

LKV: End-to-End Learning of Head-wise Budgets and Token Selection for LLM KV Cache Eviction

Enshuai Zhou^{1,2}, Yifan Hao^{2,3}, Chao Wang¹, Rui Zhang^{2,3}, Di Huang^{2,3},
Jiaming Guo^{2,3}, Xing Hu^{2,3}, Zidong Du^{2,3}, Qi Guo^{2,3}, Yunji Chen^{4,2,3,*}

¹University of Science and Technology of China, Hefei, China

²State Key Lab of Processors, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China

³University of Chinese Academy of Sciences, Beijing, China

⁴Institute of AI for Industries, Chinese Academy of Sciences, Nanjing, China

enszhou@mail.ustc.edu.cn, cyj@ict.ac.cn

Abstract

Long-context inference in Large Language Models (LLMs) is bottlenecked by the linear growth of Key-Value (KV) cache memory. Existing KV cache compression paradigms are fundamentally limited by heuristics: heuristic budgeting relies on statistical priors rather than task objectives, causing resource misallocation, while heuristic selection relies on coupled query-key interactions or static inductive biases (e.g., attention sinks). To address this limitation, we introduce LKV (Learned KV Eviction), which formulates KV compression as an end-to-end differentiable optimization problem. LKV integrates LKV-H to learn task-optimized global budgets, and LKV-T to derive intrinsic KV importance without materializing attention matrices. This design bypasses heuristic proxies, strictly aligning compression with task objectives. Extensive evaluations demonstrate that LKV achieves state-of-the-art performance on both LongBench and RULER benchmarks at high compression rates. In particular, on LongBench, LKV achieves near-lossless performance with only 15% KV cache retention, and delivers up to $3.19\times$ decoding throughput gains with negligible ($\sim 0.7\%$) overhead. Crucially, our analysis identifies learned budgeting as the dominant driver of fidelity, demonstrating that data-driven allocation is essential to overcome the limitations of hand-crafted heuristics.¹

1 Introduction

Processing long contexts is essential for Large Language Models (LLMs) applications like repository-level code analysis (Zhang et al., 2023a) and multi-document question answering (Liu et al., 2024a). However, processing these extensive inputs imposes a prohibitive *memory wall* (Gholami et al.,

2024). As the model processes a long sequence, the accumulation of Key-Value (KV) states triggers an immediate bottleneck (Kwon et al., 2023), which severely limits context length and throughput on memory-constrained hardware (Sheng et al., 2023). KV cache eviction (Liu et al., 2023; Zhang et al., 2023b) mitigates this by dynamically discarding non-essential tokens to reduce memory footprint. However, current approaches remain constrained by two structural limitations.

First, budget allocation relies on hand-crafted rules: uniform (Zhang et al., 2023b; Li et al., 2024), layer-wise decay (Cai et al., 2025), or statistical proxies (Feng et al., 2025a; Wan et al., 2025). These heuristics are hard-coded and cannot be optimized against a task objective, forcing reliance on manual safeguards (minimum budgets, attention sinks (Xiao et al., 2023)) that constrain the discovery of complex sparsity patterns. Second, token selection typically depends on attention scores, creating a circular dependency: identifying which tokens to evict requires the $\mathcal{O}(t^2)$ computation that eviction aims to avoid (Adnan et al., 2024).

We posit that optimal KV compression should be an end-to-end learned capability rather than a collection of heuristic rules. Instead of relying on manual rules, the task objective itself should directly guide the optimization of both macro-level budget allocation (e.g., distinguishing critical retrieval heads from streaming heads) and micro-level token utility (e.g., balancing distant anchors against recent local context). We introduce **LKV** (Learned KV Eviction), which reformulates KV cache compression as a unified, end-to-end differentiable optimization problem.

As shown in Figure 1, unlike uniform, layer-wise, or heuristic allocation strategies, LKV learns a global, constraint-free budget policy. **LKV-H**

*Corresponding author.

¹Code is available at <https://github.com/enszhou/LKV>.

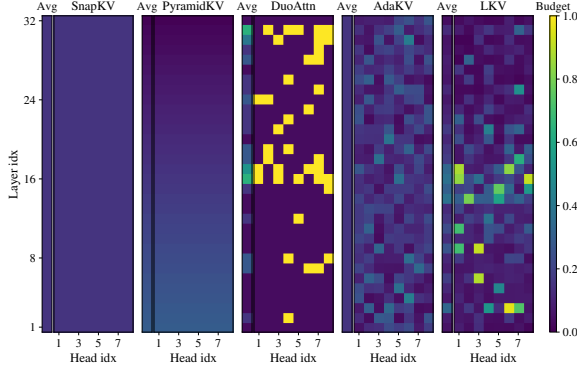


Figure 1: KV budget allocation policies (15% retention). (a) SnapKV: Uniform. (b) PyramidKV: Layer-wise decay. (c) DuoAttention: Rigid binary classification (Retrieval vs. Streaming). (d) Ada-SnapKV: Adaptive within layers but with uniform layer priors. (e) LKV (Ours): Learned global, fine-grained policy optimizing task objectives without rigid priors.

(Global Learned Budgeting) flattens all heads into a single competition via learnable embeddings, enabling cross-layer resource transfer unrestricted by structural priors.

We train LKV end-to-end via self-distillation, with the LLM backbone fully frozen. Only the lightweight LKV modules ($\sim 8.5\text{M}$ parameters, $\sim 0.1\%$ of an 8B backbone) are optimized, taking under 2 GPU-hours—a one-time offline cost. Extensive experiments on LongBench (Bai et al., 2024) and RULER (Hsieh et al., 2024) demonstrate that LKV significantly outperforms both training-free heuristics and trained methods like DuoAttention, while delivering $1.99\text{--}3.19\times$ decoding throughput gains with $\sim 0.7\%$ overhead. Our analysis further reveals that the *learned global budget* contributes more to performance gains than the selection policy itself, demonstrating the importance of precise, head-level resource allocation. Our contributions are summarized as follows:

- We propose LKV, an end-to-end differentiable framework for KV eviction, training only lightweight external modules with a frozen backbone.
- We introduce LKV-H, a global budgeting mechanism that uncovers intrinsic head importance, and LKV-T, a matrix-free selector for efficient inference.
- We achieve SOTA performance with up to $3.19\times$ decoding throughput gains, and show that learned budgeting dominates compression quality.

2 Related Work

2.1 Eviction-based KV Cache Compression

Budget Allocation Strategies. Early KV eviction methods use uniform allocation across all layers and heads (Zhang et al., 2023b; Li et al., 2024). Recent works introduce non-uniform strategies: PyramidKV (Cai et al., 2025) uses layer-wise decay, D2O (Wan et al., 2025) and CAKE (Qin et al., 2025) adapt budgets based on attention density, and Ada-KV (Feng et al., 2025a) tunes budgets across heads. RocketKV (Behnam et al., 2025) combines SnapKV-based prefill eviction with decode-phase sparse attention. These methods rely on statistical or rule-based heuristics. In contrast, LKV-H treats budgeting as an end-to-end differentiable optimization, enabling global head-wise competition.

Token Selection Mechanisms. Static strategies like StreamingLLM (Xiao et al., 2023) preserve initial and recent tokens to stabilize decoding. Dynamic methods assess importance via attention scores (Zhang et al., 2023b; Li et al., 2024) or value norms (Feng et al., 2025b). KVzip (Kim et al., 2025) is query-agnostic but requires a second inference pass for context reconstruction. Learning-based methods like DuoAttention (Xiao et al., 2024) and PruLong (Bhaskar et al., 2025) identify retrieval heads via distillation. However, most selectors require materializing the attention matrix. In contrast, LKV-T employs matrix-free scoring from intrinsic KV states.

2.2 Other LLM Efficiency Paradigms

LKV is orthogonal to sparse attention (e.g., Quest (Tang et al., 2024), MInference (Jiang et al., 2024)), prompt compression (e.g., LLMLingua-2 (Pan et al., 2024)), KV quantization (e.g., KIVI (Liu et al., 2024b)), and hybrid systems like ShadowKV (Sun et al., 2025) (SVD-based key compression with CPU offloading). LKV reduces token cardinality through learned eviction and integrates with these methods for complementary gains.

3 Methodology

3.1 Preliminaries and Problem Formulation

Consider a Decoder-only LLM with L layers and H KV heads per layer, accommodating both MHA and GQA (Vaswani et al., 2017; Ainslie et al., 2023). Given input tokens $x_1, \dots, x_t$, the model maintains a KV cache $\mathcal{C}_t = \{(\mathbf{K}_{\leq t}^{(l,h)}, \mathbf{V}_{\leq t}^{(l,h)})\}_{l=1, h=1}^{L,H}$ where $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{t \times d}$. As

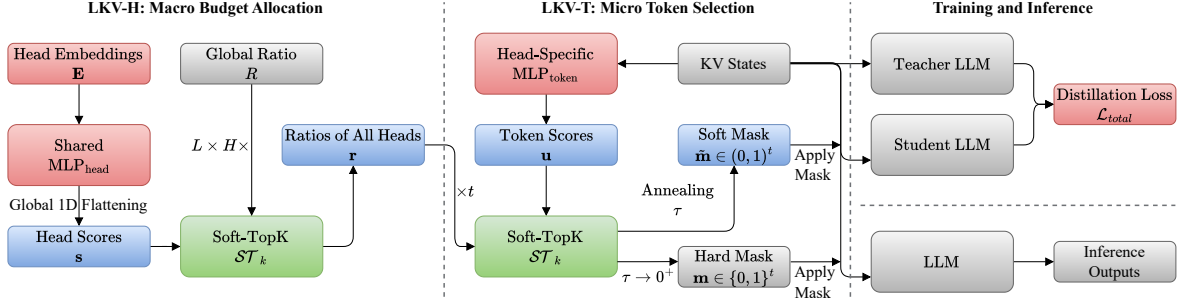


Figure 2: Overview of LKV. **Left (LKV-H):** Learns global budget ratios $\mathbf{r}$ from head embeddings. **Middle (LKV-T):** Performs differentiable, query-agnostic token selection via Soft-TopK. **Right:** End-to-end optimization via self-distillation against a frozen teacher.

t grows, the memory footprint of $\mathcal{C}_t$ scales linearly, imposing a *memory wall* on throughput.

Problem Formulation. Our objective is to compress $\mathcal{C}_t$ by learning a binary retention mask $\mathbf{M} \in \{0, 1\}^{L \times H \times t}$ for all KV cache items. Crucially, this mask operates at the KV-head level, ensuring that any eviction decision is consistently applied across all query heads sharing that KV cache. We define a target global retention ratio $R \in (0, 1)$, which determines the total KV budget at step t as $B_{\text{total}} = \lfloor R \cdot L \cdot H \cdot t \rfloor$. We formulate LKV as an end-to-end optimization problem to minimize the discrepancy between the full-cache and compressed models:

$$\begin{aligned} \min_{\phi} \quad & \mathbb{E}_{x \sim \mathcal{D}} [\mathcal{L}(\text{LLM}(\mathcal{C}_{\text{full}}), \text{LLM}(\mathbf{M} \odot \mathcal{C}_{\text{full}}; \phi))] \\ \text{s.t.} \quad & \|\mathbf{M}\|_0 = \sum_{l=1}^L \sum_{h=1}^H \|\mathbf{m}^{(l,h)}\|_0 \leq B_{\text{total}}. \end{aligned} \quad (1)$$

where ϕ represents the learnable parameters of LKV, and $\mathcal{L}$ denotes the distillation loss function, which will be detailed in Section 3.5. The overall framework of LKV is illustrated in Figure 2. We now describe the core components of our method in detail.

3.2 The Differentiable Soft-TopK Operator

End-to-end learning requires a differentiable approximation of the Top- k selection. Heuristic methods often rely on non-differentiable sorting, which block the gradient flow required for optimizing the budget allocation and selection policy. To bridge this gap, we adopt the **Soft-TopK Operator**, denoted as $\mathcal{ST}_k(\cdot)$, which is based on the smooth approximation framework explored by Su (2024). Unlike simple temperature-scaled approximations,

this formulation guarantees rigorous constraint satisfaction and numerical stability.

Mathematical Definition. Let $\mathbf{x} \in \mathbb{R}^n$ be the input score vector and $k \in (0, n)$ be the target parameter. We define the element-wise Soft-TopK operator with an input-dependent threshold $\lambda(\mathbf{x})$ as:

$$\begin{aligned} \mathcal{ST}_k(\mathbf{x})_i &\triangleq f(x_i - \lambda(\mathbf{x})) \\ \text{s.t.} \quad & \sum_{i=1}^n f(x_i - \lambda(\mathbf{x})) = k. \end{aligned}$$

We employ the CDF of the Laplace distribution as f for its symmetric gradients and analytical tractability: $f(z) = \frac{1}{2}(1 + \text{sgn}(z)(1 - e^{-|z|}))$. Consequently, the core challenge lies in analytically determining $\lambda(\mathbf{x})$ such that the summation constraint is strictly satisfied.

Closed-Form Solution for λ . To strictly enforce the budget constraint, we derive an analytical solution for the threshold $\lambda(\mathbf{x})$. Assuming sorted inputs, the constraint $\sum f(x_i - \lambda) = k$ yields a quadratic equation in terms of e^λ . The valid root provides the explicit closed-form:

$$\lambda(\mathbf{x}) = \log \left(\frac{\sqrt{(k - m)^2 + S_1 S_2} - (k - m)}{S_1} \right),$$

where the partial sums are defined as $S_1 = \sum_{i=1}^m e^{-x_i}$ and $S_2 = \sum_{i=m+1}^n e^{x_i}$. This analytical formulation guarantees exact budget satisfaction and maintains end-to-end differentiability with respect to both input scores $\mathbf{x}$ and the budget parameter k . Detailed derivation, numerical stability analysis, and gradient proofs are provided in Appendix A.

Theoretical Properties. This formulation constructs a mapping $\mathcal{ST}_k : \mathbb{R}^n \rightarrow (0, 1)^n$ satisfying three properties:

- **Normalization:** The output vector always sums to k , enforcing the budget constraint.
- **Order Preservation:** $x_i > x_j \implies \mathcal{ST}_k(\mathbf{x})_i \geq \mathcal{ST}_k(\mathbf{x})_j$, ensuring prioritization of important tokens.
- **Translation Invariance:** $\mathcal{ST}_k(\mathbf{x} + c) = \mathcal{ST}_k(\mathbf{x})$. This decouples selection from absolute magnitude, stabilizing training.

Theoretical Unification. Our framework bridges established selection mechanisms through a generalized perspective. A temperature parameter $\tau > 0$ can be naturally introduced (via input scaling $\mathbf{x}/\tau$) to adjust the sharpness of the distribution. This reveals two critical limiting cases: the operator approximates to the standard Softmax function when $k = 1$ (at $\tau = 1$), and converges to the discrete Hard Top- k selection in the limit as $\tau \rightarrow 0^+$.

3.3 LKV-H: Global Budget Allocation

A critical limitation of existing compression paradigms is their fragmented scope of budget allocation. Heuristic methods typically operate within restricted structural boundaries: methods like AdaKV focus solely on *intra-layer* redistribution, while approaches like PyramidKV enforce fixed *inter-layer* decay patterns. Achieving a holistic allocation often requires combining multiple heuristics, which is suboptimal and complex. LKV-H removes these constraints by formulating budget allocation as a **unified, global competition**. Instead of optimizing layer-wise or head-wise separately, we flatten the entire model’s attention structure into a single dimension, allowing every head to compete for the global memory capacity directly.

Head Embeddings and Scoring. We assign a learnable embedding vector $\mathbf{e}^{(l,h)} \in \mathbb{R}^{d_e}$ to each KV head. A shared MLP maps these embeddings to scalar importance scores:

$$s^{(l,h)} = \text{MLP}_{\text{head}}(\mathbf{e}^{(l,h)}).$$

Unlike statistical proxies (e.g., attention concentration) used in prior works, these scores quantify the intrinsic contribution of each head to the training objective.

Global 1D Competition. Let $R \in (0, 1)$ be the target global retention ratio. The aggregate retention ratio (sum of ratios among all heads) is $R_{\text{agg}} = R \cdot L \cdot H$. Crucially, we bypass hierarchical rigidities by flattening the score tensor $\mathbf{s} = \{s^{(l,h)}\}^{L \times H}$ into a one-dimensional vector

$\mathbf{s}_{\text{flat}} \in \mathbb{R}^{LH}$. This formulation transforms allocation into a global competition, enabling *cross-layer budget transfer*: a critical head in the deepest layer can effectively claim capacity from a redundant head in the first layer, forcing resources to align strictly with information density rather than structural depth. We then apply the Soft-TopK operator (with $\tau = 1$) to this global vector:

$$\mathbf{r}_{\text{flat}} = \mathcal{ST}_{R_{\text{agg}}}(\mathbf{s}_{\text{flat}}), \quad \text{where} \quad \sum_{i=1}^{L \cdot H} (\mathbf{r}_{\text{flat}})_i = R_{\text{agg}}.$$

By reshaping $\mathbf{r}_{\text{flat}}$ back to $\mathbf{r} \in \mathbb{R}^{L \times H}$, we obtain the specific retention ratio $r^{(l,h)} \in (0, 1)$ for each head. This *flattened optimization* enables cross-layer resource transfer: the model can automatically discover complex patterns—such as allocating heavy budgets to middle-layer retrieval heads while sparsifying both initial and final layers—that rigid layer-wise heuristics can never approximate.

Zero-Overhead Budgeting at Inference. Since the learned ratios $\mathbf{r}$ depend solely on static embeddings, they are pre-computed once and frozen. At inference step t , the discrete token budget for head (l, h) is simply $b^{(l,h)} = \lfloor r^{(l,h)} \cdot t \rfloor$, incurring zero runtime overhead. With the precise budget established, the remaining challenge is to select specific tokens, which we address next.

3.4 LKV-T: Matrix-Free Token Selection

Having established the optimal capacity (budget) for each head via LKV-H, the framework must now determine the specific content (tokens) to retain. While prior methods like H2O and SnapKV rely on query-key attention scores to assess token utility, they suffer from a "chicken-and-egg" problem: evaluating utility requires computing the heavy attention matrix that compression aims to avoid.

LKV-T resolves this by introducing a **Matrix-Free** selection mechanism. We posit that a token’s utility is intrinsic to its latent features and can be predicted independently of the current query.

Head-Specific Intrinsic Scoring. To capture the distinct sparsity patterns of different attention heads, we assign a unique, lightweight MLP to each head. For the i -th token in head h of layer l , we concatenate its key $\mathbf{k}_i^{(l,h)}$ and value $\mathbf{v}_i^{(l,h)}$ to predict an intrinsic importance score u_i :

$$u_i^{(l,h)} = \text{MLP}_{\text{token}}^{(l,h)}([\mathbf{k}_i^{(l,h)}; \mathbf{v}_i^{(l,h)}]).$$

This scoring is *query-agnostic*, meaning the importance is derived solely from the KV states themselves, avoiding dependency on the current query $\mathbf{q}_t$.

Hierarchical Application of Soft-TopK. Note that the Soft-TopK operator here differs from its usage in LKV-H (Sec. 3.3). In LKV-H, the operator is used to *generate the retain ratios* $\mathbf{r} = \{r^{(l,h)}\}$ for all heads. Subsequently in LKV-T, we employ the operator to *select tokens* based on the allocated budget. To enable joint optimization, we feed the continuous budget $b^{(l,h)} = r^{(l,h)} \cdot t$ into the operator during training, allowing gradients to propagate from token selection to the budget allocation due to differentiability with respect to b . We apply the discrete floor operation $b^{(l,h)} = \lfloor r^{(l,h)} \cdot t \rfloor$ only at inference time.

Gumbel-Based Relaxation and Annealing. A major challenge in learning binary masks is the discrepancy between the differentiable soft mask used during training and the discrete hard selection required for inference. To bridge this gap, we adopt a Gumbel-Noise relaxation combined with temperature annealing.

During **training**, to encourage exploration and differentiability, we inject Gumbel noise $\mathbf{g} \sim \text{Gumbel}(0, 1)$ into the scores and apply the Soft-TopK operator with a temperature parameter τ :

$$\tilde{\mathbf{m}} = \mathcal{ST}_b(\mathbf{u} + \mathbf{g}; \tau), \quad \text{where} \quad \sum_{i=1}^t \tilde{m}_i = b.$$

Here, $\tilde{\mathbf{m}} \in (0, 1)^t$ represents the continuous relaxation of the retention mask. The temperature τ controls the sharpness of the distribution. We employ a *temperature annealing schedule*, gradually decaying τ from a high value (high randomness, soft selection) to a near-zero value (deterministic, hard selection) over the course of training.

During **inference**, we remove the stochasticity and soft approximation. We perform a deterministic Hard Top-K selection on the raw scores $\mathbf{u}$ to retain exactly b tokens:

$$\mathbf{m} = \text{TopK}(\mathbf{u}, b), \quad \mathbf{m} \in \{0, 1\}^t.$$

Thanks to the annealing process, the distributional shift between the training mask $\tilde{\mathbf{m}}$ (at late stages) and the inference mask $\mathbf{m}$ is minimized.

3.5 Training Objective

Distillation Loss. We train LKV via self-distillation (Zhang et al., 2019) while keeping the LLM backbone entirely **frozen**. The Student (S , compressed) learns to mimic the Teacher (T , full-cache) by minimizing a joint objective that aligns both output distributions and intermediate representations:

$$\mathcal{L}_{\text{total}} = D_{\text{KL}}(P_T \parallel P_S) + \frac{\beta}{L} \sum_{l=1}^L \|\mathbf{H}_T^{(l)} - \mathbf{H}_S^{(l)}\|_2^2$$

where P denotes output probabilities, $\mathbf{H}^{(l)}$ represents hidden states at layer l , and β balances the two terms.

Numerical Stability. Standard *additive* attention masking ($QK^\top + \log M$) introduces gradient singularities ($\nabla M \propto 1/M$) as masks approach zero. To ensure robust training, we implement a custom differentiable FlashAttention kernel. This kernel employs a *multiplicative* formulation that is algebraically equivalent to the standard attention but numerically stable even under BF16 precision (details in Appendix B).

4 Experiments

4.1 Experimental Setup

Models. We evaluate LKV on two widely adopted open-weights models: Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen3-8B (Yang et al., 2025). Both models inherently employ GQA for efficient inference. Crucially, our method is fully compatible with the GQA structure and achieves significant *further* compression on top of these already efficient architectures, adding only $\sim 0.1\%$ trainable parameters (see architecture details in Appendix C.1).

Benchmarks. To evaluate the quality of compression, we adopt two standard benchmarks. Long-Bench (Bai et al., 2024) is used as our primary benchmark for understanding long-contexts, where we select 16 representative subtasks. See Appendix D.1 for detailed configurations and filtering criteria. We further evaluate on RULER (Hsieh et al., 2024), a synthetic benchmark that systematically probes model performance under compressed long-context inputs across varying sequence lengths and task complexities.

Training Details. LKV is trained via self-distillation on our **Loom** dataset ($\sim 23k$ samples), strictly decontaminated from evaluation benchmarks (see Appendix C.2). The process is cost-effective, taking < 2 hours on $8 \times A100$ GPUs (hyperparameters in Appendix C.3).

Baselines. Implemented via KVPress (Devoto et al., 2025), we compare LKV against Full Cache and several baselines: SnapKV (Li et al., 2024) (uniform budgets); PyramidKV (Cai et al., 2025) (layer-wise budget decay); DuoAttention (Xiao et al., 2024) (offline optimized head patterns); and AdaKV (Feng et al., 2025a) (specifically *Ada-SnapKV*, in-layer adaptive budgeting). Additionally, Expected Attention (ExpAttn) (Devoto et al., 2025) is included for component analysis. All experiments follow a question-agnostic (Feng et al., 2025a) setup, where the question is unavailable during context compression.

4.2 General Long-Context Capabilities

We evaluate the comprehensive capabilities of LKV on LongBench. Table 1a presents the comparison against representative baselines on Llama-3.1-8B-Instruct under a strict 15% KV budget ($R = 0.15$); for the complete task-wise breakdown, please refer to Table 12. Figure 3 visualizes the performance trends across varying retention ratios.

Overall Performance Comparison. LKV achieves an average score of 46.73, consistently outperforming representative baselines. It recovers 98.4% of the Full Cache performance (46.73 vs. 47.48), demonstrating virtually lossless context retention. Compared to the strongest heuristic baseline (Ada-SnapKV) in our main comparison, our method leads by a substantial margin of 7.67 points. Against two recent adaptive baselines, D2O and CAKE, it further leads by 4.58 and 2.91 points, respectively (Appendix E.2). This gap highlights the structural advantage of our approach: while heuristic methods relying on local statistics often misallocate resources, LKV learns a global budgeting policy that directly aligns with the generation objective. We observe consistent superiority on Qwen3-8B and Mistral-7B-Instruct-v0.3 as well (see Tables 13 and 16).

Overcoming Rigid Priors. A key advantage of LKV is its ability to break free from hand-crafted structural constraints. Baselines like PyramidKV and DuoAttention enforce pre-defined patterns—

specifically, a rigid layer-wise decay or a fixed offline binary classification. As shown in Table 1a, these methods suffer severe degradation on retrieval-intensive tasks. For instance, in the Synthetic category (heavily impacted by the *Passage Retrieval* task, see full breakdown in Table 12), DuoAttention and PyramidKV drop to 17.75 and 24.99, respectively. These rigid priors aggressively prune deep layers or specific heads that are critical for retrieving specific information. In contrast, LKV achieves a high score of 53.38 (matching Full Cache) by automatically identifying and preserving these critical capacities.

Learned vs. Heuristic Adaptation. Although Ada-SnapKV introduces adaptivity, its marginal improvement over the uniform SnapKV baseline (39.06 vs. 38.05) exposes the fundamental limitations of heuristic-based allocation. First, Ada-SnapKV relies on *attention concentration* as a proxy for budget demand. This heuristic is insufficient because a head’s concentration pattern does not necessarily correlate with its task importance, especially when the method is still constrained by a rigid uniform layer-wise budget. It fails to reallocate resources across layers to preserve deep-context information. Second, the underlying token selection still depends on accumulated attention scores, which rely on historical statistics rather than predicting future retrieval needs. LKV overcomes these dual barriers by learning a global policy that directly optimizes the generation objective, unrestricted by layer boundaries or heuristic proxies. Consequently, this structural advantage substantially reduces reliance on runtime query access. While query-aware evaluation boosts heuristics by +7–14 points, none surpass LKV’s query-agnostic baseline (Appendix 17). LKV’s negligible delta (+0.72) confirms it inherently captures query-invariant patterns, thereby avoiding the $\mathcal{O}(t^2)$ cost of query-dependent selection.

Robustness across Budgets. Figure 3 illustrates the performance consistency of LKV on both Llama-3.1-8B-Instruct and Qwen3-8B models. Our method consistently maintains a superior performance-efficiency trade-off compared to baselines. Notably, the performance advantage remains highly significant as the budget decreases (e.g., at 10% budget). This demonstrates that LKV is also exceptionally robust in extreme low-resource regimes.

Table 1: Main results on LongBench and RULER length generalization (Llama-3.1-8B-Instruct, $R = 0.15$).

(a) LongBench								(b) RULER (32K–128K)			
Method	Avg.	S-QA	M-QA	Summ	FewS	Synth	Code	Method	32K	64K	128K
Full Cache	47.48	43.57	45.41	28.89	65.43	53.48	51.42	Full Cache	88.51	81.76	62.30
SnapKV	38.05	27.27	34.08	22.93	58.11	40.73	50.05	SnapKV	35.31	27.87	12.82
PyramidKV	32.12	25.17	26.75	22.57	49.29	24.99	46.29	PyramidKV	31.82	24.42	10.53
DuoAttention	31.03	21.76	25.94	21.66	50.71	17.75	50.40	DuoAttention	12.04	10.94	7.26
Ada-SnapKV	39.06	27.38	35.46	23.29	59.28	43.77	50.56	Ada-SnapKV	41.50	32.36	18.69
LKV (Ours)	46.73	41.66	42.79	28.76	66.30	53.38	51.25	LKV (Ours)	82.40	74.36	55.34

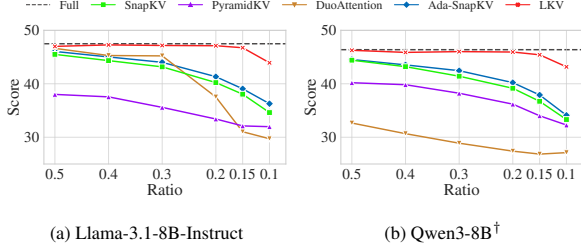


Figure 3: Performance on LongBench across varying KV cache retention ratios. LKV shows robustness in low-resource regimes.

[†]For Qwen3, we employ an on-the-fly approximation due to the lack of official pre-computed patterns. (see Appendix D.2)

4.3 Length and Compression Robustness

We further evaluate the robustness of LKV on Llama-3.1-8B-Instruct by varying the context length t and the global retention ratio R . The results are visualized in Figure 4. Results on Qwen models show consistent trends and are visualized in Figure 6.

Scalability to Long Contexts. Figure 4a and Table 1b evaluate RULER performance from 4k to 128k ($R = 0.15$). Despite training within 16k, LKV generalizes well across the full range. At 128K— $8\times$ beyond the training window—LKV recovers 88.8% of Full Cache (55.34 vs. 62.30), leading Ada-SnapKV by +36.7 points. We attribute this to the length invariance of the learned policy: LKV-H captures head-level sparsity patterns independent of sequence position, while LKV-T derives utility strictly from the KV states themselves. Heuristic baselines collapse beyond 32K (Ada-SnapKV drops from 41.50 at 32K to 18.69 at 128K), confirming that statistical proxies fail under extreme extrapolation.

Performance under Extreme Compression.

Figure 4(b) evaluates performance on fixed length inputs (16k) across varying retention ratios (from 0.5 down to 0.1). The advantage of LKV becomes

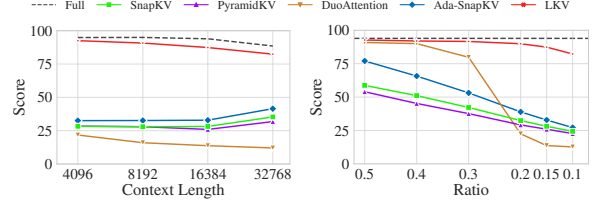


Figure 4: Robustness evaluation on RULER (Llama-3.1-8B-Instruct).

more pronounced in low-resource regimes. Even when the retention ratio is reduced to $R = 0.1$, LKV retains a high degree of performance. While AdaKV performs better than rigid priors (e.g., PyramidKV, DuoAttention) due to its adaptive nature, it still lags behind LKV under extreme compression. Structural priors like DuoAttention experience significant degradation at lower ratios, failing to adapt their rigid head-pruning strategies to tight budgets. This confirms that LKV utilizes the limited budget more effectively than both heuristic adaptive methods and static priors.

4.4 Ablation Study: Dissecting the Gains

To disentangle the contributions of budget allocation (LKV-H) and token selection (LKV-T), we conduct a comprehensive grid analysis on LongBench using Llama-3.1-8B-Instruct (15% budget). We combine three budgeting strategies (Uniform, AdaKV, LKV-H) with three token selection policies: SnapKV, LKV-T, and ExpAttn, a recent method that selects tokens by estimating their expected future attention. Table 2 reports the macro-average score (Avg.) across all 16 subtasks, along with results on representative retrieval-intensive subsets.

Dominance of Learned Budgeting (LKV-H).

The results reveal that budget allocation is the primary driver of performance. While heuristic adap-

Table 2: Impact of different budgeting and selection strategy combinations. Evaluated on LongBench (Llama-3.1-8B-Instruct) under 15% KV budget.

Selection	Budgeting	Avg.	Δ_{Base}	Multi-Doc QA	Synthetic
SnapKV	Uniform	38.05	-	34.08	40.73
	AdaKV	39.06	+1.01	35.46	43.77
	LKV-H	44.06	+6.01	41.75	52.87
ExpAttn	Uniform	37.12	-	37.39	17.77
	AdaKV	38.16	+1.04	37.33	23.26
	LKV-H	45.72	+8.60	43.27	45.00
LKV-T	Uniform	39.74	-	40.27	22.41
	AdaKV	43.45	+3.71	38.83	47.76
	LKV-H	46.73	+6.99	42.79	53.38

tation (AdaKV) provides only marginal gains over the Uniform baseline, our learned policy (LKV-H) delivers massive improvements across all sectors. For instance, LKV-H boosts ExpAttn by +8.60 points on average. Crucially, under uniform budgeting, ExpAttn suffers severe degradation on retrieval-heavy tasks (e.g., 17.77 on Synthetic); however, LKV-H acts as a universal enhancer, rescuing it to 45.00. We observe consistent gains on real-world tasks like Multi-Doc QA (e.g., +7.67 gain for SnapKV), proving that LKV-H effectively corrects the flaws of heuristic selectors.

Superiority of LKV-T. Comparing the columns in Table 2, our LKV-T selector consistently outperforms existing methods. Under the Uniform budget, LKV-T achieves 39.74, surpassing both SnapKV (38.05) and ExpAttn (37.12). When combined with our learned budget, the full LKV system achieves the highest average score of 46.73. This confirms that LKV-T provides a highly accurate yet efficient (Matrix-free) selection mechanism.

Training Data Decontamination. To rule out the possibility that our gains reflect adaptation to LongBench-overlapping task families, we re-train LKV on two decontaminated Loom variants: **Loom2**, which replaces the overlapping families (HotpotQA, MultiNews, NarrativeQA, TREC; 65% of Loom) with LongAlign-10k, and **Loom3**, a removal-only control that drops them entirely (composition in Table 6). As shown in Table 3, both variants retain a 6+ point margin over Ada-SnapKV, with Loom2 within 0.2 points and Loom3 within 1.5 points of Loom. Under Loom3, the directly overlapping tasks degrade only marginally relative to Full Cache: HotpotQA and MultiNews drop by 0.80 and 0.50 points and NarrativeQA improves by 0.59; the single meaningful drop is TREC (-5.00), reflecting the removal of its 5-

Table 3: Decontamination analysis on LongBench (Llama-3.1-8B-Instruct, $R = 0.15$). Loom2 replaces the LongBench-overlapping task families with LongAlign-10k; Loom3 removes them entirely.

Method	Avg.	S-QA	M-QA	Summ	FewS	Synth	Code
Full Cache	47.48	43.57	45.41	28.89	65.43	53.48	51.42
Ada-SnapKV	39.06	27.38	35.46	23.29	59.28	43.77	50.56
LKV (Loom)	46.73	41.66	42.79	28.76	66.30	53.38	51.25
LKV (Loom2)	46.53	41.72	44.80	28.54	64.29	53.35	49.84
LKV (Loom3)	45.25	42.81	43.08	28.45	63.46	44.51	50.81

shot formatting rather than task leakage. This confirms that LKV learns a transferable eviction policy rather than task-specific heuristics.

4.5 Efficiency Analysis

We evaluate the computational and memory efficiency of LKV using the Llama-3.1-8B-Instruct model ($R = 0.15$) on an A100-80GB GPU. All experiments use a custom implementation based on `flash_attn_varlen_func` (Dao et al., 2022), which flattens the compressed KV cache into a compact format to eliminate padding overhead.

Latency and Throughput. LKV incurs a modest 7.7% prefill latency overhead, comparable to heuristic baselines (Appendix E.1). During decoding, this cost becomes negligible. Table 4 reports decoding throughput and LKV overhead. LKV uses chunk-based score-and-replace (chunk size 16): LKV scores only new tokens and overwrites low-importance entries in-place, with zero memory reallocation. LKV overhead remains $\sim 0.7\%$ across all configurations, while throughput gains reach $1.99\text{--}3.19\times$ over Full Cache. Notably, LKV operates at $4\times 64\text{K}$ and $1\times 256\text{K}$ where Full Cache runs out of memory.

Memory Efficiency. Figure 5 dissects the memory consumption across context lengths. First, regarding system stability, Full Cache exhibits rapid memory growth, crashing with OOM errors at 225k tokens. In contrast, LKV successfully scales to 262k and beyond. This capability stems from the aggressive compression of the KV tensors (Figure 5b). By maintaining a 15% budget, LKV reduces the actual storage cost at 200k length from 25.0 GB (Full) to 3.75 GB (Ours), achieving a $6.6\times$ reduction. It is worth noting that the reduction in total peak memory (Figure 5a) appears less aggressive than the storage compression ratio. This is because the peak footprint is dominated by static model weights and transient activation buffers, which partially mask the savings from the

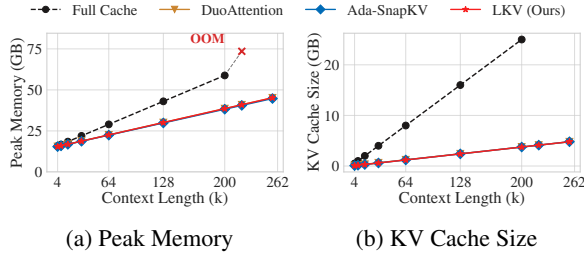


Figure 5: Memory profiling on Llama-3.1-8B-Instruct ($R = 0.15$).

Table 4: Decoding throughput (tokens/s) and LKV scoring overhead (% of total inference time). $R = 0.15$

BS×Ctx	Full	LKV	Speedup	Overhead
8×16K	138.9	276.7	1.99×	0.63%
4×32K	70.1	148.3	2.12×	0.67%
4×64K	OOM	147.9	—	0.67%
1×200K	12.6	40.3	3.19×	0.73%
1×256K	OOM	40.2	—	0.70%

compressed KV cache. Nevertheless, the substantial reduction in KV storage is sufficient to extend the maximum context length significantly beyond the hardware limit of Full Cache.

5 Conclusion

We reformulate KV cache eviction as an end-to-end differentiable optimization problem, breaking the reliance on rigid heuristics. By combining global learned budgeting (LKV-H) and matrix-free token selection (LKV-T), LKV achieves SOTA fidelity on LongBench and RULER, delivering near-lossless performance with only 15% retention, a $6.6\times$ storage reduction, and up to $3.19\times$ decoding throughput gains. Crucially, learned budgeting dominates compression quality, enabling extended-context deployment on commodity hardware. While requiring lightweight training, LKV establishes a data-driven framework for efficient long-context inference.

Limitations

LKV’s learned components are model-dependent: the budget allocation patterns and token selection policy are optimized for specific backbone architectures, and their generalization to substantially different models warrants further study. Additionally, while our current implementation is compatible with FlashAttention’s variable-length interface, full integration with PagedAttention-based serving frameworks requires further engineering, which we

leave as future work.

Acknowledgements

This work is partially supported by the NSF of China (Grants No.62525203, 62302478, 62341411, U22A2028, 6240073476), the Science and Technology Major Special Program of Jiangsu (Grant No. BG2024028), Strategic Priority Research Program of the Chinese Academy of Sciences (Grants No.XDB0660200, XDB0660201, XDB0660202) and Youth Innovation Promotion Association CAS.

We used generative AI tools to polish the language of this manuscript.

References

- Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J. Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. [Keyformer: KV Cache Reduction through Key Tokens Selection for Efficient Generative Inference](#).
- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. [GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints](#). *Preprint*, arXiv:2305.13245.
- Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. [LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.
- Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. 2025. [RocketKV: Accelerating Long-Context LLM Inference via Two-Stage KV Cache Compression](#). In *Forty-Second International Conference on Machine Learning*.
- Adithya Bhaskar, Alexander Wettig, Tianyu Gao, Yihe Dong, and Danqi Chen. 2025. [Cache Me If You Can: How Many KV’s Do You Need for Effective Long-Context LMs?](#) *Preprint*, arXiv:2506.17121.
- Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2025. [PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling](#). *Preprint*, arXiv:2406.02069.
- Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Re. 2022. [FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness](#). In *Advances in Neural Information Processing Systems*.

- Alessio Devoto, Maximilian Jeblick, and Simon Jégou. 2025. [Expected Attention: KV Cache Compression by Estimating Attention from Future Queries Distribution](#). *Preprint*, arXiv:2510.00636.
- Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. 2025a. [Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. 2025b. [Identify Critical KV Cache in LLM Inference from an Output Perturbation Perspective](#). *Preprint*, arXiv:2502.03805.
- Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. 2024. [AI and Memory Wall](#). *IEEE Micro*, 44(3):33–39.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. [The Llama 3 Herd of Models](#). *Preprint*, arXiv:2407.21783.
- Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekeshe, Fei Jia, and Boris Ginsburg. 2024. [RULER: What’s the Real Context Size of Your Long-Context Language Models?](#) In *First Conference on Language Modeling*.
- Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. [MInference 1.0: Accelerating Pre-filling for Long-Context LLMs via Dynamic Sparse Attention](#). *Preprint*, arXiv:2407.02490.
- Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. 2025. [KVzip: Query-Agnostic KV Cache Compression with Context Reconstruction](#). *Preprint*, arXiv:2505.23416.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient Memory Management for Large Language Model Serving with PagedAttention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP ’23*, pages 611–626, New York, NY, USA. Association for Computing Machinery.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. [SnapKV: LLM Knows What You are Looking for Before Generation](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranajpe, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. [Lost in the Middle: How Language Models Use Long Contexts](#). *Transactions of the Association for Computational Linguistics*, 12:157–173.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. [Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time](#). In *Thirty-Seventh Conference on Neural Information Processing Systems*.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhao Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. [KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache](#). In *Proceedings of the 41st International Conference on Machine Learning*, pages 32332–32344. PMLR.
- Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. [LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression](#). *Preprint*, arXiv:2403.12968.
- Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. 2025. [CAKE: Cascading and Adaptive KV Cache Eviction with Layer Preferences](#). *Preprint*, arXiv:2503.12491.
- Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. [FlexGen: High-throughput generative inference of large language models with a single GPU](#). In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *ICML’23*, pages 31094–31116, Honolulu, Hawaii, USA. JMLR.org.
- Jianlin Su. 2024. [Post-Softmax: Searching for Smooth Approximations of Top-k](#). Scientific Spaces (Blog post in Chinese). Title translated from Chinese.
- Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2025. [ShadowKV: KV Cache in Shadows for High-Throughput Long-Context LLM Inference](#). *Preprint*, arXiv:2410.21465.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. [Quest: Query-Aware Sparsity for Efficient Long-Context LLM Inference](#). *Preprint*, arXiv:2406.10774.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is All you Need](#). In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, and Mi Zhang. 2025. [D2O:](#)

Dynamic Discriminative Operations for Efficient Long-Context Inference of Large Language Models. *Preprint*, arXiv:2406.13035.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024. **DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads**. In *The Thirteenth International Conference on Learning Representations*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. **Efficient Streaming Language Models with Attention Sinks**. In *The Twelfth International Conference on Learning Representations*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. **Qwen3 Technical Report**. *Preprint*, arXiv:2505.09388.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zhan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023a. **RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation**. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2471–2484, Singapore. Association for Computational Linguistics.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. 2019. **Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation**. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3712–3721.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023b. **H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models**. In *Thirty-Seventh Conference on Neural Information Processing Systems*.

A Theoretical Analysis of Soft-TopK Operator

This appendix establishes the rigorous mathematical foundation for the Soft-TopK operator introduced in Section 3.2. We provide the derivation of the closed-form threshold and analyze its gradient properties to ensure end-to-end differentiability.

A.1 Derivation of the Closed-Form Solution

The derivation follows the smooth approximation framework for Top-K operators explored by Su (2024). Our goal is to find a scalar threshold $\lambda(\mathbf{x})$

such that the sum of activated scores equals the target budget k :

$$\sum_{i=1}^n f(x_i - \lambda(\mathbf{x})) = k, \quad (2)$$

where the activation function $f(z)$ is based on the Cumulative Distribution Function (CDF) of the Laplace distribution:

$$f(z) = \frac{1}{2}(1 + \text{sgn}(z)(1 - e^{-|z|})). \quad (3)$$

Expanding this function for non-negative and negative inputs, we obtain:

$$f(z) = \begin{cases} 1 - \frac{1}{2}e^{-z} & z \geq 0 \\ \frac{1}{2}e^z & z < 0 \end{cases} \quad (4)$$

Given input scores $\mathbf{x} \in \mathbb{R}^n$ sorted in descending order ($x_1 \geq \dots \geq x_n$), let m be the split index such that $x_m \geq \lambda(\mathbf{x}) \geq x_{m+1}$. We decompose the summation constraint into two parts: the "head" ($i \leq m$) and the "tail" ($i > m$).

For the head elements, where $x_i - \lambda \geq 0$:

$$f(x_i - \lambda) = 1 - \frac{1}{2}e^{-(x_i - \lambda)} = 1 - \frac{1}{2}e^{-x_i}e^{\lambda}.$$

For the tail elements, where $x_i - \lambda < 0$:

$$f(x_i - \lambda) = \frac{1}{2}e^{x_i - \lambda} = \frac{1}{2}e^{x_i}e^{-\lambda}.$$

Substituting these expressions back into the budget constraint yields:

$$\begin{aligned} k &= \sum_{i=1}^m \left(1 - \frac{1}{2}e^{-x_i}e^{\lambda}\right) + \sum_{i=m+1}^n \left(\frac{1}{2}e^{x_i}e^{-\lambda}\right) \\ k &= m - \frac{1}{2}e^{\lambda} \underbrace{\sum_{i=1}^m e^{-x_i}}_{S_1} + \frac{1}{2}e^{-\lambda} \underbrace{\sum_{i=m+1}^n e^{x_i}}_{S_2}, \end{aligned}$$

where S_1 and S_2 are the partial sums of the exponential scores. To solve for λ , we introduce the substitution $y = e^{\lambda}$. Multiplying the equation by $2y$ (noting $y > 0$) transforms it into a standard quadratic equation:

$$S_1 y^2 + 2(k - m)y - S_2 = 0. \quad (5)$$

Applying the quadratic formula and selecting the positive root (since $y = e^{\lambda}$ must be positive, and $S_1, S_2 > 0$):

$$\begin{aligned} y &= \frac{-2(k - m) + \sqrt{4(k - m)^2 + 4S_1 S_2}}{2S_1} \\ &= \frac{-(k - m) + \sqrt{(k - m)^2 + S_1 S_2}}{S_1}. \end{aligned}$$

Finally, taking the natural logarithm gives the closed-form solution:

$$\lambda(\mathbf{x}) = \ln \left(\frac{\sqrt{(k-m)^2 + S_1 S_2} - (k-m)}{S_1} \right). \quad (6)$$

Numerical Stability at Boundaries. The closed-form solution above relies on the denominator $S_1 = \sum_{i=1}^m e^{-x_i}$ being non-zero. A potential singularity arises only if $m = 0$ (selecting zero tokens). Conversely, if $m = n$ (selecting all tokens), $S_2 \rightarrow 0$, which is handled by the numerator but suggests a threshold $\rightarrow -\infty$. In our framework, we strictly restrict the target budget k to the open interval $(0, n)$. Specifically, we clamp the retention ratio R such that the derived budget k never strictly reaches 0 or n . This constraint ensures that the split index m satisfies $1 \leq m < n$, guaranteeing $S_1 > 0$ and $S_2 > 0$. Consequently, the closed-form solution remains numerically stable and bounded throughout the training process.

A.2 Gradient Analysis and Differentiability

To explicitly demonstrate the end-to-end differentiability of the operator, we derive the gradients of λ with respect to the input scores $\mathbf{x}$ and the budget k . Instead of differentiating the complex closed-form solution (Eq. 6) directly, we apply the Implicit Function Theorem to the constraint equation:

$$F(\lambda, \mathbf{x}, k) = \sum_{i=1}^n f(x_i - \lambda) - k = 0. \quad (7)$$

Differentiating implicitly with respect to a specific input score x_j , we have:

$$\begin{aligned} \frac{\partial F}{\partial x_j} + \frac{\partial F}{\partial \lambda} \frac{\partial \lambda}{\partial x_j} &= 0 \\ \Rightarrow f'(x_j - \lambda) - \left(\sum_{i=1}^n f'(x_i - \lambda) \right) \frac{\partial \lambda}{\partial x_j} &= 0. \end{aligned}$$

Rearranging terms yields the gradient with respect to the input:

$$\frac{\partial \lambda}{\partial x_j} = \frac{f'(x_j - \lambda)}{\sum_{i=1}^n f'(x_i - \lambda)}. \quad (8)$$

Similarly, differentiating with respect to the budget k :

$$\frac{\partial \lambda}{\partial k} = -\frac{1}{\sum_{i=1}^n f'(x_i - \lambda)}. \quad (9)$$

Conclusion: The derivative of the activation function, $f'(z) = \frac{1}{2}e^{-|z|}$, corresponds to the Probability Density Function (PDF) of the Laplace

distribution. Since the exponential function is strictly positive for all finite inputs, the denominator $\sum_{i=1}^n f'(x_i - \lambda)$ is always strictly positive (non-zero). This guarantees that the gradients exist and are numerically well-behaved, proving that $\lambda(\mathbf{x})$ is a smooth, differentiable function of both $\mathbf{x}$ and k .

B Derivation and Analysis of Custom Backward Kernel

In Section 3.5, we introduced a custom Triton kernel to address the numerical instability of standard masking mechanisms. This appendix analyzes the gradient stability by comparing the backward passes of the standard log-space formulation and our proposed multiplicative formulation.

B.1 Instability of Standard Log-Space Masking

Standard attention implementations, such as PyTorch’s `F.scaled_dot_product_attention`, typically apply masks as an additive bias in the logit domain. Let s_i be the raw attention score and $m_i \in (0, 1)$ be the learnable soft mask value for the i -th token. The attention probability scalar p_i is computed as:

$$p_i = \text{softmax}(s_i + \log m_i) = \frac{e^{s_i + \log m_i}}{\sum_j e^{s_j + \log m_j}}. \quad (10)$$

To analyze numerical stability, we examine the gradient flow during backpropagation. In automatic differentiation frameworks, gradients are computed via the Chain Rule through the nodes of the computational graph. The graph for the mask branch involves a logarithmic operation:

$$m_i \xrightarrow{\log} z_i \xrightarrow{+s_i} a_i \xrightarrow{\text{softmax}} p_i,$$

where $z_i = \log m_i$ is the bias term. The gradient of the loss $\mathcal{L}$ with respect to the mask m_i is given by:

$$\frac{\partial \mathcal{L}}{\partial m_i} = \frac{\partial \mathcal{L}}{\partial z_i} \cdot \frac{\partial z_i}{\partial m_i} = \frac{\partial \mathcal{L}}{\partial z_i} \cdot \frac{1}{m_i}. \quad (11)$$

The Singularity Problem ($1/m_i$): The critical instability arises from the derivative of the logarithm, $\frac{1}{m_i}$. As the LKV training progresses, the model learns to evict tokens by driving their mask values $m_i \rightarrow 0$. In this regime, the local gradient term $\frac{1}{m_i}$ approaches infinity (∞). In low-precision floating point formats (e.g., BF16 or FP16), this infinity causes numerical overflow, resulting in “Inf”

or “NaN” gradients. This instability occurs at the “log” node independently of the downstream gradients, causing the optimization to collapse whenever the mask attempts to act as a strict filter (i.e., approaching zero).

B.2 Proposed Multiplicative Formulation

To resolve the numerical instability identified in Appendix B.1, we propose a multiplicative formulation that operates directly on the probability domain. This method eliminates the problematic logarithm operator while maintaining exact mathematical equivalence to the standard formulation.

Formulation. The computation proceeds in three steps using the standard FlashAttention tiling pattern:

1. **Raw Softmax:** Compute the standard attention probability on the raw scores $\mathbf{s}$:

$$p_{\text{raw},i} = \text{softmax}(\mathbf{s})_i = \frac{e^{s_i}}{\sum_j e^{s_j}}.$$

2. **Multiplicative Masking:** Apply the soft mask m_i element-wise to the raw probabilities:

$$\tilde{p}_i = p_{\text{raw},i} \cdot m_i.$$

3. **Re-normalization:** Normalize the masked probabilities to ensure they sum to 1:

$$p_i = \frac{\tilde{p}_i}{\sum_j \tilde{p}_j}.$$

Proof of Algebraic Equivalence. Substituting the expression for $p_{\text{raw},i}$ into the re-normalization step:

$$p_i = \frac{\frac{e^{s_i}}{\sum_k e^{s_k}} \cdot m_i}{\sum_j \left(\frac{e^{s_j}}{\sum_k e^{s_k}} \cdot m_j \right)}. \quad (12)$$

The common denominator term $\sum_k e^{s_k}$ (the partition function of the raw softmax) appears in both the numerator and the denominator summation, thus cancelling out:

$$p_i = \frac{e^{s_i} \cdot m_i}{\sum_j e^{s_j} \cdot m_j}. \quad (13)$$

Using the identity $x = e^{\log x}$, we can rewrite m_i as $e^{\log m_i}$:

$$\begin{aligned} p_i &= \frac{e^{s_i} \cdot e^{\log m_i}}{\sum_j e^{s_j} \cdot e^{\log m_j}} = \frac{e^{s_i + \log m_i}}{\sum_j e^{s_j + \log m_j}} \\ &= \text{softmax}(s_i + \log m_i). \end{aligned}$$

Conclusion: Our multiplicative kernel produces output strictly identical to the standard softmax($s + \log m$) formulation during the forward pass, but avoids the explicit computation of $\log m$.

Compatibility with FlashAttention. Crucially, this formulation is fully compatible with the IO-aware design of FlashAttention (Dao et al., 2022). In the standard FlashAttention kernel, the mask is applied as an additive bias ($s_i \leftarrow s_i + \text{bias}_i$) before the exponentiation step in the online softmax algorithm. In our custom Triton kernel, we simply replace this additive operation with a multiplicative one after exponentiation ($e^{s_i} \leftarrow e^{s_i} \cdot m_i$). Since this modification only alters the on-chip arithmetic operations (ALU) without affecting the global memory access patterns or the block-wise tiling structure, it preserves the $O(t^2)$ memory savings and IO efficiency of the original FlashAttention. Consequently, LKV achieves numerical stability with negligible runtime overhead compared to standard kernels.

B.3 Gradient Stability Analysis

We now analyze the gradient stability of our proposed formulation by examining the backward pass. The fundamental difference lies in how the gradient of the loss $\mathcal{L}$ propagates back to the mask m_i .

Gradient Flow Comparison. To pinpoint the source of stability, we compare the partial derivative $\frac{\partial p_i}{\partial m_i}$ (the sensitivity of the output probability to the mask) in both formulations.

1. Standard Log-Space Formulation (Unstable): Recall the computational graph: $m_i \xrightarrow{\log} z_i \xrightarrow{+s_i} \dots \rightarrow p_i$. Applying the chain rule, the gradient contains an inverse term:

$$\frac{\partial p_i}{\partial m_i} = \frac{\partial p_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial m_i} = \frac{\partial p_i}{\partial z_i} \cdot \frac{1}{m_i}. \quad (14)$$

Instability: The term $\frac{1}{m_i}$ acts as a gradient scaling factor. As the model learns to evict tokens ($m_i \rightarrow 0$), this factor approaches infinity, causing gradient explosion regardless of the downstream gradient magnitude.

2. Proposed Multiplicative Formulation (Stable): Recall our new computational graph from Appendix B.2:

$$m_i \xrightarrow{\times p_{\text{raw},i}} \tilde{p}_i \xrightarrow{\text{Normalize}} p_i.$$

Here, the mask m_i interacts directly with the bounded raw probability $p_{\text{raw},i}$ via multiplication.

The chain rule yields:

$$\frac{\partial p_i}{\partial m_i} = \sum_j \frac{\partial p_i}{\partial \tilde{p}_j} \cdot \frac{\partial \tilde{p}_j}{\partial m_i}. \quad (15)$$

Since the operation is element-wise multiplication ($\tilde{p}_j = p_{\text{raw},j} \cdot m_j$), the derivative is non-zero only when $j = i$, and crucially:

$$\frac{\partial \tilde{p}_i}{\partial m_i} = p_{\text{raw},i}. \quad (16)$$

Substituting this back, the gradient flow becomes:

$$\frac{\partial p_i}{\partial m_i} = \underbrace{\frac{\partial p_i}{\partial \tilde{p}_i}}_{\text{Normalization Jacobian}} \cdot p_{\text{raw},i}. \quad (17)$$

Conclusion on Stability. Our formulation effectively replaces the unstable scaling factor $\frac{1}{m_i}$ with the stable factor $p_{\text{raw},i}$.

- **Boundedness:** The raw probability $p_{\text{raw},i}$ is the output of a standard softmax, strictly bounded in $(0, 1]$. It represents the intrinsic attention weight of the token before masking.
- **Zero-Preserving:** When a token is evicted ($m_i \rightarrow 0$), the gradient does not explode; instead, it is scaled by the token’s original importance $p_{\text{raw},i}$.

By shifting the masking operation from the logit domain to the probability domain, we guarantee that the gradient ∇_m remains finite and well-behaved throughout the entire training process, enabling stable optimization even in low-precision (BF16) environments.

C Training Implementation Details

C.1 Model Architecture and Parameter Efficiency

In this section, we detail the architecture and parameter statistics of the LKV modules. The system introduces negligible parameter overhead compared to the backbone LLMs (approximately 0.1% of the total parameters for an 8B model).

LKV-H (Global Budgeting). This module consists of learnable head embeddings and a shared scoring network.

- **Embeddings:** We assign a unique embedding vector $\mathbf{e}^{(l,h)} \in \mathbb{R}^{d_{\text{head}}}$ to each KV head ($L \times H_{kv}$ embeddings in total).

- **Shared Predictor:** A single, shared 2-layer MLP processes these embeddings to predict global budget ratios. Since this predictor is shared across all heads, its parameter count is constant.

Parameter Analysis. We analyze the parameter overhead for both Llama-3.1-8B-Instruct ($L = 32, H_{kv} = 8$) and Qwen3-8B ($L = 36, H_{kv} = 8$). The head dimension d_{head} is 128 for both models. Table 5 shows the detailed parameter count for each component.

For Llama-3.1-8B, the LKV-H module comprises $32 \times 8 = 256$ head embeddings plus a single shared MLP, totaling approximately 0.04M parameters. The LKV-T module consists of 256 independent MLPs (one per KV head), each with roughly 33k parameters, summing to 8.45M. The total added overhead is approximately 8.5M parameters, representing 0.11% of the backbone.

For Qwen3-8B, due to its deeper architecture (36 layers), the overhead is slightly higher but remains negligible. The LKV-H embeddings scale to 288 instances, and the LKV-T module also scales to 288 independent instances. The total added parameters are approximately 9.6M, which corresponds to 0.12% of the backbone model.

Table 5: Architecture and parameter breakdown of LKV modules. "Config / Arch." details the vector dimension or MLP structure (Input $\rightarrow$ Hidden $\rightarrow$ Output). All MLPs use SiLU activation. "Unit Params" denotes the parameter count of a single instance.

Module	Config / Arch.	Unit Params	Llama-3.1-8B		Qwen3-8B	
			Count	Total Params	Count	Total Params
LKV-H Embeds	Vector ($d=128$)	128	256	32,768	288	36,864
LKV-H Predictor	$128 \rightarrow 64 \rightarrow 1$	8,320	1	8,320	1	8,320
LKV-T Predictor	$256 \rightarrow 128 \rightarrow 1$	33,024	256	8,454,144	288	9,510,912
Total Added	-	-	-	≈ 8.50 M	-	≈ 9.56 M

C.2 Training Data Construction

To effectively train the LKV framework for long-context capabilities, we constructed a composite dataset named **Loom**. The dataset is curated to cover a diverse range of tasks, including multi-hop reasoning, summarization, code completion, and knowledge-based QA.

Data Composition. The dataset comprises approximately **23,000 samples** mixed from seven public sources. The composition ratios and task types are detailed in Table 7.

- **Long-Context QA & Reasoning (45%):** We integrate *HotpotQA* (30%) for multi-hop reasoning and *NarrativeQA* (5%) for long-story

comprehension. Additionally, we include *SQuAD* (10%) formatted as a few-shot task (8-shot) to encourage in-context learning capabilities.

- **Summarization (20%)**: We utilize *Multi-News* to train the model’s ability to aggregate information across long documents.
- **Code Completion (20%)**: We employ *The Stack* (Python, Java, C++, C#) to enhance long-range dependency modeling in code, constructing samples by concatenating multiple code blocks.
- **Classification & Knowledge (15%)**: We include *TREC* (10%) for fine-grained classification and *MMLU* (5%) for general knowledge, both formatted as 5-shot tasks to simulate long-context retrieval scenarios.

Preprocessing and Filtering. All data is formatted into a standardized Chat template. We strictly applied length filtering using the Qwen3-8B tokenizer, retaining only samples with a token length between **2,000 and 16,000 tokens**. This ensures the model learns effective eviction policies under sufficient KV cache pressure.

Decontamination Strategy. We rigorously address the potential risk of data leakage through a multi-level strategy. First, we strictly excluded **VC-SUM** from our training pipeline, as it is the only dataset in LongBench that utilizes training data for evaluation due to data scarcity. For the remaining datasets, we ensure strict separation at both the task and sample levels:

- **Limited Task Overlap**: The intersection of tasks is minimal. Our training data covers only a small subset of the tasks found in LongBench. The vast majority of LongBench tasks (e.g., Synthetic tasks) are unseen during training, ensuring that our evaluation reflects true generalization capabilities.
- **Sample-Level Disjointness**: For the few tasks that overlap between our training data and LongBench (e.g., HotpotQA, MultiNews), the specific samples are mutually exclusive. As explicitly stated in the LongBench paper regarding their data extraction process:

“Since LLMs may have already been trained on the training set

of some of our collected public datasets, to avoid test leakage, we extract data from the **test sets** of these public datasets, with the exception of VCSUM due to its insufficient data in its test set.” (Bai et al., 2024)

Since LKV is trained exclusively on the **training splits** of these public sources, there is zero overlap in sample instances.

To further verify that our results are not driven by task-family overlap, we additionally trained two decontaminated variants of Loom, analyzed in Section 4.4: **Loom2** replaces the four overlapping task families (HotpotQA, MultiNews, NarrativeQA, TREC; 65% of Loom) with LongAlign-10k, while **Loom3** removes them entirely and re-allocates the freed budget to the remaining non-overlapping sources (TheStack, SQuAD, MMLU). Table 6 summarizes their composition.

Table 6: Composition of the decontaminated Loom variants (% of the training set).

Component	Loom	Loom2	Loom3
Overlapping tasks (HotpotQA, MultiNews, NarrativeQA, TREC)	65%	0%	0%
LongAlign-10k (substitute)	0%	40%	0%
Non-overlapping remainder (TheStack, SQuAD, MMLU)	35%	60%	100%

Table 7: Composition of the Loom training dataset. All subsets are sourced from their official training splits and filtered to the 2k-16k token range.

Dataset Source	Task Type	Ratio	Configuration
HotpotQA	Multi-hop QA	30%	Fullwiki
MultiNews	Summarization	20%	-
The Stack	Code Completion	20%	3-concat (Py, Java, C#, C++)
SQuAD	Reading Comprehension	10%	8-shot Few-shot
TREC	Classification	10%	5-shot / Fine-grained
MMLU	Knowledge QA	5%	5-shot / All subsets
NarrativeQA	Long-doc QA	5%	-

C.3 Hyperparameters and Training Schedule

We train LKV using the AdamW optimizer with BFloat16 precision to ensure training stability and efficiency. We apply gradient clipping with a maximum norm of 1.0. The training process spans 1,000 steps with a global batch size of 64 (1 sample per GPU $\times$ 8 GPUs $\times$ 8 gradient accumulation steps).

Learning Rate Schedule. Since the LKV modules (MLPs) are lightweight and initialized from scratch, we utilize a relatively large learning rate of 1.0×10^{-3} . We apply a polynomial learning rate decay (power=1.0, equivalent to linear decay)

down to a final value of 1.0×10^{-4} , following a 20-step (2%) warmup phase.

Dynamic Annealing Strategy. A critical component of our training is the dynamic annealing of the Soft-TopK temperature τ and the target retention ratio R . We implement an exponential decay schedule for both parameters to facilitate a curriculum learning effect—starting with a "soft" selection and high budget, and gradually converging to a "hard" selection with the target sparsity. For the current step t and total steps T , the values are updated as:

$$\tau_t = \tau_{init} \left(\frac{\tau_{final}}{\tau_{init}} \right)^{t/T}$$

$$R_t = R_{init} \left(\frac{R_{final}}{R_{init}} \right)^{t/T}$$

Specifically, τ decays from 1.0 to 0.001, and the training budget ratio R decays from 0.5 (50% retention) to the target 0.15 (15% retention). This prevents the model from collapsing into trivial solutions early in training.

Loss Configuration. We set the weight for the hidden state alignment loss to $\beta = 0.5$. The distillation temperature is set to 1.0.

Table 8 summarizes the detailed hyperparameters used in our experiments.

D Evaluation Setup Details

D.1 Benchmark Filtering and Configuration

To ensure a fair and unified comparison with prior state-of-the-art methods, our evaluation on LongBench strictly follows the settings adopted by SnapKV (Li et al., 2024) and Ada-KV (Feng et al., 2025a).

Task Selection. LongBench consists of 21 subtasks covering multi-language scenarios. Following standard practice in KV cache compression literature, we select the 16 English and Code tasks for evaluation. The 5 excluded tasks are exclusively Chinese-language datasets: *MultiFieldQA-zh*, *DuReader*, *VCSUM*, *LSHT*, and *PassageRetrieval-zh*. We exclude these tasks for two reasons:

1. **Baseline Alignment:** Representative baselines (e.g., SnapKV, PyramidKV, AdaKV) report results on this specific subset of 16 tasks. Adhering to this protocol guarantees an apples-to-apples comparison.

Table 8: Hyperparameters for training LKV on Llama-3.1-8B and Qwen3-8B.

Hyperparameter	Value
<i>Optimization</i>	
Optimizer	AdamW
Precision	BFloat16
Peak Learning Rate	1.0×10^{-3}
End Learning Rate	1.0×10^{-4}
LR Scheduler	Polynomial (Power=1.0)
Warmup Steps	20 (2%)
Global Batch Size	64
Gradient Accumulation	8 steps
Max Gradient Norm	1.0
Max Steps	1,000
<i>Annealing Schedules</i>	
Temperature Mode	Exponential Decay
Temperature Range (τ)	$1.0 \rightarrow 0.001$
Budget Ratio Mode	Exponential Decay
Budget Ratio Range (R)	$0.5 \rightarrow 0.15$
<i>Loss Coefficients</i>	
Hidden Loss Weight (β)	0.5
Distillation Temp	1.0

2. **Modality Focus:** The base models (Llama-3.1-8B-Instruct and Qwen3-8B) are trained primarily on English. Evaluating on English tasks ensures a fair comparison of model capabilities in their primary modality.

Included Tasks. Table 9 lists the details of the 16 evaluated tasks, categorizing them into Single-Document QA, Multi-Document QA, Summarization, Few-Shot Learning, Synthetic, and Code tasks. The metric for code generation is Edit Similarity (Edit Sim), while others use F1 Score, ROUGE-L, or Accuracy.

D.2 Baseline Implementation Details

We implement all baselines using KVPress (Devoto et al., 2025) for unified evaluation.

DuoAttention Implementation. DuoAttention (Xiao et al., 2024) prunes “Streaming Heads” while retaining “Retrieval Heads”:

- **Llama-3.1-8B:** We utilize the official pre-computed attention patterns provided by the authors to identify head types.
- **Qwen3-8B (On-the-Fly Approximation):** Since Qwen3-8B was released after DuoAttention and lacks official compatible patterns,

Table 9: Details of the 16 LongBench tasks used in our evaluation. This selection aligns with the evaluation protocols of SnapKV and AdaKV.

Category	Task	Metric	Language
Single-Doc QA	NarrativeQA	F1	EN
	Qasper	F1	EN
	MultiFieldQA-en	F1	EN
Multi-Doc QA	HotpotQA	F1	EN
	2WikiMQA	F1	EN
	MuSiQue	F1	EN
Summarization	GovReport	ROUGE-L	EN
	QMSum	ROUGE-L	EN
	MultiNews	ROUGE-L	EN
Few-Shot Learning	TREC	Accuracy	EN
	TriviaQA	F1	EN
	SAMSum	ROUGE-L	EN
Synthetic	PassageCount	Accuracy	EN
	PassageRetrieval-en	Accuracy	EN
Code	LCC	Edit Sim	Python/C#/Java
	RepoBench-P	Edit Sim	Python/Java

we adopt an *on-the-fly* estimation method proposed by Devoto et al. (2025). Instead of expensive offline calibration, we synthesize attention patterns by probing the model with random context samples. By measuring the attention distribution on these synthetic inputs, we dynamically classify heads as retrieval or streaming, effectively extending DuoAttention’s structural prior to Qwen3 without compatibility issues.

Other Baselines. We adhere to standard configurations: SnapKV uses the recommended observation window size; PyramidKV follows the standard layer-wise decay profile; and Ada-SnapKV employs attention concentration scores for adaptive budgeting.

E Additional Experimental Results

E.1 Prefill Latency

Table 10 reports prefill latency at 200k context length. LKV’s dynamic token scoring ($O(T)$) incurs a 7.7% overhead, comparable to Ada-SnapKV’s $O(W \cdot T)$ filtering (+7.2%).

Table 10: Eviction complexity and prefill latency at 200k context length. W : observation window, T : context length.

Method	Complexity	Time (s)	Overhead
Full Cache	–	82.40	–
DuoAttention	$O(1)$	87.21	+5.8%
Ada-SnapKV	$O(W \cdot T)$	88.37	+7.2%
LKV (Ours)	$O(T)$	88.76	+7.7%

E.2 Comparison with Additional Baselines

We additionally benchmark D2O (Wan et al., 2025) and CAKE (Qin et al., 2025), two recent adaptive eviction methods, on LongBench (Llama-3.1-8B-Instruct, $R = 0.15$). Table 11 reports the category-wise results. LKV outperforms D2O and CAKE by 4.58 and 2.91 points on average, winning 13/16 and 12/16 subtasks respectively.

Table 11: Comparison with additional adaptive baselines on LongBench (Llama-3.1-8B-Instruct, $R = 0.15$).

Method	Avg.	S-QA	M-QA	Summ	FewS	Synth	Code
Full Cache	47.48	43.57	45.41	28.89	65.43	53.48	51.42
D2O	42.15	28.93	36.74	24.06	66.14	46.28	57.14
CAKE	43.82	32.63	40.10	23.69	64.24	52.84	56.76
LKV (Ours)	46.73	41.66	42.79	28.76	66.30	53.38	51.25

E.3 Detailed LongBench Breakdown

In this section, we provide the fine-grained performance breakdown for all 16 evaluated LongBench subtasks. Tables 12 and 13 present the results for Llama-3.1-8B-Instruct and Qwen3-8B, respectively, under a strict 15% KV budget ($R = 0.15$).

Analysis of Structural Priors. The breakdown reveals why heuristic methods with rigid structural priors suffer in specific domains. For instance, on the *PassageRetrieval-en* (PRe) task using Llama-3.1-8B-Instruct (Table 12), DuoAttention and PyramidKV drop to scores of 29.00 and 44.65, respectively, compared to the Full Cache score of 99.50. This confirms that static priors (e.g., heavy penalization of distant tokens or specific heads) aggressively evict needle-in-a-haystack information. In contrast, LKV achieves a score of 97.00, demonstrating its ability to dynamically identify and retain critical retrieval tokens regardless of their position.

E.4 Qwen3-8B performance on RULER benchmark

To verify the generalization of LKV across different architectures, we further evaluate Qwen3-8B on the RULER benchmark. Figure 6 visualizes the results under two critical settings: (a) **Length Scalability**: Performance across increasing context lengths (4k, 8k, 16k, 32k) with a fixed retention ratio ($R = 0.15$). (b) **Compression Robustness**: Performance across varying retention ratios ($R \in [0.1, 0.5]$) at a fixed context length of 16k.

Consistent with the findings on Llama-3.1-8B-Instruct, LKV demonstrates superior robustness compared to baselines. Notably, at 32k context

Table 12: Llama-3.1-8B-Instruct: Detailed performance breakdown on 16 LongBench tasks ($R = 0.15$).

Method	Avg.	NrivQA	Qasper	MF-en	HoppsQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P
Full Cache	47.48	32.38	45.22	53.12	56.84	48.04	31.35	34.63	25.25	26.78	68.50	90.79	36.99	7.45	99.50	50.88	51.96
SnapKV	38.05	26.73	24.81	30.28	48.35	29.48	24.40	26.02	20.82	21.94	50.00	91.81	32.53	6.46	75.00	49.95	50.16
PyramidKV	32.12	22.12	23.97	29.43	39.05	27.38	13.82	25.44	20.41	21.86	38.50	87.96	21.40	5.33	44.65	50.11	42.47
DuoAttention	31.03	19.63	16.41	29.23	38.09	22.68	17.04	24.10	17.94	22.93	39.50	89.12	23.51	6.50	29.00	49.34	51.46
Ada-SnapKV	39.06	25.98	24.07	32.10	49.85	32.58	23.94	25.93	21.89	22.05	52.00	92.16	33.68	7.54	80.00	50.18	50.94
LKV (Ours)	46.73	31.87	43.16	49.94	57.16	42.18	29.04	34.44	25.04	26.79	69.50	91.80	37.60	9.75	97.00	49.99	52.50

Table 13: Qwen3-8B: Detailed performance breakdown on 16 LongBench tasks ($R = 0.15$). Note that DuoAttention utilizes on-the-fly approximation.

Method	Avg.	NrivQA	Qasper	MF-en	HoppsQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P
Full Cache	46.37	28.99	40.06	51.98	56.93	42.99	32.29	33.57	23.98	24.72	71.50	88.69	39.74	2.75	100.00	50.71	53.03
SnapKV	36.72	21.59	22.43	27.02	39.62	29.90	21.17	27.42	19.76	19.23	49.50	89.44	38.37	3.26	76.00	48.52	54.26
PyramidKV	34.01	18.42	22.38	25.25	31.97	28.17	17.31	26.33	19.29	19.34	43.50	87.51	37.16	4.57	60.50	48.78	53.68
DuoAttention	26.85	11.21	13.07	24.06	28.48	25.27	9.26	22.37	17.05	20.63	30.50	84.74	35.65	0.42	7.50	45.81	53.59
Ada-SnapKV	37.90	21.40	23.12	27.68	42.16	32.09	20.83	27.02	20.13	19.73	54.00	89.11	39.30	3.97	83.50	48.88	53.51
LKV (Ours)	45.41	24.06	38.23	50.29	57.60	43.59	31.37	32.60	24.10	24.26	69.50	88.77	38.54	3.81	100.00	45.04	54.88

Table 14: Llama-3.1-8B-Instruct: Average LongBench scores across varying retention ratios (R).

Method	R=0.5	R=0.4	R=0.3	R=0.2	R=0.15	R=0.1
SnapKV	45.49	44.33	43.16	40.22	38.05	34.62
PyramidKV	38.01	37.55	35.59	33.41	32.12	31.97
DuoAttention	46.62	45.28	45.21	37.53	31.03	29.72
Ada-SnapKV	46.04	45.02	44.01	41.32	39.06	36.28
LKV (Ours)	46.99	47.26	47.16	47.11	46.73	43.93

Table 15: Qwen3-8B: Average LongBench scores across varying retention ratios (R).

Method	R=0.5	R=0.4	R=0.3	R=0.2	R=0.15	R=0.1
SnapKV	44.41	43.20	41.41	39.16	36.72	33.31
PyramidKV	40.20	39.82	38.23	36.19	34.01	32.29
DuoAttention	32.66	30.67	28.89	27.40	26.85	27.13
Ada-SnapKV	44.53	43.53	42.44	40.21	37.90	34.10
LKV (Ours)	46.25	45.86	46.01	45.93	45.41	43.17

length, heuristic methods like SnapKV experience significant degradation, whereas LKV maintains high effective context retention.

Table 16: LongBench results on Mistral-7B-Instruct-v0.3 ($R = 0.15$).

Method	Avg.	SD-QA	MD-QA	Summ	FewS	Synth	Code
Full Cache	44.19	39.01	37.41	28.95	65.20	51.75	45.86
SnapKV	34.81	21.69	26.82	22.92	56.05	39.53	47.72
PyramidKV	33.62	20.86	25.46	22.49	55.13	34.35	48.69
DuoAttention	26.68	16.00	21.29	20.43	50.80	5.83	44.84
Ada-SnapKV	36.47	23.83	29.23	23.25	57.80	43.27	47.30
LKV (Ours)	42.75	37.04	36.37	28.98	61.65	50.67	45.25

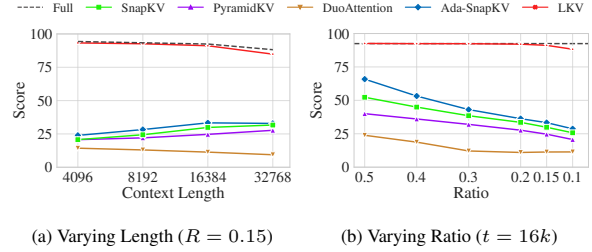


Figure 6: Qwen3-8B performance on RULER. (a) Effective context length evaluation up to 32k tokens. (b) Performance degradation analysis under aggressive compression ratios. LKV maintains superior stability across both dimensions.

Table 17: Query-agnostic vs. query-aware evaluation on LongBench (Llama-3.1-8B-Instruct, $R = 0.15$).

Method	Agnostic	Aware	Δ
Full Cache	47.48	47.48	—
SnapKV	38.05	45.89	+7.84
PyramidKV	32.12	45.68	+13.56
DuoAttention	31.03	42.87	+11.84
Ada-SnapKV	39.06	45.98	+6.92
LKV (Ours)	46.73	47.45	+0.72