7B</i>	94.1	91.7	90.8	89.1	80.7	63.6	85.0
ThinK, 70%	86.7	80.6	80.5	81.7	67.1	60.4	62.8
LeanK, 70%	93.6	89.8	90.6	88.5	78.0	64.5	84.2

Table 3: Performance of different methods and models on RULER. Detailed results are in Appendix G.

4.2 Effectiveness of LeanK

Table 2, Table 3 and Table 4 present performance results on three benchmarks, respectively. LeanK’s effectiveness is demonstrated in four aspects:

High Compression Ratio with Minimal Performance Loss. Results show that LeanK maintains near-lossless performance under a 70% compression ratio, while ThinK experiences significant degradation. On RULER, ThinK at a 70% pruning ratio shows performance drops of 52.8% on Llama and 26.1% on Qwen; even at 60% pruning, ThinK still suffers an 8.0% drop on Llama. In contrast, LeanK only shows minimal drops of 0.3% and 0.1% on the two models under the same 70% compression setting. On LongBench, ThinK drops by 5.7% and 4.8% under 70% compression, while LeanK maintains strong performance with only 0.4% and 3.1% degradation.

Strong Staticity in K Channels. On RULER evaluation tasks, which features diverse input lengths ranging from 4K to 128K, LeanK demonstrates consistently strong performance using the learned static channel patterns. This indicates that the importance of K cache channels in pretrained LLMs exhibits a largely static nature.

Methods	Medium			Hard			Avg.
	8K	16K	32K	8K	16K	32K	
<i>Llama-3.1-8B</i>	0.56	0.35	0.30	1.07	0.65	0.46	0.56
ThinK, 70%	0.26	0.17	0.15	0.28	0.14	0.12	0.19
LeanK, 70%	0.70	0.49	0.40	1.14	0.65	0.50	0.65
<i>Qwen2.5-7B</i>	1.08	0.92	1.06	1.01	0.93	0.88	0.98
ThinK, 70%	0.96	0.75	0.75	0.76	0.69	0.64	0.76
LeanK, 70%	1.06	0.86	0.76	1.03	0.85	0.74	0.88

Table 4: Performance of different methods and models on GSM-Infinite. For each subset, we calculate AUC score for op=2,4,6,8,10,12 with 256 samples for each op. Detailed results are in Appendix H.

Methods	Avg.	2Wiki	GovRep	Hotpot	LCC	MNews	MF-en	PsgCnt	PsgRtr	Qasper	Rbench	Samsun	TREC	TrvQA	QMSum
<i>Llama-3.1-8B-Instruct</i>	52.4	48.5	34.5	58.1	63.4	27.1	56.6	9.3	99.5	44.7	56.7	43.8	73.0	91.7	27.1
ThinK 60%	51.5	48.1	30.3	57.9	63.1	25.3	57.6	9.9	99.5	44.9	55.5	41.1	72.0	90.1	25.2
ThinK 70%	49.4	47.1	27.0	56.9	62.7	25.9	50.5	9.3	99.5	35.9	54.9	42.2	65.0	90.5	23.7
LeanK 70%	52.2	48.3	33.3	58.1	63.2	26.5	56.3	9.6	99.5	46.5	57.0	42.7	73.0	92.1	25.1
<i>Qwen2.5-7B-Instruct</i>	51.7	47.1	31.8	57.7	60.6	23.9	52.6	8.5	100.0	43.6	66.8	46.2	71.5	89.3	23.8
ThinK 70%	49.2	44.9	28.7	54.2	59.8	23.1	49.9	8.5	99.0	37.2	64.5	44.0	65.5	87.5	22.7
LeanK 70%	50.1	46.7	30.7	57.7	59.1	23.9	52.1	9.0	100.0	42.1	65.8	43.9	71.5	87.9	22.8

Table 2: Performance of different methods and models on LongBench.

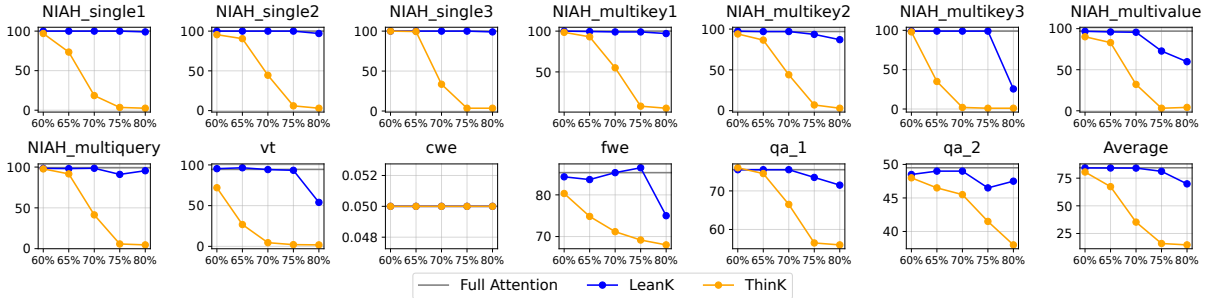


Figure 4: Comparison of performance on RULER 64K under different pruning ratios.

Strong Generalizability. As illustrated in Table 4, LeanK’s learned channel patterns generalize effectively to long-generation reasoning tasks, which are typically sensitive to compression (Li et al., 2025). Under a 70% pruning ratio, ThinK suffers performance drops of 67% on Llama and 20% on Qwen. LeanK outperforms ThinK on both models, and even improves the performance of Llama by 13%. Given the growing importance of reasoning tasks, LeanK offers a promising approach to enhancing efficiency of model inference.

Resilience Under Extreme Sparsity. Furthermore, we applied our method to Llama-3.1-8B-Instruct model across varying compression ratios and tested performance on RULER 64K tasks. As illustrated in Figure 4, LeanK consistently surpasses ThinK under diverse pruning ratios, while also maintaining higher performance under aggressive pruning settings.

4.3 Efficiency of LeanK

Kernel Design. We implement custom decoding kernel to accelerate attention computation using TileLang (Wang et al., 2025). After loading the model weights, we group attention heads in each layer based on their remained channel count, and reorder the Q, K, V, O projection weights accordingly. For each group, we store a separate pruned K cache, along with the full K cache for sink and local tokens. At each decoding step, a fused decoding kernel is launched, which directly reads from the grouped K cache, performs FlashAttention, and outputs the final attention results. By reading sig-

nificantly fewer K channels, the kernel reduces memory bandwidth usage and accelerates decoding. This approach achieves 1.3× and 1.6× average speedup in attention computation on Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct, respectively, as shown in Figure 5 and Figure 10.

GPU Memory Reduction. Under a 70% pruning ratio, LeanK achieves approximately 70% GPU memory reduction in the K cache for long-context inputs. Moreover, when all channels of a head’s K cache are pruned, the corresponding V cache can also be safely removed (§3.3). This occurs in approximately 18% heads in Llama-3.1-8B-Instruct and 16% in Qwen2.5-7B-Instruct. We evaluate the memory reduction on a single 80GB A100 GPU with an input length of 4096 and an output length of 1024. As shown in Figure 6, LeanK enables a 20% larger batch size and saves approximately 10GB of memory when the batch size is 64.

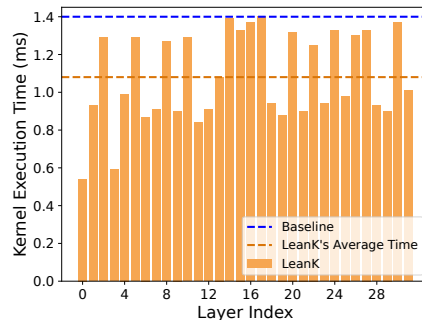


Figure 5: Kernel execution time of each layer on Llama-3.1-8B-Instruct. LeanK uses 70% pruning ratio. Both Baseline and LeanK use Tilelang implementation.

Method	K Ratio	Acc
Original	-	84.38
DuoAttn	50%	83.94
DuoAttn + LeanK	80%	83.53

Table 5: LeanK on top of DuoAttn. Performance on RULER 64K.

Method	K Ratio	Acc
Original	-	84.38
Quest	-	72.41
Quest + LeanK	70%	75.14

Table 6: LeanK on top of Quest. Performance on RULER 64K.

Method	K Ratio	Acc
Original	-	86.03
KIVI	-	84.67
KIVI + LeanK	70%	84.16

Table 7: LeanK on top of KIVI. Performance on RULER 32K.

Increased End-to-end Throughput. Combining our efficient decoding kernel, enlarged batch size and a more efficient KV cache management strategy, LeanK achieves a 1.2 \times increase in end-to-end throughput on Llama-3.1-8B-Instruct, as shown in Table 8².

	Batch size	Gen length	Gen Time	Throughput
Baseline	52	128	47.27 s	141 tokens/s
LeanK	64	128	47.62 s	172 tokens/s

Table 8: End-to-end generation time and throughput. Tested with Huggingface transformers framework, with input sequence length 4096.

4.4 Orthogonality with Other Methods

We emphasize that K channel sparsity is orthogonal to existing approaches. LeanK can be combined with other KV cache optimization methods for further acceleration, especially in resource-constrained environments.

DuoAttention. DuoAttention (Xiao et al., 2024a) is a KV cache eviction method that categorizes heads into Streaming heads and Retrieval heads, evicting the KV cache of the former. LeanK can be applied to the remaining Retrieval Heads, boosting KV cache memory reduction from 50% to 65%, without performance degradation (Table 5).

Quest: Quest (Tang et al., 2024) is a KV cache selective reading method that identifies and loads only critical pages during decoding. LeanK can be applied to both the critical page selection and loading phases, reducing memory reads by 70% and improving model accuracy (Table 6).

KIVI: KIVI (Liu et al., 2024b) is a KV cache quantization method. LeanK can be applied beforehand to prune unimportant KV entries, and then KIVI quantizes remaining cache. This combination improves compression ratio from 5.3 \times to 9.7 \times , using 2-bit quantization for both K and V (Table 7)³.

²Huggingface Baseline uses a batch size of 52 since larger batch sizes would lead to OOM.

³When our method is not applied and only KIVI is used, we encounter an OOM issue with the official implementation at 64K input length, so we conduct experiments at 32K instead.

4.5 Ablation Study

LeanK involves two design decisions for pruning K cache channels: (1) employing a learned pruning mask rather than relying on norm-based selection, and (2) allocating pruning budgets globally across all heads, resulting in varying numbers of retained channels per head. In contrast, Think utilizes a uniform budget across heads and a dynamic, norm-based mask. To clearly assess these design choices, we apply LeanK’s per-head learned budget but replace its learned mask with the dynamic norm-based mask, with results presented in Table 9. We observe the per-head budget alone significantly boosts accuracy from 35.29 to 76.59, underscoring the advantage of adaptive budget allocation across heads. Furthermore, replacing the dynamic norm-based mask with LeanK’s learned mask further boosts accuracy to 84.10, demonstrating the effectiveness of the learned sparsity.

Method	Ratio	Acc
Original	-	84.38
Uniform Budget, Dynamic Mask (Think)	70%	35.29
Per-head Budget, Dynamic Mask	70%	76.59
Per-head Budget, Learned Mask (LeanK)	70%	84.10

Table 9: Evaluation results on Llama-3.1-8B-Instruct, with accuracy measured on RULER 64K.

5 Analysis

In this section, we analyze the learned channel importance distribution to gain deeper insight into model’s behavior.

5.1 Frequency and Channel Importance

RoPE assigns specific frequencies to every pair of channels in the K matrix. We introduce the term "channel pair index" to indicate each pair of channels, where smaller indices correspond to higher frequencies and larger indices to lower frequencies. Both Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct contain 64 channel pairs per head. Figures 7 and 9 illustrate the retained ratio of channels relative to their channel pair indices.

We have two observations: (1) Channel pairs with lower frequencies generally exhibit higher

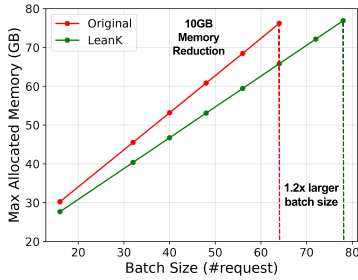


Figure 6: Batch Size and Memory. LeanK enables a 20% larger batch size, saving 10GB memory.

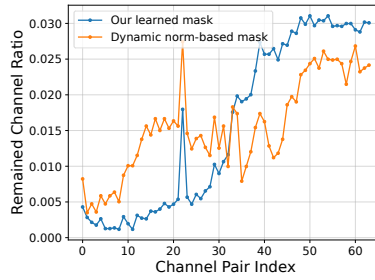


Figure 7: Channel Pair Index and Remained Channel Ratio on Llama-3.1-8B-Instruct.

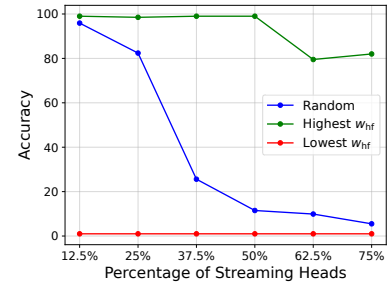


Figure 8: Converting heads with highest or lowest w_{hf} values into streaming heads and performance.

importance, aligning with previous research indicating that semantic information crucial for long-context understanding is encoded mainly in low-frequency channels (Barbero et al., 2025). Our pruning method retains significantly fewer high-frequency channels than dynamic norm-based methods. (2) However, exceptions exist – channel pair 22 in Llama and channel pair 31 in Qwen, despite being high-frequency, show considerable importance. A further investigation into their specific functions remains for future work.

5.2 High Frequency Ratio of Each Head

Inspired by Retrieval Heads identification methods such as Wu et al. (2025) and the RoPE frequency analysis in Section 5.1, we want to investigate the relationship between retrieval ability of different heads and the ratio of their high frequency components w_{hf} , defined as:

$$w_{hf} = \frac{\|q_{[:high]} K_{[:high]}^T\|_2}{\|q K^T\|_2}$$

where $q_{[:high]}$ and $K_{[:high]}$ represent the higher-frequency half of channel dimensions.⁴ Since high frequency channels mainly contain less informative noises, heads with higher w_{hf} are more likely to be influenced by such noises and might be less crucial for capturing semantic information. We compute w_{hf} using a single input sequence from RULER NIAH_multikey3 task for Llama-3.1-8B-Instruct, and examine the importance of heads with different w_{hf} values through head pruning.

We convert heads with highest or lowest w_{hf} into streaming heads and evaluate performance on NIAH_multikey3 task. For comparison, we randomly select the same number of heads to convert and report average performance over four trials. Results in Figure 8 suggest that heads with low w_{hf}

⁴We use half (high=32) as the boundary between high and low frequency components for simplicity.

are crucial for long-context understanding while heads with high w_{hf} could be pruned with minimal impact. Results in Appendix D show that it also generalizes well on other tasks. This opens the opportunity of an effective, training-free head pruning strategy with minimal calibration cost.

6 Related Works

KV Cache Optimization. Large KV caches in long-context LLMs lead to significant GPU memory overhead and increasing output latency. To mitigate this, various optimization methods have been proposed. Eviction-based methods discard KV entries of less important tokens, such as H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024b), which rely on attention scores, or DuoAttention (Xiao et al., 2024a), which prunes KV cache at the head level. Selection-based methods like SparQ (Ribar et al., 2024), Quest (Tang et al., 2024), and Double Sparsity (Yang et al., 2024) retain the full KV cache in memory but selectively read relevant entries to reduce memory bandwidth usage. Quantization methods such as KIVI (Liu et al., 2024b) compress KV caches by reducing numerical precision. Our method is orthogonal to these eviction, selection, and quantization approaches and can be combined with them for further gains (see §4.4).

Structured Pruning. Traditional structured pruning methods for LLMs target hidden states (Ma et al., 2023), layers (Gromov et al., 2025), and expert components (Lu et al., 2024), but they are limited to small task ranges and often suffer from poor performance. Other pruning methods primarily target weights and activations (Frantar and Alistarh, 2023; Sun et al., 2024). In contrast, our approach focuses on pruning the KV cache and attention computations, incorporating the unstructured attention sink and local window mechanism into algorithm design, which serves as a valuable complement to existing weight pruning methods for LLMs.

7 Conclusion

We propose LeanK, a learning-based method for pruning the channel dimension of K cache to enable efficient LLM decoding. LeanK employs a double-stage optimization process to learn a static pruning mask. Experiments demonstrate that LeanK reduces GPU memory usage by up to 70% for the K cache and 16%–18% for the V cache, achieving a $1.45\times$ speedup during inference, while preserving model accuracy.

Limitations

We observe significant redundancy along the channel dimension of pretrained LLMs. Improving positional embeddings and conducting more thorough pretraining over this dimension may enhance the model’s long-context processing ability and reduce memory consumption. We leave this exploration for future work.

Acknowledgement

This work was supported in part by National Key Research and Development Program of China under Grant No. 2020YFA0804503 and National Natural Science Foundation of China under Grant No. 62272264.

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. 2024. **LongBench: A bilingual, multi-task benchmark for long context understanding**. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.
- Federico Barbero, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Velickovic. 2025. **Round and round we go! what makes rotary positional encodings useful?** In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Léon Bottou, Zhihao Jia, and Beidi Chen. 2025. **Magicpig: LSH sampling for efficient LLM generation**. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Elias Frantar and Dan Alistarh. 2023. **Sparsegpt: Massive language models can be accurately pruned in one-shot**. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. **The llama 3 herd of models**. *Preprint*, arXiv:2407.21783.
- Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. 2025. **The unreasonable ineffectiveness of the deeper layers**. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Xiangyu Hong, Che Jiang, Biqing Qi, Fandong Meng, Mo Yu, Bowen Zhou, and Jie Zhou. 2024. **On the token distance modeling ability of higher RoPE attention dimension**. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5877–5888, Miami, Florida, USA. Association for Computational Linguistics.
- Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. **Ruler: What’s the real context size of your long-context language models?** *Preprint*, arXiv:2404.06654.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. **SWE-bench: Can language models resolve real-world github issues?** In *The Twelfth International Conference on Learning Representations*.
- Diederik P. Kingma and Jimmy Ba. 2015. **Adam: A method for stochastic optimization**. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. 2024a. **LooGLE: Can long-context language models understand long contexts?** In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16304–16333, Bangkok, Thailand. Association for Computational Linguistics.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024b. **Snapkv: Llm knows what you are looking for before generation**. *Advances in Neural Information Processing Systems*, 37:22947–22970.

- Zhen Li, Yupeng Su, Runming Yang, Congkai Xie, Zheng Wang, Zhongwei Xie, Ngai Wong, and Hongxia Yang. 2025. Quantization meets reasoning: Exploring llm low-bit quantization degradation for mathematical reasoning. *arXiv preprint arXiv:2501.03035*.
- Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. 2024a. [Retrievalattention: Accelerating long-context llm inference via vector retrieval](#). *Preprint*, arXiv:2409.10516.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. [KIVI: A tuning-free asymmetric 2bit quantization for KV cache](#). In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.
- Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. 2024. [Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 6159–6172. Association for Computational Linguistics.
- Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.
- Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. [Yarn: Efficient context window extension of large language models](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, and 25 others. 2025. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.
- Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. 2024. [Sparq attention: Bandwidth-efficient LLM inference](#). In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.
- Philip Sedgwick. 2012. Pearson’s correlation coefficient. *Bmj*, 345.
- Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. [A simple and effective pruning approach for large language models](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. [QUEST: query-aware sparsity for efficient long-context LLM inference](#). In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.
- Lei Wang, Yu Cheng, Yining Shi, Zhengju Tang, Zhiwen Mo, Wenhao Xie, Lingxiao Ma, Yuqing Xia, Jilong Xue, Fan Yang, and 1 others. 2025. [Tilelang: A composable tiled programming model for ai systems](#). *arXiv preprint arXiv:2504.17577*.
- Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. 2025. [Retrieval head mechanistically explains long-context factuality](#). In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024a. [Duoattention: Efficient long-context llm inference with retrieval and streaming heads](#). *Preprint*, arXiv:2410.10819.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024b. [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.
- Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. 2025. [Think: Thinner key cache by query-driven pruning](#). In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Shuo Yang, Ying Sheng, Joseph E. Gonzalez, Ion Stoica, and Lianmin Zheng. 2024. [Post-training sparse attention with double sparsity](#). *Preprint*, arXiv:2408.07092.
- Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. 2024. [A survey on recent advances in llm-based multi-turn dialogue systems](#). *arXiv preprint arXiv:2402.18013*.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. [H2o: Heavy-hitter oracle for efficient generative inference of large language models](#). *Advances in Neural Information Processing Systems*, 36:34661–34710.
- Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. 2025. [Gsm-infinite: How do your llms behave over infinitely increasing context length and reasoning complexity?](#) *Preprint*, arXiv:2502.05252.

A Quantifying K Channel Staticity

We aim to quantify the *staticity* of K channels across various tasks and input lengths. For a specific channel i within an attention head, given an input sample, we measure the following norm ratio:

$$r_i = \frac{\|Q_{[i]}K_{[i]}^T\|_2}{\|QK^T\|_2},$$

where Q represents the query matrix corresponding to an observation window located at the last part of the input, K is the complete key matrix, and $Q_{[i]}$, $K_{[i]}$ denote the i -th channel of Q and K , respectively. The ratio r_i captures the relative importance of channel i within an attention head.

We aggregate these channel-wise ratios r_i into a single vector r of shape $(L \times n \times d)$, where L , n , and d correspond to the number of layers, attention heads per layer, and channels per head, respectively.

To evaluate staticity across tasks, we measure r separately for different tasks and compute the Pearson correlation coefficient between these vectors. Specifically, for tasks 1 and 2, we obtain vectors $r^{(1)}$ and $r^{(2)}$. If channel importance significantly differs between tasks, we would expect a low correlation between $r^{(1)}$ and $r^{(2)}$. However, as illustrated in Figure 1, Pearson correlations between different RULER tasks consistently remain close to 1. This indicates that the high-importance channels in one task generally remain highly important in another, highlighting a static sparsity pattern in the K channels. Similar results are observed across varying input lengths, reinforcing the notion of channel staticity.

B Comparison with Double Sparsity

Double Sparsity (Yang et al., 2024) proposes a KV selective-reading strategy based on offline identification of outlier channel dimensions. Specifically, it computes the norm of the QK product to select high-norm (outlier) channels, which are then used during inference to retrieve critical tokens.

In this section, we compare our learned channel importance scores with the outlier channel selection criterion used in Double Sparsity to evaluate whether their method can be used for channel pruning. As shown in Table 10, we find that Double Sparsity’s method exhibits several limitations that lead to suboptimal performance:

1. The outlier channel selection process is not

inherently designed for channel-wise pruning, and may lack careful design considerations.

2. During offline norm collection, the method splits QK product on context dimension into chunks of smaller sizes, which overlooks the unstructured composition of QK channel dimension (namely, the attention sink and local window) and may lead to inaccurate norm calculation.
3. As discussed in Section 2, relying solely on norm (magnitude) is an insufficient proxy for estimating channel importance. This limitation may restrict the pruning ratio of local norm-based pruning methods.

	Method	Ratio	Acc
Llama-3.1-8B-Instruct	Original	-	84.4
	DS	60%	29.9
	DS	70%	14.0
	LeanK	70%	84.1

Table 10: Comparison with Double Sparsity. Methods are tested on RULER 64K, with 200 samples taken from each subtask. Detailed results are in Table 19.

C Channel Frequency Analysis for Qwen2.5-7B-Instruct

We visualize the learned mask for Qwen2.5-7B-Instruct in Figure 9. Similar as Section 5.1, we observed that LeanK’s learned pruning pattern preserves more low frequency channels compared with ThinK’s norm-based method, which might contribute to LeanK’s effectiveness.

Furthermore, a similar outlier channel with channel pair index 31 appears to have relatively high norm and large impact on model performance.

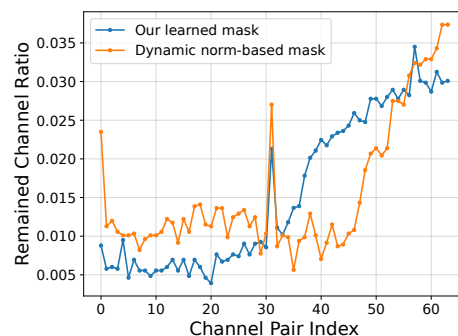


Figure 9: Remained Ratio and Channel Pair Index on Qwen2.5-7B-Instruct.

Method	Ratio	Llama-3.1-8B-Instruct													
		niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	Avg.
Original	-	100.0	100.0	100.0	100.0	97.0	99.0	97.0	99.1	94.5	0.1	85.3	75.5	49.5	84.4
Highest w_{hf}	25%	99.5	99.5	100.0	100.0	96.5	98.5	96.6	98.6	93.5	0.15	85.5	76.5	49.5	84.2
Highest w_{hf}	50%	99.5	100.0	100.0	100.0	96.0	99.0	93.8	98.9	92.2	0.4	83.0	75.5	48.5	83.6
Lowest w_{hf}	25%	2.5	3.0	3.5	5.5	3.0	1.0	4.5	5.0	1.5	0.1	67.2	53.0	40.0	14.6
Random	25%	99.6	98.2	99.2	99.5	93.6	82.4	71.7	93.8	70.1	0.1	80.1	73.9	46.8	77.6
DuoAttn	50%	100.0	100.0	100.0	99.0	96.5	99.0	95.8	99.3	91.2	0.05	84.5	76.0	50.0	84.0

Table 11: Head pruning and classification based on **high frequency ratio** of each head.

D Head Pruning based on High Frequency Ratio

In Section 5.2, we define the *High Frequency Ratio* of each attention head as:

$$w_{hf} = \frac{\|q_{[:high]}K_{[:high]}^T\|_2}{\|qK^T\|_2}$$

We compute w_{hf} using a single 64K-token input from the RULER VT task. We observe that this score effectively distinguishes retrieval heads. To evaluate its utility, we modify Llama-3.1-8B-Instruct by converting a subset of heads into streaming heads based on their w_{hf} values. Specifically, we convert heads with the highest or lowest w_{hf} values and measure end-to-end performance on RULER 64K. For comparison, we also convert an equal number of randomly selected heads into streaming heads, repeating the experiment four times and reporting the average result (Table 11).

Our findings show that converting heads with the lowest w_{hf} severely degrades performance, while converting the top 50% highest w_{hf} heads has minimal impact. This matches the performance of DuoAttention (Xiao et al., 2024a), which requires an additional training phase. In contrast, our method achieves comparable results using only a single input sequence for calibration.

E Necessity of Training Stage 2

We verified that applying Top-K on scaling factor α trained from the training stage 1 leads to sub-optimal result on model performance, results are shown in Table 12, suggesting that the second training stage is necessary for pruning under predefined sparsity ratios.

The primary reason for the performance disparity lies in the misalignment between the scaling operation in Stage1 and the ultimate objective of channel masking for deployment. For instance, some channels may be robust to scaling, but entirely masking them out can lead to severe performance degradation. Stage2 could help avoid these channels from being pruned.

Furthermore, Direct Top-K on α violates alignment requirements (aligning to multiples of 16 or 32) and might be inefficient for hardware execution.

	Method	Ratio	Acc
Qwen2.5-7B-Instruct	Original	-	80.7
	w/o 2nd stage	70%	70.7
	w/ 2nd stage	70%	78.0

Table 12: Necessity of 2nd stage of training. Methods are tested on RULER 64K, with 200 samples taken from each subtask. Using only the first stage with a 70% pruning ratio yields a RULER 64K accuracy of just 70.73. With the second stage added, accuracy improves to 77.97 on Qwen2.5-7B-Instruct. Detailed results are in Table 22.

F Kernel Benchmarking on Qwen

LeanK could achieve a 1.6x speedup on attention computation on Qwen2.5-7B-Instruct. Execution time of each layer is shown in Figure 10.

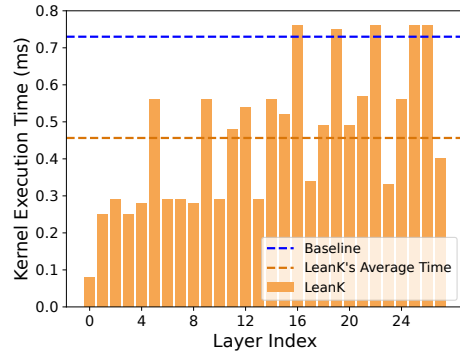


Figure 10: Kernel execution time of each layer on Qwen2.5-7B-Instruct. LeanK uses 70% pruning ratio.

G Full Evaluation Results on RULER

Full evaluation results on RULER (Hsieh et al., 2024) are shown in Table 13. We evaluate all tasks in RULER across input lengths ranging from 4K to 128K. For Qwen testing on input lengths larger than 32K, we apply Yarn extrapolation with a factor of 4 as suggested by its official documentation.

		Llama-3.1-8B-Instruct														Avg.
	Method	Ratio	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	
4K	Original	-	100.0	100.0	100.0	99.5	99.0	100.0	99.8	99.6	99.4	99.6	93.7	85.0	61.0	95.1
	ThinK	60%	93.0	88.0	99.5	88.5	97.0	99.0	90.0	92.0	76.9	99.1	93.3	86.5	61.0	89.5
	ThinK	70%	31.0	49.5	43.0	66.0	80.0	35.5	49.3	53.9	23.6	97.4	84.5	86.5	58.0	58.3
	Ours	70%	100.0	100.0	100.0	99.5	99.5	99.0	99.8	99.6	99.3	99.6	94.3	86.5	61.5	95.3
8K	Original	-	100.0	100.0	100.0	99.5	99.5	99.5	99.8	99.8	98.9	94.4	84.5	78.0	56.0	93.1
	ThinK	60%	75.0	88.5	99.0	87.5	99.0	98.5	87.1	93.0	76.2	42.3	84.5	76.0	56.5	81.7
	ThinK	70%	12.5	33.0	19.5	64.0	66.0	12.5	29.3	46.9	15.5	20.4	68.7	68.5	53.0	39.2
	Ours	70%	100.0	100.0	100.0	100.0	100.0	99.5	99.3	99.6	99.2	94.7	87.0	78.5	56.0	93.4
16K	Original	-	100.0	100.0	100.0	99.5	100.0	99.5	99.4	98.9	99.3	53.3	90.7	79.5	53.0	90.2
	ThinK	60%	99.0	88.5	99.0	95.0	99.5	98.5	92.4	94.8	66.8	0.9	96.7	78.0	51.5	79.7
	ThinK	70%	9.5	28.5	23.0	60.5	65.5	5.0	31.5	46.8	9.6	0.4	92.3	66.5	48.5	37.1
	Ours	70%	100.0	100.0	100.0	99.5	100.0	99.5	99.0	98.1	98.6	36.0	91.7	79.0	53.0	88.8
32K	Original	-	100.0	100.0	100.0	99.0	99.0	100.0	98.9	99.4	97.6	2.7	93.3	76.0	51.5	86.0
	ThinK	60%	88.0	94.0	100.0	97.5	98.5	100.0	95.3	98.5	70.0	0.0	96.2	75.5	49.5	81.8
	ThinK	70%	9.5	32.5	21.0	58.5	54.5	2.0	37.9	48.6	9.2	0.0	91.7	62.5	45.0	36.4
	Ours	70%	100.0	100.0	100.0	99.5	99.5	100.0	98.4	98.5	97.0	1.2	94.2	76.5	51.0	85.8
64K	Original	-	100.0	100.0	100.0	97.0	97.0	99.0	97.0	99.1	94.5	0.1	85.3	75.5	49.5	84.4
	ThinK	60%	97.0	95.5	100.0	98.5	94.0	98.0	90.1	97.8	72.0	0.1	80.3	76.0	48.0	80.6
	ThinK	70%	18.5	44.5	33.5	55.0	44.0	2.0	32.1	41.4	4.5	0.1	71.2	66.5	45.5	35.3
	Ours	70%	100.0	100.0	100.0	99.0	97.0	99.0	95.6	98.5	94.3	0.1	85.3	75.5	49.0	84.1
128K	Original	-	100.0	99.0	100.0	97.0	75.0	53.5	93.1	97.3	54.5	0.2	76.3	71.5	37.5	73.5
	ThinK	60%	98.5	97.0	99.5	97.0	71.0	26.0	91.9	97.4	40.0	0.3	80.2	70.0	38.5	69.8
	ThinK	70%	29.0	67.0	28.9	75.5	40.0	0.0	50.9	52.9	8.3	0.5	73.2	59.0	35.0	40.0
	Ours	70%	100.0	99.0	100.0	96.5	75.0	51.5	88.1	96.6	61.6	0.4	74.8	70.5	38.0	73.2
		Qwen2.5-7B-Instruct														Avg.
	Method	Ratio	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	
4K	Original	-	100.0	100.0	100.0	100.0	100.0	100.0	94.6	100.0	100.0	99.7	84.2	84.5	60.0	94.1
	ThinK	70%	97.0	94.0	98.0	96.0	95.0	78.0	82.4	94.4	80.5	98.6	73.3	84.0	55.5	86.7
	Ours	70%	100.0	100.0	100.0	100.0	100.0	100.0	100.0	93.8	100.0	99.7	99.4	80.0	84.0	60.5
8K	Original	-	100.0	100.0	100.0	100.0	100.0	99.0	89.8	99.9	100.0	94.0	88.0	69.5	52.0	91.7
	ThinK	70%	98.5	90.0	98.0	93.0	93.5	60.5	72.1	90.5	69.3	86.3	83.7	61.5	50.5	80.6
	Ours	70%	100.0	100.0	100.0	100.0	100.0	98.0	88.0	99.6	99.1	90.6	89.5	67.5	53.0	89.8
16K	Original	-	100.0	100.0	100.0	99.5	100.0	92.5	95.1	99.9	98.1	85.4	92.0	63.5	54.0	90.8
	ThinK	70%	97.5	98.0	97.5	93.0	94.0	59.5	77.9	89.4	69.1	66.0	91.7	59.5	53.5	80.5
	Ours	70%	100.0	100.0	100.0	99.5	99.5	93.0	92.9	99.8	96.4	84.1	94.7	64.0	54.0	90.6
32K	Original	-	100.0	100.0	99.5	99.0	98.5	94.0	91.4	98.9	96.9	67.7	87.8	68.0	57.0	89.1
	ThinK	70%	99.5	99.0	98.0	98.0	94.0	67.5	70.9	91.4	75.7	57.1	91.0	66.0	53.5	81.7
	Ours	70%	100.0	100.0	100.0	99.0	97.5	93.5	85.3	98.9	96.0	67.8	90.5	65.5	56.0	88.5
64K	Original	-	100.0	98.0	98.0	95.5	82.5	47.5	82.8	97.4	95.3	55.4	82.5	69.5	44.5	80.7
	ThinK	70%	95.0	91.5	95.0	85.0	63.0	24.5	56.1	79.4	65.7	35.7	74.0	61.5	46.0	67.1
	Ours	70%	100.0	98.0	100.0	96.5	83.0	48.0	73.1	95.4	81.1	47.1	78.0	66.5	47.0	78.0
128K	Original	-	100.0	99.5	66.0	92.5	56.0	8.5	63.1	90.5	81.2	36.5	57.2	43.5	34.5	63.6
	ThinK	70%	97.0	93.5	93.5	91.5	39.5	8.0	55.4	72.8	55.8	36.1	61.3	46.5	34.5	60.4
	Ours	70%	100.0	99.5	96.5	93.0	54.0	17.5	66.6	85.6	60.9	29.0	59.0	43.5	33.0	64.5

Table 13: Performance of Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct on RULER. 200 samples are taken for each task.

H Full Evaluation Results on GSM-Infinite

We evaluate Medium and Hard tasks in GSM-Infinite across input lengths ranging from 8K to 32K. Full results are shown in Table 14.

I Additional Experimental Results

Table 20, Table 21 and Table 23 provide other supplementary results.

Specifically, Table 20 evaluated static norm-based selection method’s performance on RULER, suggesting that a static channel pruning strategy could achieve comparable performance with dynamic methods such as ThinK. The implementation of the method is illustrated in Section 2.

Table 21 verified that LeanK is orthogonal with token pruning (Quest), head pruning (DuoAttention) and quantization (KIVI) methods. For Quest, block size is 64 and token budget is 1024. For DuoAttention, 50% heads are pruned. KIVI are tested with 2 bits for both K and V cache, group size 32 and residual length 128.

Table 23 conducted ablation experiments to justify our design choices, showing that both (1) a more fine-grained head-wise budget allocation and (2) a learning-based channel-wise importance score that does not rely on channel’s magnitude contribute to the effectiveness of our method.

J Choice of Hyperparameter Lambda

The hyperparameter λ is multiplied to the regularization loss (L1 norm of scaling factor α) in training Stage1, with the total loss defined as: $L_{1st} = L_{dist} + \lambda L_{reg}$.

We trained models with varying λ values on Llama-3.1-8B-Instruct and evaluated performance on the RULER 32K benchmark.

	Method	Ratio	Acc
Llama-3.1-8B-Instruct	Original	-	86.0
	$\lambda = 0.04$	70%	84.4
	$\lambda = 0.06$	70%	85.8
	$\lambda = 0.08$	70%	85.5
	$\lambda = 0.10$	70%	85.1

Table 15: RULER 32K performance with different choices of training hyperparameter λ on Llama-3.1-8B-Instruct.

Results in Table 15 show that LeanK is robust to a wide range of λ values (0.04 to 0.10), achieving the best performance around $\lambda = 0.06$. Setting λ too small (e.g., 0.04) under-regularizes α , while excessively large λ values (e.g., 0.10) can lead to less accurate learning during Stage 1, slightly harming the final results.

K Choice of Training Task

LeanK is robust to the choice of training task. Specifically, we trained Qwen2.5-7B-Instruct using DuoAttention[1]’s task and evaluated performance on RULER 16K. Results in Table 16 verified that LeanK could still effectively learn channel importance with different tasks (with training hyperparameters changed accordingly), suggesting that LeanK’s effectiveness is not sensitive to specific training tasks.

	Method	Ratio	Acc
Qwen2.5-7B-Instruct	Original	-	90.8
	Our Task	70%	90.6
	DuoAttn Task	70%	90.2

Table 16: LeanK’s performance on RULER 16K with different choices of training task λ on Qwen2.5-7B-Instruct.

L Comparison with Static Channel Pruning Baseline

We evaluated a static norm-based channel pruning baseline on Llama-3.1-8B-Instruct using the

RULER benchmark. The static mask was generated by averaging channel norm distribution (as described in Section 2.2) across 100 NIAH_multikey3 sequences with 64K context length. Results are shown in Table 17.

Method	Ratio	4K	8K	16K	32K	64K	128K	Avg.
Original	-	95.1	93.1	90.2	86.0	84.4	73.5	87.1
ThinK	70%	58.3	39.2	37.1	36.4	35.3	40.0	39.4
Static	70%	68.7	57.5	54.8	55.9	54.0	55.5	57.7
LeanK	70%	95.3	93.4	88.8	85.8	84.1	73.2	86.8

Table 17: Different method’s performance with Llama-3.1-8B-Instruct on the RULER dataset.

The results show that averaging channel norms across multiple inputs yields more effective pruning than dynamic methods like ThinK. However, there remains a substantial performance gap compared to LeanK (i.e., Learned Pattern > Static Pattern > Dynamic Pattern). This further supports both the static nature of channel importance and the effectiveness of LeanK’s design.

M Analysis of V-Cache Pruning Conditions

We analyzed both the distribution and characteristics of the completely pruned heads:

1) In terms of **Distribution**, these heads are distributed across all layers, with a higher concentration in the shallow layers (e.g., in Qwen, 43% of them appear in the first 5 layers).

2) Speaking of **Characteristics**, We examined the channel frequency distribution of these heads (as described in Section 5.2). These heads exhibit high *high-frequency ratio* w_{hf} and relatively large norms on high-frequency channels. This indicates that these heads mainly rely on local context and patterns for next-token prediction, while contributing less to semantic extraction from the context.

N Comparison with SnapKV

We compared LeanK with SnapKV, a token-level KV cache pruning method, with results shown in 18. Under the same overall KV cache reduction ratio (44% for Llama, 43% for Qwen), LeanK achieves higher average performance. SnapKV underperforms LeanK on complex KV retrieval tasks such as NIAH_multikey3.

		8K Medium							
	Method	Pruning ratio	op=2	op=4	op=6	op=8	op=10	op=12	AUC
Llama-3.1-8B-Instruct	Original	-	0.1429	0.1389	0.1508	0.1349	0.0516	0.0278	0.5615
	ThinK	70%	0.1825	0.0556	0.0675	0.0159	0.0278	0.0079	0.2620
	Ours	70%	0.1746	0.1944	0.2183	0.1389	0.0437	0.0397	0.7024
Qwen2.5-7B-Instruct	Original	-	0.2857	0.3651	0.2619	0.1865	0.0952	0.0556	1.0793
	ThinK	70%	0.4405	0.3532	0.1984	0.1190	0.0437	0.0437	0.9564
	Ours	70%	0.3770	0.3333	0.2222	0.1587	0.1310	0.0476	1.0575
		8K Hard							
	Method	Pruning ratio	op=2	op=4	op=6	op=8	op=10	op=12	AUC
Llama-3.1-8B-Instruct	Original	-	0.4127	0.1865	0.2540	0.1905	0.1706	0.1190	1.0675
	ThinK	70%	0.0833	0.0833	0.0595	0.0198	0.0595	0.0357	0.2816
	Ours	70%	0.4762	0.1944	0.2659	0.2103	0.1706	0.1270	1.1428
Qwen2.5-7B-Instruct	Original	-	0.4643	0.2817	0.2262	0.1032	0.1151	0.1032	1.010
	ThinK	70%	0.3492	0.1944	0.1746	0.0913	0.0913	0.0635	0.7580
	Ours	70%	0.3373	0.3532	0.2381	0.1389	0.0833	0.0952	1.0298
		16K Medium							
	Method	Pruning ratio	op=2	op=4	op=6	op=8	op=10	op=12	AUC
Llama-3.1-8B-Instruct	Original	-	0.0913	0.0675	0.1270	0.0833	0.0159	0.0159	0.3473
	ThinK	70%	0.1389	0.0397	0.0357	0.0159	0.0040	0.0119	0.1707
	Ours	70%	0.1468	0.1071	0.1706	0.0913	0.0317	0.0317	0.4900
Qwen2.5-7B-Instruct	Original	-	0.2738	0.2937	0.2381	0.1429	0.0833	0.0556	0.9227
	ThinK	70%	0.4008	0.2778	0.1429	0.0833	0.0397	0.0119	0.7501
	Ours	70%	0.4365	0.2302	0.1825	0.1468	0.0437	0.0714	0.8572
		16K Hard							
	Method	Pruning ratio	op=2	op=4	op=6	op=8	op=10	op=12	AUC
Llama-3.1-8B-Instruct	Original	-	0.3214	0.0992	0.1429	0.1190	0.1071	0.0476	0.6527
	ThinK	70%	0.1032	0.0159	0.0198	0.0238	0.0159	0.0198	0.1369
	Ours	70%	0.2857	0.1230	0.1349	0.1310	0.0833	0.0754	0.6528
Qwen2.5-7B-Instruct	Original	-	0.4722	0.2619	0.1706	0.1389	0.0873	0.0635	0.9265
	ThinK	70%	0.3770	0.1786	0.1429	0.0754	0.0635	0.0873	0.6926
	Ours	70%	0.4246	0.2579	0.1548	0.0873	0.0913	0.0952	0.8512
		32K Medium							
	Method	Pruning ratio	op=2	op=4	op=6	op=8	op=10	op=12	AUC
Llama-3.1-8B-Instruct	Original	-	0.0873	0.0675	0.1151	0.0675	0.004	0	0.2978
	ThinK	70%	0.0913	0.0437	0.0437	0.0198	0	0	0.1529
	Ours	70%	0.1190	0.0992	0.1429	0.0754	0.0198	0.0079	0.4007
Qwen2.5-7B-Instruct	Original	-	0.3016	0.3730	0.2778	0.1468	0.0794	0.0714	1.0635
	ThinK	70%	0.4206	0.2183	0.1627	0.0913	0.0476	0.0437	0.7520
	Ours	70%	0.3294	0.2381	0.2063	0.0992	0.0397	0.0317	0.7639
		32K Hard							
	Method	Pruning ratio	op=2	op=4	op=6	op=8	op=10	op=12	AUC
Llama-3.1-8B-Instruct	Original	-	0.1984	0.1190	0.1111	0.0675	0.0437	0.0437	0.4624
	ThinK	70%	0.0675	0.0278	0.0119	0.0119	0.0238	0.0278	0.1231
	Ours	70%	0.2579	0.1468	0.0794	0.0714	0.0516	0.0516	0.5040
Qwen2.5-7B-Instruct	Original	-	0.4603	0.2579	0.1468	0.1151	0.1032	0.0556	0.8810
	ThinK	70%	0.3929	0.1429	0.1429	0.0714	0.0635	0.0476	0.6409
	Ours	70%	0.3929	0.1905	0.1587	0.1071	0.0675	0.0437	0.7421

Table 14: Performance of different methods and models on GSM-Infinite, 256 samples are taken for each op. Generation temperature is set to 0.

Model	Method	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	Avg.
Llama-3.1-8B	SnapKV	100	100	100	99.5	99.0	89.5	99.1	99.4	96.9	5.5	92.5	76.5	49.5	85.2
	LeanK	100	100	100	99.5	99.5	100.0	98.4	98.5	97.0	1.2	93.3	76.0	51.5	85.8
Qwen2.5-7B	SnapKV	100	100	100	99.0	93.5	79.0	90.6	98.9	97.8	72.5	88.2	67.0	55.0	87.8
	LeanK	100	100	100	99.0	97.5	93.5	85.3	98.9	96.0	67.8	90.5	65.5	56.0	88.5

Table 18: Performance of LeanK and SnapKV on RULER 32K under the same pruning ratio.

		Llama-3.1-8B-Instruct														
	Method	Ratio	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	Avg.
RULER 64K	Original	-	100.0	100.0	100.0	100.0	97.0	99.0	97.0	99.1	94.5	0.1	85.3	75.5	49.5	84.4
	Double Sparsity	60%	45.5	37.0	37.5	35.0	36.0	1.0	8.9	20.9	14.8	0.1	71.5	66.0	45.0	29.9
	Double Sparsity	70%	3.0	5.5	3.5	9.0	3.0	1.0	3.5	5.6	1.1	0.1	68.8	54.0	37.0	14.0
	Ours	70%	100.0	100.0	100.0	99.0	97.0	99.0	95.6	98.5	94.3	0.1	85.3	75.5	49.0	84.1

Table 19: Full comparison results comparing our method with Double Sparsity.

		Qwen2.5-7B-Instruct														
	Method	Ratio	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	Avg.
	Original	-	100.0	98.0	98.0	95.5	82.5	47.5	82.8	97.4	95.3	55.4	82.5	69.5	44.5	80.7
	ThinK (Dynamic norm)	60%	97.0	95.5	100.0	98.5	94.0	98.0	90.1	97.8	72.0	0.1	80.33	76.0	48.0	80.6
	Static norm	60%	96.5	96.5	100.0	96.0	95.0	99.5	86.4	96.1	91.5	0.1	81.3	74.0	47.5	81.6

Table 20: Static norm-based selection. Tested on RULER 64K, with 200 samples from each subtask.

		Llama-3.1-8B-Instruct														
	Method	Ratio	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	Avg.
64K	Original	-	100.0	100.0	100.0	100.0	97.0	99.0	97.0	99.1	94.5	0.1	85.3	75.5	49.5	84.4
	DuoAttn	50%	100.0	100.0	100.0	99.0	96.5	99.0	95.8	99.3	91.2	0.1	84.5	76.0	50.0	84.0
	DuoAttn + Ours	80%	100.0	100.0	100.0	99.0	96.0	99.0	95.0	99.4	89.8	0.1	83.7	76.0	48.0	83.5
	Quest	-	100.0	98.5	82.0	98.5	84.0	5.0	93.3	95.1	83.2	0.6	83.7	72.5	45.0	72.4
	Quest + Ours	70%	100.0	100.0	99.0	99.0	84.0	11.5	93.0	96.9	87.6	0.05	87.3	74.0	44.5	75.1
		Llama-3.1-8B-Instruct														
	Method	Ratio	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	Avg.
32K	Original	-	100.0	100.0	100.0	100.0	99.0	100.0	98.9	99.4	97.6	2.7	93.3	76.0	51.5	86.0
	KIVI	-	100.0	100.0	100.0	98.5	96.5	95.3	97.5	98.8	91.4	5.8	93.5	75.5	48.0	84.7
	KIVI + Ours	70%	100.0	99.5	100.0	99.0	98.0	91.5	97.1	98.4	92.3	1.0	96.3	73.5	47.5	84.2

Table 21: LeanK applied on top of other pruning methods. Tested on RULER 64K (32K for KIVI to avoid OOM), with 200 samples from each subtask.

		Qwen2.5-7B-Instruct														
	Method	Ratio	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	Avg.
	Original	-	100.0	98.0	98.0	95.5	82.5	47.5	82.8	97.4	95.3	55.4	82.5	69.5	44.5	80.7
	w/o 2nd stage	70%	99.5	86.5	99.0	92.0	75.5	10.5	77.4	91.1	58.3	48.5	72.7	64.5	44.0	70.7
	w/ 2nd stage	70%	100.0	98.0	100.0	96.5	83.0	48.0	73.1	95.4	81.1	47.1	78.0	66.5	47.0	78.0

Table 22: Necessity of the second stage of training. Tested on RULER 64K, with 200 samples from each subtask.

		Llama-3.1-8B-Instruct														
	Method	Ratio	niah_s1	niah_s2	niah_s3	niah_mk1	niah_mk2	niah_mk3	niah_mv	niah_mq	vt	cwe	fwe	qa_1	qa_2	Avg.
	Original	-	100.0	100.0	100.0	100.0	97.0	99.0	97.0	99.1	94.5	0.05	85.3	75.5	49.5	84.4
	Uneven Dynamic	70%	100.0	100.0	98.5	99.0	92.5	92.5	86.9	93.3	39.6	1.0	70.5	74.0	48.0	76.6
	LeanK	70%	100.0	100.0	100.0	99.0	97.0	99.0	95.6	98.5	94.3	0.05	85.3	75.5	49.0	84.1

Table 23: Using our trained mask’s budget allocation and Think’s dynamic norm-based channel selection strategy for pruning. Tested on RULER 64K, with 200 samples from each subtask.