RECENT KV CACHE OPTIMIZATION STRATEGIES FOR EFFICIENT LARGE LANGUAGE MODEL INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive inference in Large Language Models (LLMs) is bottle necked by the memory capacity and bandwidth demands of the Key-Value (KV) cache. Although many KV cache optimization strategies have been proposed, making optimal deployment choices is difficult due to inconsistent evaluation practices and difficult to make direct comparisons. We present a systematic review and quantitative meta analysis of KV cache optimization strategies. We organized these approaches families of strategies, including quantization, token eviction, architectural reductions, and paged memory management. By extracting and normalizing the reported data from existing studies, we examined the impact of memory footprint, latency, throughput, and model quality on KV cache compression strategies. Our findings show that there is no single method that is universally superior; rather, the optimal strategy selection is highly dependent on the specific deployment constraints. We provide an evidence based recommendation matrix to guide future deployments and highlight gaps in current evaluation methods.

1 INTRODUCTION

As Large Language Models (LLMs) are deployed with longer context windows, larger batch sizes, and increasingly demanding inference workloads, memory efficiency has become a central constraint on scalable autoregressive generation. A major source of this pressure is the Key-Value (KV) cache, which stores the key and value tensors generated for previously processed tokens. Since the introduction of the standard multi-head attention mechanism Vaswani et al. (2017), caching these tensors has been essential to avoid redundant computation during autoregressive decoding. However, the size of the KV cache grows with sequence length, causing memory capacity and bandwidth requirements to increase rapidly. At scale, these demands can increase operational costs as well as limit decoding throughput and inference efficiency Dao et al. (2022); Pope et al. (2022); Aminabadi et al. (2022); Sheng et al. (2023); Liu et al. (2023a).

To mitigate these memory and bandwidth bottlenecks, the literature has seen an explosion of KV cache optimization strategies. Recent advances have proposed highly specialized architectures to push context boundaries to millions of tokens DeepSeek-AI (2026); Hooper et al. (2024), formalized novel mathematical interpretations of attention bounds Geh et al. (2026); Yang et al. (2026), introduced geometry aware or stochastic algorithms Hsu et al. (2026); Wang et al. (2026a); Su et al. (2026), and designed routing and compression managers Wang et al. (2026b); Yoo et al. (2026); Liu et al. (2025c); Lu et al. (2026). Despite the surge of different KV cache optimization strategies, differing evaluation practices prevent direct comparisons. Different studies measure success under very different constraints, including: varying model scales, attention architectures, evaluation metrics, and hardware accelerators Shen et al. (2026); Prucs et al. (2025); Xu et al. (2026); Luohe et al. (2024). To date, existing qualitative narrative surveys and reviews on LLM inference efficiency have documented these techniques but have largely remained qualitative Khoshnoodi et al. (2024); Miao et al. (2023); Jiang et al. (2026); Zhen et al. (2025); Kim et al. (2026). The literature therefore lacks a quantitative meta-analysis that identifies which KV cache optimization strategies achieve the optimal trade-offs among memory consumption, latency, throughput, and model quality, and how these trade-offs vary with deployment constraints.

In this work, we address this gap by performing a systematic review and quantitative meta analysis of recent KV cache optimization strategies published between January 2022 and September 2026.

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

We constructed a unified taxonomy of KV cache optimizations and compare results within each study by calculating relative measures, such as memory reduction and throughput ratios. Through hardware aware normalization, our meta analysis maps previously isolated claims onto a unified framework. Our findings show that no single KV cache optimization strategy is universally optimal and is used to inform an evidence-based decision framework for selecting efficient strategies that meet deployment-specific memory budgets, as well as latency requirements and target context lengths.

2 Related Work

2.1 Qualitative Surveys on LLM Inference

The rapid evolution of LLM inference has led to multiple surveys of optimized methods, memory bottlenecks, and KV cache management Ma et al. (2026); Zhu et al. (2024); Tian et al. (2025); Sun et al. (2025); Xiao et al. (2026); Li et al. (2024a); Pan & Li (2025). These surveys do not pool results across studies or normalize performance metrics for direct comparison. As a result, they cannot support reliable conclusions about the trade off between memory use and computational latency between each optimization strategy. Our approach directly overcomes these limitations by performing a systematic review and meta analysis that synthesizes data from literature to map the trade offs between memory footprint and computational latency.

2.2 Foundational and Isolated KV-Cache Optimizations

The academic literature is rich with foundational architectural and systems interventions designed to minimize KV cache overhead. Foundational efforts successfully reduced memory bandwidth by sharing keys and values across heads, as seen in Multi-Query Attention (MQA) Shazeer (2019) and Grouped-Query Attention (GQA) Ainslie et al. (2023). Concurrently, systems like Paged Attention treated the KV cache as virtual memory to eliminate fragmentation Kwon et al. (2023). Building on these baselines, numerous token eviction and pruning mechanisms were introduced, assuming that only a fraction of tokens (such as "heavy hitters" or persistent contexts) need to be preserved Zhang et al. (2023); Liu et al. (2023b); Li et al. (2024b).

However, individual proposals continuously evaluate their methods in differing configurations with varying hardware and baseline configurations, making claims of "universal superiority" unreliable Agrawal & Mayer (2026); Qiu et al. (2026); Liu et al. (2025a;b). Recent studies show that the performance gains from KV cache optimization depend strongly on interactions with other optimization techniques and deployment-specific cost constraints Chitty-Venkata et al. (2024); Ahmed et al. (2025); Zellinger & Thomson (2025); Yang et al. (2024b). By extracting and standardizing relative effects within the study, our meta analysis circumvents hardware isolation and effectively compares these strategies.

2.3 Empirical Benchmarking Frameworks

To combat evaluation inconsistency, recent literature has proposed large scale LLM benchmarking suites and standardized system evaluations Agrawal et al. (2025; 2024); Zhong et al. (2024); Wang et al. (2026c); Wu et al. (2024); Majidi et al. (2026); Patel et al. (2024); Dai et al. (2025); Zhu et al. (2026). These frameworks evaluate serving strategies holistically, capturing phenomena like prefill decode interference, phase splitting, and multi class throughput scheduling. Despite their immense value for standardized inference evaluation, these benchmarking frameworks are constrained by implementation overhead and integration complexity. They are typically capable of natively implementing and testing only a tiny fraction of the theoretical KV cache optimization strategies available in the broader academic literature. Our meta analysis acts orthogonally to these benchmarks: by aggregating reported data across all published literature, we transcend the engineering and implementation limits of any single benchmarking suite to capture the entirety of the optimization landscape.

3 METHODS

3.1 RESEARCH QUESTIONS

Three questions guide the analysis of deployment-specific trade-offs: (RQ1) does the cache actually stay within the memory budget set by a compression setting; (RQ2) which memory and performance comparisons remain valid when we match the same prompts, account for repeated runs, or change how results are combined; and (RQ3) which configurations still offer the best trade-offs when we account for uncertainty about which configurations can fairly be compared? Together, these questions inform deployment recommendations by establishing each strategy's actual memory requirements and identifying fair performance comparisons.

We distinguish between a paper, a set of experiments reported in that paper, a configuration, and an individual request and its result. A configuration specifies the model checkpoint, attention architecture, cache representation, runtime, hardware, context length, generation settings, batch settings, and measurement method. Multiple prompts or repeated runs from one benchmark do not count as independent studies. We compare results only when the details needed for a particular claim match, or when we explicitly account for their differences. For example, using the same prompts and maximum generation length lets us compare recorded speeds, but it does not mean the models generated the same number of tokens.

3.2 SOURCE CORPUS AND COMPLETE REPORTING WITHIN THE SELECTED SCOPE

The starting corpus contains 17 primary reports selected through draft references, method-specific retrieval, and exploratory web discovery on 18 September 2026. The target publication window was January 2022 through that date. Queries and source URLs are recorded in `evidence/data/search_log.json` and the source manifest. Searches covered arXiv, PMLR, ACL Anthology, OpenReview, and USENIX. All selected reports were retrieved; nine supplied published-table observations, one supplied numerical benchmark logs, and seven remained contextual. Table 3 and Appendix A specify coverage.

We reanalyze comparable numerical results and draw on contextual evidence to explain how the strategy families in Table 2 address memory constraints. Quantization reduces the bytes used per stored value; architectural changes such as grouped-query attention (GQA) reduce the number of KV heads; and eviction reduces the number of retained tokens. Paging limits wasted memory allocation, prefix reuse avoids storing duplicate state, and offloading changes where state is stored. These distinct mechanisms help explain which strategies suit different deployment needs.

3.3 BENCHMARK PAIRING AND MEASUREMENT AUDIT

We reanalyze the public artifact of Agrawal & Mayer (2026), fixed to commit `925fadf5 4a351b389e26d15535c1129aaefe61e2`. Its 28 system-run files cover FP16, KIVI-2, KIVI-4, SnapKV, CaM, and two TurboQuant settings on Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3. Comparisons between KIVI/SnapKV/CaM were kept and cover GovReport, NarrativeQA, and Qasper, with 200 requests per dataset and model and two recorded runs. These are existing A100 measurements; we ran no new GPU inference.

We pair observations only when their model identifier, dataset, request index, prompt hash, recorded document length, generation budget, and run identifier match exactly. We also check the GPU name and count, along with the Torch and CUDA versions, against the FP16 baseline. These checks do not establish that kernels or all software settings are identical. For each method-request pair, we average the two recorded measurements before calculating ratios. We also repeat the comparisons within each run separately. The prompt hashes allow us to verify pairing without redistributing the prompt text.

By performing source inspection, we identified three outcomes suitable for analysis: KV tensor bytes measured after prefill, the reported sum of prefill time and the first cached-forward time, and cached-forward tokens per second. The two timing metrics do not represent conventional time to first token (TTFT) or end-to-end throughput. In the field labeled TTFT, the first output token has already been selected before the additional cached forward occurs. Actual generated sequence lengths are

not recorded. We excluded CaM's memory field from post-prefill comparisons because it measures cache size after generation. We also exclude TurboQuant's timing and memory fields because its timing covers only decoder operations and, with packing disabled, its memory estimate assumes a hypothetically packed index storage.

3.4 Byte accounting and the distinction between document and prompt length

For a batch-one FP16 cache with L layers, H_{KV} key-value heads, head dimension d, and n cached tokens, tensor storage is

$$M_{16}(n) = 2LH_{\mathrm{KV}}dn\,(2\text{ bytes}), \tag{1}$$

where the leading factor of two accounts for both keys and values. Both models examined have $L = 32$, $H_{\mathrm{KV}} = 8$, and $d = 128$, corresponding to 0.125 MiB per cached token (Meta, 2024; Mistral AI, 2024). We therefore infer the full cached-prompt length by dividing the reported FP16 tensor storage, expressed in MiB, by 0.125 and checking that the result is an integer. It assumes the recorded batch-one, full-cache FP16 layout; the calculation were adjusted for sliding-window caches and shared tensors and cannot be applied directly to peak GPU memory.

The KIVI implementation packs values into b bits and stores two FP16 metadata values for each group of g scalars. The quantized portion therefore occupies a fraction $q_b = b/16 + 2/g$ of the corresponding FP16 storage. KIVI also retains some keys and values in full precision. For residual length R and the observed prompt lengths $n \geq R$, the numbers of full-precision residual tokens are $r_K = n \bmod R$ for keys and $r_V = \min(n, R)$ for values. Combining quantized storage with these residuals gives the predicted ratio of KIVI tensor storage to FP16 tensor storage:

$$\widehat{m}_b(n) = q_b + (1 - q_b)\frac{r_K + r_V}{2n}, \qquad q_b = \frac{b}{16} + \frac{2}{g}. \tag{2}$$

The run metadata specify $g = R = 32$. Equation 2 follows from the existing packing and residual-storage implementation (Liu et al., 2024).

We compare this prediction with every KIVI memory measurement using the inferred full cached-prompt length. We also record the discrepancy obtained when the log's `context_tokens` field is used instead. Source inspection shows that this field counts tokens in the dataset document before the question, instructions, and chat template are added. It therefore represents document length being read in. We report all resulting changes within the existing length buckets: < 4096, 4096-8191, and ≥ 8192 tokens.

3.5 Paired comparisons and aggregation sensitivity

For each request, we compare method a with comparator b using the memory ratio $m_{a/b} = M_a/M_b$ and the recorded-rate ratio $t_{a/b} = T_a/T_b$. Method a meets the criterion for lower memory use and a higher recorded rate at margin ϵ when both

$$m_{a/b} < \frac{1}{1+\epsilon} \quad \text{and} \quad t_{a/b} > 1 + \epsilon.$$

We report results for $\epsilon \in \{0, 0.01, 0.05, 0.10\}$ in both source runs. We examine all six method pairs among KIVI-2, KIVI-4, SnapKV, and CaM. These comparisons describe the implemented configurations, including their different budgets. Separately, for each observed combination of model, dataset, context, and outcome, we calculate the median and geometric mean of the prompt-level ratios, as well as the ratio of the method's mean outcome to the comparator's mean outcome. For baseline values x_i and comparison values y_i, the ratio of means weights each prompt-level ratio by its baseline magnitude:

$$\frac{\sum_i y_i}{\sum_i x_i} = \sum_i w_i \frac{y_i}{x_i}, \qquad w_i = \frac{x_i}{\sum_j x_j}.$$

These summaries therefore answer different descriptive questions. We report changes in both the full method ranking and the top-ranked method. The thresholds are sensitivity margins, not significance levels. Empirical ranges, interquartile ranges (IQRs), and agreement across repeated runs do not provide confidence intervals for a population of implementations.

3.6 CANDIDATE-SET SENSITIVITY AND DEPLOYMENT ADMISSIBILITY

Within each experiment, we compare configurations whose memory use and throughput were measured jointly. Configuration a dominates configuration b if it uses no more memory ($m_a \leq m_b$) and achieves at least as much throughput ($t_a \geq t_b$), with a improvement in at least one measure.

Let C contain the recorded configuration blocks with unambiguous method identities, and let U contain the additional blocks with ambiguous identities. To assess sensitivity to including these additional blocks, we consider every candidate set S that includes all of C and any subset of U: $C \subseteq S \subseteq C \cup U$. For each candidate set, $F(S)$ is the non-dominated subset: configurations that no other configuration in S dominates. We then report

$$F_{\text{certain}} = \bigcap_S F(S), \qquad F_{\text{possible}} = \bigcup_S F(S). \tag{3}$$

Thus, F_{certain} contains configurations that are non-dominated in every candidate set, whereas F_{possible} contains configurations that are non-dominated in at least one candidate set. These sets describe sensitivity to including or excluding recorded blocks with ambiguous identities. They do not assign probabilities or bound the frontier across all algorithms. Our preferred comparison includes all valid recorded blocks; the enumeration shows how excluding blocks with ambiguous identities affects the result. Multiplying all memory values by one common positive constant and all throughput values by another preserves dominance.

Finally, recommending a configuration under a system constraint requires quality and systems evidence that both apply to that configuration. We present quality differences from the literature separately by task and in their original metrics. We also do not estimate I^2, test moderator effects, or identify a universally optimal deployment configuration. We release source hashes, complete paired rows, code-derived checks, exclusions, and plot inputs with the project. Appendix E provides the procedure for reproducing the analysis.

4 RESULTS AND DISCUSSION

4.1 EVIDENCE COVERAGE AND LIMITS OF THE COMPARISON

The published-table corpus contains 433 observations from nine papers. Of these, 377 are retained and 56 are excluded, with a reason recorded for each exclusion. Only 28 retained observations meet the original metadata-completeness rule, which requires information on model, hardware, context, batch, and workload. The numerical benchmark contains 15,200 records across 28 run files. The existing pairing procedure yields 13,200 request-outcome ratios and 165 method-specific cells. Our new enumeration of all method pairs produces 7,200 paired rows across 36 strata, each defined by a model, dataset, and method pair. These rows represent 1,200 model-prompt instances.

The main findings address how storage is counted, how configurations compare when tested on the same prompts, and how stable selection decisions are under the specified constraints. These analyses show how deployment constraints determine which configurations are eligible and which performance comparisons can help select among them. Section 4.6 draws on the contextual literature to explain how these findings relate to broader deployment settings.

4.2 STORAGE OVERHEAD CHANGES BUDGET ELIGIBILITY

The implemented KIVI settings require one metadata bit per scalar in addition to the quantized values. Before including residual storage, KIVI-2 therefore uses 3 bits per scalar and KIVI-4 uses 5 bits. Relative to FP16, their asymptotic storage fractions are $q_2 = 3/16$ (18.75%) and $q_4 = 5/16$ (31.25%). These exceed the nominal fractions of 12.5% and 25%, which count only the quantized values. Including residual storage, the observed storage ratios across all retained prompts range from 18.77% to 19.86% for KIVI-2 and from 31.26% to 32.19% for KIVI-4. SnapKV's tested retention setting uses approximately 25% of FP16 tensor storage (Figure 1). The additional KIVI storage is explained by metadata and residuals. Budget eligibility must therefore be assessed using total storage, including these overheads, rather than nominal bit width alone.

Equation 2 agrees with all 4,800 individual KIVI run measurements to an absolute ratio error below 10^{-15} when using the full-prompt length inferred from FP16 bytes. Using recorded document length

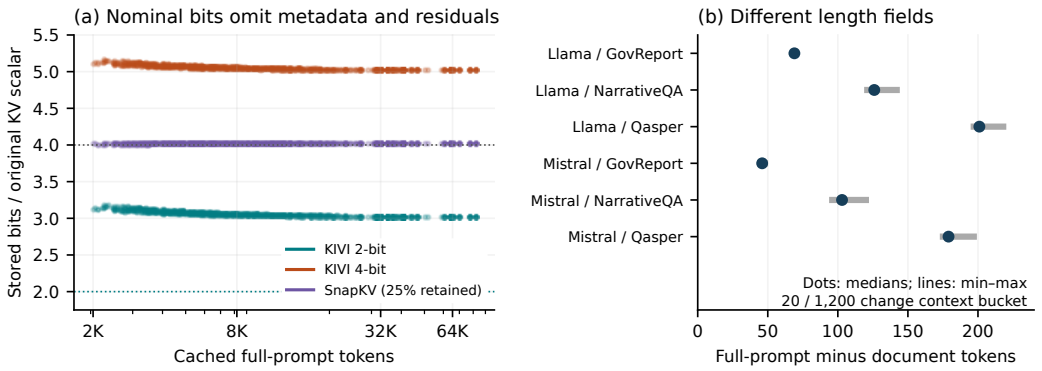


Figure 1: Storage accounting across all retained model-prompt instances. Left: effective stored bits include quantization metadata and full-precision residuals; dotted lines mark nominal two/four-bit payloads. Right: full cached-prompt length inferred from FP16 tensor bytes differs from the logged document length. The plotted values concern stored tensors, not peak device memory.

instead produces an error as large as 0.00575 in the storage ratio. The inferred additional length is 45-218 tokens, and 20 of 1,200 model-prompt instances cross an existing context-bucket boundary.

Under a post-prefill KV budget of at most 25% of FP16, nominal four-bit reasoning would admit KIVI-4, but none of its 1,200 observed configurations fits. KIVI-2 and the tested SnapKV setting both fit every instance. At a 20% budget, only KIVI-2 among these settings fits every instance. This is a cache-storage eligibility result: it neither establishes total-device feasibility nor chooses the method with the best task quality. The storage formula is established implementation accounting; the contribution here is its exhaustive application to detect a length-field mismatch and correct the budget decision.

4.3 Complete paired comparisons distinguish dominance from trade-offs

Across all 1,200 matched model-prompt instances, SnapKV stores less cache than KIVI-4 and achieves a higher recorded cached-forward rate. Both inequalities hold with a 10% multiplicative margin and in each of the two source runs separately. The comparison covers every request in all six model-dataset strata. Table 4 summarizes the results by stratum, and the artifact provides all individual comparisons, including CaM timing. The comparison with KIVI-2 reveals a trade-off: KIVI-2 stores less cache than SnapKV but has a lower recorded cached-forward rate on every matched instance. Both inequalities again hold with a 10% multiplicative margin and in both runs. Thus, the same SnapKV configuration dominates KIVI-4 on the two recorded measures, while its higher rate relative to KIVI-2 comes with greater memory use. Figure 2 shows the rate ratios for every matched request, without selecting favorable context buckets.

Prior work already shows that compression need not improve serving performance (Gao et al., 2025; Agrawal & Mayer, 2026). Our contribution is to establish the consistency and margin of the paired ordering under these specific configurations, while making explicit the limits of a deployment claim based on recorded rates. Determining the quality cost requires evaluation on the same prompts at matched memory budgets, which these systems measurements do not provide.

4.4 Method rankings are largely stable across aggregation rules

Grouping the original 165 method-specific cells by model, dataset, context, and outcome yields 45 sets of directly comparable method results. The median, geometric mean, and ratio of means identify the same top method in all 45 cells and preserve the full ranking in 43. Only two cells show changes in lower ranks: Mistral's cached-forward rate on NarrativeQA in the longest document-context bucket, and its reported prefill-plus-first-cached-forward time on Qasper below 4096 document tokens (Figure 3). We retain the original document-context buckets to keep the quantity being estimated unchanged after the length audit.

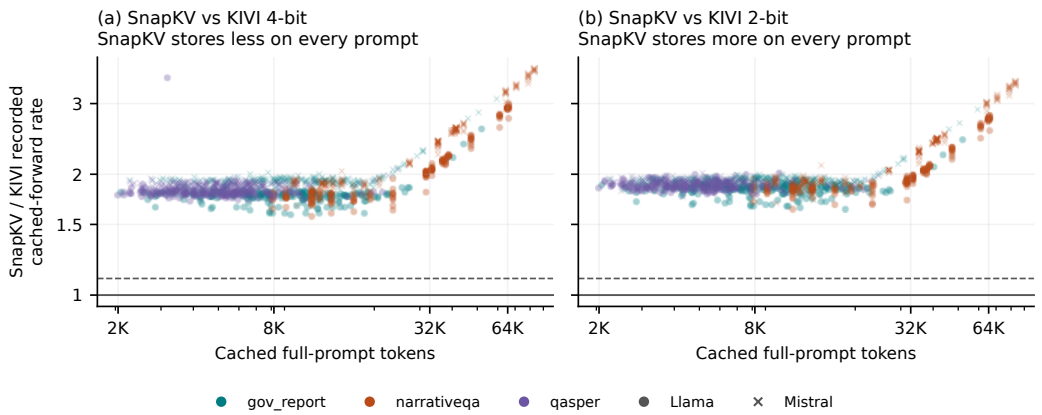


Figure 2: Recorded cached-forward rate ratios for all paired comparisons. Each panel includes 1,200 matched model-prompt instances, with repeated runs averaged within each request. Solid and dashed reference lines mark ratios of 1 and 1.1, respectively. SnapKV stores less cache than KIVI-4 but more than KIVI-2 on every paired request. Cached-forward rates are performance proxies, not measurements of end-to-end throughput; complete generated lengths and joint task-quality measurements are unavailable.

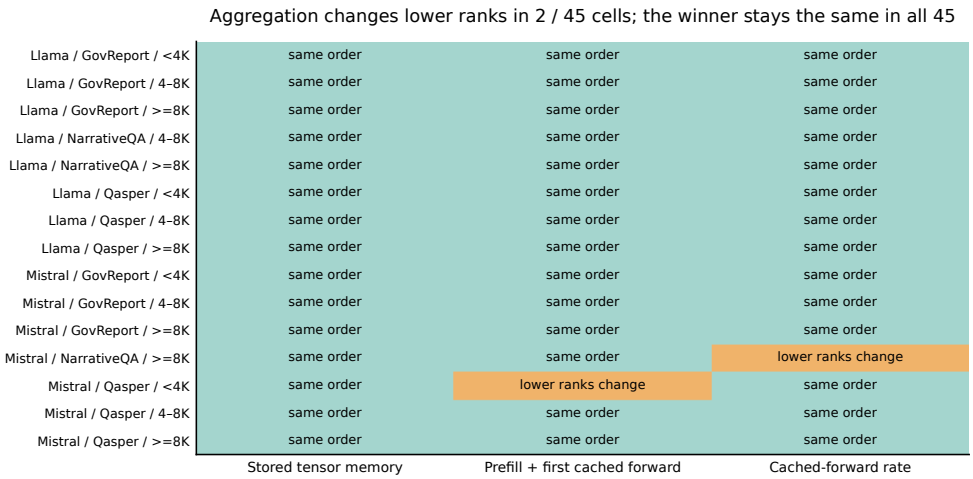


Figure 3: Method-ranking stability under three aggregation rules across 45 comparable cells, using the source's document-context buckets. Green cells preserve the full ranking; orange cells change only lower ranks. The top method is unchanged in every cell. This stability limits claims that aggregation alone makes these rankings unreliable.

In the earlier median-versus-ratio-of-means comparison, only one of the 165 method-specific cells crosses unity, moving from 0.998 to 1.003. The two aggregation rules can nevertheless yield different magnitudes: the largest relative discrepancy is 16.5%. Storage definitions, timed operations, and candidate eligibility can invalidate a conclusion even when rankings are stable across aggregation rules.

4.5 Candidate eligibility can change frontier membership

In the PyramidInfer source, PyramidInfer continues to perform the best out of the four fixed configurations: it uses less KV memory and achieves higher throughput than every other listed configuration. Its throughput ratio is 2.237 relative to Accelerate and 1.487 relative to DeepSpeed. Using

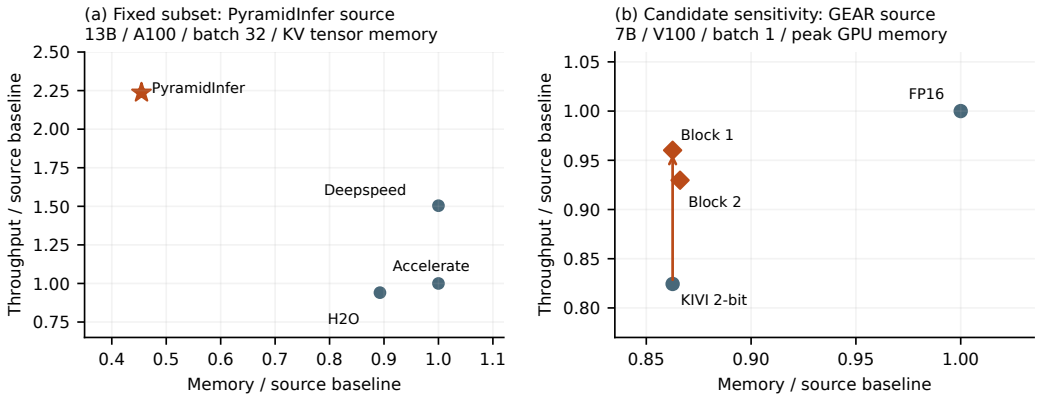


Figure 4: Local frontiers and candidate-set sensitivity. Left: the fixed PyramidInfer source set. Right: both ambiguous GEAR blocks are shown as numerical configurations. Block 1 dominates KIVI; FP16 remains non-dominated in every admission scenario. The panels use different memory definitions and cannot be merged. Neither has a jointly measured quality constraint.

a stronger common throughput baseline changes the magnitude of the reported gain but preserves dominance, because all throughput values are divided by the same baseline (Yang et al., 2024a).

The GEAR source illustrates a different uncertainty: which configurations belong in the comparison. When the batch-one candidate set is restricted to configurations unambiguously labeled FP16 and KIVI, both are non-dominated. The first ambiguously labeled GEAR block reports the same memory use as KIVI, 7.28 GB, but higher throughput: 4.10 versus 3.52 tokens/s. Including this block therefore removes KIVI from the frontier, regardless of the block's unresolved method name. A second ambiguously labeled block reports 7.31 GB and 3.97 tokens/s. We retain the two source-block identities rather than assign unverified algorithm variants (Kang et al., 2024).

Across the four admission scenarios, only FP16 belongs to F_{certain}; KIVI and the ambiguous blocks are possible members in some scenarios. With all recorded blocks admitted, FP16 and Block 1 form the observed frontier. This is not a newly discovered advantage of GEAR: those measurements already appear in its paper. The methodological lesson is that uncertainty about an algorithm label must not be hidden inside a definitive numerical frontier. Candidate-set sensitivity and baseline-magnitude sensitivity are distinct operations with different consequences.

4.6 Decision implications and limitations

Table 1 connects the quantitative findings to deployment constraints. It distinguishes whether a configuration meets the audited requirements from broader, conditional guidance drawn from the literature. The budget and paired-rate results show why the preferred configuration depends on the objective: KIVI-2 uses less stored cache than SnapKV, whereas SnapKV achieves a higher recorded cached-forward rate. Other strategy families address bottlenecks outside the scope of these paired measurements.

For memory-constrained edge inference, quantization and grouped-query attention (GQA) offer complementary ways to reduce cache storage: quantization reduces bytes per KV scalar, while GQA reduces the number of KV heads (Liu et al., 2025c; Ainslie et al., 2023). The audited models already use GQA, so the reported quantization ratios measure savings within that architecture rather than an additional gain from GQA. Section 4.2 also shows why nominal precision is insufficient: the tested four-bit configuration exceeds the 25% stored-cache budget once metadata and residuals are included. Selecting precision therefore requires measuring realized storage, checking kernel support, and evaluating task quality on the target device. For high-throughput batch serving, paged allocation reduces fragmentation, while exact prefix reuse avoids repeated computation across requests (Kwon et al., 2023). These mechanisms preserve the represented KV state, provided prefix reuse is valid for the same model and prefix. Their benefits depend on concurrency and the frequency of reusable prefixes; neither reduces the unique KV payload of every request. They therefore com-

Table 1: Decision matrix for the audited evidence. Eligibility concerns the named measurement only; it does not certify task quality.

Decision	Supported conclusion	Additional evidence needed
Post-prefill KV budget $\leq 20\%$	KIVI-2 fits all 1,200 instances; tested SnapKV and KIVI-4 do not	Peak memory and task-quality tolerance
Post-prefill KV budget $\leq 25\%$	KIVI-2 and SnapKV fit all instances; KIVI-4 fits none	Quality and complete-request performance
Choose by recorded decode rate	SnapKV exceeds both KIVI variants on all matched prompts	Realized output lengths, end-to-end timing, quality
Use the GEAR local frontier	FP16 survives every candidate scenario; KIVI membership is conditional	Resolve ambiguous source-block identities
Recommend by quality and cost	Not identified by these joint systems rows	Co-measured quality, performance and deployment constraints

plement cache compression and should be evaluated alongside it, with throughput measured under the required request-latency limits.

For very long contexts, token eviction and some sparse or streaming attention schemes can limit retained state when the task tolerates restricted access to earlier tokens (Li et al., 2024a; Xiao et al., 2026). However, reducing decode-cache storage does not establish that prefill is feasible or that full-context recall is preserved. When earlier KV state must remain available, offloading offers a capacity option at the cost of data transfers. FlexGen specifically motivates this trade-off for latency-insensitive workloads (Sheng et al., 2023). Even when KV state is preserved or reconstructed, successful retrieval of all task-relevant information remains an empirical question.

5 Conclusion

5.1 Evidence-Based Deployment Recommendations

Through our systematic review and quantitative meta analysis, we found there is no universal "silver bullet" for KV cache optimization. The efficacy of any strategy is directly linked to the specific deployment constraints.

Based on our synthesis, we propose the following framework:

- **Memory Constrained Edge Inference:** For deployments limited by strict VRAM capacity (e.g., mobile devices), aggressive KV cache quantization (e.g., 4-bit) combined with grouped-query attention provides the optimal balance, maximizing context length while tolerating the slight computational overhead.
- **High Throughput Batch Serving:** Datacenter environments prioritizing total throughput should leverage paged memory management and prefix caching. These system level optimizations eliminate fragmentation and redundency without altering the underlying model mathematics, ensuring zero quality degradation.
- **Ultra Long Context Workloads:** Applications requiring massive context windows (e.g., document analysis) benefit most from token eviction and sparse attention, provided the specific downstream task is robust to the loss of intermediate token representations. If full recall is required, host memory or SSD offloading is mandatory, accepting the penalty to inter token latency.

Future research must prioritize standardized evaluation methodologies. The current literature is hampered by isolated benchmarking on disparate hardware. We advocate for the adoption of unified empirical frameworks that report not only theoretical memory savings, but also end to end latency, throughput, and rigorous long context quality metrics.

AI USE STATEMENT

During the preparation of this work, generative AI tools were utilized to assist with text formatting. We affirm that all quantitative data extraction and arguments made were independently conducted and rigorously verified by the authors. The authors bear full responsibility for the contents, findings, and conclusions presented in this paper.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our meta-analysis, we have made our complete screening logs openly available. The calculated within-study ratios, raw extracted metrics, and the Python scripts utilized for the random-effects models and Pareto-frontier generation are provided to help with independent verification and future extensions of this work.

REFERENCES

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S. Gulavani, R. Ramjee, and Alexey Tumanov. Vidur: A Large-Scale Simulation Framework For LLM Inference. *arXiv preprint arXiv:2405.05465*, 2024. URL https://arxiv.org/abs/2405.05465.

Amey Agrawal, Nitin Kedia, Anmol Agarwal, Jayashree Mohan, Nipun Kwatra, Souvik Kundu, R. Ramjee, and Alexey Tumanov. On Evaluating Performance of LLM Inference Serving Systems. *arXiv preprint arXiv:2507.09019*, 2025. URL https://arxiv.org/abs/2507.09019.

Niki Agrawal and Ruben Mayer. Benchmarking KV-Cache Optimizations across Task Quality and System Performance for Long-Context Serving. *arXiv preprint arXiv:2607.05399*, 2026. URL https://arxiv.org/abs/2607.05399.

Ammar Ahmed, Sheng Di, Franck Cappello, Zi-Rui Liu, Jingoo Han, and Ali Anwar. Systematic Evaluation of Optimization Techniques for Long-Context Language Models. *arXiv preprint arXiv:2508.00305*, 2025. URL https://arxiv.org/abs/2508.00305.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, A. Awan, Cheng Li, Duo Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2022.

Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharati Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, M. Emani, and Venkat Vishwanath. LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1362–1379, 2024.

J.-G. Dai, Tian-Ze Deng, Yueying Li, and Tianyi Peng. Throughput-Optimal Scheduling Algorithms for LLM Inference and AI Agents. *arXiv preprint arXiv:2504.07347*, 2025. URL https://arxiv.org/abs/2504.07347.

Tri Dao, Daniel Y. Fu, S. Ermon, A. Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arXiv preprint arXiv:2205.14135*, 2022. URL https://arxiv.org/abs/2205.14135.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the Limits of KV Cache Compression. *arXiv preprint arXiv:2609.19969*, 2026. URL https://arxiv.org/abs/2609.19969.

Wei Gao, Xinyu Zhou, Peng Sun, Tianwei Zhang, and Yonggang Wen. Rethinking key-value cache compression techniques for large language model serving. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/26289c647c6828e862e271ca3c490486-Abstract-Conference.html.

Renato Geh, Alex Chen, Daniel Israel, Aditya Grover, and Guy Van den Broeck. A Probabilistic Interpretation of KV Cache Eviction. *arXiv preprint arXiv:2608.28293*, 2026. URL https://arxiv.org/abs/2608.28293.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. *arXiv preprint arXiv:2401.18079*, 2024. URL https://arxiv.org/abs/2401.18079.

Chin Ting Hsu, Yu-Syuan Xu, Ling Zou, Hsien-Kai Kuo, and Wen-Huang Cheng. ECOKV: Geometry-Aware KV Cache Eviction via Complementary Diversity Metrics. *arXiv preprint arXiv:2609.06663*, 2026. URL https://arxiv.org/abs/2609.06663.

Jiantong Jiang, Peiyu Yang, Rui Zhang, and Feng Liu. Towards Efficient Large Language Model Serving: A Survey on System-Aware KV Cache Optimization. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 38450–38476, 2026. doi: 10.18653/v1/2026.findings-acl.1916. URL https://aclanthology.org/2026.findings-acl.1916/.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. GEAR: An Efficient KV Cache Compression Recipe for Near-Lossless Generative Inference of LLM. *arXiv preprint arXiv:2403.05527*, 2024. URL https://arxiv.org/abs/2403.05527.

Mahsa Khoshnoodi, Vinija Jain, Ming Gao, Malavika Srikanth, and Aman Chadha. A Comprehensive Survey of Accelerated Generation Techniques in Large Language Models. *arXiv preprint arXiv:2405.13019*, 2024. URL https://arxiv.org/abs/2405.13019.

Wonung Kim, Hyunmin Choi, Minsu Kim, Jaehong Cho, Yeongwook Kim, and Jongse Park. Simthesizer: An Agent-Driven Simulation Framework for LLM Serving Systems. *arXiv preprint arXiv:2608.24650*, 2026. URL https://arxiv.org/abs/2608.24650.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, C. Yu, Joseph E. Gonzalez, Haotong Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A Survey on Large Language Model Acceleration based on KV Cache Management. *arXiv preprint arXiv:2412.19442*, 2024a. URL https://arxiv.org/abs/2412.19442.

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. *arXiv preprint arXiv:2404.14469*, 2024b. URL https://arxiv.org/abs/2404.14469.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring Attention with Blockwise Transformers for Near-Infinite Context. *arXiv preprint arXiv:2310.01889*, 2023a. URL https://arxiv.org/abs/2310.01889.

Minghui Liu, Aadi Palnitkar, Tahseen Rabbani, Hyunwoo Jae, Kyle Sang, Dixi Yao, Shayan Shabihi, Fuheng Zhao, Tian Li, Ce Zhang, Furong Huang, and Kun-Peng Zhang. Hold Onto That Thought: Assessing KV Cache Compression On Reasoning. *arXiv preprint arXiv:2512.12008*, 2025a. URL https://arxiv.org/abs/2512.12008.

Xiang Liu, Zhenheng Tang, Hong Chen, Peijie Dong, Zeyu Li, Xiuze Zhou, Bo Li, Xuming Hu, and Xiaowen Chu. Semantic Integrity Matters: Benchmarking and Preserving High-Density Reasoning in KV Cache Compression. *arXiv preprint arXiv:2502.01941*, 2025b. URL https://arxiv.org/abs/2502.01941.

Zeyu Liu, Woomin Song, Xuandi Fu, Sai Muralidhar Jayanthi, Vivek Govindan, Aram Galstyan, Sravan Babu Bodapati, and Srikanth Ronanki. SGD-KV: Summarization Guided KV Cache Compression. In *39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning*, 2025c.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. *arXiv preprint arXiv:2305.17118*, 2023b. URL https://arxiv.org/abs/2305.17118.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. *arXiv preprint arXiv:2402.02750*, 2024. URL https://arxiv.org/abs/2402.02750.

Jiaxun Lu, Xiang Zhang, and Yunfeng Shao. Unified AI Gateway: A Framework for Joint Model Routing and KV Cache Management. *arXiv preprint arXiv:2609.06940*, 2026. URL https://arxiv.org/abs/2609.06940.

Shi Luohe, Hongyi Zhang, Yao Yao, Z. Li, and Zhao Hai. Keep the Cost Down: A Review on Methods to Optimize LLM's KV-Cache Consumption. *arXiv preprint arXiv:2407.18003*, 2024. URL https://arxiv.org/abs/2407.18003.

Xiaoxiao Ma, Kuofeng Gao, Zeyi Lu, Wenxi Jiang, Hao Fang, Hao Wu, Bin Chen, and Shu-Tao Xia. When Efficiency Meets Safety: A Benchmark Security Analysis of KV Cache Compression in Large Language Models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24472–24485, 2026. URL https://aclanthology.org/2026.acl-long.1123/.

Forough Majidi, Mohammad Mehdi Morovati, Foutse Khomh, and Heng Li. LLM Serving in the Wild: An Empirical Study of Frameworks, Methods, and System Designs. *arXiv preprint arXiv:2608.03036*, 2026. URL https://arxiv.org/abs/2608.03036.

Meta. Llama-3.1-8B-Instruct: Model configuration. Hugging Face model repository, 2024. URL https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct/blob/main/config.json. Configuration file: config.json.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems. *arXiv preprint arXiv:2312.15234*, 2023. URL https://arxiv.org/abs/2312.15234.

Mistral AI. Mistral-7B-Instruct-v0.3: Model configuration. Hugging Face model repository, 2024. URL https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3/blob/main/config.json. Configuration file: config.json.

J. Pan and Guoliang Li. A Survey of LLM Inference Systems. *arXiv preprint arXiv:2506.21901*, 2025. URL https://arxiv.org/abs/2506.21901.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132, 2024. URL https://www.microsoft.com/en-us/research/publication/splitwise-efficient-generative-llm-inference-using-phase-splitting/.

Reiner Pope, Sholto Douglas, A. Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, J. Heek, Kefan Xiao, Shivani Agrawal, and J. Dean. Efficiently Scaling Transformer Inference. *arXiv preprint arXiv:2211.05102*, 2022. URL https://arxiv.org/abs/2211.05102.

Ákos Prucs, Nara Csutora, Mátyás Antal, and Márk Marosi. Compute-Accuracy Pareto Frontiers for Open-Source Reasoning Large Language Models. *arXiv preprint arXiv:2512.24776*, 2025. URL https://arxiv.org/abs/2512.24776.

Chen Qiu, Ziwu Liu, Chao Fei, Guozhong Li, and Panos Kalnis. KVDiagnosis: A Diagnostic Benchmark for KV-Cache Compression in Long-Context Language Models. *arXiv preprint arXiv:2608.09412*, 2026. URL https://arxiv.org/abs/2608.09412.

Noam Shazeer. Fast Transformer Decoding: One Write-Head is All You Need. *arXiv preprint arXiv:1911.02150*, 2019. URL https://arxiv.org/abs/1911.02150.

Bin-Qi Shen, Li Jin, Han-Yu Cai, Lang-Xiang Hu, and Yu-Ting Xin. The Efficiency Frontier: A Unified Framework for Cost–Performance Optimization in LLM Context Management. In *2026 International Conference on Large Models and Intelligent Agent Technologies (LMIAT)*, pp. 15–22, 2026.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark W. Barrett, Joseph Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. High-throughput Generative Inference of Large Language Models with a Single GPU. In *International Conference on Machine Learning*, pp. 31094–31116, 2023.

Yi Su, Hong Liu, Guanghua Yu, and Jianchen Zhu. D-Quant: Driftable Entropy Coding for KV Cache Quantization. *arXiv preprint arXiv:2609.19880*, 2026. URL https://arxiv.org/abs/2609.19880.

Si-Yuan Sun, Jin-Ling Yu, Han Liu, Hanyu Guo, Yang Cao, Shouhua Zhang, and Jie-Han Zhou. Optimizing Attention for Efficient LLM Inference: A Review. In *2025 8th World Conference on Computing and Communication Technologies (WCCCT)*, pp. 482–491, 2025.

Kaiyuan Tian, Linbo Qiao, Baihui Liu, Gongqingjian Jiang, Shan-Shan Li, and Dongsheng Li. A survey on memory-efficient transformer-based model training in AI for science. *Frontiers of Computer Science*, 20, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and I. Polosukhin. Attention is All you Need. In *Neural Information Processing Systems*, pp. 5998–6008, 2017.

Heng Wang, Jielin Qiu, Wenting Zhao, Cheng Qian, Liangwei Yang, Jiawei Han, Heng Ji, Silvio Savarese, Shelby Heinecke, and Huan Wang. Random Attention: Rethinking KV Cache Eviction for Efficient Reasoning. *arXiv preprint arXiv:2609.03430*, 2026a. URL https://arxiv.org/abs/2609.03430.

Michael Wang, Keith Li, and Roozbeh Bostandoost. MetaKV: Adaptive KV Cache Compression for Constrained LLM Inference. *arXiv preprint arXiv:2609.07966*, 2026b. URL https://arxiv.org/abs/2609.07966.

Michael Wang, Yikang Yue, Shaobo Li, Yirui Eric Zhou, Chen Wang, and Jian Huang. Benchmarking LLM Serving Systems for Agentic AI Workloads with XPerf. *arXiv preprint arXiv:2608.20370*, 2026c. URL https://arxiv.org/abs/2608.20370.

Bing-Ya Wu, Sheng-Yu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 2024.

Linhui Xiao, Guiping Cao, Mingyue Guo, Xianchao Guan, Fan Yang, Ming Tao, X. Li, Yuxing Peng, and Yaowei Wang. A Survey on the Green Development of Large Models: From Resource-Efficient Architectures to Hardware-Software Co-Design. *arXiv preprint arXiv:2607.09084*, 2026. URL https://arxiv.org/abs/2607.09084.

Yichun Xu, Navjot K. Khaira, and Tejinder Singh. KV Cache Optimization Strategies for Scalable and Efficient LLM Inference. *arXiv preprint arXiv:2603.20397*, 2026. URL https://arxiv.org/abs/2603.20397.

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. PyramidInfer: Pyramid KV Cache Compression for High-throughput LLM Inference. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3258–3270. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.findings-acl.195. URL https://aclanthology.org/2024.findings-acl.195/.

Fan Yang, Xinhao Yang, Hongyi Wang, Zehao Wang, Zhenhua Zhu, Shulin Zeng, and Yu Wang. GLITCHES: GPU-FPGA LLM Inference Through a Collaborative Heterogeneous System. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, 2024b.

Jiaming Yang, Chenwei Tang, Liangli Zhen, Chenyang Zhang, and Jiancheng Lv. Jacap: Robust KV Cache Eviction via Jacobian-Based Nonlinear Information Capacity Preservation. *arXiv preprint arXiv:2609.08131*, 2026. URL https://arxiv.org/abs/2609.08131.

Suho Yoo, Hyunjong Ok, Jongmin Choi, Jihoo Jung, and Joon Son Chung. OmniKVQuant: KV Cache Quantization for Omni-LLMs. *arXiv preprint arXiv:2609.11582*, 2026. URL https://arxiv.org/abs/2609.11582.

Michael J. Zellinger and Matt Thomson. Economic Evaluation of LLMs. *arXiv preprint arXiv:2507.03834*, 2025. URL https://arxiv.org/abs/2507.03834.

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. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

Ranran Zhen, Juntao Li, Yixin Ji, Zhenlin Yang, Tong Liu, Qingrong Xia, Xinyu Duan, Zhefeng Wang, Baoxing Huai, and Min Zhang. Taming the Titans: A Survey of Efficient LLM Inference Serving. *arXiv preprint arXiv:2504.19720*, 2025. URL https://arxiv.org/abs/2504.19720.

Yinmin Zhong, Sheng-Yu Liu, Junda Chen, Jianbo Hu, Yi-Bo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *USENIX Symposium on Operating Systems Design and Implementation*, pp. 193–210, 2024.

Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. TraceLab: Characterizing Coding Agent Workloads for LLM Serving. *arXiv preprint arXiv:2606.30560*, 2026. URL https://arxiv.org/abs/2606.30560.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A Survey on Model Compression for Large Language Models. *Transactions of the Association for Computational Linguistics*, 12: 1556–1577, 2024. doi: 10.1162/tacl_a_00704. URL https://aclanthology.org/2024.tacl-1.85/.

A Coverage, source selection, and measurement definitions

Initial queries included "KV cache quantization KIVI paper throughput peak memory table" and "KV cache compression SnapKV H2O performance memory throughput paper." Broader queries combined quantization, eviction, compression, paging, offloading, prefix caching, sparse attention, grouped queries, retrieval, and benchmark terms. Versions are linked by paper ID and PDF SHA-256.

Table 2: Core mechanism taxonomy.

ID	Strategy family
1	KV-cache quantization and mixed precision
2	Token eviction or pruning
3	Cache compression, merging, or low-rank approximation
4	Paged or block-based memory management
5	CPU, SSD, distributed, or host-memory offloading
6	Prefix caching and cross-request reuse
7	Sliding-window, sparse, streaming, or recurrent attention
8	Multi-query or grouped-query attention
9	Retrieval-based or selective KV-cache reconstruction

Table 3: Targeted source corpus. Family IDs refer to Table 2; multiple IDs can denote distinct evaluated methods.

Source	Year	Family IDs	Numerical
KIVI	2024	1	Yes
KVQuant	2024	1	Yes
SnapKV	2024	2	Yes
H2O	2023	2	Yes
GQA	2023	8	Yes
PagedAttention	2023	4	No
FlexGen	2023	1, 5	No
StreamingLLM	2024	7	No
Quest	2024	9	No
SGLang	2024	6	No
CaM	2024	2, 3, 7	Yes
PyramidInfer	2024	2	Yes
Yuan et al. benchmark	2024	1, 2, 7, 9	Yes
Agrawal-Mayer benchmark	2026	1, 2, 3	Yes
RocketKV	2025	2, 9	No
GEAR	2024	1, 3	Yes
PagedEviction	2026	2, 4	No

The 56 excluded published observations comprise three undefined contrasts with out-of-memory baselines, 15 rows without same-batch baselines, and 38 rows with duplicated ambiguous GEAR labels. Full raw values, missing fields, exclusion reasons, and source locations remain in `evidence/data/published_observations.csv`.

The benchmark field `cache_size_mb` counts tensor elements times element size and divides by 2^{20}, so we use MiB. It is not peak allocator memory. KIVI/SnapKV measure it after prefill; CaM measures it after generation. The incremental allocation field has a different meaning and is excluded from isolated-cache comparisons. The reported timing field includes a first cached forward after the first token has been chosen; cached-forward rate excludes parts of complete token generation. TurboQuant's decoder-only timing and hypothetical packed-storage accounting are excluded rather than mixed with the retained definitions. Copied source code and a source audit ledger accompany the release.

B Accounting derivation

For a quantized group of g scalars, packed payload requires $bg/8$ bytes. One FP16 scale and one FP16 minimum add four bytes. Relative to $2g$ FP16 bytes, the fraction is $b/16 + 2/g$. For keys,

quantization proceeds over complete residual blocks; values retain the most recent R positions. Separating the two residual contributions gives Equation 2. The examined heads have dimensions compatible with packing, and the stored tensors retain the original KV heads rather than the repeated attention heads. This derivation describes the audited implementation; alternative metadata precision, group size, residual policy, or packing changes the expression.

At batch one, each cached token contributes $2 \times 32 \times 8 \times 128 \times 2 = 131072$ FP16 bytes, or 0.125 MiB. The baseline cache field therefore supplies a candidate full-prompt length without fitting the quantized result. Applying that length to the independently recorded KIVI bytes yields the reported consistency check. The analysis initially used the document-length field and failed to obtain exact agreement; both predictions and their discrepancies are preserved. The prompt construction routine explains why the document count omits instructions and question tokens. The full-prompt inference does not establish the number of generated tokens, which remains unavailable.

All 1,200 KIVI-2 configurations meet a 20% stored-cache budget, whereas the tested KIVI-4 and SnapKV configurations do not. At 25%, both KIVI-2 and SnapKV meet the budget for every instance. A hard bound on KV tensors does not constrain weights, temporary buffers, allocator reservation, host storage, or peak GPU use. Such quantities require distinct measurements.

C Complete paired and aggregation results

Table 4: All six model-dataset strata for SnapKV relative to KIVI-4. Values are medians of paired request ratios; throughput is the recorded cached-forward proxy.

Model	Dataset	n	Memory ratio	Rate ratio
Llama	gov_report	200	0.795	1.785
Llama	narrativeqa	200	0.799	2.016
Llama	qasper	200	0.791	1.808
Mistral	gov_report	200	0.796	1.938
Mistral	narrativeqa	200	0.799	2.426
Mistral	qasper	200	0.792	1.895

`paired_requests.csv` contains all 7,200 pairwise rows, identified by model, dataset, sample and method pair. `paired_strata.csv` contains all 36 strata at each of four sensitivity margins. `run_sensitivity.csv` repeats the three memory-compatible pair comparisons in both runs. These are descriptive robustness checks over the recorded artifact. We do not attach a binomial confidence interval to 1,200 correlated model-prompt instances or treat 4,800 storage rows as independent experimental replications.

All 45 ranking cells are in `ranking_stability.csv`, with their three explicit orders in `rankings.csv`. Methods are ranked by smaller memory/time or larger recorded rate. Context buckets remain those of the original document-length field for this particular comparison. The paired main results use all prompts and do not depend on a bucket choice. The 20 affected bucket assignments and their inferred full-prompt lengths are individually listed in `context_audit.csv`.

D Frontier sensitivity

Let a source group contain mandatory configurations C and optional source-block IDs U. For every subset $V \subseteq U$, calculate $F(C \cup V)$ using jointly observed memory and throughput. Intersecting these sets yields configurations that survive all enumerated admission decisions; taking their union yields configurations that survive at least one. Optional configurations cannot be certain because they are absent from a permitted scenario. A mandatory point dominated by a mandatory point cannot be possible.

For GEAR, the optional blocks have coordinates $(7.28/8.44, 4.10/4.27)$ and $(7.31/8.44, 3.97/4.27)$. The first dominates KIVI's $(7.28/8.44, 3.52/4.27)$ and the second block. FP16 remains non-dominated because it has higher throughput than every compressed point, at higher memory. Removing an uncertain method label does not erase the underlying measured

configuration. Admission scenarios carry no probability, and the resulting sets do not bound unobserved algorithms or alternative implementations.

Finally, replacing positive baseline denominators rescales each axis by a common positive constant. Such transformations preserve both weak inequalities and at least one strict inequality, so dominance is unchanged.

E REPRODUCTION AND VERIFICATION

The project contains the original extraction under `evidence/` and the additional analysis under `overhaul/`. Run `python evidence/analysis.py`, then `python overhaul/analyze.py` and `python overhaul/render.py`. The first two analyses use the Python standard library; rendering requires Matplotlib and NumPy. Tests exercise accounting on independent synthetic layouts, pairing, full-denominator result checks, candidate enumeration, and baseline invariance. Original extracted values are preserved. Generated macros and plot/table inputs link the main numerical claims to outputs. The analysis plan documents exploratory amendments and the verification report distinguishes computational replay from source verification and GPU replication.