WAVEKV: WHERE ATTENTION MEETS SIGNAL PROCESSING FOR KV-CACHE COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Key-value (KV) caching enables efficient text generation with Large Language Models (LLMs), but its near-linear growth with generated tokens creates increasing memory demands and a memory wall. Existing compression methods rely on query-dependent token eviction or model fine-tuning, which either discard useful KV states or require model adaptation, limiting simultaneous reductions in KV storage and attention computation. Motivated by structured attention patterns and signal-energy redundancy of KV states in the frequency and discrete-wavelet domains, we propose **WaveKV**, a training-free KV-cache compression framework that combines discrete cosine transforms (DCT) and discrete wavelet transforms (DWT) to jointly reduce KV-cache storage and attention computation. **WaveKV** first performs a one-time, query-independent attention calibration to capture positional attention structure and retrieval-sensitive layers. The resulting positional policy generalizes across datasets and models to guide adaptive DCT-based compression. It then uses adaptive, attention-guided DWT-based hierarchical routing to selectively access the compressed representation during attention computation without directly evicting tokens. Across LLaMA-3.1-8B, Phi3-medium-128k, and Qwen3-8B, **WaveKV** achieves $\sim$**2**$\times$ KV-cache compression, improving average LongBench and RULER scores over similarly compressed baselines by **40%** and **75%**, respectively, while maintaining competitive language-modeling and long-context capability. At the system level, **WaveKV** reduces attention scoring compared to dense full-KV by up to **11.7**$\times$ and peak memory by **40%** at 64K context, while supporting 128K contexts where full KV runs out of memory. Our code is available at `https://anonymous.4open.science/r/WaveKV-95DF`

1 INTRODUCTION

LLMs rely on KV caches for efficient autoregressive generation by storing and reusing key and value states from previously processed tokens, avoiding repeated computation during decoding. However, appending new states for every token causes KV-cache memory to grow linearly with context length, increasing memory-capacity, bandwidth, and attention-computation demands and limiting inference throughput. A principled approach is therefore to compress the KV cache.

Existing KV cache compression methods reduce attention overhead by selecting or evicting KV states according to estimated token importance. SnapKV Li et al. (2024), PyramidKV Cai et al. (2024b), FastKV Jo et al. (2026), and PyramidInfer Yang et al. (2024) rely primarily on attention scores from recent queries, while TreeKV He et al. (2025) models regional variation in token contributions and WaveFilter Yang et al. (2026) performs DWT-based token selection. However, once tokens are evicted, they cannot contribute to future attention, which can degrade performance when relevance shifts or information is distributed across the context. Figure 1 compares these methods, where Figure 1a illustrates the degradation of PyramidInfer relative to vanilla LLaMA-3.1-8B (Dense) on aggregation-heavy RULER tasks. To avoid explicit token eviction, LoCoCo Cai et al. (2024a) and Activation Beacon Zhang et al. (2025) learn compact KV representations, but require additional memory and model-specific fine-tuning. Signal-processing approaches instead exploit sequence-level structure. FreqKV Kai et al. (2026) uses DCT-based frequency compression based on low-frequency KV energy concentration, while FAEDKV Li et al. (2025) applies Fourier-domain compression to preserve contextual information without token eviction. However, frequency compression alone does not efficiently determine which compressed representations should participate

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

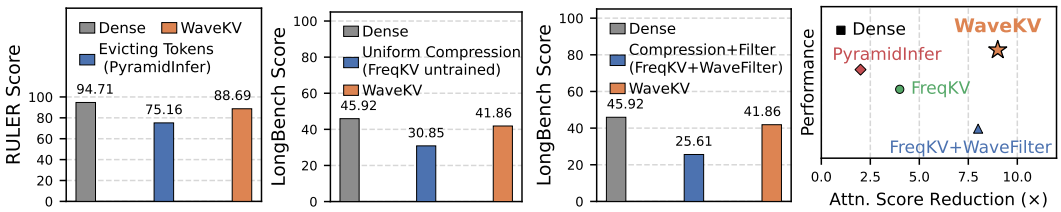


Figure 1: Component analysis: (a) token eviction, (b) uniform DCT, and (c) naive DCT+DWT degrade quality (d) WaveKV outperforms baselines while achieving the same compression rate.

in attention. Effective KV-cache compression therefore requires both **frequency-level conciseness** to reduce representation cost and **token-level selection** to focus computation on attention-relevant information.

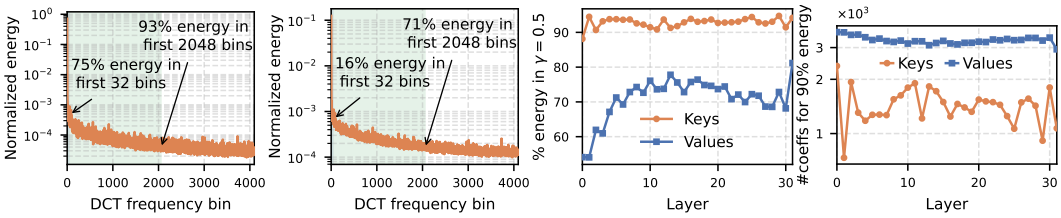
However, existing methods largely optimize only one of these objectives, while jointly achieving high compression and strong inference performance introduces three challenges. **1) Training-free compression:** The method should avoid model-specific fine-tuning and perform *query-independent cache compression*, rather than relying on query-derived importance scores. **2) Non-uniform compression:** KV information and spectral characteristics vary across context regions He et al. (2025); Yang et al. (2024), making uniform compression inefficient. As shown in Figure 1b, uniform DCT truncation in untrained FreqKV at 0.5 retention degrades performance, showing that frequency compression alone is insufficient and motivating adaptive compression. **3) Token selection in the compressed representation:** Frequency compression breaks the direct correspondence between transformed components and original tokens, complicating token selection. Figure 1c further shows that naively combining WaveFilter Yang et al. (2026) with FreqKV Kai et al. (2026) substantially degrades performance, motivating adaptive and coordinated DCT compression with DWT routing.

In this paper, we propose **WaveKV**, a training-free hierarchical DSP framework for KV-cache compression and efficient attention in off-the-shelf LLMs. **WaveKV** exploits attention structure to jointly reduce cache storage and attention workload, with query-independent compression and routing policies but query-dependent tree traversal, enabling scalable long-context inference while maintaining model performance (Figure 1d). The main contributions are as follows:

- We propose **WaveKV**, a novel training-free DSP-based framework that couples *KV-cache representation conciseness* and *attention sparsification*, using one-time attention calibration to guide adaptive DCT compression and Haar-DWT-based hierarchical token selection across inference inputs. Rather than simply combining independent compression and selection techniques, WaveKV uses attention to coordinate *what to compress* and *where to compute*.
- We evaluate **WaveKV** across long-context language modeling and understanding workloads without model fine-tuning. WaveKV achieves $\sim$**2**$\times$ KV-cache compression, improving average LongBench and RULER scores over similarly compressed baselines by **40%** and **75%** respectively, demonstrating competitive long-context capability under compressed KV representations.
- We develop an optimized implementation of **WaveKV** with batched decoding and custom Triton kernels, reducing KV/tree states scored by up to **11.7**$\times$ and peak memory by **40%** at 64K compared with full-KV attention, while supporting 128K contexts where full KV runs out of memory. For long-context prefill, **WaveKV** reduces time-to-first-token (TTFT) by $\sim$**2**$\times$ over baselines.

2 Background and Motivation

Frequency level conciseness using DCT: DCT-based compression of pre-RoPE key (K) and value (V) states exhibits strong energy concentration in lower-frequency bins as shown in Figures 2a and 2b, consistent with FreqKV Kai et al. (2026). However, K and V exhibit substantially different frequency redundancy. Averaged across layers and heads, for 4K context the first 2K coefficients ($\gamma = 0.5$) retain approximately 93% of K energy (Figure 2a) (L2 norm of each key vector), but only 71% of V energy (Figure 2b), as highlighted by the green boxes. Energy con-

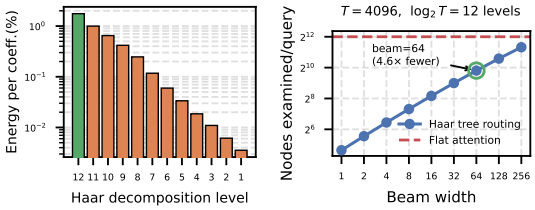


(a) (b) (c) (d)

Figure 2: (a-b) Frequency redundancy varies across K and V. (c) Fixed retention affects efficient energy preservation. (d) To preserve accuracy, WaveKV fixes the energy budget.

centration also varies substantially across layers (Figure 2c), thus making a fixed retention ratio insufficient for consistent information preservation across KV states and layers. Consequently, **WaveKV** allocates coefficients independently for each layer and for KV according to a target energy threshold, maintaining consistent energy preservation across the KV cache (Figure 2d).

Token selection using Wavelet:Haar-wavelet decomposition Zhang (2019) reveals scale-dependent structure in the key states, with coefficient energy decreasing from coarse level $\log_2 T$ to fine levels, as shown for $T = 4096$ in Figure 3a. This enables hierarchical tree routing, retaining the top-k branches (beam width) at each level and progressively expanding to finer resolutions. With $k = 64$, the router examines 895 nodes instead of 4096, reducing the search space by $4.6\times$ (Figure 3b). More generally, routing over T tokens examines $O(2k \log_2 T)$ nodes versus $O(T)$, with greater savings at longer contexts. However, applying DWT-based token selection directly to the compressed KV cache degrades attention performance (Figure 3b).



(a) (b)

Figure 3: Characterization of DWTs in terms of (a) energy of K vectors in coarse levels; and (b) attention-scoring reduction and DWT search beam.

3 WaveKV for KV-Cache Compression

In this section, we present **WaveKV**, shown in Figure 4. WaveKV first performs one-time offline calibration to construct a generalizable attention profile that remains fixed during inference and determines where and how aggressively KV states are compressed (Section 3.1). For retrieval-based workloads, the profile can be calibrated further for the target model. This profile guides adaptive DCT energy retention across KV-cache regions to concentrate compression where model quality is least affected (Section 3.2). During inference, Haar-DWT-based hierarchical routing uses the same profile to adaptively select relevant regions for attention computation (Section 3.3).

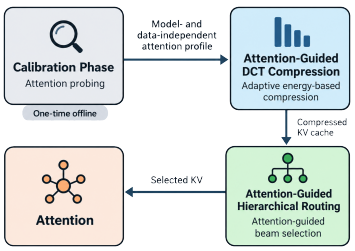


Figure 4: Overview of WaveKV.

3.1 Attention-Aware Calibration

The calibration phase characterizes inherent attention structure for KV-cache compression through one-time calibration on representative contexts, producing a fixed profile reused across inputs, models, and context lengths without model-weight optimization or task-specific fine-tuning. Figure 5 shows 4096-token attention heatmaps for Llama-3.1-8B on Timaeus Pile-GitHub Gao et al. (2020). Similar structures are observed across Qwen3-8B, Pile tasks, and context lengths as shown in Appendix D. Despite variations in content, length, and model, attention exhibits a consistent positional structure. This leads to the following key findings:

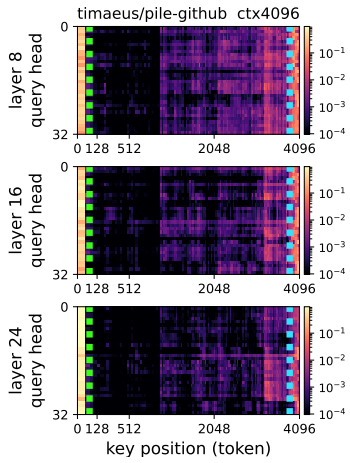
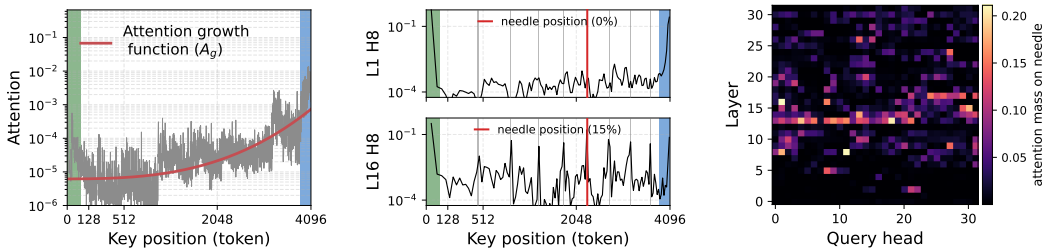


Figure 5: Attention heatmap across query heads and key positions at context length of 4096.

Finding 1: Three-part attention structure. Across layers and heads, attention consistently exhibits a three-part positional structure: a strong attention-sink *Head* covering the first ∼64 tokens, a relatively sparse *Info* region, and a high-attention *Tail* covering the last ∼256 tokens. The Head and Tail boundaries are determined using a cumulative attention-mass (defined in Appendix D.1) threshold and remain fixed across context lengths, as indicated by the green and blue-dotted lines in Figure 5.

Finding 2: Attention growth. Attention increases gradually for older tokens and rapidly toward recent tokens. Aggregating attention in the *Info* region yields a smoothly increasing profile, approximated by a convex quadratic A_g on a logarithmic scale (red line in Figure 6a). This function termed as *attention growth function*, guides position-adaptive compression.

Finding 3: Retrieval-aware calibration. The positional structure may fail for retrieval workloads, where relevant tokens can occur anywhere. A key observation is that attention mass at the needle position varies substantially across layer-heads, as shown in (Figures 6b–6c). **WaveKV** therefore performs a one-time needle-in-a-haystack calibration to identify retrieval-sensitive layer-heads, which guide compression and routing. The calibration is model-specific but performed only once and reused during inference. Details regarding the calibration settings are discussed in Appendix D.1.



(a) Convex quadratic attention growth function

(b) Needle attention mass varies across layers.

(c) Retrieval-sensitive layer guides compression.

Figure 6: Key findings leading to attention-aware calibration

3.2 Attention-guided DCT based compression

Adaptive DCT Blocks: The calibrated attention-growth function A_g provides a position-dependent estimate of attention demand across the *Info* region. Because this profile varies spatially, a fixed DCT block size would either over-compress regions with rapidly changing attention or under-compress slowly varying regions. We therefore adapt block sizes according to the local attention slope. Let $s(p)$ denote the local log-slope of A_g at position p (red line in Figure 7):

$$s(p) = \frac{d \ln(A_g(p))}{dpos}. \tag{1}$$

Since A_g is convex, $|s(p)|$ is smaller in the distant *Info* region and increases toward the recent tail. We define s_{ref} as the 25th percentile of $|s(p)|$ (Appendix. D.2) and determine the continuous block size as

$$B(p) = \text{clip}\left(B_{\max}\frac{s_{\text{ref}}}{|s(p)|}, B_{\min}, B_{\max}\right). \tag{2}$$

Thus, slowly varying regions receive larger blocks, while rapidly changing regions receive smaller ones. We set $B_{\max} = 256$ (Appendix D.2) and snap block sizes to the dyadic palette $256, 128, 64, \ldots$. For example, $|s| \approx 2s_{\text{ref}}$ gives a 128-token block. The calibrated slope increases by $\sim 3.6\times$ toward the recent tail, producing the $256 \to 128 \to 64$ LADDER shown in Figure 7. Across context lengths, B_{Info} comprises approximately 41% 256-token, 37% 128-token, and 22% 64-token blocks. The final structure is $\{B_{\text{Head}}, B_{\text{Info}}, B_{\text{Tail}}\}$, with fixed $B_{\text{Head}} : 64$ and $B_{\text{Tail}} : 256$.

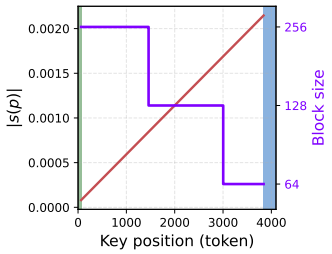


Figure 7: Adaptive block-size ladder for the *Info* region guided by attention profile.

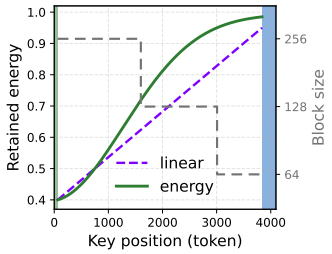


Figure 8: Energy allocation for DCT blocks based on attention growth function $A_g(p)$

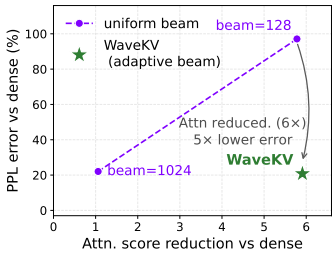


Figure 9: WaveKV achieves same performance at 6× attention-scoring reduction

Per-Block Energy Allocation: While block size determines representation granularity, a complementary energy-allocation rule controls the amount of information retained within each block. Using $A_g(p)$ as the importance profile, we assign a target retained-energy fraction:

$$e(p) = \text{clip}\left(1 - (1 - e_{\min})\frac{A_g(p_{\text{far}})}{A_g(p)}, e_{\min}, 1\right) \tag{3}$$

Let $\rho = A_g(p_{\text{tail}})/A_g(p_{\text{far}})$ denote the measured attention dynamic range, which is ≈ 50–67 across our calibration results. Assuming a near-lossless tail energy cap of $e_{\max} = 0.99$, we get

$$e_{\min} = 1 - (1 - e_{\max})\rho, \tag{4}$$

giving $e_{\min} \approx 0.4$. Rather than directly following the nonlinear profile (green line in Figure 8), we use a simple linear schedule between the calibrated endpoints, $e_{\min} = 0.4$ in the distant region and $e'_{\max} = 0.95$ near the recent tail (purple dotted line Figure 8). This avoids profile-specific fitting while preserving the observed increase in information allocation toward recent tokens. For each block, we retain the lowest-frequency DCT coefficients until their cumulative energy reaches the target $e(p)$. Consequently, older regions receive more aggressive frequency compression, while regions near the recent tail retain more DCT energy.

Retrieval-Aware Energy Allocation. In retrieval-oriented workloads, relevant tokens may occur anywhere in the *Info* region, while only a subset of layers may retrieve them. A uniform compression budget can therefore discard information needed by retrieval-sensitive layers. **WaveKV** addresses this using the layer-wise retrieval profile obtained from the one-time needle-in-a-haystack calibration (Section 3.1). For calibration trial d, a head h in layer ℓ is considered to retrieve the needle if its peak attention within the *Info* region is within $\pm T_{\text{needle}}$ tokens of the needle position and exceeds $\theta = 0.05$:

$$\text{hit}_{\ell,h}^{(d)} = \begin{cases} 1, & \text{if } |p_{\ell,h}^{(d)} - p_{\text{needle}}^{(d)}| \le T_{needle} \text{ and } w_{\ell,h}^{(d)} \ge \theta, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

where $p_{\ell,h}^{(d)}$ is the peak-attention position, $p_{\text{needle}}^{(d)}$ is the needle position, and $w_{\ell,h}^{(d)}$ is the corresponding peak attention weight. We then define the layer retrieval frequency as the fraction of N trials in which at least one head in layer ℓ retrieves the needle:

$$f_\ell = \frac{1}{N}\sum_{d=1}^{N}\left[\max_h \text{hit}_{\ell,h}^{(d)} = 1\right]. \tag{6}$$

Based on retrieval frequency f_ℓ, layers are assigned to PROTECT, SOFT, or BLIND tiers using consistency threshold, $\phi = 0.5$:

$$\text{tier}(\ell) = \begin{cases} \text{PROTECT}, & f_\ell \ge \phi, \\ \text{SOFT}, & 0 < f_\ell < \phi, \\ \text{BLIND}, & f_\ell = 0, \end{cases} \qquad e_\ell = \begin{cases} 1.0, & \text{PROTECT}, \\ 0.5, & \text{SOFT}, \\ 0.2, & \text{BLIND}. \end{cases} \tag{7}$$

Thus, retrieval-sensitive layers receive larger energy budgets, while retrieval-blind layers are compressed more aggressively. The DCT block structure remains unchanged and operates only along the token dimension. e_ℓ determines the number of low-frequency coefficients retained in each layer.

3.3 COMPUTE ON COMPRESSED CACHE THROUGH ADAPTIVE DWT

Haar-DWT Hierarchical Routing: **WaveKV** uses the Haar-DWT hierarchy as a query-independent index with query-dependent traversal. That is, the compressed KV representation, tree topology, and calibrated beam configuration are constructed without using the current decode query. The current query is used only to score nodes while traversing the fixed hierarchy. After prefill, retained KV rows form a binary tree with subtree-mean keys. During decoding, the router scores node means against the query, keeps the top-(k) nodes, expands them, and repeats until at most ($2k$) leaves remain for exact attention. The routing-to-dense compute ratio is $\frac{C_r}{C_d} = \frac{k(\log_2 P+2)}{T}$, where ($T$) is the retained KV rows and $P = 2^{\lceil \log_2 T \rceil}$ is the padded tree size. Routing is beneficial when $k < T/(\log_2 P + 2)$, exposing the tradeoff between beam width, pruning error, and routing cost.

Attention-Adaptive Beam Width: To address this tradeoff, **WaveKV** uses the calibrated attention profile from Section 3.1 to jointly set block sizes, energy budgets, and beam widths. In the LADDER variant, each region uses a tree, with wider beams for high-attention regions, narrower beams for low-attention regions, and dense attention for recent blocks. In the TIER variant, PROTECT layers are attended densely, SOFT layers use wider beams, and BLIND layers use narrower beams. Thus, routing adapts by position or layer instead of using a uniform beam. As shown in Figure 9, **WaveKV** achieves $\approx 6\times$ attention-scoring reduction (as defined in Appendix E.5) while reducing model perplexity error by about $5\times$ relative to a uniform beam. Matching the same $6\times$ gain with a uniform beam at $k = 128$ increases model perplexity error to 97%. This isolates the benefit of attention-adaptive routing over a uniform beam. The beam configuration and hierarchy remain fixed across decode queries, while selected branches are query-dependent. Detailed settings and ablation studies are in Appendix D.2.

4 EVALUATIONS

In this section, we evaluate **WaveKV** on language modeling and long-context understanding tasks (Section 4.1). We also assess the throughput, latency, and memory requirements of **WaveKV** (Section 4.2). We evaluate **WaveKV** on three decoder-only LLMs from different families: **LLaMA-3.1-8B-Instruct** Grattafiori et al. (2024), **Qwen3-8B** Yang et al. (2025), and **Phi-3-medium-128k** Abdin et al. (2024), to generalize across architectures. We use **PG-19** Rae et al. (2020) for language modeling, **LongBench** Bai et al. (2023) and **RULER** Hsieh et al. (2024) for long-context understanding, and **GSM8K** Cobbe et al. (2021) and **MMLU** Hendrycks et al. (2021) for downstream reasoning. These cover long-sequence generation, retrieval, and reasoning. We report **PPL** Jurafsky & Martin (2025) on PG-19, task-specific metrics on LongBench, and **accuracy** on RULER, GSM8K, and MMLU. System efficiency is measured by **latency**, **throughput**, **attention-compute reduction**, and **peak memory**. More details are in Appendix E.

Table 1: PG-19 perplexity on LLaMA-3.1-8B.

Method	1K	2K	4K	8K	16K	32K
Full KV	7.90	8.42	7.30	7.93	6.68	8.25
FreqKV (fine-tuned)	7.51	7.93	7.71	8.20	6.97	8.63
FreqKV$_0$ (untrained)	7.90	8.44	8.50	8.77	7.47	9.30
PyramidInfer (eviction)	8.41	10.29	8.58	8.91	7.17	10.47
WaveFilter (KV intact)	7.87	8.38	7.33	7.87	6.65	8.21
FreqKV+WaveFilter	7.76	8.48	7.97	8.60	7.36	9.01
WaveKV	8.01	8.97	7.66	8.24	7.14	9.22

4.1 MODEL PERFORMANCE

4.1.1 LANGUAGE MODELING

We evaluate language-modeling quality on PG-19 using perplexity (PPL). Table 1 reports LLaMA-3.1-8B across context lengths. Full results are in Appendix G.1. WaveKV limits PPL degradation relative to Full KV, WaveFilter (DWT only), which retains the full cache, and fine-tuned FreqKV (DCT only) at a comparable $\sim$50% compression rate. Beyond 4K, WaveKV substantially outperforms the FreqKV+WaveFilter (naive DCT+DWT) combination, as aggressive frequency compression makes the naive composition increasingly lossy. At 32K, WaveKV reduces PPL from 9.30 to 9.22 on LLaMA-3.1-8B, from 11.51 to 10.53 on Qwen3-8B, and from 8.24 to 7.42 on Phi3-medium-128k over untrained FreqKV$_0$. Despite being both training-free and token-selective, WaveKV out-

Table 2: LongBench performance on LLaMA-3.1-8B.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Code		Avg.
	NarrativeQA	Qasper	MFQA-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RepoBench-P	
Full KV	24.38	39.26	55.60	55.23	47.17	31.43	34.54	25.11	26.40	73.97	92.04	44.06	63.73	57.10	47.86
GemFilter	19.59	8.01	29.53	19.41	11.65	11.44	34.98	24.14	24.53	69.86	93.07	43.77	32.29	43.70	35.66
SnapKV	0.57	0.28	2.17	3.74	1.72	2.01	0.57	8.92	0.00	34.29	16.39	9.45	3.06	6.48	37.40
PyramidKV	5.37	0.41	2.65	3.59	2.76	1.47	0.54	5.95	0.00	32.88	17.63	8.79	3.65	6.65	8.59
LaCache	24.88	10.88	25.01	16.98	12.90	11.56	31.61	23.49	25.47	72.60	92.50	43.64	44.12	46.41	36.54
FastKV	1.14	0.00	1.63	0.77	1.46	0.29	0.06	4.60	0.11	30.14	8.43	6.77	6.82	5.27	4.22
CaM	25.01	8.67	26.39	15.13	15.37	10.86	33.00	22.62	25.69	69.86	92.25	42.59	46.65	46.08	35.88
D2O	22.76	7.23	20.62	14.85	12.12	9.25	29.17	22.27	24.91	69.86	90.62	42.01	43.18	46.28	34.96
ThinK	3.42	0.28	2.59	4.22	3.01	2.56	0.53	8.88	0.00	35.62	20.02	8.50	3.12	6.31	8.96
StarKV	1.68	4.12	8.46	3.50	3.16	2.38	14.11	8.66	8.60	16.44	11.71	6.08	15.59	18.43	7.90
WaveFilter	24.18	23.02	45.93	51.66	33.72	26.33	17.53	22.54	7.61	57.53	83.30	37.03	39.59	43.71	36.69
FreqKV	19.03	26.06	35.28	45.51	30.11	18.25	27.24	21.70	22.21	52.05	84.89	40.33	59.27	50.74	38.05
PyramidInfer	23.57	31.53	34.60	50.89	41.85	24.95	29.06	24.04	24.07	63.01	91.00	42.33	63.05	59.19	<u>43.08</u>
WaveKV	22.59	28.52	38.84	51.31	45.79	27.45	28.43	23.10	23.99	64.38	91.67	43.88	70.91	62.96	**44.56**

performs training-free FreqKV$_0$, eviction-based PyramidInfer, and the naive FreqKV+WaveFilter combination, demonstrating the advantage.

4.1.2 Long Context Understanding

We evaluate **WaveKV** on LongBench, RULER, and Needle-in-a-Haystack against eviction-based GemFilter Shi et al. (2026), SnapKV Li et al. (2024), PyramidKV Cai et al. (2024b), LaCache Shi et al. (2025), and FastKV Jo et al. (2026), merging-based CaM Zhang et al. (2024), KVMerger Wang et al. (2024b), and D2O Wan et al. (2024), channel-compression ThinK Xu et al. (2025) and WaveFilter, and fine-tuning-based StarKV Bhatnagar et al. (2026) and FreqKV. Since **WaveKV's** compression decisions are independent of the current decode query, we also construct query-free variants of query-dependent baselines using key-only importance while preserving cache budgets, protected recent-token regions, and attention computation. Only cache construction and compression are modified. Queries are used normally during attention. Details are in Appendix E.7.

LongBench Evaluation: We report LongBench results in Table 2 for LLaMA-3.1-8B-Instruct, with complete task-level results for all models provided in Appendix G.2. All methods are evaluated at a comparable $\sim$**2**$\times$ KV-cache compression rate. On LLaMA-3.1-8B, **WaveKV** achieves an average score of 44.56, compared with 43.08 for PyramidInfer. Similar results are observed across Qwen3-8B and Phi3-medium-128k, where **WaveKV** achieves average scores of 43.19 and 37.12, respectively. Across the three model families, **WaveKV** improves the LongBench score by $\sim$**40%** over the average of the baselines, demonstrating that the proposed compressed representation preserves long-context understanding at a comparable compression rate.

RULER Evaluation: We report RULER results at 16K context in Table 3. **WaveKV** achieves 86.57% average accuracy on LLaMA-3.1-8B, remaining close to the full-KV baseline (89.08%) while outperforming WaveFilter (54.60%). Similar gains across Qwen3-8B and Phi-3-Medium-128K are shown in Appendix G.2. WaveKV outperforms the average of all baselines by $\sim$**70%**, indicating that WaveKV preserves long-context retrieval and reasoning more effectively at similar compression rate.

Table 3: RULER accuracy on LLaMA-3.1-8B at 16K context.

Method	FWE	QA	VT	NIAH	Avg.
Full KV	85.87	66.40	96.88	94.18	89.08
FreqKV	90.67	41.10	20.68	23.51	31.80
PyramidInfer	73.07	52.00	63.56	38.42	45.66
WaveFilter	28.00	62.50	65.20	54.63	54.60
WaveKV	**78.40**	**68.50**	**88.28**	**91.89**	**86.57**

Needle-in-a-Haystack: Figure 10 reports Needle-in-a-Haystack results, including the challenging NIAHmultikey2 setting with near-identical distractors. FreqKV is effective up to its 8K training regime but degrades once frequency compression is applied, while PyramidInfer suffers from irreversible token eviction. In contrast, WaveFilter and **WaveKV** preserve access to information across the context. WaveKV maintains a multi-resolution compressed representation, allowing tokens critical for retrieval to remain represented even at long context lengths.

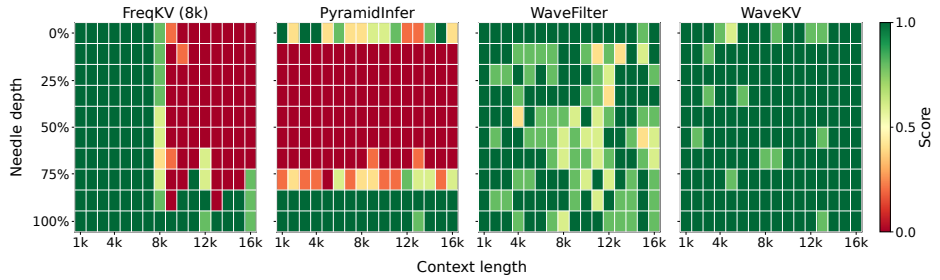


Figure 10: The results on Needle-in-a-Haystack for KV compression methods on LLaMA-3.1-8B.

4.1.3 Long Text Generation

We next evaluate whether KV-cache compression preserves autoregressive reasoning and knowledge performance on **GSM8K** and **MMLU**. Since generated information is repeatedly reused through the KV cache, aggressive compression can accumulate approximation errors across decoding steps. As shown in Table 4, **WaveKV** provides the strongest overall trade-off among methods that compress the KV cache similarly ($\sim 2\times$). On MMLU, WaveKV nearly matches the full-KV baseline with 69.39% accuracy, while GSM8K is more sensitive to compression, consistent with prior work showing that arithmetic reasoning is more vulnerable than world-knowledge tasks Liu et al. (2025). MMLU is predominantly knowledge-driven and comparatively less demanding of extended reasoning Wang et al. (2024a). Although WaveFilter achieves higher GSM8K accuracy, it retains the full KV cache and provides no KV-cache storage compression. Additional analysis is in Appendix G.

Table 4: Long text generation with reasoning on LLaMA-3.1-8B.

Method	GSM8K	MMLU
Full KV	82.50	69.04
GemFilter	1.00	15.50
StarKV	1.00	4.00
FastKV	3.00	34.00
FreqKV	43.50	64.65
PyramidInfer	42.00	68.16
WaveFilter	84.00	65.70
WaveKV	**74.50**	**69.39**

4.2 Hardware Performance

4.2.1 Attention-Scoring Reduction

We next evaluate the reduction in attention work achieved by WaveKV, measured as the ratio of KV rows scored by dense attention to the KV/tree rows scored by each method. This metric captures the reduction in attention scoring and KV reads and should not be interpreted as end-to-end wall-clock speedup. As shown in Figure 11, WaveKV scales favorably with context length, reaching $9.3\times$ and $11.7\times$ reduction in KV rows scored at 16K and 32K, respectively. In contrast, fixed-fraction methods like WaveFilter and PyramidInfer remain near $2\times$, while FreqKV trained at 8K reaches $3.8\times$ at 16K. Even though naive FreqKV+WaveFilter combination can reach similar reduction, its performance collapses.

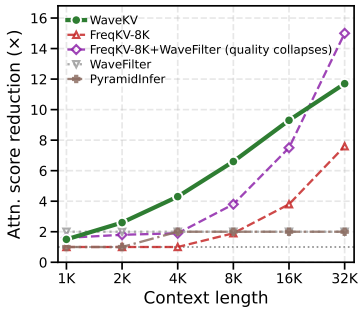


Figure 11: Reduction in KV/tree states scored during attention relative to dense attention.

4.2.2 Throughput

We next evaluate inference throughput, measured as the number of generated tokens processed per unit time. All throughput experiments are conducted with a batch size of four sequences, and the reported tokens/s values are normalized by the batch size to reflect per-sequence throughput. Table 5 reports realized end-to-end decoding throughput, which should be distinguished from the attention-scoring reduction in Figure 11. Where Full KV fits, WaveKV incurs a modest throughput overhead (22.4/23.1/22.4 vs. 25.3/25.4/24.9 tokens/s at 16K/32K/64K), indicating that reduced atten-

Table 5: PG-19 decoding throughput (tokens/s).

Context Length	Full KV	FreqKV	WaveKV
16K / 32K / 64K / 128K	25.3 / 25.4 / 24.9 / OOM	13.2 / 23.4 / 23.3 / 23.3	22.4 / 23.1 / 22.4 / 21.5

Table 6: Peak memory and recompressions on PG-19 at 16K, 32K, 64K, and 128K generated tokens.

Metric	Full KV	FreqKV	WaveKV
Peak Mem. (GB)	42.3 / 37.9 / 55.4 / OOM	44.5 / 30.3 / 30.3 / 30.3	46.8 / 32.1 / 33.2 / 35.4
# Recomp.	0 / 0 / 0 / –	5 / 9 / 17 / 33	4 / 9 / 18 / 36

tion work primarily improves memory scalability rather than wall-clock speed over optimized dense attention. WaveKV nevertheless sustains 21.5 tokens/s at 128K, where Full KV hits OOM.

4.2.3 Memory Requirement

We evaluate KV-cache memory usage in Table 6. At 16K, WaveKV incurs a modest overhead over Full KV (46.8 vs. 42.3 GB), however, the compression benefit grows with context length as WaveKV maintains a fixed cache. WaveKV uses 32.1 vs. 37.9 GB at 32K and 33.2 vs. 55.4 GB at 64K. Thus, WaveKV becomes more memory-efficient than Full KV between 16K and 32K, reducing peak memory by 40.1% at 64K and supporting 128K where Full KV runs out of memory.

Table 7: TTFT on PG-19 at 16K, 32K, 64K, and 128K context lengths, respectively.

Full KV	FreqKV	WaveFilter	WaveKV
1.680 / 4.030 / 10.703 / OOM	2.429 / 5.158 / 10.705 / 22.016	1.616 / 3.912 / 10.446 / OOM	2.424 / 4.674 / 9.421 / 19.301

4.2.4 Prefill Latency

Adaptive DCT compression (Section 3.2) during long-context prefill ($\geq 16K$) increases TTFT due to repeated ladder-based DCT computation, while A_g shows that attention-aware calibration is primarily needed for the most recent 8K–10K tokens, as earlier tokens carry negligible attention mass. **WaveKV** addresses this with lightweight DCT recompression during rolling prefill and full attention-guided compression only for recent tokens, improving TTFT scaling with bounded memory. Table 7 reports TTFT for LLaMA-3.1-8B with batch size two against Full KV, FreqKV, and WaveFilter. **WaveKV** achieves comparable TTFT across context lengths and, at 128K, remains operational while Full KV and WaveFilter run out of memory, with $\sim$15% lower TTFT than FreqKV. Further implementation details for long-context prefill, together with an ablation across batch sizes and context lengths, are provided in Appendix F.

5 Conclusion

We presented **WaveKV**, the first training-free compression framework to couple KV-cache compression with attention-guided token routing guided by a shared calibrated attention structure, combining adaptive DCT compression and Haar-DWT routing to reduce storage and attention computation. Across LLaMA-3.1-8B, Phi3-medium-128k, and Qwen3-8B, **WaveKV** achieves $\sim\mathbf{2}\times$ KV-cache compression, improving average LongBench and RULER scores over similarly compressed baselines by **40%** and **75%**, respectively, while maintaining competitive language-modeling and long-context capability. At long contexts, WaveKV reduces peak memory by $\mathbf{40.1}\%$ at 64K, providing $\mathbf{1.7}\times$ higher decoding throughput than FreqKV, sustains 21.5 tokens/s at 128K where Full KV is OOM, and reduces attention-scored KV/tree states by up to $\mathbf{11.7}\times$. These results demonstrate that attention-guided compression and routing preserve long-context quality while reducing memory and attention overhead without fine-tuning.

AI USE STATEMENT

An LLM coding assistant was used during this project for two tasks. This includes drafting and iterating on Python functions and Triton kernels. The authors reviewed and executed that code, and verified it against an independent unit test cases for the basic algorithms used in the kernels and functions. Furthermore, generative AI tools were also used for drafting and editing portions of this manuscript, which the authors reviewed, fact-checked against the measurements, and revised. Generative AI tools were not used to produce the reported experimental results. Citations have not been generated or selected without verification. The authors take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our work pertains to key-value compression using digital signal processing concepts for large language models. Only publicly available resources – such as datasets, models and baseline algorithms – have been used. The paper poses no ethical issues, including its motivation and experiments.

REPRODUCIBILITY STATEMENT

We have provided detailed implementation details at each step of our methodology in the paper. This is comprehensively elaborated in Section 3, along with a detailed background in Appendices B, C, and D. Detailed settings of our experiments are outlined in Section 4 and Appendix E. With this submission, we provide an anonymized code repository for reproducibility. We would be releasing a detailed version of the repository upon acceptance of the paper.

REFERENCES

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Nasir Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, 1974. doi: 10.1109/T-C.1974.223784.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Priyansh Bhatnagar, Ashkan Moradifirouzabadi, Se-Hyun Yang, SeungJae Lee, Jungwook Choi, and Mingu Kang. Star-kv: Low-rank kv cache compression via soft thresholding for adaptive rank control. *arXiv preprint arXiv:2606.08382*, 2026.

Ruisi Cai, Yuandong Tian, Zhangyang Wang, and Beidi Chen. Lococo: Dropping in convolutions for long context compression. *arXiv preprint arXiv:2406.05317*, 2024a.

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

Hanting Chen, Zhicheng Liu, Xutao Wang, Yuchuan Tian, and Yunhe Wang. Dijiang: Efficient large language models through compact kernelization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 7103–7117. PMLR, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359. Curran Associates, Inc., 2022. doi: 10.52202/

068431-1189. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf.

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. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11091–11104. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/ding24i.html.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ziwei He, Jian Yuan, Haoli Bai, Jingwen Leng, and Bo Jiang. Treekv: smooth key-value cache compression with tree structures. *arXiv preprint arXiv:2501.04987*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

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

Dongwon Jo, Jiwon Song, Yulhwa Kim, and Jae-Joon Kim. Fastkv: Decoupling of context reduction and kv cache compression for prefill-decoding acceleration. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32167–32186, 2026.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2025. Online manuscript, 3rd edition draft.

Jushi Kai, Yixuan Wang, Boyi Zeng, Haoli Bai, Bo Jiang, Ziwei He, and Zhouhan Lin. Freqkv: Key-value compression in frequency domain for context window extension. In *International Conference on Learning Representations*, volume 2026, pp. 105568–105587, 2026.

Runchao Li, Yao Fu, Mu Sheng, Xianxuan Long, Haotian Yu, and Pan Li. Faedkv: Infinite-window fourier transform for unbiased kv cache compression. In *EMNLP (Findings)*, pp. 16856–16866, 2025.

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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Xiang Liu, Zhenheng Tang, Hong Chen, Peijie Dong, Zeyu Li, Xiuze Zhou, Bo Li, Xuming Hu, and Xiaowen Chu. Can llms maintain fundamental abilities under kv cache compression? *arXiv preprint arXiv:2502.01941*, 2025.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 31932–31951, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/874a4d89f2d04b4bcf9a2c19545cf040-Paper-Conference.pdf.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

Dachuan Shi, Yonggan Fu, Xiangchi Yuan, Zhongzhi Yu, Haoran You, Sixu Li, Xin Dong, Jan Kautz, Pavlo Molchanov, and Yingyan Celine Lin. LaCache: Ladder-shaped KV caching for efficient long-context modeling of large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54892–54903. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/shi25b.html.

Zhenmei Shi, Yifei Ming, Xuan-Phi Nguyen, Yingyu Liang, and Shafiq Joty. Discovering the gems in early layers: Accelerating long-context llms with 1000x input token reduction. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 13839–13857, 2026.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.127063. URL https://www.sciencedirect.com/science/article/pii/S0925231223011864.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, and Mi Zhang. D2o: Dynamic discriminative operations for efficient generative inference of large language models. *arXiv preprint arXiv:2406.13035*, 2:3, 2024.

Xuefei Wang, Haoyu Tang, Tianyuan Liang, Zhibin Wang, Yupeng Hu, and Weili Guan. Resonating with rope: Spectral quantization for high-fidelity key cache compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 37328–37348, 2026. doi: 10.18653/v1/2026.acl-long.1732.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, 2024a.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*, 2024b.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 62143–62156, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/9b77f07301b1ef1fe810aae96c12cb7b-Paper-Conference.pdf.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5e5fd18f863cbe6d8ae392a93fd271c9-Paper-Conference.pdf.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 37228–37253, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5c1ddd2e59df46fd2aa85c833b1b36ed-Paper-Conference.pdf.

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. In *International Conference on Learning Representations*, volume 2025, pp. 56691–56709, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3258–3270, 2024.

Jinnan Yang, Yan Wang, Zhen Bi, Kehao Wu, Xiaojie Li, Jungang Lou, Zechao Li, and Jing Liu. Wavefilter: Enhancing the long-context capability of diffusion llms via wavelet-guided kv cache filtering. *arXiv preprint arXiv:2606.00724*, 2026.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving. In M. Zaharia, G. Joshi, and Y. Lin (eds.), *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/file/dbf02b21d77409a2db30e56866a8ab3a-Paper-Conference.pdf.

Dengsheng Zhang. Wavelet transform. In *Fundamentals of image data mining: Analysis, Features, Classification and Retrieval*, pp. 35–44. Springer, 2019.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. Long context compression with activation beacon. In *International Conference on Learning Representations*, volume 2025, pp. 101932–101948, 2025.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient llms inference. In *Forty-first international conference on machine learning*, 2024.

A Discussions and Limitations

Attention reduction and throughput. WaveKV reduces the number of KV/tree states scored during attention by up to $11.7\times$. However, this does not translate proportionally to end-to-end throughput. Prior work has shown that lower attention FLOPs do not always yield wall-clock speedup because GPU execution also depends on memory traffic, locality, and kernel efficiency Dao et al. (2022); Ye et al. (2025). WaveKV also introduces hierarchical traversal, top-k selection, and sparse KV accesses. These operations are not captured by the attention-scoring metric. The remaining Transformer computation is also unchanged. Since we do not separately profile these costs, we do not attribute the throughput gap to a specific component. Future work can explore fused routing and attention kernels, IO-aware execution, and hardware-aware cache layouts to better translate reduced attention work into end-to-end speedup Dao et al. (2022); Ye et al. (2025).

Generalization across model families. WaveKV applies the same attention-based structural policy across model families. This includes the Head-Info-Tail decomposition, attention-growth formulation, and compression and routing policy. Prior work shows that long-context retrieval is often concentrated in a subset of attention heads, while the identity and behavior of these heads can vary across models Wu et al. (2025); Xiao et al. (2025). Attention-sink behavior has also been observed across multiple Transformer families Xiao et al. (2024). These observations support the use of a common structural prior, but do not imply that the same profile is optimal for every model. This is particularly relevant to Phi-3-128K. Its long-context extension uses LongRoPE, which changes positional scaling in attention Ding et al. (2024). RoPE-based position handling can directly affect long-context attention structure Su et al. (2024); Peng et al. (2024). Although WaveKV remains competitive on Phi3-medium-128k, more model-specific calibration may further improve performance. Future work can adapt the profile shape, region boundaries, energy allocation, and routing parameters more closely to each model family.

B Discrete Cosine Transform for KV-Cache Compression

B.1 Discrete Cosine Transform

The Discrete Cosine Transform (DCT) represents a finite sequence as a weighted sum of cosine basis functions and is widely used for transform coding because correlated signals often concentrate

a large fraction of their energy in a small number of coefficients Ahmed et al. (1974). Among its variants, the DCT-II is commonly used for compression and is the transform employed in our KV-cache compression pipeline.

Consider a length-T sequence $\mathbf{x} = [x_0, \ldots, x_{T-1}]^\top$. Its orthonormal DCT-II is defined as

$$X_k = \alpha_k \sqrt{\frac{2}{T}} \sum_{t=0}^{T-1} x_t \cos\left[\frac{\pi}{T}\left(t + \frac{1}{2}\right) k\right], \qquad k = 0, \ldots, T-1, \tag{8}$$

where

$$\alpha_k = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0, \\ 1, & k > 0. \end{cases} \tag{9}$$

The inverse transform is

$$x_t = \sqrt{\frac{2}{T}} \sum_{k=0}^{T-1} \alpha_k X_k \cos\left[\frac{\pi}{T}\left(t + \frac{1}{2}\right) k\right]. \tag{10}$$

For a KV tensor $\mathbf{K} \in \mathbb{R}^{T \times d}$, we apply the DCT independently along the token dimension for each feature channel, i.e.,

$$\widehat{\mathbf{K}}_{k,j} = \alpha_k \sqrt{\frac{2}{T}} \sum_{t=0}^{T-1} \mathbf{K}_{t,j} \cos\left[\frac{\pi}{T}\left(t + \frac{1}{2}\right) k\right], \tag{11}$$

and analogously for $\mathbf{V}$. Direct evaluation of Eq. 11 requires $\mathcal{O}(T^2 d)$ operations. However, the DCT can be reduced to FFT-like computations, resulting in $\mathcal{O}(Td \log T)$ complexity Ahmed et al. (1974). For a multi-head attention layer, d can equivalently be interpreted as the total dimensionality across KV heads; hence, the transform cost remains quasi-linear in context length.

The use of frequency-domain representations has recently been explored for efficient LLM inference. DiJiang Chen et al. (2024) uses DCT-based frequency-domain kernelization to reduce Transformer computation, while FreqKV Kai et al. (2026) observes that KV states along the sequence dimension exhibit substantial low-frequency energy concentration and compresses the cache by retaining a subset of DCT coefficients. More recently, frequency-domain analyses of RoPE-based key caches have shown that positional rotations induce structured spectral behavior that can also be exploited for cache compression Wang et al. (2026). These observations motivate treating the token dimension of the KV cache as a signal and exploiting its spectral redundancy during autoregressive generation.

B.2 DCT Before and After RoPE

An important consideration for frequency-domain KV compression is whether the transform is applied before or after Rotary Position Embedding (RoPE) Su et al. (2021). Let $\mathbf{k}_t \in \mathbb{R}^{d_h}$ denote the key associated with token position t before RoPE. RoPE rotates every pair of key channels by a position-dependent angle. For channel pair j, this can be expressed using a complex representation

$$z_j(t) = k_{t,2j} + \mathrm{i} k_{t,2j+1}. \tag{12}$$

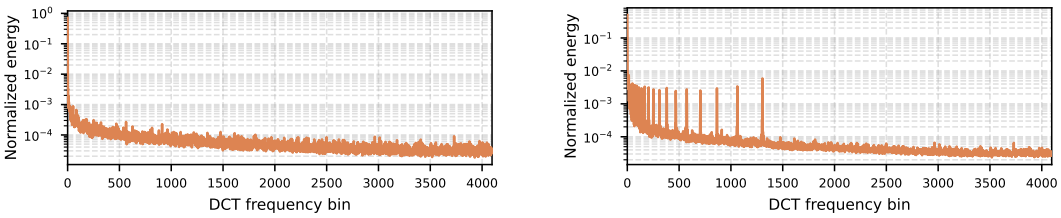
After RoPE,

$$z_j^{\mathrm{RoPE}}(t) = z_j(t) \exp\left(\mathrm{i} t \theta_j\right), \tag{13}$$

where θ_j is the rotation frequency associated with the j-th channel pair.

Equation 13 shows that RoPE acts as a position-dependent modulation of the key signal along the token dimension. Consequently, even when the pre-RoPE key varies slowly with t, the corresponding post-RoPE representation contains oscillations whose frequencies depend on θ_j. In the Fourier-domain interpretation, multiplication by $\exp(\mathrm{i} t\theta_j)$ shifts spectral content away from its original frequency. The analogous DCT representation therefore no longer necessarily concentrates its dominant energy in the first few low-frequency coefficients. Recent analyses similarly observe that RoPE induces structured spectral peaks in post-RoPE key representations Wang et al. (2026).

Figure 12 illustrates this difference. Before RoPE, the key states exhibit stronger energy concentration toward the lower DCT bins, making low-frequency truncation an effective compression



(a) Pre-RoPE key spectrum.

(b) Post-RoPE key spectrum.

Figure 12: Comparison of sequence-domain DCT spectra before and after RoPE. Pre-RoPE keys exhibit stronger low-frequency concentration, whereas position-dependent RoPE rotations redistribute energy across frequency bins. This motivates applying DCT-based low-frequency compression before RoPE.

mechanism. After RoPE, positional rotations redistribute this energy across the spectrum. Therefore, WaveKV performs frequency-level compression on the *pre-RoPE* KV representation, before position-dependent rotations are introduced. We note that this distinction primarily affects the key cache because RoPE is applied to queries and keys, while values are not normally subjected to the rotary transformation.

Position Indices after Frequency Compression. Retaining only $M < T$ DCT coefficients and reconstructing them into a shorter sequence produces M compressed KV states representing an original block of T tokens. From the attention interface, this operation can be viewed as downsampling the original token-domain representation. However, each reconstructed KV state summarizes information from multiple original token positions and therefore no longer has a direct one-to-one correspondence with an original position. Applying RoPE using consecutive indices $\{0, \ldots, M-1\}$ would consequently contract the positional span of the original block.

We instead associate every compressed state with an intermediate position spanning the original block. For a block beginning at position p_s and ending at p_e, the m-th compressed state is assigned

$$\widetilde{p}_m = p_s + \frac{m}{M-1}(p_e - p_s), \qquad m = 0, \ldots, M-1, \tag{14}$$

for $M > 1$. RoPE is subsequently evaluated using $\widetilde{p}_m$ rather than the compacted cache index m. Thus, frequency compression reduces the number of stored KV states while the intermediate position indices preserve the positional extent of the original sequence when rotary embeddings are applied.

C Discrete Wavelet Transform for Hierarchical Token Selection

C.1 Hierarchical Interpretation of the DWT

The Discrete Wavelet Transform (DWT) decomposes a signal into components that are localized jointly in position and scale Zhang (2019). Unlike the DCT, whose basis functions span an entire transform window, wavelet coefficients describe the signal at progressively finer spatial resolutions. This multiresolution structure makes the DWT particularly suitable for constructing a hierarchical representation of KV states along the token dimension.

We use the Haar wavelet because of its simple binary decomposition. Let $\mathbf{x} = [x_0, \ldots, x_{T-1}]$ denote a sequence, where, for simplicity, T is a power of two. At the finest level, we define

$$a_i^{(0)} = x_i. \tag{15}$$

Each subsequent level combines neighboring coefficients into an *approximation* coefficient and a *detail* coefficient:

$$a_i^{(j+1)} = \frac{a_{2i}^{(j)} + a_{2i+1}^{(j)}}{\sqrt{2}}, \tag{16}$$

$$d_i^{(j+1)} = \frac{a_{2i}^{(j)} - a_{2i+1}^{(j)}}{\sqrt{2}}. \tag{17}$$

The approximation coefficient $a_i^{(j+1)}$ summarizes the average content of a progressively larger region of the sequence, whereas $d_i^{(j+1)}$ captures the variation within that region.

Consequently, the Haar-DWT naturally induces a binary hierarchy. The leaf nodes correspond to fine-grained token-level representations, while increasingly higher levels summarize larger contiguous regions. For example, a node at level j represents a window of approximately 2^j input positions. The root therefore provides a coarse summary of the complete sequence, whereas descending the tree progressively resolves smaller regions.

The complete Haar transform can be computed in $\mathcal{O}(T)$ operations because the number of coefficients processed decreases by a factor of two at every level:

$$T + \frac{T}{2} + \frac{T}{4} + \cdots < 2T. \tag{18}$$

This makes the transform attractive for constructing a multiresolution index over long KV sequences.

C.2 DWT Hierarchy for Attention Token Selection

The hierarchical interpretation of the DWT provides a natural mechanism for selecting attention-relevant regions without exhaustively scoring every KV entry. Rather than viewing the compressed KV cache as a flat sequence of T states, WaveKV organizes it as a multiresolution tree. Each leaf corresponds to a retained KV state, while an internal node represents the aggregate key information of all leaves in its subtree. Thus, the hierarchy provides progressively finer views of the same sequence.

Given a decode query $\mathbf{q}$, routing begins at the coarse levels of the tree. For a node with representative key $\bar{\mathbf{k}}_i$, its relevance can be estimated using

$$s_i = \frac{\mathbf{q}^\top \bar{\mathbf{k}}_i}{\sqrt{d_h}}, \tag{19}$$

where d_h is the attention-head dimension. Only the top-k nodes at the current level are retained and expanded into their children. The remaining subtrees are pruned without examining their individual tokens. This process is repeated from coarse to fine resolution until the selected branches reach the leaf level, where exact attention is computed over the corresponding KV states.

The routing process can therefore be interpreted as a coarse-to-fine search over the context:

$$\text{context} \rightarrow \text{coarse regions} \rightarrow \text{candidate subregions} \rightarrow \text{selected KV states}. \tag{20}$$

If the beam width is k, approximately $2k$ candidate nodes are scored at each tree level. For a tree over T KV states, this results in a routing complexity of approximately

$$\mathcal{O}(k \log_2 T)\,, \tag{21}$$

rather than scoring all T states at every decode step. This advantage increases with context length when $k \ll T$.

Importantly, the DWT hierarchy is constructed independently of the current query, while the traversal itself remains query-dependent. This separates the representation of the KV cache from the decision of which regions should participate in attention. In WaveKV, the beam width is further adapted using the calibrated attention profile: regions or layers with diffuse attention are searched with a wider beam, whereas regions with concentrated attention are explored with a narrower beam. This allows the routing cost to be reduced without applying a fixed token-retention fraction across the entire context.

Wavelet-based token selection has also been explored by WaveFilter Yang et al. (2026), which exploits the multiresolution structure of DWT coefficients to reduce attention computation. In WaveKV, however, the hierarchy is applied over the already compressed KV representation and is coupled with the same attention-aware calibration used for DCT compression, enabling storage compression and token selection to be jointly coordinated.

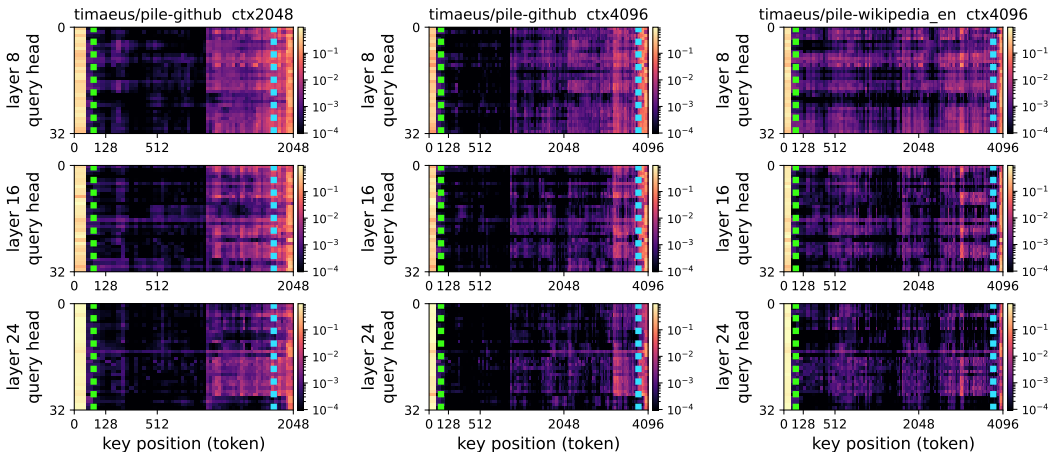


Figure 13: Attention heatmaps of the query heads across key positions for different context lengths (ctx), datasets, and layers show a similar underlying structure.

D Inherent Positional Structure in Attention Mechanism

D.1 Calibration Findings

The primary objective of the calibration phase is to quantify the inherent attention structure of a model and establish a stable basis for KV-cache compression. Rather than adapting the cache-compression policy to each inference query or evaluation sample, WaveKV performs a one-time, model-specific calibration over representative contexts and queries. The resulting parameters are fixed before deployment and reused across inference queries. We first evaluate Llama-3.1-8B across different context lengths and datasets. Figures 13a-13b show attention heatmaps for 2048- and 4096-token contexts from the Timaeus Pile-GitHub dataset Gao et al. (2020), across heads from layers 8, 16, and 24, while Figure 13c shows the corresponding 4096-token heatmap from the Timaeus Pile-Wiki dataset Gao et al. (2020). Figure 14 shows similar attention heatmap for Qwen3-8B model. Despite differences in length, content, and model type, WaveKV shows there is an underlying consistent positional structure in attention.

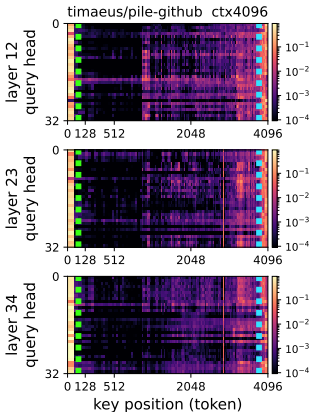


Figure 14: Similar attention structure observed across Qwen3-8B

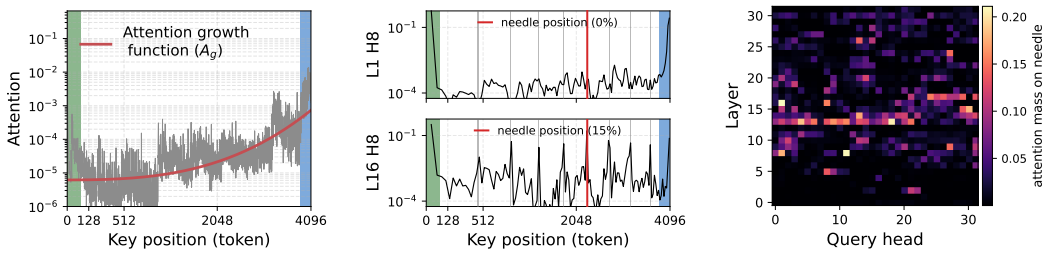
Attention growth: Attention grows slowly for older tokens and increasingly rapidly toward recent tokens. To quantify this behavior, **WaveKV** measures attention growth in the *Info* region by taking the maximum attention over query heads sharing each KV entry, followed by averaging across KV heads and layers. As shown in Figure 15a (Paper Figure 6a), the resulting profile is smoothly increasing and is approximated on a logarithmic scale by a convex quadratic A_g (red line), termed the *attention growth function*. This growth function is subsequently used by the downstream system blocks to determine position-adaptive compression schemes.

Retrieval-Aware Calibration: The positional attention structure above does not necessarily hold for retrieval-oriented queries, where a relevant token can occur at an arbitrary position within the context. If a needle lies within the *Info* region, aggressive compression can therefore cause retrieval errors. To quantify how strongly different parts of the model attend to such retrieval targets, we define the attention mass at key position p as the normalized attention weight averaged across layers

and heads:

$$A(p) = \frac{1}{LH} \sum_{\ell=1}^{L} \sum_{h=1}^{H} a_{\ell,h}(p), \tag{22}$$

where $a_{\ell,h}(p)$ denotes the normalized attention weight assigned to position p by head h in layer ℓ, and L and H denote the number of layers and heads. Using this measure, we perform an additional calibration to identify layers that require full-resolution KV access. WaveKV constructs a needle-attention map by embedding a randomly generated key-value needle at a random position in a long haystack and measuring the attention assigned to the needle for each layer and head. Repeating this procedure across random draws produces a layer-head retrieval profile. As shown in Figure 15b (Paper Figure 6b), the attention assigned to the needle varies substantially across layer-head pairs. For example, L_1H_8 assigns 0% attention to the needle, while $L_{16}H_8$ assigns 15%. The overall heatmap in Figure 15c (Paper Figure 6c) further shows that only a subset of layers and heads consistently exhibits high needle attention, while many remain largely needle-blind. This retrieval-aware profile allows downstream blocks to adapt compression according to retrieval sensitivity. Although the resulting calibration is model-specific because layer and head behavior differs across architectures, it is performed only once and reused during inference.



(a) Convex quadratic attention growth function

(b) Needle attention mass varies across layers.

(c) Retrieval-sensitive layer guides compression.

Figure 15: Trends of queries and keys leading to attention-aware calibration.

D.2 Calibration Setup

The attention profile used by WaveKV is obtained through a one-time offline calibration phase. Calibration sequences are sampled from representative documents from the Pile corpus Gao et al. (2020) and are disjoint from the evaluation datasets. The calibration procedure determines the Head–Info–Tail structure, attention-growth profile, adaptive DCT block sizes, retained-energy budgets, retrieval-sensitive layers, and DWT-routing beam widths. Once obtained, these parameters remain fixed during evaluation and are not adapted using evaluation queries or labels.

Head-Info-Tail Structure The *Head* boundary is identified using the cumulative attention-mass threshold from the beginning of the sequence, while the *Tail* boundary is determined analogously from the end of the sequence. Aggregating these boundaries across layers and heads yields the fixed 64-token Head and $\sim$256-token Tail regions. As shown by the green and blue dotted lines in Figure 15b, these boundaries remain stable across context lengths, supporting their use as fixed regions during inference.

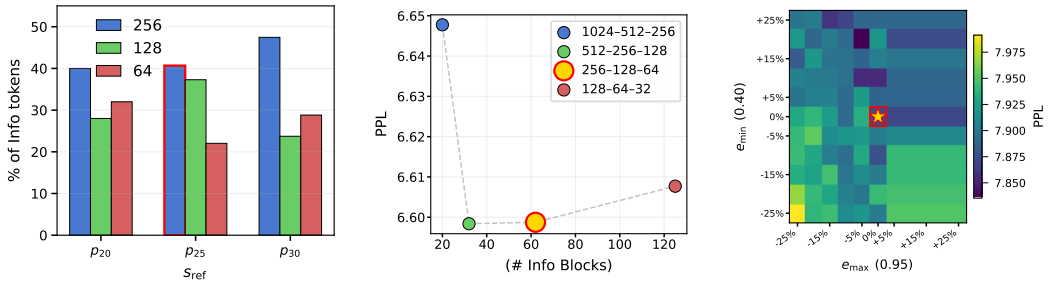
The calibration profile sets $H = 64$ as the fixed *Head* length and T_{des} as the desired *Tail* length (256 tokens in our default configuration). To ensure that the *Info* region is compatible with the DCT block granularity, we round its length to the nearest multiple of 64:

$$I = 64 \cdot \text{round}\left(\frac{C - H - T_{\text{des}}}{64}\right), \tag{23}$$

where C is the prefilling context length. The realized Tail length is then

$$T = C - H - I, \tag{24}$$

which may differ slightly from T_{des} due to the rounding constraint. For example, with $C = 8191$, $H = 64$, and $T_{\text{des}} = 256$, this yields $I = 7872$ and $T = 255$. Thus, the Tail is a flexible region



(a) s_{ref} sensitivity (b) B_{max} sensitivity (c) Energy sensitivity

Figure 16: Ablation study for Info block structure and DCT energy

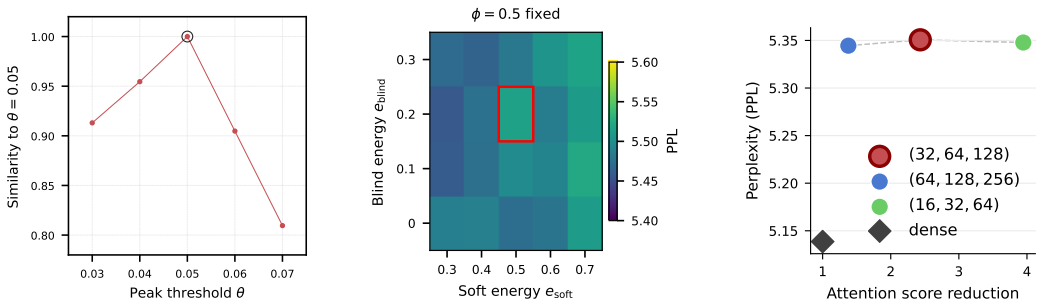
whose realized length depends on the context length and block-alignment constraint, while T_{des} serves as its calibration target.

Choice of Reference Slope We set s_{ref} to the 25th percentile of $|s(p)|$ over the *Info* region to obtain a robust low-slope reference that captures the slowly varying distant portion of the region without being overly influenced by steeper positions near the *Tail*. This choice anchors the largest block size $B_{\max}$ to the typical low-slope regime, while progressively steeper regions are assigned smaller dyadic blocks. Figure 16a therefore marks (p_{25}) using the corresponding calibration settings of (41/37/22)% for the *Info* blocks {256,128,64} as shown in Sec. 3.2. The points (p_{20}) and (p_{30}) show the renormalized {256,128,64} split obtained from the same attention envelope when (s_{ref}) is decreased or increased by one typical step. These perturbations modestly shift the allocation between coarse and fine blocks from (40.0/28.0/32.0) at (p_{20}) to (47.5/23.7/28.8) at (p_{30}) for 4K context while preserving the same coarse-to-fine structure. Together with the fixed (41/37/22) ladder, these results indicate that (p_{25}) provides a stable default rather than relying on a narrowly tuned operating point.

Choice of Maximum Block Size. We set $B_{\max} = 256$ to balance compression locality and DCT efficiency. Smaller blocks such as 128 unnecessarily fragment slowly varying regions, increasing the number of transforms and making computation less efficient at very small block sizes, while larger blocks such as 1024 weaken locality and provide coarser frequency resolution. As shown in Figure 16b, finer $B_{max} = 128$ {128-64-32} increases the *Info* block count without improving calibration perplexity. The $B_{max} = 512$ {512-256-128} and $B_{max} = 256$ {256-128-64} configurations exhibit similar perplexity, indicating limited sensitivity to moderate changes in block granularity. In contrast, $B_{max} = 1024$ {1024-512-256} yields the largest perplexity degradation, consistent with reduced token locality from overly coarse blocks. We therefore use $B_{max} = 256$ as a training-free default that balances block granularity without excessive fragmentation or overly coarse partitioning. Thus, longer contexts scale by increasing the number of blocks rather than their size, preserving consistent compression granularity and DCT efficiency.

Info blocks structure The *Info* span is tiled using the non-uniform block ladder derived from Equations 1–3, producing the $256 \rightarrow 128 \rightarrow 64$ hierarchy. The calibration profile determines how the *Info* tokens are distributed across the three block sizes: approximately 41% of the *Info* tokens are covered by 256-token blocks, 37% by 128-token blocks, and 22% by 64-token blocks. These fractions are applied to the total *Info* span at each context length, so the number of blocks changes with context length. A coarse-to-fine cascade then fills the span from distant to recent positions, placing 256-token blocks first, followed by 128- and 64-token blocks toward the Tail. For example, at 8K, the 7,872-token *Info* span is tiled by 13 blocks of 256, 22 blocks of 128, and 27 blocks of 64 tokens, exactly covering the span.

DCT energy allocation per block. Each block carries its own DCT energy target. The linear profile introduced in Equations 3 and 4 ramps linearly across the ladder from 0.40 of DCT energy at the oldest block to 0.95 energy at the newest, a step of ~ 0.009 per block over the 62 blocks for the 8K evaluation length. Older, coarser blocks are thus compressed hardest, and blocks adjacent to the exact tail are nearly lossless. The *Head* and *Tail* don't get compressed. We set the



(a) θ sensitivity (b) Retrieval energy sensitivity (c) Beam width sensitivity

Figure 17: Ablation study for retrieval layer structure and energy and beam width design

upper energy-retention target to $e'_{\max} = 0.95$, slightly below the empirically observed near-lossless regime. This provides a conservative retention target for the high-attention recent region while retaining compression headroom and avoiding exact retention at $e = 1.0$. The choice also provides a simple 5-percentage-point operating point for the linear energy schedule. Figure 16c shows the heatmap that varies ($e_{\min}$) and ($e_{\max}$) around (0.40→0.95) by $\pm 25\%$ with calibrated block structure fixed. Lowering ($e_{\min}$) increases compression of distant blocks, leading to higher perplexity, while increasing ($e_{\min}$) retains more tokens and improves perplexity. Perturbing both endpoints by ($\pm 10\%$) results in only minor changes in calibration perplexity, indicating that the default endpoints provide a stable operating point rather than a tightly tuned setting.

Retrieval-oriented workloads Equations 5–6 are used during calibration to identify retrieval-sensitive layers and assign them to PROTECT, SOFT, or BULK tiers. The resulting layer assignments are model-specific as different models have different architectures and layers can behave differently. But this is a one-time thing and can be calibrated before deployment. For Llama-3.1-8B-Instruct (32 layers), the PROTECT layers are $\{5, 6, 8\text{–}11, 13\text{–}17, 19\text{–}21, 24\text{–}26\}$, SOFT layers are $\{2, 7, 18, 22, 23, 27\text{–}30\}$, and BLIND comprises $\{0, 1, 3, 4, 12, 31\}$. For Qwen3-8B (36 layers), PROTECT comprises layers 1–15 and 17–23, SOFT comprises $\{16, 24, 26, 28\text{–}34\}$, and BLIND comprises $\{0, 25, 27, 35\}$. For Phi-3-medium-128k-instruct (40 layers), PROTECT comprises layers $\{8\text{–}28\}$ and $\{30\text{–}31\}$, SOFT comprises $\{2\text{–}7, 29, 32, 33, 35\}$, and BLIND comprises $\{0, 1, 34, 36\text{–}39\}$. These assignments are obtained once during calibration and remain fixed during inference.

Retrieval-oriented energy allocation

We count a head as retrieval-sensitive only when its maximum attention weight satisfies θ, and its argmax lies within T_{needle} of the needle position as shown in Eq. 6. This peak threshold filters diffuse heads whose argmax may fall near the needle by chance, distinguishing concentrated retrieval behavior from weak or diffuse attention. We set $\theta = 0.05$ as a fixed, conservative threshold. At long contexts, it is substantially above the $\mathcal{O}(1/S)$ peak expected under near-uniform attention, where S denotes the number of attended tokens, while remaining low enough to capture moderately concentrated retrieval heads. We vary θ around 0.05 and measure the retrieval similarity compared to the same. Figure 17a shows small variations around $\theta = 0.05$ (0.03-0.06) produce similar retrieval-sensitive layer profiles (90% similarity), indicating that the calibration is not sensitive to the precise threshold choice.

The energy budgets are derived from the empirical retrieval frequency measured during needle-in-a-haystack calibration. Retrieval frequency f_ℓ uses a 0.5 cutoff (ϕ) in Eq. 7, as a layer is classified as retrieval-critical only if it exhibits a needle peak in at least half of the calibration probes, capturing majority retrieval behavior while avoiding a more aggressive threshold such as 0.6 that would protect fewer layers. We then discretize energy retention into three levels, $\{1.0, 0.5, 0.2\}$. PROTECT layers retain the full DCT budget ($e_\ell = 1.0$), SOFT layers use half the budget ($e_\ell = 0.5$) for occasional retrieval ($0 < f_\ell < 0.5$), and BLIND layers use $e_\ell = 0.2$ when $f_\ell = 0$. The value 0.2 is deliberately below the usual far-*Info* retention range ($\sim$ 0.4–1.0) to enable aggressive compression of needle-insensitive layers, while remaining strictly positive so their routed KV states remain usable. Figure. 17b shows the heatmap of perplexity observed around the defined energy budget (e_l) and

shows that small variations have negligible effect on the perplexity, indicating that the calibration is not sensitive to the precise values, however, if the energy value increases, the rate of compression decreases.

Adaptive Beam-Width Ladder. The LADDER variant assigns beam widths k by block granularity. Finer-grained regions receive progressively broader candidate coverage, while coarse regions use narrower beams to limit routing overhead. For a 8192-token context, the *Info* region is partitioned into 256-, 128-, and 64-token blocks, with corresponding beam widths $(k = 32, 64, 128)$ respectively. In the retrieval TIER variant, we use the same principle, but across the retrieval-sensitive layers, PROTECT layers are completely used for attention, while SOFT layers get $k = 128$ and BLIND gets $k = 64$. Fig. 17c shows that deviations around the beam-widths $(k = 32, 64, 128)$ to $(k = 64, 128, 256)$ or $(k = 16, 32, 64)$ change the perplexity negligibly, however, the attention score reduction alters compared to dense changes

All parameters are fixed before evaluation and are not tuned on evaluation queries or labels.

E Detailed Evaluation Setup

E.1 Models and Inference Configuration

We evaluate WaveKV on LLaMA-3.1-8B-Instruct, Qwen3-8B, and Phi-3-medium-128k using their publicly released checkpoints and tokenizers. Unless otherwise stated, all models are evaluated in their original pretrained/instruction-tuned form without additional fine-tuning. We use the native attention-head, KV-head, and positional encoding configurations of each model. For each experiment, the same model checkpoint, tokenizer, prompt format, generation parameters, and numerical precision are used across all KV-cache methods. This ensures that differences in model quality or system performance arise from the KV-cache strategy rather than changes to the underlying model.

E.2 Hardware and System Configuration

Experiments are conducted on a server equipped with two NVIDIA H100 NVL GPUs with 94 GB HBM3 memory per GPU, dual Intel Xeon Platinum 8490H processors with 120 cores per socket and 240 hardware threads, and 1 TiB of system DRAM. The server runs Ubuntu Linux. Unless otherwise stated, all experiments use a single H100 GPU. The same hardware configuration is used across methods for the corresponding model-quality and system-level comparisons.

E.3 Dataset Details

PG-19. We use PG-19 for language-modeling evaluation because its full-length books naturally contain long coherent contexts. We evaluate perplexity at context lengths of 1K, 2K, 4K, 8K, 16K, and 32K tokens.

LongBench. For LongBench, we evaluate the standard tasks spanning single-document QA, multi-document QA, summarization, few-shot learning, and code understanding. We follow the benchmark's standard prompting and task-specific evaluation metrics.

RULER. We use RULER to evaluate controlled long-context retrieval and reasoning. Unless otherwise stated, we evaluate a nominal context length of 16K tokens with input sequences of at least 8K tokens and report accuracy over the selected RULER task categories.

Needle-in-a-Haystack. We additionally evaluate retrieval capability using Needle-in-a-Haystack, where target information is inserted at different positions within long contexts. This isolates the ability of each compression method to preserve information that may occur far from the current decoding position.

GSM8K and MMLU. GSM8K and MMLU are used to evaluate whether KV-cache compression preserves general mathematical reasoning and knowledge capabilities during autoregressive generation.

E.4 Evaluation Metrics

For PG-19, we report perplexity (PPL) as defined in Jurafsky & Martin (2025). Lower perplexity indicates better language-modeling performance. For LongBench, we use the standard metric associated with each task and report the average across tasks. For RULER, Needle-in-a-Haystack, GSM8K, and MMLU, we report accuracy or task-specific retrieval accuracy, as appropriate. For system evaluation, we report Time-to-First-Token (TTFT), normalized per-sequence decoding throughput, peak allocated GPU memory, number of KV-cache recompressions, and attention-compute reduction.

E.5 System-Level Measurement

All throughput measurements use a batch size of four sequences. The reported tokens/s values are divided by the batch size and therefore represent per-sequence decoding throughput. The peak allocated memory is measured during the complete inference run, including model parameters, KV-cache storage, temporary compression buffers, and routing-related allocations. We additionally report the number of cache recompressions required as the generated sequence grows.

Attention-scoring reduction is reported as

$$S_{\text{attn}} = \frac{\text{KV rows scored by dense attention}}{\text{KV/tree rows scored by the evaluated method}}. \tag{25}$$

This metric represents a proxy for attention FLOPs and KV-memory reads per decoding step and should not be interpreted as end-to-end wall-clock speedup.

E.6 Baseline Configuration

We compare WaveKV against representative KV-cache methods including FreqKV Kai et al. (2026), WaveFilter Yang et al. (2026), PyramidInfer Yang et al. (2024), GemFilter Shi et al. (2026), SnapKV Li et al. (2024), PyramidKV Cai et al. (2024b), LaCache Shi et al. (2025), FastKV Jo et al. (2026), CaM Zhang et al. (2024), D2O Wan et al. (2024), ThinK Xu et al. (2025), and StarKV Bhatnagar et al. (2026) where applicable. For FreqKV, we evaluate both fine-tuned and training-free variants and use the reported retention settings corresponding to each experiment. For WaveFilter, we use the original uncompressed KV cache and apply DWT-based token selection during attention. For combined FreqKV+WaveFilter experiments, WaveFilter is applied to the FreqKV-compressed representation to study whether token selection can be directly composed with frequency-domain compression. Unless otherwise stated, all methods use the same context length, model, batch size, numerical precision, and generation settings.

E.7 Query-Free Baseline Construction

WaveKV performs KV-cache compression independently of the current decode query. To enable a controlled comparison with methods whose original compression decisions depend on query-derived attention scores, we construct query-free variants of these baselines. We remove the query only from the *cache-compression or selection decision*. The original query is still used normally during the subsequent attention computation. All other method-specific settings, including cache budgets, protected recent-token windows, and merge or selection ratios, are kept unchanged.

Eviction-based methods. Methods such as SnapKV, PyramidKV, and FastKV typically rank historical KV states using a query-dependent importance score

$$s_t = f(Q, K_t), \tag{26}$$

which is commonly derived from recent-query attention scores $QK^\top/\sqrt{d}$. For the query-free variant, we replace this score with a key-only magnitude statistic computed independently for each native KV head h:

$$s_{h,t} = \text{Pool}_t\left(\|k_{h,t}\|_2\right). \tag{27}$$

The highest-scoring historical positions and their corresponding value states are retained under the same cache budget as the original method. Any protected recent-token window used by the baseline remains unchanged.

Merging-based methods. For CaM, KVMerger, and D2O, query-derived importance used to determine which KV states should be retained or merged is replaced with the same key-only importance measure. Once a state is selected for merging, its representative is determined using key–key similarity rather than a query-dependent criterion. In particular, we use cosine similarity between the candidate key and retained representatives:

$$\mathrm{sim}(\mathbf{k}_i, \mathbf{k}_j) = \frac{\mathbf{k}_i^\top \mathbf{k}_j}{\|\mathbf{k}_i\|_2 \|\mathbf{k}_j\|_2}. \tag{28}$$

This preserves the key-space geometry used for merging while removing dependence on the current query.

Channel-compression methods. For methods whose channel-selection criterion depends on query–key interaction, we replace the query-dependent channel score with the average key energy of each channel:

$$s_{h,c} = \frac{1}{|T|} \sum_{t \in T} k_{h,t,c}^2, \tag{29}$$

where T denotes the set of cached positions, h the KV head, and c the channel. The channels with the largest energy are retained under the same channel budget as the original method. Components that are already query-independent are left unchanged.

Rationale for key-only scoring. The key magnitude provides a simple query-agnostic proxy for a token's potential attention contribution. From the Cauchy–Schwarz inequality,

$$\left|\mathbf{q}^\top \mathbf{k}\right| \le \|\mathbf{q}\|_2 \|\mathbf{k}\|_2, \tag{30}$$

so a key with small norm has a correspondingly bounded contribution to the unnormalized query–key interaction for any fixed query norm. Similarly, key–key similarity provides a query-independent measure for preserving geometric proximity during merging.

We emphasize that these key-only scores are intended as a controlled query-free adaptation of existing baselines rather than as universally optimal importance measures. The purpose of this protocol is to compare compression methods under the same query-independent constraint as WaveKV, while otherwise preserving each baseline's original operating configuration.

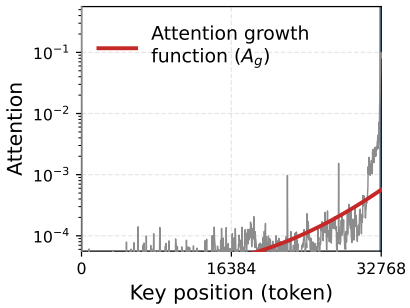
E.8 Implementation Details

WaveKV is implemented with batched decoding and custom Triton kernels for DCT-based KV compression and hierarchical routing. DCT transforms are applied along the token dimension of pre-RoPE KV states. During decoding, Haar-DWT-derived hierarchical routing is used to identify the subset of compressed KV states participating in attention. All reported model-quality and system measurements are obtained using the same implementation environment and hardware for the corresponding baseline comparison.

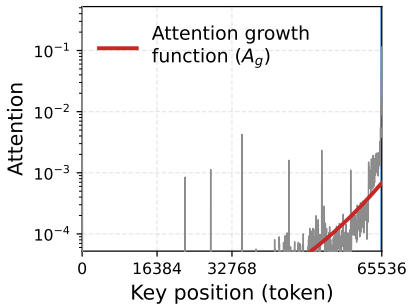
F Long-Prefill implementation and evaluation of WaveKV

Attention-guided non-uniform DCT-based compression, as proposed in Section 3.2, preserves the required attention structure; however, its full application during long-context prefill ($\geq 16K$) increases time-to-first-token (TTFT) relative to dense attention (Full KV) and FreqKV due to the repeated ladder-based DCT computation. Calibration shows that, at these lengths, most attention is concentrated within the most recent 8K-10K tokens, while earlier tokens receive sparse attention. Figure 18 shows that the proposed attention growth function A_g (Section 3.2) identifies that attention-aware calibration is primarily needed for the most recent 8K-10K tokens.

Applying the full WaveKV ladder to these low-impact regions during prefill therefore introduces unnecessary overhead. We address this with **WaveKV-FF** (Fast Forward), which uses a fixed N=8192 token per-layer cache and performs rolling prefill. The first 64 positions are retained as an exact *Head*. During prefill, whenever the live cache exceeds N, the far *Info* region is recompressed using a single orthonormal DCT, retaining $L = \lfloor \gamma(N-64) \rfloor$ coefficients with a retention ratio of $\gamma = 0.4$.



(a) Attention pattern for 32K Prefill



(b) Attention pattern for 64K Prefill

Figure 18: Attention growth function A_g shows only the last 8K-10K tokens actually require the attention aware DCT compression.

Although this fixed coefficient-retention approach is computationally efficient, Section 2 shows that it performs poorly when attention to the compressed region is non-negligible. In long-context prefill, however, Figure 18 shows that attention to these earlier tokens is sparse, making this simplified compression highly effective for reducing compute overhead without degrading performance. This keeps the rolling prefill bounded without repeatedly applying the full adaptive DCT ladder.

On the final prefill compression, preserving attention becomes relevant for the most recent 8K-10K tokens. After the complete prefill has been ingested, guided by A_g, the *Info* region is recompressed using the attention-guided DCT blocks and Energy Allocation methodology from Section 3.2, together with the global exact *Tail*, yielding the standard WaveKV approach. The decode pipeline remains unchanged and uses the same calibrated beam-routed attention as mentioned in Section 3.3.

Table 8 evaluates TTFT and peak GPU memory usage across prefill contexts from 16K to 128K and different batch sizes $\{1,2,4,8\}$. As an ablation, we evaluate WaveKV-Streaming, which uses the same fixed N=8192 cache and rolling prefill as WaveKV-FF, but applies the full ladder-based DCT at every recompression. We compare WaveKV-FF against Full KV, FreqKV, WaveFilter, and WaveKV-Streaming. Full KV and WaveFilter achieve similar TTFT and outperform the other methods, however, because they perform no compression, both reach OOM at the highest context length (64K Batch-4 onwards, 128K Batch-2 onwards). WaveKV-Streaming keeps peak memory bounded as the prefill length increases for a given batch size due to its fixed cache size. However, its repeated ladder-based DCT computation at every recompression increases TTFT, reaching $\sim$2$\times$ - 4$\times$ that of Full KV and WaveFilter. WaveKV and FreqKV also keep peak memory bounded through their fixed cache sizes while achieving TTFT comparable to Full KV and WaveFilter and $\sim$2$\times$ - 4$\times$ reduction compared to WaveKV-Streaming. However, at longer prefill contexts (64K and 128K), WaveKV-FF outperforms all baselines. At 128K, WaveKV-FF achieves a $\sim$2$\times$ reduction in TTFT and $\sim$40% lower peak GPU memory than Full KV and WaveFilter, $\sim$3$\times$ reduction of TTFT compared to WaveKV-Streaming, while reducing TTFT by $\sim$10% and peak GPU memory by $\sim$30% compared to FreqKV, with stronger compression and adaptively preserving the attention-high cache region.

G Detailed Comparison of Language Tasks

In this appendix, we present a detailed comparison of the WaveKV performance on the language tasks evaluated in Section 4.1 using the evaluation setup in Appendix E.

G.1 Language Modeling

Table 9 provides the complete PG-19 perplexity results for LLaMA-3.1-8B, Qwen3-8B, and Phi3-medium-128k across context lengths from 1K to 32K. These results complement the main-paper evaluation by showing how the effect of KV-cache compression evolves with context length and across model families. Since lower perplexity indicates better language-modeling quality, the com-

Table 8: TTFT and peak GPU memory across context lengths and batch sizes. For each batch size, the first row reports **TTFT (s)** and the second row reports **Peak Memory (GB)**.

Context	Batch	Full KV	FreqKV	WaveFilter	WaveKV-Streaming	WaveKV-FF
16K	1	1.182s 22.6GB	1.521s 21.0GB	0.857s 23.0GB	11.102s 27.9GB	1.867s 18.2GB
	2	1.680s 29.1GB	2.426s 25.8GB	1.616s 27.1GB	4.652s 30.3GB	2.474s 20.3GB
	4	3.377s 42.0GB	4.719s 35.6GB	3.267s 35.3GB	6.050s 35.2GB	4.480s 24.5GB
	8	6.927s 68.0GB	9.363s 55.1GB	6.662s 51.7GB	9.250s 44.8GB	8.452s 33.0GB
32K	1	2.014s 29.1GB	2.618s 25.3GB	1.950s 27.1GB	8.913s 36.5GB	2.654s 24.2GB
	2	4.030s 42.0GB	5.158s 34.5GB	3.912s 35.3GB	8.743s 41.0GB	4.674s 28.3GB
	4	8.178s 68.0GB	10.243s 52.9GB	7.891s 51.7GB	13.512s 50.0GB	8.959s 36.5GB
	8	OOM	20.869s 89.8GB	15.818s 84.5GB	18.604s 68.1GB	17.946s 52.9GB
64K	1	5.325s 42.0GB	5.372s 34.0GB	5.197s 35.4GB	14.264s 48.1GB	4.971s 32.5GB
	2	10.703s 68.0GB	10.705s 51.9GB	10.446s 51.7GB	17.428s 56.6GB	9.421s 40.5GB
	4	OOM	21.534s 87.7GB	20.970s 84.5GB	24.183s 73.5GB	18.758s 56.4GB
	8	OOM	OOM	OOM	OOM	OOM
128K	1	17.009s 68.0GB	10.964s 51.3GB	16.838s 51.3GB	25.406s 58.9GB	9.703s 43.2GB
	2	OOM	22.016s 86.6GB	OOM	33.261s 75.0GB	19.301s 59.1GB
	4	OOM	OOM	OOM	OOM	OOM
	8	OOM	OOM	OOM	OOM	OOM

parison characterizes how closely each compressed representation preserves the behavior of the full-KV baseline.

Across all three models, WaveKV remains close to Full KV while becoming increasingly favorable relative to frequency-only compression as the context grows. On LLaMA-3.1-8B, WaveKV improves over untrained FreqKV from 4K onward, reaching a PPL of 9.22 at 32K compared with 9.30 for untrained FreqKV. The trend is stronger on Qwen3-8B, where WaveKV reduces PPL from 10.79 to 9.60 at 4K and from 11.51 to 10.53 at 32K. On Phi3-medium-128k, WaveKV remains particularly close to Full KV through 16K, achieving 8.32 compared with 8.28 for Full KV, while substantially improving over untrained FreqKV at longer contexts, including 7.42 versus 8.24 at 32K. The results also show that the benefit of adaptive compression is most evident once stronger compression is required. At shorter contexts, the differences among methods are relatively small, whereas at 16K and 32K the gap between WaveKV and uniform frequency-domain compression becomes more pronounced. This behavior is consistent across model families and indicates that adapting the retained

Table 9: PPL of WaveKV against state-of-the-art on PG-19 at different evaluation context lengths.

Model	Method	Evaluation Context Length					
		1024	**2048**	**4096**	**8192**	**16384**	**32768**
LLaMA-3.1-8B	Full KV	7.90	8.42	7.30	7.93	6.68	8.25
	FreqKV (fine-tuned)	7.51	7.93	7.71	8.20	6.97	8.63
	FreqKV_0 (untrained)	7.90	8.44	8.50	8.77	7.47	9.30
	PyramidInfer (eviction)	8.41	10.29	8.58	8.91	7.17	10.47
	WaveFilter (KV intact)	7.87	8.38	7.33	7.87	6.65	8.21
	FreqKV+WaveFilter	7.76	8.48	7.97	8.60	7.36	9.01
	WaveKV	8.01	8.97	7.66	8.24	7.14	9.22
Qwen3-8B	Full KV	10.02	10.86	9.18	8.96	7.87	9.29
	FreqKV	9.93	10.73	10.69	9.99	8.79	11.32
	FreqKV_0	10.03	10.85	10.79	10.14	8.87	11.51
	PyramidInfer	10.72	12.33	10.47	10.64	8.71	10.56
	WaveFilter	10.46	11.44	9.69	9.67	8.15	9.65
	FreqKV + WaveFilter	10.45	11.31	11.04	10.64	9.16	11.61
	WaveKV	10.33	12.03	9.60	9.26	8.49	10.53
Phi3-medium-128k	Full KV	5.82	6.28	6.36	6.12	8.28	6.40
	FreqKV	5.80	6.26	7.58	7.22	8.99	8.22
	FreqKV_0	5.82	6.27	7.59	7.22	9.01	8.24
	PyramidInfer	7.38	7.33	7.87	7.19	10.22	7.84
	WaveFilter	6.26	6.93	6.97	6.87	9.22	6.92
	FreqKV + WaveFilter	6.67	7.23	7.81	7.56	9.50	8.82
	WaveKV	6.06	6.39	6.47	6.28	8.32	7.42

spectral information across the cache better preserves next-token prediction quality than applying a uniform frequency-retention strategy.

G.2 Long-Context Understanding

This section provides the complete long-context understanding results on LongBench and RULER, complementing the summarized results reported in the main paper. LongBench evaluates performance across diverse long-document tasks, including single-document and multi-document question answering, summarization, few-shot learning, and code understanding, while RULER provides controlled stress tests for retrieval and reasoning over extended contexts. Together, these benchmarks characterize how well WaveKV preserves information required for long-range attention under KV-cache compression. We report the full task-level LongBench results for all evaluated models and the complete RULER results across the considered task categories. These detailed results allow us to examine the consistency of WaveKV across model families and across different forms of long-context retrieval and reasoning.

LongBench Evaluation: Table 10 reports complete LongBench results across 14 tasks for LLaMA-3.1-8B, Qwen3-8B, and Phi3-medium-128k. On LLaMA-3.1-8B, WaveKV achieves the highest average score of 44.56, outperforming PyramidInfer at 43.08 and FreqKV at 38.05, with notable gains on 2WikiMQA (45.79), LCC (70.91), and RepoBench-P (62.96). A similar trend holds for Qwen3-8B, where WaveKV achieves the strongest compressed-method average of 43.19 compared with 41.55 for PyramidInfer and 36.82 for FreqKV, with improvements across document QA, multi-document retrieval, and code tasks. On Phi3-medium-128k, WaveKV remains competitive with an average score of 37.12, compared with 38.63 for D2O, 38.15 for CaM, and 37.20 for PyramidInfer, while still improving over PyramidInfer on tasks such as Musique, GovReport, SAMSum, and LCC. Overall, the results indicate that WaveKV is particularly effective on tasks requiring information to remain accessible across distant or distributed context regions, supporting the use of adaptive compression and hierarchical routing instead of uniform compression or irreversible token eviction.

Table 10: Performance of different KV-cache compression methods on LongBench. We report the task-specific LongBench scores for each dataset and the average score across all evaluated tasks. The best scores are in **bold**, while the second best are underlined.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Code		Avg.
	NarrativeQA	Qasper	MFQA-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RepoBench-P	
LLaMA-3.1-8B															
Full KV	25.00	14.20	27.98	16.37	12.75	11.54	34.24	24.00	24.05	73.97	93.07	43.26	46.82	45.79	37.19
GemFilter	19.59	8.01	29.53	19.41	11.65	11.44	34.98	**24.14**	24.53	69.86	**93.07**	43.77	32.29	43.70	35.66
SnapKV	0.57	0.28	2.17	3.74	1.72	2.01	0.57	8.92	0.00	34.29	16.39	9.45	3.06	6.48	37.40
PyramidKV	5.37	0.41	2.65	3.59	2.76	1.47	0.54	5.95	0.0	32.88	17.63	8.79	3.65	6.65	8.59
LaCache	24.88	10.88	25.01	16.98	12.90	11.56	31.61	23.49	25.47	**72.60**	92.50	43.64	44.12	46.41	36.54
FastKV	1.14	0.00	1.63	0.77	1.46	0.29	0.06	4.60	0.11	30.14	8.43	6.77	6.82	5.27	4.22
CaM	**25.01**	8.67	26.39	15.13	15.37	10.86	33.00	22.62	**25.69**	69.86	92.25	42.59	46.65	46.08	35.88
D2O	22.76	7.23	20.62	14.85	12.12	9.25	29.17	22.27	24.91	69.86	90.62	42.01	43.18	46.28	34.96
ThinK	3.42	0.28	2.59	4.22	3.01	2.56	0.53	8.88	0.00	35.62	20.02	8.50	3.12	6.31	8.96
StarKV	1.68	4.12	8.46	3.50	3.16	2.38	14.11	8.66	8.60	16.44	11.71	6.08	15.59	18.43	7.9
WaveFilter	24.18	23.02	**45.93**	**51.66**	33.72	26.33	17.53	22.54	7.61	57.53	83.30	37.03	39.59	43.71	36.69
FreqKV	19.03	26.06	35.28	45.51	30.11	18.25	27.24	21.70	22.21	52.05	84.89	40.33	59.27	50.74	38.05
PyramidInfer	23.57	**31.53**	34.60	50.89	41.85	24.95	**29.06**	24.04	24.07	63.01	91.00	42.33	63.05	59.19	43.08
WaveKV	22.59	28.52	38.84	51.31	**45.79**	**27.45**	28.43	23.10	23.99	64.38	91.67	**43.88**	**70.91**	**62.96**	**44.56**
Qwen3-8B															
Full KV	24.10	43.56	49.89	57.31	45.72	36.02	33.13	24.07	22.13	72.60	92.51	45.99	73.96	62.89	48.85
GemFilter	3.29	4.72	13.77	6.29	7.79	3.65	18.68	16.20	20.72	38.36	0.00	29.25	26.40	25.30	13.71
SnapKV	3.18	3.93	20.84	11.71	11.21	5.98	28.79	22.78	18.83	69.86	89.69	41.94	51.65	49.97	32.83
PyramidKV	2.81	2.98	17.06	9.38	10.63	6.37	24.32	22.17	19.45	67.12	74.73	37.36	41.30	41.97	29.18
LaCache	3.68	4.20	24.12	11.81	11.57	6.04	29.75	19.85	19.89	71.23	89.92	44.87	59.00	56.06	40.03
FastKV	3.51	7.80	19.41	11.85	10.55	6.83	**30.87**	20.38	23.55	69.86	88.33	42.15	56.45	54.82	32.51
CaM	3.30	6.75	25.62	11.84	10.88	5.54	29.30	20.03	20.77	69.86	**92.60**	**45.16**	64.80	54.90	34.48
D2O	3.54	6.18	21.57	11.49	11.11	5.91	29.18	20.57	22.22	68.49	92.60	45.09	66.70	54.43	34.94
ThinK	2.98	5.50	21.89	12.08	12.35	7.00	29.11	**23.47**	20.93	**72.60**	90.08	43.14	51.75	53.79	33.71
StarKV	1.09	0.71	4.12	0.98	0.46	1.05	2.79	8.95	1.29	10.96	0.80	2.66	8.35	10.25	3.60
WaveFilter	**21.98**	19.04	29.31	41.88	**38.47**	**23.16**	18.13	21.34	10.99	67.12	78.48	40.75	51.56	47.23	36.39
FreqKV	15.50	26.35	29.28	34.37	34.05	8.73	26.63	19.86	22.56	61.64	81.05	42.23	61.64	51.60	36.82
PyramidInfer	20.22	18.58	30.99	46.50	33.70	23.00	26.68	21.54	**23.64**	63.01	92.32	44.97	**75.88**	60.69	41.55
WaveKV	18.73	**35.27**	**37.46**	**52.10**	37.39	21.52	29.46	22.08	22.50	63.50	89.22	42.85	66.95	**65.65**	**43.19**
Phi3-medium-128k															
Full KV	16.27	16.67	45.36	33.22	21.86	19.61	33.40	25.05	24.54	79.79	89.87	45.87	58.32	60.48	40.74
GemFilter	1.29	2.91	5.00	3.47	3.35	8.16	11.45	3.27	4.01	57.45	0.00	8.89	22.86	23.07	10.10
SnapKV	8.81	17.43	16.37	22.40	15.05	15.70	3.22	15.35	3.29	68.09	75.00	31.19	26.93	32.93	22.82
PyramidKV	7.20	15.24	11.62	20.22	18.40	13.33	6.19	13.50	6.65	68.09	66.62	26.16	26.07	32.44	22.51
LaCache	23.26	16.08	34.16	40.85	28.44	**29.38**	32.35	21.10	23.45	76.60	88.08	**45.45**	49.31	54.22	34.10
FastKV	11.99	11.07	10.93	21.59	8.80	15.45	1.56	11.57	0.95	68.09	67.79	20.11	14.34	21.87	17.92
CaM	**24.74**	14.28	**37.04**	39.63	26.76	28.45	**32.88**	20.59	22.64	**77.66**	87.81	44.25	47.14	52.57	38.15
D2O	22.58	12.85	29.83	**41.09**	**29.52**	23.90	30.14	20.20	**23.82**	76.60	87.33	44.72	47.59	51.90	**38.63**
ThinK	10.58	8.52	23.14	32.76	18.78	20.82	7.39	17.72	9.06	70.21	81.33	34.68	30.41	38.24	27.72
StarKV	0.28	4.91	5.63	2.94	4.00	4.60	6.70	11.39	5.77	13.83	17.78	5.37	15.10	20.19	7.68
WaveFilter	4.85	7.76	25.59	17.30	12.36	9.42	11.94	20.27	8.87	65.96	72.81	35.19	35.88	46.34	26.75
FreqKV	13.85	**21.79**	32.90	19.30	24.32	7.47	28.07	21.35	22.89	64.89	83.59	39.73	54.66	51.50	34.74
PyramidInfer	20.79	15.50	29.24	32.61	21.31	15.80	27.21	22.58	22.71	71.28	**89.99**	37.64	54.59	**59.61**	37.20
WaveKV	16.19	11.92	31.28	33.88	21.39	18.94	29.35	**22.84**	22.69	65.96	89.24	43.32	**55.15**	57.51	37.12

RULER Evaluation: Table 11 reports the complete RULER evaluation at 16K context, using inputs of at least 8K tokens. For LLaMA-3.1-8B, WaveKV achieves an average accuracy of 86.57%, remaining within 2.51 percentage points of Full KV at 89.08% while substantially outperforming WaveFilter at 54.60%, PyramidInfer at 45.66%, and FreqKV at 31.80%. The task-level results show that WaveKV maintains strong performance across QA, variable tracking, and NIAH, achieving 68.50%, 88.28%, and 91.89%, respectively. The same trend extends across model families. On Qwen3-8B, WaveKV achieves an average accuracy of 78.75%, exceeding the strongest competing compressed baseline, WaveFilter at 40.06%, by 38.69 percentage points. On Phi3-medium-128k, WaveKV reaches 88.75%, close to the Full-KV accuracy of 92.15% and 54.72 percentage points above the strongest competing compressed baseline, PyramidInfer at 34.03%. These results indicate that WaveKV preserves retrieval and long-context reasoning substantially better than the evaluated compression baselines across different model families.

G.3 Long Text Understanding

We further evaluate whether KV-cache compression preserves information during long-form autoregressive generation on **GSM8K** and **MMLU**. For both benchmarks, generation is limited to 4096 tokens, and each prompt requires the model to provide its reasoning before producing the final answer. This setting stresses the decode-time KV cache because intermediate reasoning states are repeatedly reused as generation progresses. Errors introduced by compression can therefore propagate across later decoding steps. As shown in Table 4, this effect is particularly visible on GSM8K. Full KV achieves 82.50%, while WaveKV reaches 74.50%. However, WaveKV substantially outperforms

Table 11: RULER accuracy across LLaMA-3.1-8B, Qwen3-8B, and Phi3-medium-128k at 16K context. Inputs are mimimum of 8K length.

Model	Method	FWE	QA	VT	NIAH	Avg.
LLaMA-3.1-8B	Full KV	85.87	66.40	96.88	94.18	89.08
	FreqKV	90.67	41.10	20.68	23.51	31.80
	FreqKV_0	88.27	39.20	15.12	23.18	30.60
	PyramidInfer	73.07	52.00	63.56	38.42	45.66
	WaveFilter	28.00	62.50	65.20	54.63	54.60
	WaveKV	**78.40**	**68.50**	**88.28**	**91.89**	**86.57**
Qwen3-8B	Full KV	86.00	68.50	100.00	99.78	93.44
	FreqKV	93.80	27.00	13.28	22.72	28.57
	FreqKV_0	93.93	26.60	8.28	22.54	27.98
	PyramidInfer	77.60	30.00	14.12	20.26	26.15
	WaveFilter	60.00	57.00	76.20	28.81	40.06
	WaveKV	71.33	53.00	78.40	86.16	**78.75**
Phi3-medium-128k	Full KV	85.33	70.50	100.00	97.44	92.15
	FreqKV	90.73	42.80	25.80	25.59	33.90
	FreqKV_0	90.87	38.00	15.44	24.07	31.24
	PyramidInfer	72.20	45.40	44.76	25.07	34.03
	WaveFilter	15.67	53.50	66.00	15.94	26.35
	WaveKV	87.33	66.50	72.40	96.59	**88.79**

similarly compressed methods such as FreqKV (43.50%) and PyramidInfer (42.00%). Prior work on KV-cache compression also finds arithmetic reasoning to be more sensitive to aggressive compression than knowledge-oriented tasks Liu et al. (2025). This suggests that preserving the semantic continuity of intermediate reasoning states is especially important for multi-step arithmetic generation. In contrast, WaveKV nearly preserves MMLU performance, achieving 69.39% compared with 69.04% for Full KV. This difference between GSM8K and MMLU is consistent with the different demands of the two benchmarks. MMLU has been characterized as predominantly knowledge-driven, with many questions requiring less multi-step reasoning than more reasoning-focused benchmarks Wang et al. (2024a). Even when the prompt requests an explicit reasoning trace, the information needed for many MMLU questions may therefore be recovered from relatively localized knowledge representations. GSM8K instead requires a sequence of dependent arithmetic operations, which makes errors in earlier reasoning more likely to affect later steps. WaveFilter achieves higher GSM8K accuracy at 84.00%, but retains the full KV cache and therefore avoids the information loss introduced by cache compression. Overall, these results show that WaveKV preserves strong knowledge performance under compression, while long-form arithmetic reasoning remains more sensitive to compression of intermediate states.

Table 12: Long text generation with reasoning on LLaMA-3.1-8B.

Method	GSM8K	MMLU
Full KV	82.50	69.04
GemFilter	1.00	15.50
LaCache	1.50	6.00
StarKV	1.00	4.00
SnapKV	3.50	54.00
FastKV	3.00	34.00
CaM	0.00	2.50
D2O	4.50	24.50
PyramidKV	7.00	40.00
ThinKV	6.50	52.00
FreqKV	43.50	64.65
PyramidInfer	42.00	68.16
WaveFilter	84.00	65.70
WaveKV	**74.50**	**69.39**