SAR-KV: Selective Anchor-Residual KV Caching for Single-GPU Long-Context LLM Inference

Anonymous ACL submission

Abstract

For single-GPU long-context large language model (LLM) inference, end-to-end peak memory often matters more than resident key-value (KV) cache size: allocator overhead, reconstruction buffers, and offloading staging can push compact caches past the device limit. **Selective Anchor-Residual KV (SAR-KV)** is a fully GPU-resident cache with full-precision chunk-mean anchors, budgeted low-bit residuals, and full-precision prompt-tail and recent windows. Queries route over chunk anchors, and only relevant chunks are reconstructed at token resolution. On Llama-3.1-8B-Instruct, anchor centering lowers 4-bit round-to-nearest (RTN) reconstruction mean-squared error (MSE) to $0.169\times$ that of direct raw-KV quantization, an 83.1% reduction. With the 0.25 cache-ratio profile, SAR-KV trails Full KV by 0.63 points on the five-category macro-average score, has the lowest observed end-to-end peak memory at 32K and 64K, and uniquely completes the 128K prefill-plus-1,000-decode-step protocol.

1 Introduction

Long-context inference is becoming central to repository-scale code understanding (Liu et al., 2024b; Khan et al., 2025), document question answering (QA) (Bai et al., 2024, 2025), and retrieval-heavy reasoning (Hsieh et al., 2024). The key bottleneck in practice is not only model capacity but the key-value (KV) cache: every past token retains a key-value state, memory grows linearly with prompt length, and the cache stays on the critical path of each decode step (Vaswani et al., 2017; Kwon et al., 2023; Gao et al., 2025). At 64K–128K input context, whether inference completes at all can depend on the gap between a method's advertised cache size and its actual peak GPU memory.

This gap is missed when compression is judged only by resident cache size. Figure 1 profiles

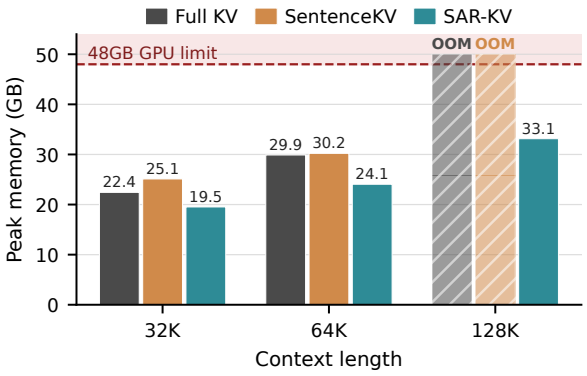

Figure 1: **Memory overflow boundary.** Peak GPU memory vs. context length for representative methods under the 0.25 profile on Llama-3.1-8B-Instruct. The dashed line marks the 48 GB budget.

it under the 0.25 cache-ratio profile on a single 48 GB GPU (Appendix D): Full KV exposes the linear KV limit, while SentenceKV lowers the final cache but still materializes prompt KV tensors and selection-time attention before compression. Thus peak memory is governed by the end-to-end inference path, not the final cache object.

Prior work usually optimizes one part of decoding: offloading and cache-placement move KV states across memory tiers (Kwon et al., 2023; Sheng et al., 2023; Xiao et al., 2024a; Xiong et al., 2024; Zhu et al., 2025); token selection and budgeted eviction choose which token states remain active (Liu et al., 2023; Zhang et al., 2023; Cai et al., 2024; Xiao et al., 2024b; Chen et al., 2024; Li et al., 2024; Tang et al., 2024; Behnam et al., 2025; Wang et al., 2025); quantization and representation compression reduce token states or residuals (Liu et al., 2024c; Hooper et al., 2024; He et al., 2024; Liu et al., 2024a; Li et al., 2025; Liu et al., 2026); and chunk/cluster designs use document structure mainly as selection granularity (Liu et al., 2025a,b). These approaches reduce resident cache size, but not necessarily end-to-end peak

memory (please refer to Appendix C for more detailed related work survey).

Selective Anchor-Residual KV (SAR-KV) instead keeps a GPU-resident representation in which every punctuation-delimited chunk has a full-precision chunk-mean anchor, budgeted low-bit residuals, and full-precision prompt-tail and recent windows (Xiao et al., 2024b). The chunk-mean anchor is the storage unit, routing key, and mean-only attention approximation; routed chunks with resident residuals are reconstructed at token resolution. This positions SAR-KV as a peak-memory method rather than only a nominal-cache-size method (Appendix C). The contributions are:

- We reframe the mismatch between nominal cache size and end-to-end peak memory as an explicit evaluation target for long-context large language model (LLM) serving.
- We propose SAR-KV, a GPU-resident selective anchor-residual cache representation with decode-time routing.
- Under the 0.25 cache-ratio profile on a 48 GB GPU, SAR-KV has the lowest observed 32K/64K end-to-end peak GPU memory and is the only evaluated implementation that completes the 128K prefill-plus-1,000-decode-step protocol (33.14 GB peak, 3.92 GB cache) while maintaining near-Full-KV accuracy in LongBench.

2 SAR-KV

"Selective" denotes two budget-constrained choices: which chunks retain residuals and which chunks are reconstructed during decoding. The design separates roles that are typically conflated in token-level caches: chunk-mean anchors make all chunks addressable, residuals provide fidelity within the allocated budget, and tail/recent windows preserve the instruction boundary and local generation state.

Storage. Let the prefill tensors of one layer be $K, V \in \mathbb{R}^{H \times L \times d}$ with H KV heads. SAR-KV partitions the prompt into ordered chunks $\mathcal{C} = \{C_i = [a_i, b_i)\}_{i=1}^{M}$ using tokenizer-level punctuation as boundary candidates, in line with recent sentence- or chunk-level KV compression designs (Zhu et al., 2025; Liu et al., 2025b). SAR-KV uses a minimum chunk length m and a hard fallback length f to prevent degenerate segments; no external parser is required. Each chunk C_i of length S_i stores a full inference-precision chunk-mean anchor (16-bit floating point (fp16) in the implementation)

$$\bar{k}_i = \frac{1}{S_i} \sum_{j \in C_i} k_j, \qquad \bar{v}_i = \frac{1}{S_i} \sum_{j \in C_i} v_j. \quad (1)$$

The per-token residuals $r^K_{i,j} = k_{i,j} - \bar{k}_i$ and $r^V_{i,j} = v_{i,j} - \bar{v}_i$ are quantized by per-block asymmetric round-to-nearest along the head dimension with block size $g = 128$ at b bits, packed into unsigned bytes, with full-precision scales and zero points stored as metadata. A fixed-size prompt-tail buffer $\mathcal{T}$ holds the last T prompt tokens in full precision; a ring-buffer $\mathcal{W}$ holds the most recent W generated tokens (Xiao et al., 2024b) so that generation cannot evict the final instruction or question.

Why anchor centering helps quantization. SAR-KV relies on the empirical observation that chunk-centered residuals quantize better than raw KV tensors. Table 1 verifies this on Llama-3.1-8B-Instruct using SAR-KV's own chunker and head-dim RTN quantizer. Per-chunk mean anchoring lowers 4-bit round-to-nearest (RTN) mean-squared error (MSE) to $0.169\times$ that of raw-KV quantization, an 83.1% reduction, and shrinks the elementwise standard deviation from 1.249 to 0.713. A per-chunk medoid anchor (the real token closest to the chunk mean) reaches only $0.378\times$ the raw-KV MSE; a single global-mean anchor reaches $0.296\times$. Aggressive low-bit storage therefore remains viable because the quantizer sees a tensor whose dynamic range has already been reduced by anchoring, not the raw activation outliers and heterogeneous token distributions reported by prior KV quantization work (Liu et al., 2024c; Hooper et al., 2024).

Coverage-aware bit choice. Let $c_{\text{fp16}} = 2Hde$ denote the per-token full-precision KV cost. A target cache ratio $\rho = C/L$, where C is the budget expressed in fp16 tokens, gives target memory $B_{\text{target}} = \rho L c_{\text{fp16}}$. Let B_{mean}, B_{tail}, and B_{recent} denote the anchor, prompt-tail, and recent-window costs, with $c_{\text{res}}(b)$ defined in Appendix A. Residual coverage at bit-width b is bounded by

$$R_b \leq \frac{B_{\text{target}} - B_{\text{mean}} - B_{\text{tail}} - B_{\text{recent}}}{c_{\text{res}}(b)}, \quad (2)$$

where R_b counts retained residual tokens and S_b counts residual chunks that keep residuals at bit-

Anchor	σ	MSE_4	MSE_8	Ratio_4
raw KV RTN	1.249	5.88×10^{-2}	2.12×10^{-4}	1.000
medoid	1.044	2.22×10^{-2}	7.99×10^{-5}	0.378
global-mean	0.938	1.74×10^{-2}	6.27×10^{-5}	0.296
chunk-mean	**0.713**	$\mathbf{9.93\times10^{-3}}$	$\mathbf{3.57\times10^{-5}}$	**0.169**

Table 1: **Residual variance reduction.** Results are on Llama-3.1-8B-Instruct, aggregated over all layers and K/V heads on eight long prompts each from LongBench NarrativeQA, Qasper, and HotpotQA. Ratio_4 denotes MSE_4 divided by the raw-KV MSE_4; lower is better.

width b, so raising the bit-width lowers error but drops both. The implemented policy plans the maximum 4-bit coverage (R_4, S_4) first and upgrades to $b \in \{8, 16\}$ only if $R_b \geq 0.85R_4$ and $S_b \geq 0.85S_4$, with the 16-bit option gated to $\rho \geq 0.75$. Under high compression pressure the planner caps anchor memory, then assigns residuals at complete-chunk granularity by uniformly interleaving chunks across up to 16 equally-sized prompt segments; chunks overlapping the pinned tail or exceeding the remaining residual budget stay mean-only. Derivation and implementation-level algorithm appear in Appendices A–B.

Decode-time routing. For grouped-query attention (Ainslie et al., 2023) query heads are first averaged within each KV-head group so the routing score lives in the same head space as the cached keys. The base score is the anchor-only product $s_i = \frac{1}{H}\sum_{h=1}^{H} q_{t,h}^{\top}\bar{k}_{i,h}/\sqrt{d}$, where H is the number of KV heads. The default ResidualMax score refines only chunks with resident residuals: SAR-KV dequantizes their residual keys, computes per-token query-key scores, and replaces s_i with the maximum within C_i; mean-only chunks keep their anchor score. Chunks are split into a high-priority set $\mathcal{H}_t$ (resident residuals reconstructed by $\tilde{k}_{i,j} = \bar{k}_i + \tilde{r}^K_{i,j}$ and an analogous $\tilde{v}$) and a medium-priority set $\mathcal{M}_t$ (one anchor token per chunk). The active KV is $\mathcal{A}_t = \mathcal{H}_t \cup \mathcal{M}_t \cup \mathcal{T} \cup \mathcal{W}_t$. Per-step rerouting wastes computation when generated text stays near the previous routed region, so SAR-KV caches the previous route and applies Adaptive State Tracking (AST). AST is controlled by a minimum reroute interval $\delta_{\min}$, a forced reroute interval $\delta_{\max}$, and a cosine-drift threshold θ. Rerouting fires only when no route exists, when $\delta_{\max}$ steps have elapsed, or when at least $\delta_{\min}$ steps have elapsed and the cosine similarity between the current query's anchor scores and those of the last routing query falls below θ. Default values are $\delta_{\min}$=4, $\delta_{\max}$=16, and θ=0.80 throughout.

3 Experiments

Setup. We evaluate Llama-3.1-8B-Instruct (Meta Ai, 2024a) with Full KV, SnapKV (Li et al., 2024), H2O (Zhang et al., 2023), SentenceKV (Zhu et al., 2025), KIVI (Liu et al., 2024c), KVTuner (Li et al., 2025), and ZipCache (He et al., 2024) on LongBench (Bai et al., 2024) and RULER (Hsieh et al., 2024). All measurements use one NVIDIA RTX PRO 5000 Blackwell GPU (48 GB). The 0.25 profile fixes the target fp16-token cache budget rather than all SAR-KV runtime options; benchmark defaults are in Appendix E. Cache, Peak, and out-of-memory (OOM) respectively denote measured resident KV cache size after prefill, end-to-end peak GPU memory, and failure to complete prefill plus 1,000 decode steps (Appendix D). Each cell reports one fixed-seed run, so 1–2 point gaps are descriptive. Appendix F provides Llama-3.2-3B-Instruct (Meta Ai, 2024b) and per-task results.[1]

Peak memory at 0.25 cache ratio. At a fixed nominal cache ratio, SAR-KV has the lowest observed 32K/64K peak GPU memory among evaluated implementations and is the only evaluated implementation that completes the 128K prefill-plus-1,000-decode-step protocol (33.14 GB peak, 3.92 GB cache). H2O/SnapKV select tokens only after forming prompt-level query/key/value tensors and repeated grouped-query KV states, so their prefill peak still contains a token-level allocation even when the final cache is small. Quantization baselines lower the storage precision but still instantiate token-level KV before packing, together with scale/zero-point and packing state; ZipCache further performs probe attention over the prompt, yielding a 44.35 GB peak at 32K despite a 0.91 GB cache. SAR-KV lowers peak memory because it avoids full prompt-token KV retention after prefill, keeping chunk anchors plus budgeted residuals and reconstructing token-level KV only for routed chunks. This changes the peak-

[1] KVTuner ships calibration presets only for Llama-3.1-8B, Mistral-7B-v0.3, and Qwen2.5-3B/7B; the Llama-3.2-3B row therefore uses the uniform-bit RTN fallback (a KIVI-style baseline without per-layer calibration), not the calibrated configuration that defines KVTuner.

Method	32K				64K				128K			
	Peak	Cache	TTFT	TPOT	Peak	Cache	TTFT	TPOT	Peak	Cache	TTFT	TPOT
Full KV	22.43	3.91	4.11	28.47	29.89	7.81	11.14	42.42	OOM	OOM	OOM	OOM
H2O	22.43	0.98	4.31	**27.17**	29.89	1.95	**11.30**	**40.31**	OOM	OOM	OOM	OOM
SnapKV	22.43	0.98	4.38	28.64	29.89	1.95	11.90	42.41	OOM	OOM	OOM	OOM
SentenceKV	25.11	0.98	12.96	163.83	30.22	1.95	28.23	170.87	OOM	OOM	OOM	OOM
KIVI	21.64	1.22	4.27	48.51	38.34	2.44	11.48	81.38	OOM	OOM	OOM	OOM
KVTuner	23.37	0.94	**4.18**	139.63	31.77	**1.88**	11.47	330.69	OOM	OOM	OOM	OOM
ZipCache	44.35	**0.91**	7.30	136.66	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
SAR-KV (Ours)	**19.52**	0.97	4.71	75.18	**24.05**	1.95	12.13	125.57	**33.14**	**3.92**	**35.90**	**251.01**

Table 2: Long-context efficiency at the 0.25 cache ratio on Llama-3.1-8B-Instruct. Peak/Cache are GB, time to first token (TTFT) seconds, and time per output token (TPOT) ms/token over 1,000 decode steps. **Bold** marks the best completed non-Full-KV result per metric/length; "OOM" denotes out of memory on one 48 GB GPU.

Method	SQA	MQA	Few	Synth	Code	Avg
Full KV	43.42	42.81	34.79	52.88	12.64	37.31
SnapKV	42.65	42.64	34.63	52.79	12.54	37.05
H2O	40.95	42.00	33.75	52.62	**12.86**	36.44
SentenceKV	42.51	42.47	34.44	52.79	12.64	36.97
KIVI	**43.32**	**42.78**	34.29	52.38	12.84	37.12
KVTuner	42.42	42.42	35.65	**53.20**	12.28	**37.19**
ZipCache	$*$	42.65	34.69	52.17	12.52	$*$
SAR-KV	40.88	42.47	**35.81**	53.00	11.24	36.68

Table 3: Category-average LongBench scores for the 13-task subset using Llama-3.1-8B-Instruct under the 0.25 profile. $*$ denotes an incomplete category due to at least one OOM task. **Bold** marks the best or tied-best non-Full-KV result; Avg is the arithmetic mean over the five displayed category scores.

Method	Single	Multi	FWE	VT	QA1
Full KV	1.00	0.98	0.85	0.94	0.55
SnapKV	**1.00**	**0.99**	0.75	0.91	**0.65**
H2O	0.99	0.94	0.82	**0.93**	**0.65**
SentenceKV	OOM	OOM	0.73	OOM	OOM
KIVI	OOM	OOM	OOM	OOM	OOM
KVTuner	**1.00**	0.98	**0.85**	0.90	0.63
ZipCache	OOM	OOM	OOM	OOM	OOM
SAR-KV	0.81	0.79	0.81	0.78	**0.65**

Table 4: RULER-64K retrieval accuracy on Llama-3.1-8B-Instruct under the 0.25 profile on one 48 GB GPU. *Single* averages Needle-in-a-Haystack (NIAH) single-needle tasks (S1, S2); *Multi* averages NIAH multi-key, multi-query, and multi-value tasks (MK1, MQ, MV). Per-task results are in Appendix G. **Bold** marks the best or tied-best compression method.

memory driver from full prompt-token materialization to a bounded anchor-residual representation.

LongBench score under compression. Table 3 shows SAR-KV preserves average LongBench performance under the strict memory profile, but it is not the strongest accuracy-only baseline: its five-category average is 36.68, compared with 37.31 for Full KV and 37.19 for KVTuner. Its advantage is memory feasibility rather than accuracy dominance. Per-task and Llama-3.2-3B results appear in Appendix F.

RULER-64K retrieval. Table 4 follows RULER (Hsieh et al., 2024) and uses exact retrieval as a failure-mode diagnostic. SAR-KV ties for best on QA1 (0.65) and reaches 0.81 on FWE, but stronger cache-resident baselines retain a 19–20 percentage-point lead on the aggregated NIAH scores.

Ablation analysis. Ablation (Appendix H) decomposes MV gains. Residuals without residual-aware routing do not help; RESIDUALMAX raises MV to 70.50, and the default AST schedule reaches 75.50. MK2 stays at 62.00 throughout.

4 Conclusion

SAR-KV separates chunk addressability, residual fidelity, and prompt/decode state, turning KV-cache compression into selective reconstruction rather than token eviction. Anchors keep chunks reachable under a tight budget, while residual routing restores token-level detail only for query-selected chunks. This design makes the compression target implementation-aware: the relevant constraint is not only resident-cache size, but the peak memory of an actual prefill-plus-decode run. In Llama-3.1-8B-Instruct evaluation, SAR-KV reduces 4-bit RTN MSE by 5.9×, stays within 0.63 LongBench points of Full KV, achieves the lowest observed 32K/64K end-to-end peak GPU memory, and uniquely completes the 128K prefill-plus-1,000-decode-step protocol, while exact retrieval and code remain challenging.

Limitations

Implementation-level comparison. Our peak-memory results compare concrete implementations under a shared PyTorch/CUDA profiling protocol. They should not be interpreted as hardware-independent lower bounds for the baseline methods: fused kernels, streaming compression, or different serving stacks could change absolute peak memory. The claim is therefore about end-to-end behavior of the evaluated implementations under the stated single-GPU protocol.

Model coverage. Memory is measured on Llama-3.1-8B-Instruct, while LongBench scores are additionally reported on Llama-3.2-3B-Instruct; Qwen, Mistral, and Gemma families remain unverified. The gap between the 8B and 3B results shows that compression headroom varies across model scales rather than being a fixed property of the method.

Statistical uncertainty. Each cell in the efficiency, RULER, and LongBench tables reports a single fixed-seed run; 1–2 point differences between methods should not be interpreted as statistically significant.

128K scope. The 128K result is a completion result for the prefill-plus-1,000-step generation protocol, not a task-level 128K accuracy evaluation.

Chunker assumptions. The chunker relies on tokenizer-level punctuation markers and length constraints. We have not separately validated code-heavy prompts beyond the LongBench code tasks, Markdown, tables, dialogue traces, or non-English inputs.

Latency. The current SAR-KV does not dominate token-selection methods in throughput when those methods fit within end-to-end peak GPU memory; overhead comes from Python-level routing, residual dequantization, and tensor-gather operations. A fused route-cache kernel in the spirit of FlashAttention (Dao et al., 2022; Dao, 2024) could make attention scale with the active routed length, but we do not pursue this engineering step here. Factorized KV representations (Novikov et al., 2015; Saxena et al., 2024) are related but not evaluated here.

Single-GPU setting. All measurements use one NVIDIA RTX PRO 5000 Blackwell GPU (48 GB); multi-GPU tensor parallelism and production serving stacks such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) remain unverified.

References

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. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*. Association for Computational Linguistics.

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)*. Association for Computational Linguistics.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2025. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*. 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 *Proceedings of the 42nd International Conference on Machine Learning*. PMLR.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *CoRR*, abs/2406.2069.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. 2024. NACL: A general and effective KV cache eviction framework for LLM at inference time. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*. Association for Computational Linguistics.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *The*

Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Wei Gao, Xinyu Zhou, Peng Sun, Tianwei Zhang, and Yonggang Wen. 2025. Rethinking key-value cache compression techniques for large language model serving. In *Proceedings of the Eighth Conference on Machine Learning and Systems, Mlsys 2025, Santa Clara, CA, USA, May 12-15, 2025*. OpenReview.net/mlsys.org.

Jitai Hao, Qiang Huang, Yaowei Wang, Min Zhang, and Jun Yu. 2026. DeltaKV: Residual-Based KV Cache Compression via Long-Range Similarity. *arXiv preprint*.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. 2024. ZipCache: Accurate and efficient KV cache quantization with salient token identification. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

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

Ishraq Khan, Assad Chowdary, Sharoz Haseeb, Urvish Patel, and Yousuf Zaii. 2025. Kodezi chronos: A debugging-first language model for repository-scale code understanding. *CoRR*, abs/2507.12482.

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*. ACM.

Xing Li, Zeyu Xing, Yiming Li, Linping Qu, Hui-Ling Zhen, Yiwu Yao, Wulong Liu, Sinno Jialin Pan, and Mingxuan Yuan. 2025. KVTuner: Sensitivity-aware layer-wise mixed-precision KV cache quantization for efficient and nearly lossless LLM inference. In *Forty-Second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net.

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 *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Reza Haffari, and Bohan Zhuang. 2024a. MiniCache: KV cache compression in depth dimension for large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. 2025a. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression. In *62nd ACM/IEEE Design Automation Conference, DAC 2025, San Francisco, CA, USA, June 22-25, 2025*. IEEE.

Tianyang Liu, Canwen Xu, and Julian J. McAuley. 2024b. RepoBench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. 2025b. ChunkKV: Semantic-Preserving KV Cache Compression for Efficient Long-Context LLM Inference. *arXiv preprint*.

Xin Liu, Xudong Wang, Pei Liu, and Guoming Tang. 2026. ZSMerge: Zero-shot KV cache compression for memory-efficient long-context LLMs.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024c. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Forty-First International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27,*

2024, Proceedings of Machine Learning Research. PMLR / OpenReview.net.

Meta Ai. 2024a. Meta llama 3.1.

Meta Ai. 2024b. meta-llama/Llama-3.2-3B-Instruct (model).

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. 2015. Tensorizing neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. 2024. Eigen attention: Attention in low-rank space for KV cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, Findings of ACL. Association for Computational Linguistics.

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 *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Proceedings of Machine Learning Research. PMLR.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Forty-First International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research. PMLR / OpenReview.net.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*.

Guangtao Wang, Shubhangi Upasani, Chen Wu, Darshan Gandhi, Jonathan Li, Changran Hu, Bo Li, and Urmish Thakker. 2025. LLMs know what to drop: Self-attention guided KV cache eviction for efficient long-context inference. *CoRR*, abs/2503.8879.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024a. InfLLM: Training-free long-context extrapolation for llms with an efficient context memory. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024b. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*.

Yi Xiong, Hao Wu, Changxu Shao, Ziqing Wang, Rui Zhang, Yuhong Guo, Junping Zhao, Ke Zhang, and Zhenxuan Pan. 2024. LayerKV: Optimizing large language model serving with layer-wise KV cache management. *CoRR*, abs/2410.428.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. 2024. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Yuxuan Zhu, Ali Falahati, David H. Yang, and Mohammad Mohammadi Amiri. 2025. SentenceKV: Efficient LLM inference via sentence-level semantic KV caching. In *Second Conference on Language Modeling*.

Algorithm 1 SAR-KV cache update in the implementation

Require: prompt IDs I, punctuation IDs $\mathcal{P}$, KV states (K, V), query q_t
Require: budget Γ, chunk limits (m, f), tail/window sizes (T, W)
Ensure: updated cache state or active KV set $\mathcal{A}_t$

```
if |K| > 1 then                                          ▷ prefill
    if I is unavailable then
        return (K, V)
    end if
    C ← CHUNK(I, P, m, f)
    choose residual bits b, tail T, and window W under Γ
    allocate residual chunks R by uniform interleaving; skip prompt-tail overlap and over-budget chunks
    for all C_i ∈ C do
        store mean anchors (k̄_i, v̄_i)
    end for
    quantize residuals K − K̄ and V − V̄ for chunks in R
    pin the last T prompt tokens and initialize the W-token recent window
    return updated SAR-KV cache state
else                                                     ▷ decode
    append the new KV token to the recent window
    if AST triggers rerouting then
        rank chunks by mean-anchor score, optionally refined by RESIDUALMAX
        select high-priority chunks H_t and mean-only chunks M_t
    end if
    reconstruct chunks in H_t with dequantized residuals
    A_t ← H_t ∪ M_t ∪ prompt tail ∪ recent window
    return A_t
end if
```

A Coverage-Aware Budget Scheduling

For one layer and batch size one, let $G = \lceil d/g \rceil$, let e be the bytes per full-precision element, and define $c_{\text{fp16}} = 2Hde$. Full KV uses $B_{\text{full}} = Lc_{\text{fp16}}$. For $b < 16$, the residual cost per retained token is $c_{\text{res}}(b) = 2H\lceil db/8 \rceil + 4HGe + c_{\text{idx}}$, with the three terms storing packed key/value residual values, key/value scale/zero-point metadata, and indices linking residual tokens to their chunks. In the reported runs $g = 128$, $e = 2$ bytes, and $c_{\text{idx}} = 8$ bytes. Hence $B_{\text{SAR}}(b) = Mc_{\text{fp16}} + (T + W)c_{\text{fp16}} + R_b c_{\text{res}}(b)$, and the resident saving relative to Full KV is $\rho_{\text{save}}(b) = 1 - B_{\text{SAR}}(b)/B_{\text{full}}$. The implementation path that realizes this accounting is described in Appendix B.

B Implementation-Level Cache Update

Algorithm 1 describes the cache-update path used by SAR-KV.

C Detailed Related Work Survey

SAR-KV is closest to methods that reduce the resident KV footprint, but it makes a different representation choice. Cache-placement and offloading systems move KV states across memory tiers (Sheng et al., 2023; Xiao et al., 2024a; Xiong et al., 2024; Zhu et al., 2025); SAR-KV instead keeps a compressed GPU-resident representation and therefore evaluates end-to-end peak memory and completion status under a fixed single-GPU budget.

Token-selection, eviction, and runtime sparse-access methods keep or fetch a subset of raw token states (Zhang et al., 2023; Li et al., 2024; Tang et al., 2024; Liu et al., 2023; Chen et al., 2024; Cai et al., 2024; Behnam et al., 2025). SAR-KV keeps every chunk addressable through a chunk-mean anchor, and reconstructs token-level KV only for routed chunks whose residuals are resident.

Quantization and representation-compression methods reduce token-level KV states, residuals, or their low-bit/low-dimensional surrogates (Liu et al., 2024c; Hooper et al., 2024; Li et al., 2025; He et al., 2024; Liu et al., 2024a; Hao et al., 2026; Liu et al., 2026). SAR-KV instead centers each punctuation-delimited chunk by a persistent chunk-mean anchor and quantizes only the budgeted residuals. Chunk- and cluster-level methods mainly use groups as selection units (Liu et al., 2025a,b); SAR-KV uses the chunk mean as the storage unit, routing key, and mean-only attention surrogate before reconstructing routed residual chunks.

D Efficiency Profiling Protocol

The memory tables use the same profiling path as the implementation benchmark. Cache-ratio profiles are target fp16-token budgets, not post hoc equalized resident-cache values. For each method and context length, the script runs a short warmup, resets method state affected by warmup when necessary, constructs the cache, synchronizes CUDA, and resets CUDA peak-memory statistics immediately before the measured run. It then executes one prefill call followed by 1,000 greedy autoregressive decode steps. TTFT is the sum of prefill time and the time of the first token output; TPOT is the decode time divided by these 1,000 steps. Peak memory is `torch.cuda.max_memory_allocated` after the reset, so it includes prefill/decode activations, reconstruction or offloading buffers, res-

Entry	Ratio(s)	T	W	Budget
LongBench	0.25	32	32	dyn./safe
RULER	0.25	32	32	dyn./safe
Memory	0.25, 0.35	32	32	dyn./safe

Table 5: SAR-KV benchmark-entry defaults. T is the full-precision prompt tail and W is the recent decode window. "dyn./safe" denotes dynamic budget allocation with coverage-safe leftover handling.

ident KV states, and cache metadata. The reported cache size is measured from the method's resident cache object after prefill. For methods whose implementation stores repeated grouped-query attention (GQA) heads before cache update, the resident cache is normalized by the ratio between attention heads and KV heads so that cache ratios are comparable. An OOM entry means that the method fails to complete this prefill-plus-decode protocol on the NVIDIA RTX PRO 5000 Blackwell GPU (48 GB). The tables aggregate the resulting per-task score files and Peak/Cache/TTFT/TPOT fields directly; incomplete or failed runs are kept as OOM/$*$ entries rather than folded into averages.

E Benchmark-Entry Configuration

Each benchmark entry sets the model, method, dataset, and length arguments, and otherwise uses the Python-entry defaults. Table 5 lists the SAR-KV defaults that differ across benchmark entries. All rows use automatic residual bits, residual block size $g = 128$, uniform residual-block allocation, RESIDUALMAX routing, update interval 4, and AST drift threshold $\theta = 0.80$. Because AST intervals are unset in the entry scripts, the implementation derives $\delta_{\min} = 4$ and $\delta_{\max} = 16$ from the update interval.

F LongBench Per-Task Results

Table 6 reports per-task scores for exactly the 13-task LongBench subset used to form the main category averages under the 0.25 profile on Llama-3.1-8B-Instruct (Meta Ai, 2024a) and Llama-3.2-3B-Instruct (Meta Ai, 2024b). "$*$" indicates an out-of-memory entry; **bold** marks the best or tied-best score among non-Full-KV methods within each model block. Failed runs are kept as OOM/$*$ rather than imputed.

For category-level patterns, on Llama-3.1-8B, SAR-KV has the highest observed HotpotQA score in this run (54.93 vs. SnapKV 53.80), is higher than KVTuner on TREC (13.50 vs. 11.00), and has the highest non-Full-KV PCount (7.00). It lags on NarrativeQA (28.06 vs. SnapKV 30.07), TriviaQA (86.37 vs. Full KV / SnapKV / KIVI 89.41), and both code-completion tasks. On Llama-3.2-3B, compression slack shrinks across the board; SAR-KV has the highest observed SAMSum, Lcc, and RB-P values and is close to the highest observed NarrativeQA value (23.83), but drops on the synthetic tasks (PCount 1.67, PRe 66.50) and on Qasper / 2WikiMultihopQA. In these two models, compression headroom is therefore model-dependent.

G RULER-64K Per-Task Results

Table 7 reports the eight RULER-64K subtask scores that underlie the *Single* and *Multi* averages in Table 4: *Single* averages S1 and S2; *Multi* averages MK1, MQ, and MV. **Bold** = best or tied-best among compression methods.

H Component Ablation and Sensitivity

Table 8 gives the four-variant ablation referenced in the main text. All variants share the 0.25 cache budget and 4-bit residual quantization with 100 examples per task; only routing and rerouting change. **V1 anchor-only**: mean anchors only, route by the mean, AST off. **V2+residual**: residuals restored to 60% of the budget but still mean-routed. **V3 +ResidualMax**: route by RESIDUALMAX under the controlled reroute schedule. **V4 full**: V3 with the default AST schedule.

Tables 9–11 report local sensitivity analyses on RULER-64K using Llama-3.1-8B-Instruct at the 0.25 cache ratio. These sweeps are not an exhaustive search over all policy thresholds: they vary residual bit-width, chunk granularity, and the Adaptive State Tracking (AST) update interval while keeping the coverage gates and drift threshold fixed. The results suggest that the reported retrieval behavior is not driven by a narrow choice of chunk size or rerouting interval. The bit-width sweep further supports the default use of 4-bit residuals under tight budgets: lower per-token residual cost allows residuals to cover more chunks, which is more beneficial on these retrieval subtasks than using higher precision for fewer chunks.

LLM	Method	Single-Doc QA			Multi-Doc QA			Few-shot			Synthetic		Code	
		NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Mus	TREC	TQA	SAMSum	PCnt	PRe	Lcc	RB-P
Llama-3.1-8B	Full KV	29.69	47.56	53.00	53.76	46.12	28.56	7.50	89.41	7.47	6.25	99.50	13.57	11.71
	SnapKV	30.07	**47.28**	50.61	53.80	45.75	28.38	7.00	**89.41**	7.48	6.08	**99.50**	13.25	11.83
	H2O	**30.25**	42.88	49.72	52.71	45.44	27.86	4.50	89.24	7.52	5.75	**99.50**	13.66	**12.05**
	SentenceKV	30.12	47.25	50.15	53.42	45.75	28.23	6.50	89.33	7.48	6.08	**99.50**	13.24	12.04
	KIVI	29.22	47.18	**53.57**	53.76	**45.96**	28.61	6.00	**89.41**	7.47	5.25	**99.50**	**13.90**	11.79
	KVTuner	27.66	47.26	52.35	53.41	45.25	28.60	11.00	88.28	7.67	6.91	**99.50**	13.84	10.71
	ZipCache	$*$	45.21	51.49	53.52	45.72	**28.71**	7.08	89.26	**7.72**	4.85	**99.50**	13.43	11.60
	SAR-KV	28.06	45.49	49.08	**54.93**	45.89	26.59	**13.50**	86.37	7.57	**7.00**	99.00	12.37	10.12
Llama-3.2-3B	Full KV	22.73	41.69	49.80	50.46	41.80	20.96	2.00	9.17	6.51	7.50	78.50	19.23	17.12
	SnapKV	23.33	36.27	48.90	**49.93**	41.07	21.40	4.50	**9.20**	6.56	6.28	80.00	19.32	16.77
	H2O	22.56	33.08	47.52	48.73	40.46	20.52	1.00	8.46	6.49	6.76	79.00	19.24	16.93
	SentenceKV	23.57	36.19	49.38	49.76	40.98	21.61	**5.50**	9.09	6.75	6.28	**80.50**	19.38	16.92
	KIVI	23.15	**40.21**	**49.63**	48.93	**41.37**	**22.25**	1.50	9.17	6.40	6.08	78.50	19.22	16.79
	KVTuner	22.54	38.63	47.90	48.60	38.47	21.72	1.50	6.76	6.84	5.28	75.00	19.27	17.11
	ZipCache	**23.88**	39.03	48.13	48.76	40.24	21.37	0.50	8.97	6.18	**7.00**	70.50	19.28	17.30
	SAR-KV	23.83	34.61	44.36	48.51	37.30	18.97	0.50	7.48	**7.29**	1.67	66.50	**19.60**	**17.94**

Table 6: LongBench 13-task subset per-task scores under the 0.25 cache-ratio profile.

Method	S1	S2	MK1	MQ	MV	FWE	VT	QA1
Full KV	1.00	1.00	1.00	0.98	0.96	0.85	0.94	0.55
SnapKV	**1.00**	**1.00**	**1.00**	**1.00**	**0.97**	0.75	0.91	**0.65**
H2O	**1.00**	0.98	0.98	0.97	0.87	0.82	**0.93**	**0.65**
SentenceKV	OOM	OOM	OOM	OOM	OOM	0.73	OOM	OOM
KIVI	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
KVTuner	**1.00**	**1.00**	**1.00**	0.98	**0.97**	**0.85**	0.90	0.63
ZipCache	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
SAR-KV	0.76	0.86	0.78	0.85	0.75	0.81	0.78	**0.65**

Table 7: RULER-64K per-task accuracy on Llama-3.1-8B-Instruct under the 0.25 profile.

Task	V1 anchor	V2 +res	V3 +RMax	V4 full
MK2	62.00	62.00	62.00	**62.00**
MV	56.00	55.50	70.50	**75.50**
QA1	65.17	65.17	64.50	64.50

Table 8: RULER-64K component ablation on Llama-3.1-8B-Instruct under the 0.25 profile.

Task	4-bit	8-bit	16-bit
MK2	**62.00**	40.00	26.00
MV	**75.50**	32.50	16.00

Table 9: Residual bit-width sensitivity on RULER-64K. Under the fixed 0.25 cache budget, lower bit-widths allow residuals to cover more chunks; 4-bit gives the best accuracy among the tested bit-widths on these retrieval subtasks.

Task	4	8	16	32	64
MK2	64.00	62.00	64.00	**70.00**	60.00
MV	73.00	75.50	**80.00**	74.50	76.00

Table 10: Chunk granularity sensitivity on RULER-64K. Header values denote `minsentencetokens`. Accuracy stays within a narrow band and shows no monotonic dependence on this chunking threshold.

Task	1	2	4	8	16	32
MK2	**62.00**	62.00	62.00	62.00	62.00	62.00
MV	**75.50**	75.50	75.50	75.00	73.00	70.50
VT	**76.40**	75.60	**76.40**	76.00	76.00	76.00

Table 11: AST update-interval sensitivity on RULER-64K. Accuracy is stable for intervals in $[1, 16]$ and drops at 32, consistent with increasingly stale routes on multi-value retrieval.

I Peak-Memory Check at 0.35

Table 12 reports a peak-memory boundary check under the relaxed 0.35 cache budget. TPOT is averaged over 1,000 decode steps. **Bold** marks the best completed non-Full-KV result per metric/length. The same pattern persists: SAR-KV retains the lowest peak memory at 32K and 64K and remains the only evaluated implementation in this check that completes the 128K prefill-plus-1,000-decode-step protocol. This relaxed-budget check is intended as peak-memory evidence rather than latency evidence, because higher residual cover-

Context	Method	Peak	Cache	TTFT	TPOT
32K	Full KV	22.43	3.91	4.10	28.50
	SnapKV	23.99	1.37	4.62	39.32
	H2O	23.99	1.37	**4.34**	**34.52**
	SentenceKV	24.22	1.37	12.18	163.88
	KIVI	21.64	1.22	4.61	53.06
	KVTuner	23.92	1.48	4.35	130.45
	ZipCache	44.35	**0.91**	7.52	148.48
	SAR-KV	**19.68**	1.12	4.76	101.48
64K	Full KV	29.89	7.81	11.36	44.01
	SnapKV	33.02	2.73	12.33	56.13
	H2O	33.02	2.73	**11.47**	**54.31**
	SentenceKV	33.44	2.73	27.36	178.09
	KIVI	38.34	2.44	12.88	87.95
	KVTuner	32.87	2.97	11.68	286.43
	ZipCache	OOM	OOM	OOM	OOM
	SAR-KV	**24.33**	**2.23**	12.25	184.27
128K	Full KV	OOM	OOM	OOM	OOM
	SnapKV	OOM	OOM	OOM	OOM
	H2O	OOM	OOM	OOM	OOM
	SentenceKV	OOM	OOM	OOM	OOM
	KIVI	OOM	OOM	OOM	OOM
	KVTuner	OOM	OOM	OOM	OOM
	ZipCache	OOM	OOM	OOM	OOM
	SAR-KV	**33.66**	**4.45**	**36.77**	**346.59**

Table 12: Long-context efficiency with 0.35 profile.

age adds routing, dequantization, and gather overhead. For KIVI, ZipCache, and similar baselines, resident cache size follows the implementation strategy and need not scale linearly with the target ratio.