MORE THAN A QUICK GLANCE: OVERCOMING THE GREEDY BIAS IN KV-CACHE COMPRESSION

Aryan Sood*, **Tanvi Sharma*** **& Vansh Agrawal**
Indian Institute of Technology, Roorkee
Roorkee, Uttarakhand, 247667, India
aryan_s2@ee.iitr.ac.in, tanvi_s@ph.iitr.ac.in & vansh_a@ph.iitr.ac.in

ABSTRACT

While Large Language Models (LLMs) can theoretically support extensive context windows, their actual deployment is constrained by the linear growth of Key-Value (KV) cache memory. Prevailing compression strategies mitigate this through various pruning mechanisms, yet trade-off semantic recall for memory efficiency. In this work, we present LASER-KV (Layer Accumulated Selection with Exact-LSH Recall), a framework designed to test the limits of KV compression under a strict accumulative budgeting policy. We deviate from the standard fixed summary size approach by implementing a block-wise accumulation strategy governed by a protection divisor (n). This allows us to isolate the effects of compression from sliding window artifacts. Our experiments on the Babilong benchmark reveal performance degradation in previous compression methods by 15–30% on various long context tasks. LASER-KV maintains stable performance, achieving superior accuracies by a margin of upto 10% at 128k. These findings challenge the prevailing assumption that attention scores alone are a sufficient proxy for token utility.

1 INTRODUCTION

Long context applications of Large Language Models (LLMs) (Radford et al., 2018) are currently limited by the physical constraints of deployment. The quadratic complexity of attention and linear KV cache growth strain the GPU VRAM. This often exceeds memory limits, even on high-end hardware. This necessitates compression strategies that maintain fixed memory budgets, i.e., fixed token budgets. However, current approaches that utilize sliding windows (Xiao et al., 2024b) or selective eviction operate under a trade-off. By aggressively pruning tokens to fit consumer-grade hardware, they tend to discard critical context, which causes a deterioration in accuracy. This prevents reliable generation in complex, multi-step reasoning tasks.

This analysis highlights the limitations of the current compression paradigm. We demonstrate that relying solely on attention scores makes long-context reasoning difficult, establishing the practical boundaries of reliable long-term LLM memory. To address this, we introduce LASER-KV (Layer Accumulated Selection with Exact-LSH Recall). Our contributions are:

- Unlike recursive methods that degrade historical context, we propose an accumulative, append-only memory mechanism governed by a protection divisor (n) that strictly isolates compression from sliding window artifacts.
- We challenge the assumption that attention scores alone are sufficient for token selection by introducing an Exact-LSH policy, combining attention scores with Locality Sensitive Hashing (LSH) (Chen et al., 2024) to recover structurally critical tokens. Our experiments on the Babilong (Kuratov et al., 2024) benchmark validate this approach: while standard policies degrade significantly at 64k+ tokens, LASER-KV maintains stability up to 128k.

*Equal contribution.

2 BACKGROUND

To address memory constraints, recent works have proposed selective retention strategies that prune the cache based on attention sparsity. SnapKV (Li et al., 2024) and H_2O (Zhang et al., 2023) identify that attention heads consistently focus on specific "heavy hitter" tokens, while methods like Quest (Tang et al., 2024) dynamically estimate token criticality based on the current query. Similarly, PyramidKV (Cai et al., 2025) allocates larger cache budgets to lower layers where attention is widely scattered, and smaller budgets to higher layers where information is concentrated.

Moving beyond static budgets, FINCH (Corallo & Papotti, 2024) introduces recursive compression to manage long contexts. It processes text in chunks, using Top-K selection to forward only high-scoring KV pairs. However, relying solely on immediate attention scores is greedy. While it captures tokens relevant to the current query, it often discards data critical for future context. Other methods, such as InfLLM (Xiao et al., 2024a), offload context to external memory, but this increases latency because of the newly introduced retrieval overhead which was not present in legacy methods.

3 METHODOLOGY

We construct *LASER-KV* (Layer Accumulated Selection with Exact-LSH Recall) to systematically improve the limitations of long-context recall under strict memory constraints. During the prefilling phase, our framework uses an *accumulative* budget to retain relevant tokens per block. Only the selected tokens are kept in the KV cache and are further used for the decoding phase. Previous techniques maintain a fixed-size summary after each iteration. This design highlights how static block-level retention impacts performance compared to maintaining a full KV cache. It specifically isolates the impact of token selection strategies.

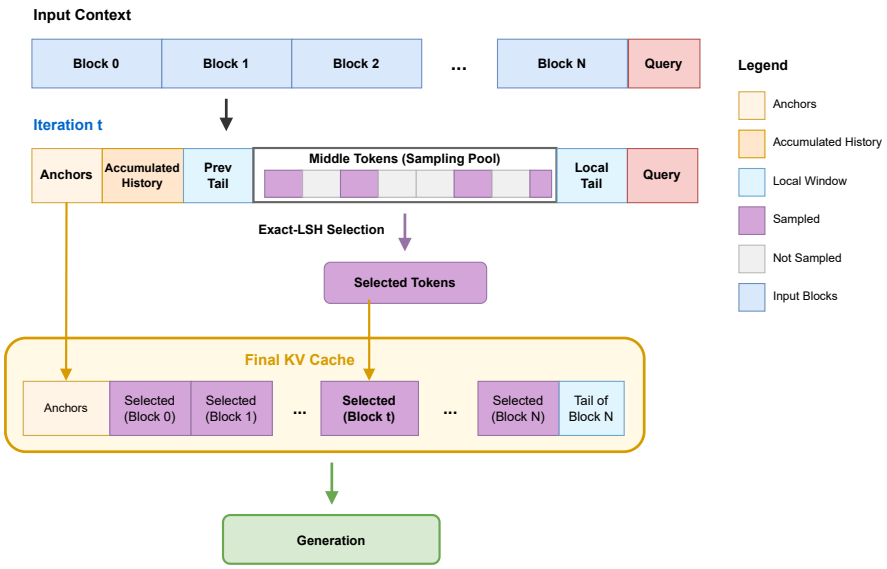

Figure 1: The LASER-KV Selection Pipeline. The input context is divided into blocks. Our Exact-LSH mechanism then selects tokens using two criteria: Exact Attention for heavy hitters and LSH for high-recall safety, to form the compressed cache.

3.1 BUDGET ALLOCATION VIA THE PROTECTION DIVISOR

A core conflict in KV compression is maintaining the balance between recent tokens and historical tokens that carry long-range semantic relevance. We manage this trade-off through a single hyper-parameter, the protection divisor n. It serves as an explicit control variable that governs generation

stability versus long-term recall capacity. Smaller values of n increase the proportion of tokens reserved for syntactic stability (i.e., anchors and local window), while larger values allocate more budget to long-term memory. In our experiments, we set $n = 4$, which induces a quarter–quarter–half allocation between global anchors, local window tokens, and recall memory. This partitioning is consistent with empirically validated attention-guided eviction strategies such as SAGE-KV (Wang et al., 2025), while remaining compatible with our accumulative memory formulation. Given a per-block budget B, n determines the cache partition boundaries.

- **The Syntactic Set ($2B/n$):** We reserve B/n tokens for Global Anchors. These preserve initial tokens to maintain attention sinks (Xiao et al., 2024b). We also reserve B/n for a Local Sliding Window. This helps maintain grammatical coherence.
- **Recall Budget** $(B - 2B/n)$, denoted as B_{long}**:** The remaining capacity is allocated to the Long-Term Memory Pool. We use our Exact-LSH selection policy for this allocation.

3.2 Block Processing and Boundary Smoothing

We process the context in sequential blocks (Acharya et al., 2025; Corallo & Papotti, 2024). Boundary artifacts can occur and tokens at the edge of a block may be dropped because they lack subsequent context during scoring. We implement a look-back strategy to prevent this (Figure 1). The scoring window for the block $\mathcal{B}_t$ is expanded to include the tail of $\mathcal{B}_{t-1}$ (Jiang et al., 2024). This ensures that bridging tokens are evaluated with sufficient local context before pruning occurs.

3.3 Exact-LSH Selection Algorithm

We hypothesize that reliance on a single metric creates brittle memory. Metrics like attention score exhibit sparse spikes. These spikes often miss supporting tokens that are structurally relevant but not currently attended to. We employ our Exact-LSH selection policy (Algorithm 1) to mitigate this.

Precision (Global Consensus): We sum attention scores across all layers and heads (Wang et al., 2025). This helps preserve tokens that are critical to specific heads (such as Induction heads (Olsson et al., 2022)). Retention occurs even if other heads ignore these tokens. This has a time complexity of $\mathcal{O}(L_q \cdot |\mathcal{C}| \cdot d_h)$, where L_q is the query length and d_h is the head dimension.

Recall (MagicPIG): We utilize Locality Sensitive Hashing (LSH) for the remaining budget. We adopt the probabilistic scoring from MagicPIG (Chen et al., 2024). This approach ranks tokens by their theoretical hash collision probability with the query. It captures structural similarity that exact attention often misses. This acts as a high-recall safety net. This introduces a scalable cost of $\mathcal{O}(|\mathcal{C}| \cdot R \cdot d_h)$ for R hash rounds, with a minimal space overhead of $\mathcal{O}(|\mathcal{C}| \cdot R)$.

This precision–recall decomposition is particularly important under our accumulative policy, where selection decisions are not revisited and early pruning errors cannot be corrected in later blocks.

Algorithm 1 Exact-LSH Selection Policy

Input: Query Q, Candidate Set $\mathcal{C}$, Budget B_{long}, Ratio $\alpha \in [0, 1]$, Hash Functions $\{h_r\}_{r=1}^{R}$
Output: Selected Token Indices $\mathcal{K}$
$S_{\text{exact}}(k) \leftarrow \sum_{l=1}^{L} \sum_{h=1}^{H} \text{Attn}_{l,h}(Q, k) \quad \forall k \in \mathcal{C}$
$\mathcal{K}_{\text{exact}} \leftarrow \text{TopK}(\mathcal{C}, \text{score} = S_{\text{exact}}, k = \alpha B_{\text{long}})$
$\mathcal{C}_{\text{residual}} \leftarrow \mathcal{C} \setminus \mathcal{K}_{\text{exact}}$
$S_{\text{lsh}}(k) \leftarrow \frac{1}{R} \sum_{r=1}^{R} \mathbf{1}[h_r(Q) = h_r(k)] \quad \forall k \in \mathcal{C}_{\text{residual}}$
$\mathcal{K}_{\text{lsh}} \leftarrow \text{TopK}(\mathcal{C}_{\text{residual}}, \text{score} = S_{\text{lsh}}, k = (1 - \alpha) B_{\text{long}})$
return $\mathcal{K}_{\text{exact}} \cup \mathcal{K}_{\text{lsh}}$

4 Experiments

We evaluated the performance of LASER-KV against several state-of-the-art baselines on the Babilong (Kuratov et al., 2024) long-context benchmark. We analyze the model's ability to retrieve single facts, chain multiple supporting facts, and handle complex argument relations across varying context lengths (16k, 64k, and 128k).

4.1 Experimental Setup and Tasks

We use Llama-3.1-8b-Instruct (Meta AI, 2024) for all the evaluations for 16k and 64k context lengths. However, the model has a 128k context limit, and to avoid truncating any samples, we used Llama-3-8b-Instruct-Gradient-1048k (Gradient AI, 2024) to evaluate the method at 128k, which relies on frequency scaling techniques similar to YaRN (Peng et al., 2023) to maintain coherence at extreme lengths. For further details, refer to Appendix A and B

4.2 Results and Analysis

We evaluate different Exact-LSH settings (Table 1) at a 16k context length to establish a comparison among them. LASER-KV was also compared to the baseline performance of Llama-3.1-8b-Instruct and the original MagicPIG configuration in our framework. Based on these results, a hybrid of Exact(0.75) and MagicPIG(0.25) was selected for comparison on longer context lengths against state-of-the-art KV compression methods.

Method	**QA1**	**QA2**	**QA3**	**QA5**	**QA6**
Full Attention	58%	19%	33%	90%	68%
FINCH	51%	**24%**	25%	85%	63%
PyramidKV	40%	17%	14%	78%	80%
SnapKV	34%	17%	13%	75%	78%
Exact	49%	20%	31%	89%	57%
MagicPIG	12%	2%	11%	35%	28%
Exact+MP(0.25)	53%	15%	31%	88%	58%
Exact+MP(0.5)	53%	15%	**32%**	89%	56%
Exact+MP(0.75)	***54%***	*18%*	*31%*	***91%***	***59%***

Table 1: Accuracies of base model (Llama-3.1-8b-Instruct) and previous methods versus various combinations of LASER-KV at 16k context length (MP stands for MagicPIG), the highlighted numbers are the best for that particular task according to our experiments.

	Context Length: 64k					**Context Length: 128k**				
Method	**QA1**	**QA2**	**QA3**	**QA5**	**QA6**	**QA1**	**QA2**	**QA3**	**QA5**	**QA6**
Full Attention	24%	6%	21%	72%	51%	31%	6%	15%	80%	55%
FINCH	22%	8%	13%	44%	45%	0%	0%	0%	0%	0%
PyramidKV	16%	**13%**	25%	72%	**70%**	25%	**10%**	19%	84%	59%
SnapKV	7%	1%	15%	51%	36%	22%	5%	19%	84%	57%
LASER-KV	***25%***	*9%*	***31%***	***87%***	*48%*	***38%***	*7%*	***25%***	***84%***	***66%***

Table 2: Performance on context lengths of 64k (Llama-3.1-8b-Instruct) and 128k (Llama-3-8b-Instruct-Gradient-1048k), the highlighted numbers are the best for that particular task according to our experiments. For LASER-KV, we keep Exact+MP(0.75) based on results from Table 1

Stability at Extreme Lengths. The most critical finding emerges at the 128k context length (Table 2), where methods such as PyramidKV maintain performance, but FINCH collapses entirely to 0% across all evaluated tasks. Although several state-of-the-art methods remain competitive with LASER-KV at shorter context lengths, LASER-KV demonstrates that it significantly outperforms them in long-context settings. This stability can be attributed to the structural design i.e, the accumulative, append-only policy which prevents early pruning errors from compounding across blocks, avoiding the irreversible information decay characteristic of recursive Top-K approaches. The protection divisor ensures a non-vanishing local window that preserves short-range coherence under aggressive compression and the hybrid Exact-LSH selection mitigates the greedy bias of attention-only pruning by explicitly reserving recall capacity for structurally similar tokens which is an important property for multi-hop reasoning tasks in Babilong, where supporting facts may not be attended.

5 Conclusion

Our results with LASER-KV demonstrate that attention scores alone are an insufficient proxy for token utility in long contexts. By implementing a protection divisor (n) to stabilize the local window and a novel Exact-LSH policy that integrates Locality Sensitive Hashing (LSH), we successfully recover structurally critical tokens often discarded by greedy pruning.

On the Babilong benchmark, LASER-KV maintains performance stability even at 128k tokens. It notably outperforms baselines like SnapKV and FINCH, which suffer significant degradation beyond 64k. These findings suggest that moving toward hybrid, structure aware selection is essential for maintaining reliable long-term memory in Large Language Models.

6 Future Work

While our hybrid LASER-KV approach helps stabilize performance, we acknowledge that our current evaluation is bounded by computational constraints, preventing exhaustive validation across a wider array of benchmarks and variable context lengths. Consequently, the sharp decline observed in baseline methods highlights that building sustainable, long-term memory remains a difficult and open challenge for the community.

References

Shantanu Acharya, Fei Jia, and Boris Ginsburg. Star attention: Efficient llm inference over long sequences, 2025. URL https://arxiv.org/abs/2411.17116.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2025. URL https://arxiv.org/abs/2406.02069.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. Magicpig: Lsh sampling for efficient llm generation, 2024. URL https://arxiv.org/abs/2410.16179.

Giulio Corallo and Paolo Papotti. Finch: Prompt-guided key-value cache compression, 2024. URL https://arxiv.org/abs/2408.00167.

Gradient AI. Llama-3-8b-instruct-gradient-1048k. https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k, 2024. Accessed: 2026.

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. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024. URL https://arxiv.org/abs/2407.02490.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack, 2024. URL https://arxiv.org/abs/2406.10149.

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, 2024. URL https://arxiv.org/abs/2404.14469.

Meta AI. Llama 3.1: Advanced open-source language model. https://ai.meta.com/blog/meta-llama-3-1/, 2024. Accessed: 2026.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022. URL https://arxiv.org/abs/2209.11895.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models, 2023. URL https://arxiv.org/abs/2309.00071.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024. URL https://arxiv.org/abs/2406.10774.

Guangtao Wang, Shubhangi Upasani, Chen Wu, Darshan Gandhi, Jonathan Li, Changran Hu, Bo Li, and Urmish Thakker. Llms know what to drop: Self-attention guided kv cache eviction for efficient long-context inference, 2025. URL https://arxiv.org/abs/2503.08879.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Infllm: Training-free long-context extrapolation for llms with an efficient context memory, 2024a. URL https://arxiv.org/abs/2402.04617.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024b. URL https://arxiv.org/abs/2309.17453.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

A BABILONG BENCHMARK TASK DESCRIPTIONS

To rigorously evaluate long-context retrieval and reasoning capabilities, we utilize the Babilong benchmark. This suite consists of various synthetic Question Answering (QA) tasks designed to test specific aspects of memory retention, logical chaining, and deduction over extended sequences. Table 3 provides a detailed summary of the specific tasks employed in our experiments, including the reasoning type required and examples of the query structure. Each task consists of 100 samples.

Table 3: Detailed descriptions of Babilong tasks. Each task targets a specific reasoning capability, ranging from single-fact retrieval to complex multi-hop deduction and state tracking.

Task ID	Task Name	Description & Reasoning Requirement
QA1	Single Supporting Fact	*Fact Retrieval.* The model must locate a single specific fact hidden within the context to answer the query. *Example:* "Where is Mary?" $\rightarrow$ "In the kitchen."
QA2	Two Supporting Facts	*Multi-hop Reasoning.* Requires chaining two separate pieces of information. The model must find an intermediate entity to locate the target. *Example:* "Where is the football?" (implied: Who had it last? $\rightarrow$ Where are they now?)
QA3	Three Supporting Facts	*Complex Chaining.* A higher-order multi-hop task requiring the retrieval and synthesis of three distinct facts to derive the correct conclusion.
QA5	Three Argument Relations	*Complex Relations.* Requires reasoning over relational dependencies among three entities. The model must track and integrate multiple interrelated facts to resolve relative positions or interactions.
QA6	Yes/No Questions	*Verification.* The model must affirm or negate a statement based on the strict presence or absence of supporting facts in the context history.

B CONFIGURATIONS

For all evaluations on the Babilong benchmark, we used a compression ratio of r = 0.25. A block size of $S_{\text{block}} = 4096$ tokens was used for evaluations on context lengths of 16k, with the notable exception of FINCH, for which the block size was set to 1024 tokens following the original implementation. Subsequently, the block size was maintained at 4k for context lengths of 64k and 128k across all compression methods. The protection divisor n was set to 4 for all experiments.

To ensure fair comparison across variable block sizes and context lengths, the effective memory budget B_{global} for the accumulative policy was calculated using the harmonic mean of the block size and total context length, scaled by the compression ratio (where T represents the context length):

$$B_{global} = \left\lfloor \frac{2 \cdot r \cdot S_{\text{block}} \cdot T}{S_{\text{block}} + T} \right\rfloor \tag{1}$$

The total memory budget B_{global} is evenly divided between all blocks. Throughout the paper, when we refer to the block budget B, we mean the per-block allocation derived from B_{global}.