LAPrune: Logits-Aligned Scoring Proxy for KV Pruning via Vector Quantization

Mingyang Yu[1] **Rong-Cheng Tu**[1†] **Yifu Ding**[1] **Hanqing Zhao**[1]
Yongcheng Jing[1] **Xiao Luo**[2] **Dacheng Tao**[1†]
[1]Generative AI Lab, College of Computing and Data Science,
Nanyang Technological University, Singapore, Singapore
[2]University of Wisconsin–Madison

Abstract

Long-context inference in Large Language Models (LLMs) suffers from high latency because attention computation and memory traffic scale linearly during decoding. KV cache pruning reduces this cost by retaining only a subset of cached tokens, but its effectiveness depends on whether the pruning proxy can recover the tokens with the largest exact attention logits under the current query. Existing proxies suffer from ranking misalignment: static heuristics rely on historical statistics and thus ignore the current query, while hashing-based retrieval proxies rank candidates in a discrete metric space that does not coincide with attention's inner product. To avoid these problems, we propose LAPrune, a KV pruning framework that uses additive vector quantization to build an efficient attention-logit-aligned proxy. LAPrune represents each cached key as a sum of learned codewords and decomposes the query-key dot product into lookup-and-add operations over query-codeword inner products. This replaces high-dimensional dot products with lightweight scalar lookups while preserving fidelity to the original attention ranking. LAPrune requires no base-model finetuning and can be integrated into long-context decoding with minimal modification. Across long-context evaluations, LAPrune improves exact-logit Top-k recovery and perplexity over hashing-based pruning proxies, achieving a $2.07\times$ decoding speedup over FlashAttention-2 at 128K context with batch size 1.

1 Introduction

Large language models are increasingly used for complex reasoning [1–3] and long-form generation over long contexts [4–8]. In practice, LLM inference consists of a *prefill* phase that processes the prompt in parallel to build the KV cache, and an autoregressive *decoding* phase that generates one token per forward pass conditioned on the cached KV states. While prefill is largely parallelizable, decoding is the dominant latency bottleneck because it is inherently sequential and repeatedly reads and attends over an ever-growing KV cache [9]. In long-context settings, each new token must compute attention against all cached keys, making the attention cost—and the associated memory traffic—grow linearly with the number of cached tokens [10]. This linear scaling makes efficient long-context decoding a central challenge in practice.

KV cache pruning has emerged as a promising strategy to reduce this cost by retaining only a subset of cached tokens during decoding. The key challenge, however, lies in deciding which tokens should be retained. Given a decoding query q_t and cached keys $\{k_i\}_{i=1}^{L}$, the ideal pruning policy would preserve the tokens with the largest exact attention logits, $\mathcal{S}_t^\star = \text{Top-}k_i(q_t^\top k_i)$, since these tokens are

[†]Corresponding authors: Rong-Cheng Tu (rongcheng.tu@ntu.edu.sg) and Dacheng Tao (dacheng.tao@ntu.edu.sg).
Code: `https://github.com/ymy9599/LAPrune`.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

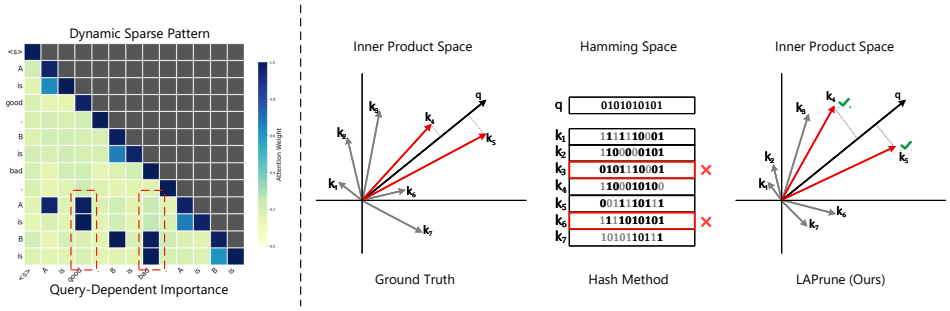

Figure 1: **Motivation of LAPrune.** (Left) LLM attention exhibits dynamic, query-dependent sparsity. (Center) Hashing-based retrieval fails due to the misalignment of space (selecting k_3 and k_6 instead of k_4 and k_5). (Right) LAPrune maintains high fidelity by operating directly in the native feature space.

most likely to dominate the attention output for the current step. Directly computing all logits $q_t^\top k_i$, however, requires the same high-dimensional key access and dot-product computation that pruning seeks to avoid. Therefore, practical KV pruning depends on a lightweight scoring proxy that is both efficient and faithful to the exact-logit ranking. If this proxy fails to reflect token importance under the current query, pruning may discard critical context and degrade generation quality [11, 12].

Existing pruning methods generally rely on two categories of coarse proxies, each characterized by a systematic misalignment in ranking, as illustrated in Fig. 1. First, static or heuristic-based proxies rank tokens based on historical attention weights or fixed positions. While these require negligible computation, they are inherently query-agnostic. By assuming a fixed importance for each token, they fail to adapt when the current query requires specific information that was historically less prominent, resulting in a contextual rigidity that limits complex reasoning. Second, query-dependent retrieval proxies attempt to capture dynamic relevance by mapping queries and keys into a compact discrete space. Although these methods enable efficient candidate selection, their ranking criteria are typically defined by bucket collisions, Hamming distance, or other surrogate metrics rather than the native attention inner product. This objective mismatch can distort the ordering of candidate tokens: tokens that are close in the retrieval space may not be the ones with the largest exact attention logits. Consequently, the selected Top-k candidates may deviate from the true attention Top-k set, reducing both pruning accuracy and downstream generation quality.

To resolve these issues, we propose LAPrune, a KV pruning framework based on additive vector quantization (AQ) that provides an efficient attention-logit-aligned scoring proxy. LAPrune represents each cached key vector as a sum of multiple learned codewords. For a current query, the query-key dot product can then be decomposed into a sum of query-codeword inner products. Unlike hashing-based retrieval, LAPrune does not use compact codes to define a separate retrieval distance. Instead, the codes parameterize a direct approximation of the original key vector, allowing token scores to approximate the native attention logits.

This structure enables a simple and efficient lookup-and-add scoring mechanism. For each decoding query, LAPrune first constructs a small lookup table containing the inner products between the query and all codebook entries. The proxy score of each cached token is then computed by fetching the entries indexed by its AQ codes and summing them. In this way, LAPrune reduces per-token scoring from a high-dimensional dot product over the key dimension to a small number of scalar lookups and additions. This substantially lowers arithmetic cost and memory traffic while maintaining high fidelity to the exact-logit Top-k ranking. As a result, LAPrune enables accurate query-dependent pruning without finetuning the base model or changing the original attention objective.

Our main contributions are summarized as follows:

- We propose LAPrune, a KV pruning framework that utilizes AQ to directly approximate inner products in their native space. This eliminates the metric misalignment inherent in hash-based proxies and improves exact-logit Top-k set recovery.
- We introduce a lookup-and-add mechanism that replaces high-dimensional computations with scalar lookups. Combined with an optimized CUDA kernel, LAPrune enables highly efficient token scoring during decoding.

- Evaluations on various models show that LAPrune consistently outperforms other pruning proxies, achieving a $2.07\times$ decoding speedup over FlashAttention-2 at 128K context with batch size 1.

2 Related Work

2.1 LLM Acceleration through KV Pruning

KV cache pruning aims to accelerate decoding by identifying and retaining only a small subset of relevant keys via a computationally efficient proxy metric. Related attribution methods identify compact input subsets or salient regions that explain foundation-model predictions [13–15]. Early approaches [16–21] primarily adopt static heuristics or permanent eviction based on historical attention weights. While computationally negligible, these strategies are inherently query-agnostic, assuming token importance remains fixed across different queries. This contextual rigidity fails to capture the dynamic retrieval patterns required for complex, long-context tasks where the relevance of a token is context-dependent.

To enable query-dependency, more recent work focuses on non-permanent, on-the-fly pruning [22–25]. Quest [26], for instance, utilizes a coarse-grained proxy by grouping tokens into blocks and estimating relevance via block-level statistics. However, this approach often suffers from resolution loss and internal fragmentation, where irrelevant tokens within a block are unnecessarily retained. To achieve finer-grained selection, hashing-based methods like MagicPIG [27] and Spotlight [28] map queries and keys into a compact binary space for fast retrieval, typically building on angular-distance LSH primitives [29]. Despite their efficiency, they introduce a fundamental objective mismatch: the ranking is performed in the Hamming space, which does not naturally coincide with inner products. This metric distortion fundamentally constrains their ranking fidelity, leading to inconsistent token selection. Learned hashing proxies also support efficient cross-modal retrieval [30].

2.2 LLM Compression By Vector Quantization

Vector quantization (VQ) represents high-dimensional vectors using combinations of learned codewords, enabling compact approximations of vector representations [31]. Variants such as product quantization (PQ) [32] and additive quantization (AQ) [33] improve representational capacity and efficiency, with AQ offering higher reconstruction fidelity under comparable storage and computational budgets. In large language models, prior work has primarily applied VQ to compress the key–value cache to reduce memory footprint [34–36], largely overlooking its potential for accelerating query-dependent computation during decoding. For example, residual-style quantization schemes [37] achieve limited compression while still relying on the standard decode-then-attention pipeline, leaving the dominant attention cost unchanged. CommVQ [38] explores communication-aware codebook designs that enable partial reordering with RoPE [39]; however, during decoding it still applies RoPE to the entire cached context at every step, introducing additional overhead that can even slow down long-context decoding rather than accelerating it. Overall, existing research on applying VQ in LLM has been largely confined to memory compression, with little attention paid to its potential for improving inference-time efficiency.

3 Method

3.1 Dynamic KV Cache Pruning

During autoregressive decoding in large language models, each newly generated token attends to all previously cached key-value (KV) states accumulated from the pre-filling phase and prior decoding steps. Given a query vector $q \in \mathbb{R}^d$ at the current decoding step and a KV cache $\{k_i, v_i\}_{i=1}^N$, standard self-attention computes:

$$\text{Self-Attn}(q) = \text{Softmax}\left(\frac{qK^\top}{\sqrt{d}}\right)V, \tag{1}$$

where $K = [k_1, \ldots, k_N]$ and $V = [v_1, \ldots, v_N]$. In long-context scenarios, the dominant bottleneck arises from the memory bandwidth required to load the ever-growing KV cache from VRAM to the GPU functional units, leading to low GPU utilization as N increases.

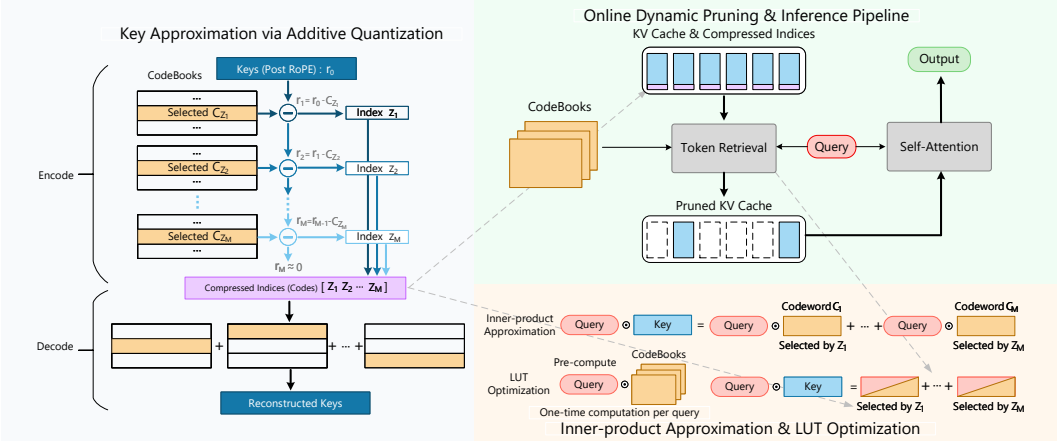

Figure 2: **LAPrune Framework Overview.** (Left) **Key Approximation**: AQ-based decomposition via iterative residual fitting. (Top-Right) **Dynamic Pruning**: Query-level token selection for sparse attention. (Bottom-Right) **LUT Optimization**: Scoring via lookup-and-add operations.

Dynamic KV cache pruning aims to accelerate decoding by restricting attention computation to a small, query-dependent subset of cached tokens. Specifically, given a pruning budget $k \ll N$, we compute an importance score s_i for each cached token and select an index set

$$\mathcal{I}(q) = \mathrm{TopK}\big(\{s_i\}_{i=1}^{N}, k\big). \tag{2}$$

This problem depends only on recovering the oracle top-k set under the same budget, rather than producing a globally correct ranking over all keys. We therefore evaluate pruning quality mainly by set overlap with the oracle set induced by true attention logits.

We then compute attention only over the corresponding subset $\{k_i, v_i\}_{i\in\mathcal{I}(q)}$. By reducing the number of tokens involved in attention, this procedure not only lowers the total computational complexity but also significantly alleviates memory bandwidth pressure during the decoding phase. Existing approaches to dynamic KV pruning mainly differ in the accuracy with which their proxies compute token importance scores. While block-level methods suffer from coarse granularity and internal fragmentation, token-level approaches enable finer selection. However, existing hashing-based token retrieval often misaligns with attention logits due to its discrete space. In this work, we propose an efficient token-level pruning mechanism with an inner-product–aligned scoring proxy that directly approximates attention logits.

3.2 AQ for Inner-Product Approximation

Additive Quantization (AQ) is a vector quantization technique that represents a high-dimensional vector as a sum of multiple codewords drawn from learned codebooks. Given a vector $x \in \mathbb{R}^d$, AQ approximates it as:

$$x \approx \sum_{m=1}^{M} c_m, \tag{3}$$

where each codeword $c_m \in \mathbb{R}^d$ is selected from the m-th codebook. This additive structure provides a compact representation while retaining high reconstruction fidelity.

In the AQ encoding stage, the vector is compressed into discrete codeword indices by minimizing the reconstruction error. In the encoding stage, indices are initialized via a greedy sequential search that minimizes the reconstruction residual and are subsequently refined through Iterated Conditional Modes (ICM). The AQ decoding process is straightforward: the quantized vector is reconstructed by retrieving the codewords from the corresponding codebooks based on the stored indices and summing them up.

A key property of AQ is that it naturally enables efficient approximation of inner products. For any query vector $q \in \mathbb{R}^d$, the inner product with x can be decomposed as:

$$q^\top x \approx \sum_{m=1}^{M} q^\top c_m. \tag{4}$$

This decomposition enables query-dependent Top-k selection using approximate inner products. Applied to post-RoPE keys, it gives an attention-logit-aligned scoring approximation for our pruning strategy.

3.3 LAPrune

Logit-Proxy–Aligned Top-k Set Selection. LAPrune performs token-level pruning by using AQ to compute a scoring proxy that approximates attention logits for Top-k set selection under a fixed budget. Specifically, within each attention head, each key vector $k_i \in \mathbb{R}^d$ stored in the KV cache is encoded using AQ as a sum of codewords selected from multiple learned codebooks. An overview of the LAPrune framework is illustrated in Figure. 2. During decoding, we define the importance score s_i by approximating the original attention logit $q^\top k_i$ through the additive structure of the quantized keys:

$$s_i = \sum_m q^\top c_{m,i} \approx q^\top k_i. \tag{5}$$

This enables efficient estimation of scores $\{s_i\}_{i=1}^N$ for selecting a top-k subset, aiming to match the oracle top-k set induced by true logits under the same budget.

Lookup-and-Add Optimization. While Equation (5) provides an logits-aligned importance score, naively calculating s_i for all N cached tokens still involves $N \times M$ inner products between the query and high-dimensional codewords. As the context length N scales, the computational overhead of selecting itself can become a new bottleneck.

To resolve this, we observe that calculating the importance score s_i involves only inner products between the current query q and the codewords. Since q remains fixed for all cached tokens at each decoding step, we can pre-compute the inner products between q and all entries in the codebooks. Consequently, the importance score for each cached key is obtained through simple lookup-and-add operations: $s_i = \sum_m \mathcal{T}_m[idx_{m,i}]$, where $\mathcal{T}_m$ denotes the pre-computed lookup table for the m-th codebook and $idx_{m,i}$ is the stored index of the m-th codeword for key k_i. This mechanism significantly accelerates the importance score estimation by bypassing redundant high-dimensional computations and focusing only on a constant number of pre-calculated inner products.

Codebook Design. Existing VQ methods for LLM compression typically quantize pre-RoPE keys. However, for dynamic pruning, this strategy incurs a prohibitive $O(N)$ overhead by necessitating repeated rotary embedding applications across the entire cache. To maximize decoding efficiency, we essentially quantize post-RoPE keys, eliminating this redundancy. Yet, this shift introduces a challenge: rotary embeddings distort the spatial distribution, resulting in a more complex data manifold than the static pre-RoPE space. To resolve this fitting challenge, we leverage the inherent basis expansion structure of additive quantization. In contrast to other vector quantization methods such as product quantization, which rely on rigid subspace partitioning and map vectors to disjoint Voronoi regions, additive quantization approximates each vector as a sum of codewords selected from learned codebooks. This additive nature enables a flexible approximation of the high-dimensional space through the superposition of codewords. Consequently, additive quantization maintains high-fidelity retrieval for rotationally distorted representations, ensuring that efficiency gains do not compromise model performance.

The codebooks are learned through a gradient-free calibration process rather than expensive end-to-end training. We first whiten collected key vectors to ensure they are isotropic. Thanks to the basis expansion property of additive quantization, we enforce a shared codebook constraint across all attention heads. This design allows the calibration process to pool diverse statistical patterns from the entire layer into a single robust feature set, rather than fitting separate, potentially unstable codebooks for each head. We then apply successive residual MiniBatch KMeans++ for initialization, which greedily fits reconstruction residuals to capture the primary data patterns. Refinement is performed

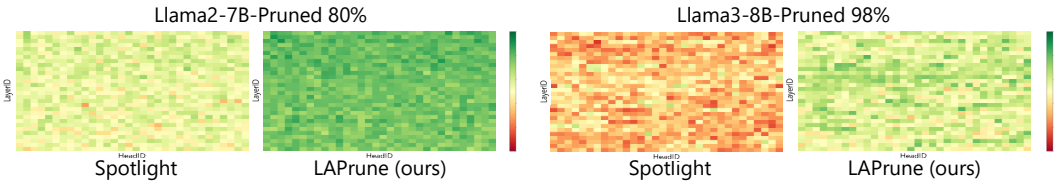

Figure 3: Selection accuracy (IoU) comparison. Heatmaps visualize the token selection accuracy across layers and heads for LLaMA2-7B and LLaMA3-8B.

Table 1: Comparison of KV cache selection accuracy (IoU (↑)) between Spotlight (MLP-based hashing) and our proposed **LAPrune**.

Method	**Pruning Ratio**							
	80%		90%		95%		98%	
	Spotlight	**LAPrune**	**Spotlight**	**LAPrune**	**Spotlight**	**LAPrune**	**Spotlight**	**LAPrune**
LLaMA2-7B	0.54	0.77 (+0.23)	0.51	0.73 (+0.22)	0.47	0.70 (+0.23)	0.42	0.65 (+0.23)
LLaMA3-8B	0.50	0.71 (+0.21)	0.43	0.66 (+0.23)	0.39	0.62 (+0.23)	0.34	0.58 (+0.24)
Qwen2.5-7B	0.53	0.69 (+0.16)	0.46	0.63 (+0.17)	0.40	0.57 (+0.17)	0.35	0.51 (+0.16)

via closed-form least-squares updates—updating each codeword to the mean of its assigned residuals through efficient indexing—obviating the need for backpropagation.

3.4 Efficiency Analysis and Hardware Optimization

The integration of LAPrune introduces three primary changes to the standard inference pipeline: an additional AQ encoding phase for new tokens, a modified attention mechanism for sparse computation, and a marginal memory footprint for storing codebook indices.

Encoding Efficiency. The AQ encoding phase occurs primarily during the prefill stage and is also performed for each newly generated key during decoding. For prefill, we implement a fused CUDA kernel that casts each residual stage as a large GEMM dispatched to cuBLAS Tensor Cores and folds the subsequent codeword assignment and residual update into a single lightweight launch, with tile-reused logits buffers and FP16 vectorized arithmetic to minimize memory traffic. For decoding, new tokens are batched through a small local buffer to amortize kernel launch overhead. For GQA-based models, we further restrict encoding to the stored KV heads rather than the expanded query heads, substantially reducing the encode workload.

Computational and Bandwidth Gains. LAPrune achieves overall acceleration by mitigating both the memory bandwidth bottleneck and the computational workload. Let ρ denote the pruning rate of the KV cache (e.g., $\rho = 0.98$). By loading only a small retained fraction $(1-\rho)$ of the original KV cache for attention computation, we drastically reduce data transfer from VRAM, directly addressing the primary memory-bandwidth constraint in long-context inference. Second, the modified attention mechanism reduces the total FLOPs required for the $QK^\top$ operation. The computational cost is approximately $(M + 2(1-\rho)d)N + 2MCd$, where $2MCd$ represents the constant cost of pre-calculating lookup tables and MN accounts for the scalar additions during token selection. Since $2(1-\rho)d \ll 2d$, the computational growth rate remains significantly lower than the original $2dN$ complexity as the context length N scales. A detailed breakdown of the computational cost is provided in Appendix E. We also implement a GEMM CUDA kernel that fuses query–codeword multiplication with lookup-and-sum operations, thereby minimizing memory movement and kernel overhead to achieve substantial end-to-end speedups.

Memory Footprint. LAPrune maintains a compact scoring index by quantizing only the key vectors for selecting. Each token is represented by codebook indices totaling $M \log_2(C)$ bits. In our standard configuration ($M = 32, C = 256$), this results in a fixed overhead of 256 bits per token. This memory footprint is comparable to hashing-based selecting methods, which typically require codes ranging from 128 up to 720 bits to achieve functional accuracy.

Table 2: Language modeling perplexity performance on PG19, ProofPile, and CodeParrot datasets. Our proposed **LAPrune** achieves lower perplexity than the hash-based Spotlight at high pruning ratios across model architectures. Δ rows report the perplexity gap to Oracle Top-K at 98% pruning. [0,1] denotes the frozen layers.

Method	Frozen	Configuration	LLaMA2-7B			LLaMA3-8B			Qwen2.5-7B		
			PG19	Math	Code	PG19	Math	Code	PG19	Math	Code
Vanilla	-	0% Pruned	6.879	3.997	3.679	8.604	3.048	5.219	11.111	3.352	4.953
Oracle Top-K	[0,1]	98% Pruned	6.941	4.060	3.729	8.881	3.109	5.353	11.506	3.423	4.989
Spotlight	[0,1]	80% Pruned	6.887	3.998	3.682	8.612	3.051	5.228	11.216	3.371	4.966
		90% Pruned	6.908	4.004	3.689	8.651	3.060	5.239	11.331	3.394	4.982
		95% Pruned	6.959	4.023	3.703	8.734	3.079	5.285	11.543	3.441	5.012
		98% Pruned	7.106	4.080	3.768	8.977	3.140	5.434	12.147	3.562	5.095
		Δ vs. Oracle	*+0.165*	*+0.020*	*+0.039*	*+0.096*	*+0.031*	*+0.081*	*+0.642*	*+0.139*	*+0.106*
LAPrune (Ours)	[0,1]	80% Pruned	6.883	3.997	3.681	8.611	3.049	5.222	11.258	3.378	4.971
		90% Pruned	6.897	4.001	3.685	8.636	3.054	5.230	11.310	3.386	4.972
		95% Pruned	6.939	4.015	3.696	8.702	3.069	5.262	11.417	3.404	4.984
		98% Pruned	7.064	4.060	3.736	8.893	3.111	5.370	11.657	3.446	5.006
		Δ vs. Oracle	*+0.123*	*+0.000*	*+0.007*	*+0.012*	*+0.002*	*+0.017*	*+0.151*	*+0.024*	*+0.017*

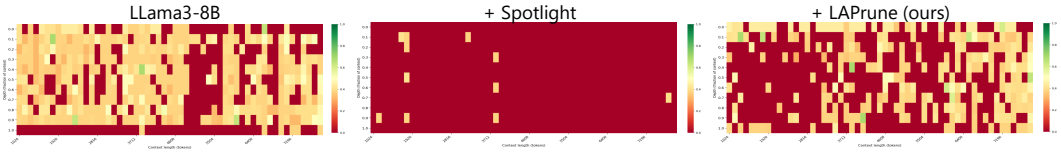

Figure 4: NIAH performance. Heatmaps visualize the ROUGE scores across varying context lengths and insertion depths on LLaMA3-8B.

4 Experiments

To verify the generalizability of our method, we evaluate LAPrune across a diverse suite of Large Language Models encompassing various versions, architectures, and parameter scales, including LLaMA-2-7B, LLaMA-2-13B, LLaMA-3-8B, LLaMA-3.1-8B-Instruct, and Qwen-2.5-7B. We compare our approach against several representative dynamic KV pruning baselines, including Quest for block-wise retrieval, MagicPIG for demand-driven dynamic retrieval, and Spotlight, a hashing-based baseline that operates at the token level similar to our method. We configure LAPrune with $M = 32$ sub-codebooks per layer, each containing $C = 256$ codewords. All attention heads within a transformer layer share a unified set of codebooks. For calibration, we sample 4K-token windows from the ArXiv and Book datasets across the model's supported context range, preserving their absolute position IDs when extracting post-RoPE keys. We use 80 calibration windows per layer for the LLaMA3.1-8B-Instruct long-context quality evaluation and five windows per layer for the other evaluations. All experiments use the PyTorch framework. Experimental details are provided in Appendix D.

4.1 Main Results

KV Cache Scoring Accuracy. To evaluate proxy accuracy, we compute IoU between the proxy-selected tokens and the Top-k tokens selected by exact full-attention logits per head (Oracle). We compare LAPrune against Spotlight, as it represents a highly competitive token-level retrieval baseline using MLP-based hashing. As shown in Table 1, LAPrune consistently achieves higher IoU across all pruning ratios, demonstrating both high precision and strong generalizability across different model architectures. At 98% pruning, our method attains 0.65 IoU on LLaMA2-7B, versus 0.42 for the hashing proxy. Figure 3 visualizes the selection stability across different layers and attention heads. While the Spotlight exhibits noticeable performance fluctuations and inconsistencies throughout the network, LAPrune maintains a uniformly high IoU.

Language Modeling Performance. We evaluate language modeling performance via perplexity across long-text (PG19 [40]), math (ProofPile), and code (CodeParrot) domains. For each dataset, we evaluate on 100 samples and perform pruning solely by selecting tokens using the proxy scores.

Table 3: Comparison with Quest and MagicPIG on LLaMA3-8B under near-98% KV pruning, reporting perplexity on PG19, ProofPile, and CodeParrot.

Method	Frozen	Configuration	PG19	Math	Code
Vanilla	-	Baseline	8.60	3.05	5.22
Oracle Top-K	[0,1]	Pruned 98%	8.88	3.11	5.35
Quest	[0,1]	Pruned 96.9%	27.51	10.83	14.21
MagicPIG	[0,16]	Pruned 98%	NaN	NaN	NaN
LAPrune (ours)	[0,1]	Pruned 98%	8.89	3.11	5.37

Table 4: Investigation of Different components in LAPrune on LLaMA2-7B with a 98% pruning ratio. Comparison of perplexity on PG19, ProofPile, and CodeParrot datasets.

Method	Local (64)	Sink (4)	Retrieval	PG19	Math	Code
Vanilla				6.88	4.00	3.68
Oracle Top-K			✓	6.94	4.08	3.73
	✓	✓	✓	7.04	4.05	3.72
LAPrune (ours)	✓	✓		7.85	4.94	4.14
			✓	7.06	4.06	3.74

Table 5: Downstream task performance on LLaMA3.1-8B-Instruct at 98% pruning, covering seven LongBench tasks and four RULER retrieval tasks. RULER is evaluated at 128K context.

		LongBench								RULER				
Method	**Config.**	Qsp	QMS	TREC	TQA	SAM	LCC	RP	**Avg.**	MK2	MK3	MQ	MV	**Avg.**
Dense	Baseline	45.45	25.22	73.00	91.65	43.81	63.38	56.61	57.02	0.90	0.71	0.96	0.93	0.875
Oracle Top-K	98% Pruned	45.86	24.88	72.00	92.24	44.51	65.01	60.30	57.83	0.89	0.63	0.94	0.94	0.850
MagicPIG	98% Pruned	44.71	25.24	70.50	80.17	37.22	49.73	51.35	51.27	0.57	0.06	0.44	0.35	0.355
Spotlight	98% Pruned	46.71	25.05	70.00	92.70	44.09	59.17	58.17	56.56	0.88	0.42	0.94	0.87	0.778
LAPrune (Ours)	98% Pruned	45.93	25.03	71.50	91.74	44.27	65.04	60.35	**57.69**	0.88	0.63	0.94	0.95	**0.850**

Table 2 shows that LAPrune achieves lower perplexity than Spotlight at high pruning ratios across the evaluated architectures. At 98% pruning on PG19, LAPrune scores 7.064 on LLaMA2-7B versus 7.106 for Spotlight, and 11.657 on Qwen2.5-7B versus 12.147. LAPrune also closely matches exact-logit Oracle selection: its Qwen perplexity gaps are only 0.151, 0.024, and 0.017 on PG19, ProofPile, and CodeParrot, respectively, while it matches Oracle's reported PPL of 4.060 on LLaMA2-7B ProofPile. These small perplexity gaps indicate that LAPrune closely approximates exact-logit selection across text, math, and code domains under aggressive pruning.

At an aggressive 0.98 pruning ratio, baseline proxies exhibit severe instability that reflects the limitations of their underlying scoring designs. As shown in Table 3, Quest suffers from a perplexity explosion because its coarse, block-level proxy cannot resolve which individual tokens within a block are truly relevant when the budget is so tight; the resulting fragmentation discards critical tokens. MagicPIG collapses to NaNs as the metric distortion of its LSH-based proxy causes the Top-k set to deviate sharply from the exact-logit ranking, leading to numerical breakdown. In contrast, LAPrune's logits-aligned scoring remains accurate and numerically stable even at this extreme regime, as further confirmed in Table 4.

NIAH. We conduct the Needle In A Haystack (NIAH) test to evaluate the ability of the LAPrune to retrieve specific information. For this experiment, we utilize an offline evaluation where retrieval success is measured by the ROUGE score between the generated output and a custom-designed answer. We evaluate context lengths ranging from 1k to 8k tokens, using manually constructed needles and answers. Figure 4 contrasts the retrieval performance of LAPrune and Spotlight. While Spotlight's accuracy collapses across most settings, LAPrune consistently identifies target information. Despite minor degradation as context grows, LAPrune exhibits a markedly more stable profile, confirming its superior fidelity in preserving long-range dependencies.

LongBench. We evaluate LAPrune on seven LongBench tasks covering question answering, summarization, classification, and code completion. As shown in Table 5, LAPrune achieves an average score of 57.69, outperforming Spotlight at 56.56 and MagicPIG at 51.27. It performs particularly well on code completion, achieving the highest scores among the compared pruning methods on LCC and RepoBench-P at 65.04 and 60.35, respectively. Overall, LAPrune reaches 99.8% of Oracle's average score, with a gap of only 0.14 points, and also exceeds Dense's average of 57.02. Together, these results show that LAPrune supports diverse downstream tasks with little loss relative to Oracle selection.

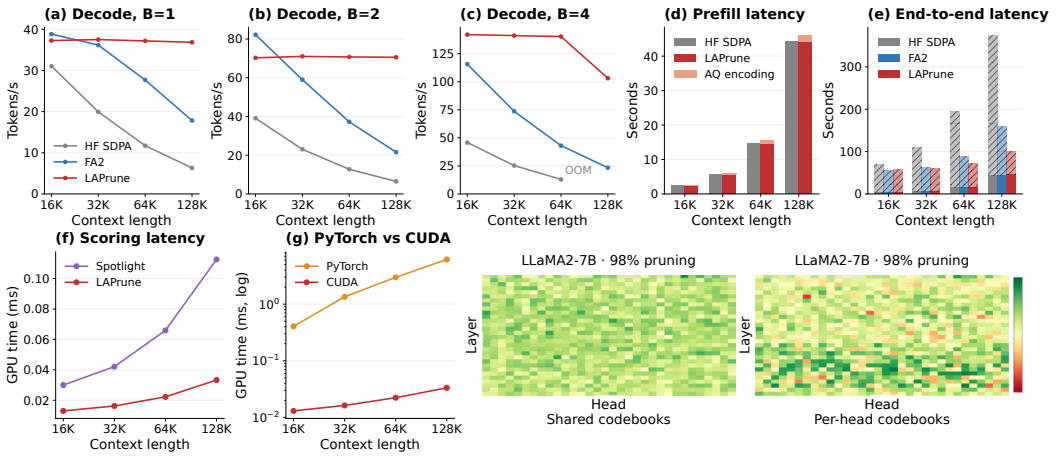

Figure 5: System efficiency on LLaMA3.1-8B-Instruct using one RTX PRO 6000D. (a–c) Decode throughput at batch sizes 1, 2, and 4. (d) Prefill latency with AQ encoding overhead. (e) End-to-end latency. (f, g) Scoring latency: vs. Spotlight, and PyTorch vs. CUDA. *Right:* shared vs. per-head codebooks on LLaMA2-7B at 98% pruning.

RULER. We further evaluate long-context retrieval at 128K on MK2, MK3, MQ, and MV. LAPrune achieves an average score of 0.850, matching Oracle and outperforming Spotlight at 0.778 and MagicPIG at 0.355. It matches Oracle on MK3 and MQ and slightly exceeds it on MV, where it scores 0.95 versus 0.94. The advantage over the other pruning methods is especially clear on the more challenging MK3 task: LAPrune scores 0.63, compared with 0.42 for Spotlight and 0.06 for MagicPIG. These results confirm that LAPrune can locate task-critical information among a large number of distractors, even at 128K context and 98% pruning.

4.2 System Efficiency

We evaluate system efficiency on LLaMA3.1-8B-Instruct from 16K to 128K context on one RTX PRO 6000D. In long-context inference, each decoding step must access an ever-growing KV cache. We benchmark prefill, decoding, and end-to-end generation under the same hardware and software settings, reporting means over three independent runs. Detailed settings are provided in Appendix F.

Figure 5(a–c) shows that LAPrune mitigates the growth in memory traffic and substantially reduces throughput degradation as the context grows. At 128K, it achieves $2.07\times$, $3.27\times$, and $4.41\times$ decoding speedups over FlashAttention-2 at batch sizes 1, 2, and 4, respectively. As shown in Figure 5(d), AQ encoding adds only approximately 3.8% to dense prefill latency at 128K, and this overhead is amortized during decoding. Figure 5(e) further shows that end-to-end generation of 2k tokens takes 101.2 s with LAPrune, compared with 160.4 s for FlashAttention-2 and 374.5 s for HF SDPA, yielding $1.58\times$ and $3.70\times$ speedups, respectively.

Beyond attention, LAPrune's GPU scoring is efficient, achieving a $2.78\times$ average speedup over Spotlight's packed Triton scorer across 16K–128K in Figure 5(f). Figure 5(g) compares our CUDA scorer with an explicit PyTorch AQ lookup-and-sum reference: CUDA scoring takes 0.013–0.033 ms, compared with 0.406–6.154 ms for the reference, demonstrating the benefit of our specialized scoring kernel.

Memory Footprint. At 128K, LAPrune requires an additional 1.104 GiB of persistent GPU memory and 0.081 GiB of overall peak GPU memory compared with Dense, corresponding to overheads of 3.61% and 0.18%, respectively. These whole-model measurements include the full KV cache and all auxiliary structures, showing that LAPrune adds only a small memory overhead. Appendix C provides the measurement settings and detailed breakdown.

Table 6: Ablation study on model size. Language modeling perplexity performance across PG19, ProofPile, and CodeParrot datasets.

Method	Configuration	LLaMA2-13B		
		PG19	Math	Code
Vanilla	0% Pruned	6.233	3.745	3.565
Oracle Top-K	98% Pruned	6.360	3.800	3.597
LAPrune (Ours)	80% Pruned	6.235	3.746	3.565
	90% Pruned	6.246	3.751	3.569
	95% Pruned	6.276	3.763	3.576
	98% Pruned	6.364	3.801	3.601

Table 7: Ablation on codebook and codeword counts under 98% pruning. Perplexity on PG19.

Method	Codebooks	Codewords	IoU (↑)	Perplexity (↓)
LAPrune (ours)	16	128	0.43	7.080
	16	256	0.47	7.069
	32	128	0.61	7.068
	32	256	0.65	7.064

Table 8: Training data sensitivity of LAPrune. IoU on PG19, Perplexity on CodeParrot.

Method	Training Data	IoU (↑)	Perplexity (↓)
LAPrune (ours)	Books + arXiv	0.58	5.370
	ProofPile (5 samples)	0.58	5.375

4.3 Ablation Study

We evaluate LAPrune on LLaMA2-13B to examine its applicability beyond the 7B setting. Table 6 shows that, even at 98% pruning, its perplexity remains within 0.004 of Oracle across all three domains. We further examine representation capacity by varying the number of codebooks M and codewords C in Table 7. At $C = 256$, increasing M from 16 to 32 raises IoU from 0.47 to 0.65, whereas increasing C from 128 to 256 at $M = 32$ raises it from 0.61 to 0.65. Increasing the number of additive components therefore has a larger effect in the tested configurations, consistent with additional components capturing residual structure left unresolved by a smaller representation. Our default $M = 32$, $C = 256$ achieves the best IoU and perplexity among these settings with a 32-byte index per encoded key vector.

We also examine whether the representation depends strongly on calibration data and codebook organization. In Table 8, calibration on Books and ArXiv or on five ProofPile samples gives the same reported IoU of 0.58, while CodeParrot perplexity changes only from 5.370 to 5.375. This suggests that a small calibration set need not closely match the evaluation domain in the tested settings. The heatmaps on the right of Figure 5 further show that head-shared codebooks yield more uniform retrieval accuracy than per-head codebooks. Pooling keys across heads provides a broader calibration distribution, which may help explain this stability, while avoiding separate codebook storage for every head. Finally, AQ achieves an IoU of 0.65 on LLaMA2-7B at 98% pruning, compared with 0.50 for PQ. Whereas PQ quantizes separate feature subspaces, AQ combines codewords spanning the full key dimension. This result supports the additive representation for capturing key structure across dimensions and preserving attention-logit rankings.

5 Conclusion

We present LAPrune, a KV cache pruning method that addresses the misalignment between pruning proxies and attention logits by directly approximating inner products via additive quantization. By enabling efficient lookup-and-add scoring aligned with the attention objective, LAPrune achieves accurate token selection without model finetuning. Empirically, this alignment yields consistently lower perplexity and a $2.07\times$ decoding speedup over FlashAttention-2 at 128K context with batch size 1, highlighting objective-aligned pruning as a promising direction for efficient LLM inference.

Acknowledgments and Disclosure of Funding

This research / project is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office. Any opinions, findings and conclusions or recommendations expressed in these materials are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore.

References

[1] Wenkai Fang, Shunyu Liu, Yang Zhou, Kongcheng Zhang, Tongya Zheng, Kaixuan Chen, Mingli Song, and Dacheng Tao. SeRL: Self-play reinforcement learning for large language models with limited data. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/95c2cbe23fb6d28a4ae908aa7f3de5bf-Paper-Conference.pdf.

[2] Kongcheng Zhang, Qi Yao, Shunyu Liu, Yingjie Wang, Baisheng Lai, Jieping Ye, Mingli Song, and Dacheng Tao. Consistent paths lead to truth: Self-rewarding reinforcement learning for LLM reasoning. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/565f995643da6329cec701f26f8579f5-Abstract-Conference.html.

[3] Kongcheng Zhang, Qi Yao, Baisheng Lai, Jiaxing Huang, Wenkai Fang, Dacheng Tao, Mingli Song, and Shunyu Liu. Incentivizing LLM reasoning via reinforcement learning with functional monte carlo tree search. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/pdf/c2993c49877279588d026101cb64146abdbe080f.pdf.

[4] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753*, 2024.

[5] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. Llm maybe longlm: Self-extend llm context window without tuning. *arXiv preprint arXiv:2401.01325*, 2024.

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

[7] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, et al. ∞Bench: Extending long context evaluation beyond 100k tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277, 2024.

[8] Yibo Wang, Yongcheng Jing, Shunyu Liu, Hao Guan, Rong-Cheng Tu, Chengyu Wang, Jun Huang, and Dacheng Tao. VTC-R1: Vision-text compression for efficient long-context reasoning. *arXiv preprint arXiv:2601.22069*, 2026. URL https://arxiv.org/abs/2601.22069.

[9] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024.

[10] Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*, 2024.

[11] Wenhao Sun, Rong-Cheng Tu, Jingyi Liao, Zhao Jin, and Dacheng Tao. AsymRnR: Video diffusion transformers acceleration with asymmetric reduction and restoration. *arXiv preprint arXiv:2412.11706*, 2024. URL https://arxiv.org/abs/2412.11706.

[12] Wenhao Sun, Rong-Cheng Tu, Yifu Ding, Jingyi Liao, Zhao Jin, Shunyu Liu, and Dacheng Tao. VORTA: Efficient video diffusion via routing sparse attention. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/0badcb4e95306df76a719409155e46e8-Abstract-Conference.html.

[13] Ruoyu Chen, Xiaoqing Guo, Kangwei Liu, Siyuan Liang, Shiming Liu, Qunli Zhang, Laiyuan Wang, Hua Zhang, and Xiaochun Cao. Where MLLMs attend and what they rely on: Explaining autoregressive token generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/papers/Chen_Where_MLLMs_Attend_and_What_They_Rely_On_Explaining_Autoregressive_CVPR_2026_paper.pdf.

[14] Ruoyu Chen, Siyuan Liang, Jingzhi Li, Shiming Liu, Li Liu, Hua Zhang, and Xiaochun Cao. Less is more: Efficient black-box attribution via minimal interpretable subset selection. *arXiv preprint arXiv:2504.00470*, 2025. URL https://arxiv.org/abs/2504.00470.

[15] Ruoyu Chen, Siyuan Liang, Jingzhi Li, Shiming Liu, Maosen Li, Zheng Huang, Hua Zhang, and Xiaochun Cao. Interpreting object-level foundation models via visual precision search. *arXiv preprint arXiv:2411.16198*, 2024. URL https://arxiv.org/abs/2411.16198.

[16] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *ICLR*, 2024.

[17] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *NeurIPS*, 2024.

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

[19] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024.

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

[21] Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, et al. NACL: A general and effective KV cache eviction framework for LLM at inference time. In *NACL*, 2024.

[22] Yuzhe Gu, Xiyu Liang, Jiaojiao Zhao, and Enmao Diao. Obcache: Optimal brain kv cache pruning for efficient long-context llm inference. *arXiv preprint arXiv:2510.07651*, 2025.

[23] Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Think: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018*, 2024.

[24] Bo Lv, Quan Zhou, Xuanang Ding, Yan Wang, and Zeming Ma. Kvpruner: Structural pruning for faster and memory-efficient large language models. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.

[25] Luning Wang, Shiyao Li, Xuefei Ning, Zhihang Yuan, Shengen Yan, Guohao Dai, and Yu Wang. Cskv: Training-efficient channel shrinking for kv cache in long-context scenarios. *arXiv preprint arXiv:2409.10593*, 2024.

[26] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. In *ICML*, 2024.

[27] Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. MagicPIG: LSH sampling for efficient LLM generation. In *NeurIPS*, 2024.

[28] Wenhao Li, Yuxin Zhang, Gen Luo, Haiyuan Wan, Ziyang Gong, Fei Chao, and Rongrong Ji. Spotlight attention: Towards efficient llm generation via non-linear hashing-based kv cache retrieval. *arXiv preprint arXiv:2508.19740*, 2025.

[29] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *NeurIPS*, 2015.

[30] Rong-Cheng Tu, Xian-Ling Mao, Wenjin Ji, Wei Wei, and Heyan Huang. Data-aware proxy hashing for cross-modal retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 686–696, 2023. doi: 10.1145/3539618.3591660.

[31] Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[32] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[33] Artem Babenko and Victor Lempitsky. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 931–938, 2014.

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

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

[36] Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *arXiv preprint arXiv:2405.03917*, 2024.

[37] Ankur Kumar. Residual vector quantization for kv cache compression in large language model. *arXiv preprint arXiv:2410.15704*, 2024.

[38] Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Binazir Karimzadeh, Colorado J Reed, Chong Wang, and Chuang Gan. Commvq: Commutative vector quantization for kv cache compression. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[39] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[40] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *ICLR*, 2020.

A Limitations

The efficiency gains of LAPrune are most pronounced in long-context regimes, where memory-bandwidth pressure dominates per-token latency. In short-context settings (below 16K), the overhead of AQ encoding and lookup-table construction is not fully amortized, and dense attention with FlashAttention-2 remains the preferable choice. LAPrune is therefore best deployed in scenarios where context lengths routinely exceed tens of thousands of tokens.

B Impact Statement

This paper presents a method designed to significantly accelerate long-context LLM inference. By lowering computational costs and enabling deployment on consumer-grade hardware, our work contributes to "Green AI" and democratizes access to advanced models. However, we acknowledge that generative language models carry inherent risks, such as producing biased, privacy-violating, or harmful content. As our method improves generation efficiency, it may inadvertently scale these potential negative impacts. Therefore, we urge users and service providers to take responsibility for the generated content, implement necessary safeguards, and strive to ensure positive social impacts.

C Whole-Model Memory Footprint

We measure whole-model GPU memory on one RTX PRO 6000D using Llama-3.1-8B-Instruct in FP16, batch size 1, a 128,000-token prompt, and 128 generated tokens. Dense uses full attention on all layers. LAPrune uses full attention on layers [0,1] and 2% Top-k retrieval on the remaining 30 layers, with an additional 64-token local encoding buffer. Table 9 reports persistent memory after decoding and separately measured prefill and decode peaks, including temporary runtime allocations. The overall peak occurs during prefill in this experiment.

Table 9: Whole-model memory at 128K context (GiB). The upper panel reports measured persistent and peak allocations; the lower panel decomposes LAPrune's persistent allocation. Values are rounded independently.

Method	Persistent	Prefill/overall peak	Decode peak
Dense	30.607	44.814	30.851
LAPrune	31.711	44.895	31.759

LAPrune persistent component	GiB
Model weights and registered buffers	14.958
Full FP16 KV cache	15.641
AQ codes	0.916
AQ codebooks	0.063
Precomputed support	0.122
Other persistent state	0.011
Total	31.711

Relative to Dense, LAPrune requires an additional 1.104 GiB (3.61%) of persistent memory, 0.081 GiB (0.18%) of overall-peak memory, and 0.908 GiB (2.94%) of decode-peak memory.

D Experimental Setup

This appendix provides comprehensive details regarding our experimental configurations to ensure the reproducibility of our results. We describe the codebook calibration process, the specific hyperparameters for various model architectures, and the implementation details of our evaluation benchmarks.

D.1 Codebook Calibration and Training

The calibration of LAPrune codebooks is a critical step to ensure that the scoring proxy remains attention logits aligned. Unlike traditional pruning methods that require end-to-end training, our approach utilizes a data-efficient, gradient-free calibration phase.

As outlined in Algorithm 1, the process consists of two primary stages. First, we perform a stable initialization in an isotropic space. By applying a whitening transform to a small sample of key vectors, we mitigate the spatial distribution distortion introduced by rotary positional embeddings. We then employ a greedy residual fitting strategy combined with MiniBatch-KMeans++ to initialize the codewords. Second, we perform iterative refinement using a closed-form least-squares approach. In each iteration, we update the codeword indices via Iterated Conditional Modes (ICM) and subsequently recompute the codewords as the mean of their assigned residuals. This ensures monotonic convergence of the reconstruction error and high selecting fidelity.

Algorithm 1 LAPrune Codebook Calibration

Input: Key features $\mathbf{X} \in \mathbb{R}^{N \times d}$, number of codebooks M, entries per codebook K, refinement iterations T.
Output: Calibrated codebooks $\{\mathbf{C}_m\}_{m=1}^{M}$, where each $\mathbf{C}_m \in \mathbb{R}^{K \times d}$.
// Phase 1: Stable Initialization via Whitening and Greedy Residuals
$\boldsymbol{\mu} \leftarrow \text{mean}(\mathbf{X}, \text{dim} = 0)$
$\boldsymbol{\Sigma} \leftarrow \text{Covariance}(\mathbf{X} - \boldsymbol{\mu})$
Compute whitening matrix $\mathbf{W} = \boldsymbol{\Sigma}^{-1/2}$ and its inverse $\mathbf{W}^{-1} = \boldsymbol{\Sigma}^{1/2}$
$\mathbf{Y} \leftarrow (\mathbf{X} - \boldsymbol{\mu})\mathbf{W}^{\top}$ ▷ Transform to isotropic space
$\mathbf{R} \leftarrow \mathbf{Y}$ ▷ Initialize residuals
for $m = 1$ **to** M **do**
    $\mathbf{C}_m^{white} \leftarrow \text{MiniBatchKMeans}(\mathbf{R}, K)$
    $z_{i,m} \leftarrow \arg\min_{k \in \{1 \dots K\}} \|\mathbf{R}_i - \mathbf{C}_{m,k}^{white}\|_2^2$ ▷ Greedy assignment
    $\mathbf{R} \leftarrow \mathbf{R} - \mathbf{C}_{m,z_m}^{white}$ ▷ Update residuals for next stage
end for
$\mathbf{C}_m \leftarrow (\mathbf{C}_m^{white}\mathbf{W}^{-\top}) + \frac{\boldsymbol{\mu}}{M}$ ▷ Project back to original space and distribute bias
// Phase 2: Iterative Refinement (Closed-form M-step)
for $t = 1$ **to** T **do**
    Update indices $\mathbf{z}$ using Iterated Conditional Modes (ICM) based on current $\{\mathbf{C}_m\}$
    for $m = 1$ **to** M **do**
        for $k = 1$ **to** K **do**
            $\mathcal{I}_{m,k} \leftarrow \{i \mid z_{i,m} = k\}$ ▷ Indices assigned to k-th codeword
            if $\mathcal{I}_{m,k} \neq \emptyset$ **then**
                $\mathbf{C}_{m,k} \leftarrow \text{mean}_{i \in \mathcal{I}_{m,k}} \left(\mathbf{X}_i - \sum_{j \neq m} \mathbf{C}_{j,z_{i,j}} \right)$
            end if
        end for
    end for
end for
return $\{\mathbf{C}_m\}_{m=1}^{M}$

D.2 Calibration Dataset and Sampling

The efficacy of LAPrune calibration relies on capturing the statistical distribution of the key cache. To maintain high efficiency, our calibration process requires only a minimal amount of data from general-purpose corpora.

Data Sources. The calibration features are extracted using text samples from two diverse sources: the ArXiv and Book datasets. Specifically, we utilize processed JSONL files where each entry contains the raw text from academic papers and literary works, respectively.

Sampling Strategy. We construct the calibration set from 4K-token windows sampled across the model's supported context range. Each window preserves its absolute position IDs when extracting

post-RoPE keys, allowing calibration features to represent keys at different sequence positions. Windows are sampled independently from the combined ArXiv and Book corpus.

Data Volume and Processing. We use 80 calibration windows per layer for the LLaMA3.1-8B-Instruct long-context quality evaluation and five windows per layer for the other evaluations, corresponding to 327,680 and 20,480 calibration tokens per layer, respectively. Calibration uses batch size 1. Details are summarized in Table 10.

Table 10: Comprehensive calibration settings and hyperparameters for LAPrune.

Category	**Parameter / Hyperparameter**	**Value**
Calibration Data	Corpus Sources	ArXiv, Book
	Segment Length (L)	4,096 tokens
	Windows per Layer (other / long-context quality)	5 / 80
	Tokens per Layer (other / long-context quality)	20,480 / 327,680
AQ Configuration	Codebook sharing strategy	Per-layer (Head-shared)
	Whitening stability constant (ϵ)	1×10^{-6}
	MiniBatch-KMeans decay (β)	0.9
	Calibration iterations per level	10
Inference / Pruning	Batch Size during Calibration	1
	Local window size	0
	Number of sink tokens	0

D.3 Evaluation Datasets and Model Configurations

We evaluate LAPrune across three representative datasets: **PG19**, consisting of long-form books from the Project Gutenberg library; **ProofPile**, a collection of mathematical scientific papers and documents; and **CodeParrot**, which contains Python source code.

For model truncation, we evaluate the LLaMA-2 series (7B and 13B) with a context length of 4,096 tokens to match their native pre-training window. All other models, including LLaMA-3-8B and Qwen-2.5-7B, are evaluated with a uniform truncation length of 8,192 tokens to assess performance in extended-context scenarios.

For Qwen2.5-7B perplexity evaluation, all methods use the same BF16 attention arithmetic: QK products are materialized in BF16, softmax is computed in FP32, and probabilities are cast back to BF16 before value aggregation.

D.4 Needle In A Haystack (NIAH) Details

To rigorously assess the retrieval fidelity of LAPrune under high sparsity, we conduct a Needle In A Haystack experiment using a challenging retrieval setup. We manually construct a "hard" needle consisting of ten semantically unrelated system-operation keywords: *"The correct response is: coffee dashboard alerts logs restart verify metrics notify summary ticket."* This manual design ensures that the model cannot rely on linguistic priors to guess the answer.

In our implementation, the background "haystack" is sampled from the proof-pile corpus. The model is tasked with answering the prompt: *"What is the correct response?"*, where the expected answer is: *"coffee dashboard alerts logs restart verify metrics notify summary ticket"*. The evaluation is conducted across context lengths from 1,024 to 8,192 tokens with a step size of 256. We utilize **ROUGE-L F1** as the primary evaluation metric to quantify the overlap between the generated output and the target answer.

D.5 Long-Context Benchmarks

We evaluate downstream task performance on two long-context benchmarks: **LongBench**, covering diverse real-world tasks such as question answering, summarization, and code completion; and

RULER, a retrieval-oriented benchmark whose synthetic tasks directly probe whether the pruning proxy faithfully preserves the relevant tokens.

Task Abbreviations. The column headers in Table 5 use the following abbreviations. LongBench: **Qsp** = Qasper, **QMS** = QMSum, **TREC** = TREC, **TQA** = TriviaQA, **SAM** = SAMSum, **LCC** = LCC, **RP** = RepoP. RULER: **MK2 / MK3** = NIAH multi-key 2 / 3, **MQ** = NIAH multi-query, **MV** = NIAH multi-value.

LongBench Configuration. On LongBench, Oracle, Spotlight, and LAPrune use no sink or local window. MagicPIG additionally retains 4 sink tokens and 16 local tokens beyond its retrieved tokens.

RULER Configuration. On RULER, all pruning methods retain 4 sink tokens and 64 local tokens. Oracle, Spotlight, and LAPrune additionally retrieve 2% of the remaining tokens, giving a total retained-token budget of $0.02(N - 68) + 68$ for a cache of N tokens, up to integer rounding. Sink and local tokens are excluded from retrieval to avoid duplicates. MagicPIG uses the same sink/local setting, with its expected LSH sampling budget specified in Appendix G.

All tasks use greedy decoding to ensure deterministic outputs and consistency across pruning configurations.

E Theoretical Complexity Analysis

To quantify the computational efficiency of LAPrune, we analyze the Floating Point Operations (FLOPs) required for the attention scoring and computation process. We focus on the primary difference in the computation of $QK^\top$ during the decoding phase.

E.1 FLOPs Calculation for LAPrune

In our approach, the $QK^\top$ computation is decomposed into three distinct components: lookup table (LUT) pre-computation, importance scoring via additions, and sparse attention computation.

1. Lookup Table Pre-computation. First, we compute the inner products between the current query $q \in \mathbb{R}^d$ and all MC codewords stored in the codebooks. The FLOPs for this dense multiplication are:

$$\text{FLOPs}_{\text{multiplication}} = 2MCd \tag{6}$$

where M is the number of codebooks, C is the number of codewords per codebook, and d is the dimensionality.

2. Importance Scoring. The importance score for each of the N cached tokens is then calculated by summing the corresponding M values retrieved from the LUT. This involves $M - 1$ scalar additions per token:

$$\text{FLOPs}_{\text{addition}} = (M - 1)N \tag{7}$$

3. Pruned Attention Computation. After selecting, we perform the standard dot-product attention only on the subset of keys retained after pruning. Let ρ denote the pruning ratio, such that $N_{\text{pruned}} = (1 - \rho)N$. The FLOPs for this sparse computation are:

$$\text{FLOPs}_{\text{pruned}} = 2d(1 - \rho)N \tag{8}$$

Total FLOPs. The total number of FLOPs for the AQ-based $QK^\top$ computation is the sum of these three parts:

$$\text{FLOPs}_{\text{LAPrune}} = (M - 1 + 2(1 - \rho)d)\, N + 2MCd \tag{9}$$

E.2 Comparison with Original Attention

In comparison, the original self-attention mechanism computes the full $QK^\top$ matrix with:

$$\text{FLOPs}_{\text{orig}} = 2dN \tag{10}$$

Given our default configuration ($M = 32, 1 - \rho = 2\%$), the growth rate $(M - 1 + 2(1 - \rho)d)$ is significantly lower than $2d$. This theoretical reduction confirms that LAPrune is substantially more efficient than the original method, especially as the context length N scales during long-context inference. To further translate these theoretical gains into wall-clock speedups, we implement a custom CUDA kernel that fuses the LUT pre-computation and the subsequent additions.

F Efficiency Evaluation Setup

We evaluate system efficiency on LLaMA3.1-8B-Instruct in FP16 using one RTX PRO 6000D. All methods run under the same software stack, and results are averaged over three independent runs.

F.1 Inference Stages

We benchmark three inference stages: prefill, including AQ encoding for LAPrune; steady-state decoding, reported as throughput in tokens/s; and end-to-end generation, including prefill and decoding of 2,048 tokens.

F.2 Hyperparameters

We use context lengths from 16K to 128K and decode batch sizes of 1, 2, and 4. LAPrune uses a 2% retrieval budget plus a 64-token local encoding buffer, with full attention in layers 0 and 1. Reported decode throughput includes the complete inference pipeline. HF SDPA runs out of memory at 128K with batch size 4.

F.3 Scoring Kernel Comparison

Figure 5(f, g) compares LAPrune's grouped-GQA CUDA scorer with Spotlight's packed Triton scorer and an explicit PyTorch AQ lookup-and-sum reference. The microbenchmarks use FP16 inputs, batch size 1, 32 query heads, 8 KV heads, head dimension 128, and 32 codebooks with 256 entries each. GPU scoring latency includes query-side preparation and scoring with precomputed key codes or hashes.

G Additional Downstream Results

For Table 5, RULER uses 100 samples per task at 128K. MagicPIG uses $K = 10$, $L = 150$, and $Q = 0$, with 4 sink tokens and a local window of 16 tokens on LongBench and 64 tokens on RULER. Its nominal 98% pruning setting refers to an expected LSH sampling budget rather than a fixed retained-token count.

For the LLaMA2-7B results in Table 11, RULER uses 100 samples per task at 4K. LongBench uses 200 samples per task, except LCC and RepoBench-P, which each use 500 samples. MagicPIG uses the same LSH and sink/local settings described above.

LLaMA2-7B Downstream Task Performance. We further evaluate LAPrune on two benchmarks: LongBench, which covers diverse real-world tasks such as QA, summarization, and code completion, and RULER, a retrieval-oriented benchmark that directly probes how faithfully the pruning proxy preserves the relevant tokens. Table 11 reports the results of LLaMA2-7B under a 98% KV pruning ratio. On LongBench, LAPrune achieves an average score of 49.20, surpassing MagicPIG at 39.26 and Spotlight at 48.61 and narrowing the gap to the Oracle Top-k to only 0.32 points. On RULER, LAPrune matches Oracle Top-k at 0.698, outperforming Spotlight at 0.665 and MagicPIG at 0.325. The advantage is especially clear on MK3, where LAPrune scores 0.36, matching Oracle, compared with 0.21 for Spotlight and 0.11 for MagicPIG. Together, these results demonstrate that LAPrune approaches exact-logit selection on both diverse downstream tasks and targeted retrieval.

H Additional Visualizations

We provide extended visualizations of the selection accuracy (IoU) across all layers and attention heads for LLaMA2-7B, LLaMA3-8B, and Qwen2.5-7B at pruning ratios of 80% and 98%. As

Table 11: Downstream results on LongBench and RULER (LLaMA2-7B, 98% pruned).

		LongBench								RULER				
Method	**Config.**	Qsp	QMS	TREC	TQA	SAM	LCC	RP	**Avg.**	MK2	MK3	MQ	MV	**Avg.**
LLaMA2-7B	Baseline	9.23	21.32	67.50	89.02	35.48	67.99	61.20	50.25	0.86	0.49	0.87	0.72	0.735
Oracle Top-K	98% Pruned	8.61	21.13	65.00	88.35	34.31	67.57	61.68	49.52	0.79	0.36	0.87	0.77	0.698
MagicPIG	98% Pruned	8.36	18.97	65.50	36.41	29.26	62.06	54.26	39.26	0.48	0.11	0.38	0.33	0.325
Spotlight	98% Pruned	8.58	20.77	63.50	88.28	30.45	67.33	61.37	48.61	0.79	0.21	0.89	0.77	0.665
LAPrune (Ours)	98% Pruned	8.87	19.98	66.00	87.78	32.34	67.98	61.46	**49.20**	0.78	0.36	0.87	0.78	**0.698**

illustrated in Figure 6, LAPrune maintains superior and more stable retrieval fidelity compared to the Spotlight baseline.

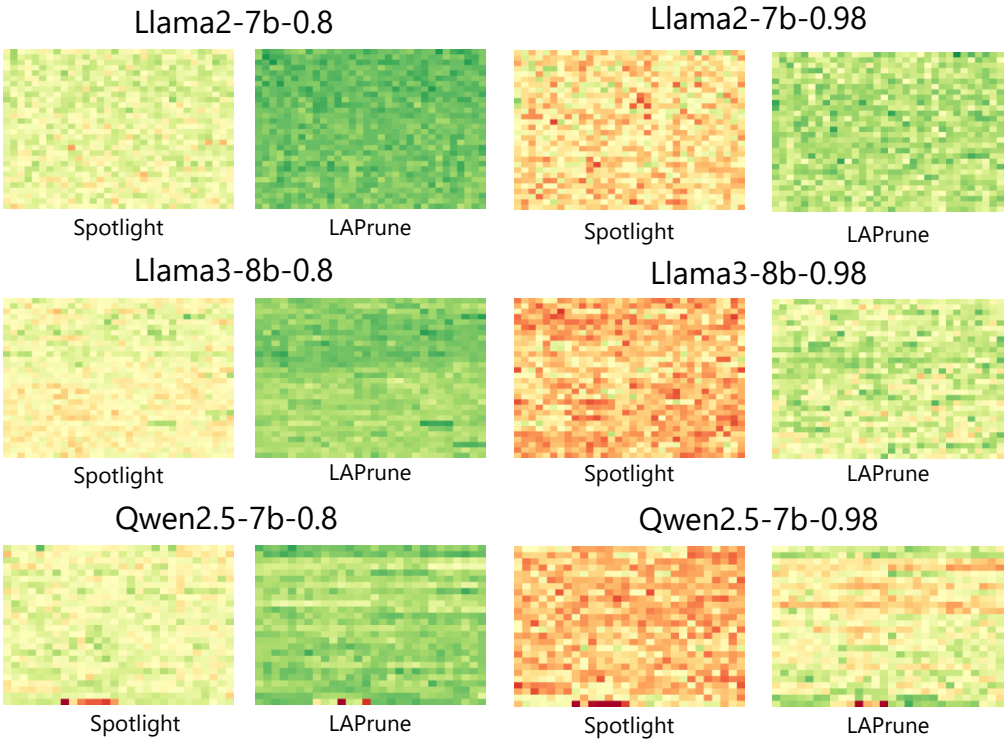

Figure 6: Complete IoU heatmaps for Spotlight and LAPrune. Higher (greener) values indicate closer alignment with the Oracle Top-k tokens.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state our main claims regarding logits-aligned proxy design, improved Top-k set recovery, and long-context decoding speedups; all are directly supported by experiments in Section 4 (Tables 1–5, Figures 3–5).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Appendix A, noting that LAPrune's efficiency advantage is concentrated in long-context regimes and that dense attention with FlashAttention-2 remains preferable below 16K context.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not contain formal theorems or proofs. Appendix E provides a FLOPs-based complexity analysis, which is a direct calculation rather than a theoretical claim requiring proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 4 specifies the models, datasets, pruning ratios, and codebook configurations. Appendix D details the calibration procedure with full pseudocode (Algorithm 1), data sources, and hyperparameters. Appendix F describes the efficiency benchmarking protocol, and Appendix D.5 specifies the LongBench and RULER configurations.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our implementation, pretrained AQ codebooks, training scripts, and evaluation instructions are publicly available at `https://github.com/ymy9599/LAPrune`. All datasets used (PG19, ProofPile, CodeParrot, LongBench, RULER) are publicly accessible.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Hyperparameters, model configurations, calibration data sampling, and evaluation protocols are detailed in Section 4 and Appendices D, F, and D.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Efficiency results are averaged over three independent runs. Quality evaluations use fixed checkpoints, codebooks, and greedy decoding. We do not report confidence intervals or multi-seed uncertainty estimates for the quality results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Efficiency and whole-model memory measurements use a single NVIDIA RTX PRO 6000D GPU. Model configurations, batch sizes, context lengths, and benchmarking settings are provided in the experimental setup and efficiency appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available pre-trained language models and standard benchmarks. It does not involve human subjects, sensitive data, or any deviation from the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: LAPrune lowers the cost of long-context LLM inference, with positive impact on accessibility and energy efficiency. As with other LLM efficiency methods, faster inference may also lower the barrier for scaling known LLM harms; standard deployment-layer mitigations remain applicable.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release new pre-trained models or datasets that pose misuse risks. The released code is an inference-time efficiency technique applied to existing public models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All models and datasets are properly cited in the paper; we use them under their respective licenses for research purposes.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The released implementation and pretrained AQ codebooks are documented in the public repository, with instructions for installation, codebook training, LongBench and RULER evaluation, and decode benchmarking.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not used as a component of the methodology.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.