KV CACHE OFFLOADING FOR CONTEXT-INTENSIVE TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

With the growing demand for long-context LLMs across a wide range of applications, the key–value (KV) cache has become a critical bottleneck for both latency and memory usage. Recently, KV-cache offloading has emerged as a promising approach to reduce memory footprint and inference latency while preserving accuracy. Most recent works on KV offloading evaluate on tasks that do not require extracting large amounts of information from the context. In this work, we study KV-cache offloading on context-intensive tasks: problems where the solution requires looking up a lot of information from the input prompt. We create and release the Text2JSON benchmark, a highly context-intensive task that requires extracting structured knowledge from raw text. We evaluate modern KV offloading on Text2JSON and other context-intensive tasks and find significant performance degradation on both Llama 3 and Qwen3 models. Our analysis identifies two key reasons for poor accuracy: low-rank projection of keys and unreliable selection, and proposes a simpler alternative strategy that significantly improves accuracy across multiple LLM families and benchmarks. These findings highlight the need for a comprehensive and rigorous evaluation of long-context compression techniques.

1 INTRODUCTION

Large Language Models (LLMs) are becoming increasingly adept at handling long-context inputs, enabling applications such as long-document summarization (Bai et al., 2024; Wang et al., 2024), large-scale codebase analysis (Liu et al., 2024a; Luo et al., 2024), and agentic workflows, and scaling to even longer tasks (Kwa et al., 2025). However, processing long contexts remains challenging because the key–value (KV) cache grows linearly with sequence length. For sufficiently long sequences, the KV cache can require more memory than the model parameters themselves (Hooper et al., 2024). Moreover, as the KV cache grows in size, it often becomes the primary bottleneck for LLM inference. Since KV cache size scales with context length, long-context inference can fit fewer sequences in device memory, drastically reducing throughput (Zhang et al., 2023; Sheng et al., 2023).

The two most popular ways to reduce KV cache footprint are quantization (Hooper et al., 2024; Liu et al., 2024b; Shutova et al., 2025; Ashkboos et al., 2024) and cache eviction (or pruning) (Xiao et al., 2024; Zhang et al., 2023; Li et al., 2024c; Tang et al., 2024). KV quantization has already gained significant popularity among practitioners (vLLM Project, 2026; NVIDIA, 2026). In turn, KV eviction not as commonly used in industrial deployments (Li et al., 2026). Several recent works attribute this to the fact that pruning KV entries can lead to significant performance drops for some tasks (Chen et al., 2026; Ananthanarayanan et al., 2026). For instance, KV eviction struggles with problems that require processing most of the input tokens: sorting a document collection, translating a book verbatim or refactoring a codebase.

A promising alternative to pruning is KV-cache offloading (Aminabadi et al., 2022; Sheng et al., 2023): instead of removing KV entries permanently, these techniques move the original entries to cheaper system memory and load them back when necessary. To avoid loading the full KV cache, they use method-specific heuristics to predict which KV entries will be used at every inference step (Lee et al., 2024; Chen et al., 2024; Sun et al., 2025; Li et al., 2025b; Yang et al., 2026), reporting little to no performance degradation compared to full attention. However, evaluations in these studies have largely been restricted to widely used long-context benchmarks such as RULER (Hsieh et al., 2024) and LongBench (Bai et al., 2024). Though popular, these benchmarks may not capture the full range of real-world applications. As a result, performance drops on practically relevant tasks may go unnoticed and could be non-negligible.

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.

In this work, we systematically evaluate KV-cache offloading techniques across more difficult workloads and identify one prominent failure mode: *context-intensive tasks*. Namely, we find that KV offloading methods struggle with problems that require looking up a lot of contextual information, even if the problem itself is not difficult. We identify several context-intensive subtasks in existing benchmarks that are not common in KV offloading research. Additionally, we create a new dataset, Text2JSON, based on the real-world task of extracting structured information from documents.

We systematically evaluate KV offloading methods on context-intensive tasks and observe significant accuracy drops. Further analysis attributes this to three main problems: 1) suboptimal key compression, 2) inaccurate selection criteria and 3) key/value grouping. Curiously, these components do not hinder accuracy on needle-in-a-haystack tasks, but become problematic when the LLM needs to retrieve much more information. Our main contributions can be summarized as follows:

1. We identify context-intensive subtasks in existing benchmarks and create a new Text2JSON dataset to evaluate KV offloading on real-world context-intensive tasks without confounding factors.
2. We evaluate several modern KV offloading methods across two LLM families (Llama and Qwen3) and find consistent accuracy drops on context-intensive problems.
3. We analyze the failure modes of KV offloading and find that the accuracy loss is caused by key selection strategies, not KV offloading in principle. Crucially, the same optimizations do not harm accuracy on non-intensive tasks such as RULER, highlighting a gap for future evaluations.
4. Using our findings, we formulate Yet Another KV Offloading[1] algorithm that replaces the problematic components with simple data-free quantization. The resulting algorithm achieves near-lossless accuracy on context-intensive tasks while maintaining high inference throughput.

2 Background

Transformer-based LLMs store token representations in a key-value (KV) cache, whose memory cost grows with context length (Vaswani et al., 2017). Quantization reduces the precision of cached vectors (Liu et al., 2024b; Hooper et al., 2024), while eviction permanently removes entries (Xiao et al., 2024; Zhang et al., 2023). Offloading instead keeps entries in system memory and retrieves selected KVs for each decoding step (Aminabadi et al., 2022; Sheng et al., 2023). It can therefore access information that permanent eviction would lose, provided its selection mechanism identifies the required entries. Modern systems select entries through compressed keys or grouped approximations (landmarks), using SVD, group means, or specialized decompositions (Lee et al., 2024; Sun et al., 2025; Chen et al., 2024; Li et al., 2025b). We investigate when these approximations fail and how quantization-based selection can improve them. We provide extended background and overview other KV compression strategies such as quantization and eviction in Appendix A.

3 Context-Intensive Tasks

The core intuition of our work is that long-context tasks vary in how much information they need to extract in order to solve the problem correctly. On one end of the spectrum Needle in the Haystack (NIAH) (Hsieh et al., 2024) requires the model to look up a single correct substring (needle) in a long prompt. On the other end, there are real-world problems that require looking up and cross-referencing many interdependent "needles". Such context-intensive tasks include context-aware document translation, programming in a pre-existing codebase, knowledge extraction, legal case analysis, and others.

Most popular benchmarks used to evaluate KV offloading are not context-intensive. The RULER benchmark (Hsieh et al., 2024), for example, consists of synthetic needle-in-a-haystack (Kamradt, 2023) tasks where the model only needs to extract a single (or very few) text "needles" from long context. More advanced NIAH benchmarks (Bianchi et al., 2025; Chen et al., 2021; Hengle et al., 2025; Wu et al., 2025a; Yuan et al., 2025b) introduce more challenging domains or distractors to make finding the correct "needle" harder; however, the total amount of information required to solve each problem remains small. Similarly, LongBench (Bai et al., 2024) aggregates 21 tasks with heterogeneous semantics, including single- and multi-document question answering, synthetic counting tasks, and code completion. Only a small subset of these tasks is relatively context-intensive (e.g., passage_count), and these tasks are both underrepresented and often excluded from evaluation protocols (Chen et al., 2024; Sun et al., 2025).

[1] YAKV Offloading

Table 1: Long-context benchmarks in terms of context intensity, scenario type and use in KV eviction, selection and offloading evaluation; ✓/✗ means only some subsets in the benchmark apply.

Benchmark Title	Context Intensity	Realistic Scenario	Used in Cache Eviction	Used in Sparse Attention	Used in KV Offloading
NIAH (Kamradt, 2023)	Low	✗	✓	✓	✓
RULER (Hsieh et al., 2024)	Low	✗	✓	✓	✓
Scrolls (Shaham et al., 2022)	Low	✓	✓	✓	✓/✗
LongBench (Bai et al., 2024; 2025)	Low	✓/✗	✓/✗	✓/✗	✓/✗
SCBench (Li et al., 2025c)	Low	✓/✗	✓	✓	✗
∞Bench (Zhang et al., 2024b)	Low	✓	✓	✓	✗
BAMBOO (Dong et al., 2024)	Low	✓/✗	✗	✗	✗
L-eval (An et al., 2024)	Low	✓	✗	✗	✗
BABILong (Kuratov et al., 2024)	Medium	✗	✗	✗	✗
Marathon (Zhang et al., 2024a)	Medium	✓	✗	✗	✗
LooGLE v2 (Li et al., 2024a; He et al., 2025)	Medium	✓	✗	✗	✗
NeedleBench v2 (Li et al., 2025a)	High	✗	✗	✗	✗
LongProc (Ye et al., 2025)	High	✗	✗	✗	✗
Loong (Wang et al., 2024)	High	✓	✗	✗	✗

We collate long-context benchmarks in Table 1. For simplicity, we consider a benchmark highly context-intensive if it requires at least 10 "needles" (substrings) to solve the problem correctly, on average using the same aggregation as the benchmark itself. Conversely, 3-10 needles are medium context intensity and less than 3 needles (e.g. NIAH) are low context intensity. For each benchmark, we check if it is used for evaluating post-training KV eviction (Zhang et al., 2023; Xiao et al., 2024; Cai et al., 2025; Li et al., 2024c; Kim et al., 2025), sparse attention (Jiang et al., 2024; Tang et al., 2024; Singhania et al., 2024; Lin et al., 2025; Zhang et al., 2025b), and KV offloading (Lee et al., 2024; Chen et al., 2024; Sun et al., 2025; Li et al., 2025b; Yang et al., 2026). To summarize, **post-training KV offloading methods evaluate almost exclusively on low context-intensity benchmarks**.

Higher context-intensity datasets do exist, typically as subsets of larger benchmarks (Li et al., 2025a; Wang et al., 2024; Wei et al., 2025; Ye et al., 2025), but they are not commonly used in KV offloading literature. We list several such benchmarks below:

- **MultiNeedle 128K:** NeedleBench v2 (Li et al., 2025a) includes a subset of synthetic retrieval tasks involving multiple independent "needles" or a connected dependency structure (e.g., a family tree). These tasks are simple and artificial, but they are more context-intensive than most NIAH benchmarks (Kamradt, 2023; Hsieh et al., 2024). Despite its simplicity, many offloading methods struggle with MultiNeedle (see Appendix B).
- **LongProc (Ye et al., 2025) HTML to TSV 8K** is a subset of 120 synthetic problems where the LLM must convert structured HTML into a TSV table. These tasks were procedurally generated using websites sourced from Arborist (Li et al., 2024b) and can largely be solved with a rules-based HTML parser, but still prove counterintuitively difficult for LLMs with KV offloading.
- **Loong** (Wang et al., 2024) contains multi-document question-answering tasks where each document contributes relevant information needed to produce the answer, making them more context-intensive than traditional document QA (Bai et al., 2024). Unlike MultiNeedle, Loong offers more realistic evaluation scenarios. However, for our purposes, Loong conflates context intensity with special domain knowledge and multilinguality. Additionally, it uses costly LLM-as-a-Judge (Zheng et al., 2023) evaluation. For this reason, we analyze other benchmarks first and use Loong for verification.

Note also that these benchmarks were not built for evaluating KV offloading or eviction: NeedleBench v2 is a NIAH-type benchmark for general LLM capabilities; LongProc is a long procedural generation benchmark where only 1 out of 17 tasks has inputs longer than 16K tokens on average. On the other hand, Loong contains real-world multi-document QA tasks that are highly context-intensive, but they require domain-specific scientific and financial knowledge, English-Chinese multilinguality and reasoning. This makes it difficult to decouple the effects of KV offloading. To address these limitations, we collect a new Text2JSON dataset to augment these three in our analysis.

3.1 Text2JSON: Multiple Needles in the Wild

To decouple context-intensity from other LLM capabilities, we gather a new dataset of context-intensive problems inspired by a popular production use case: extracting structured knowledge from unstructured text (Yates et al., 2007; Fader et al., 2011; Cui et al., 2018) using LLMs (Josifoski et al., 2022; Dagdelen et al., 2024; Xu et al., 2024). This covers a broad range of applications in search engines, chat personalization, social media, legal document analysis, and others. While not the most "exciting" use case, knowledge extraction has seen wide practical use (Goel, 2026; 567-labs, 2026; Shcherbak, 2025). For our Text2JSON benchmark, we gather 500 samples from across 4 data extraction tasks: product specifications, medical professionals, organization records and movies.

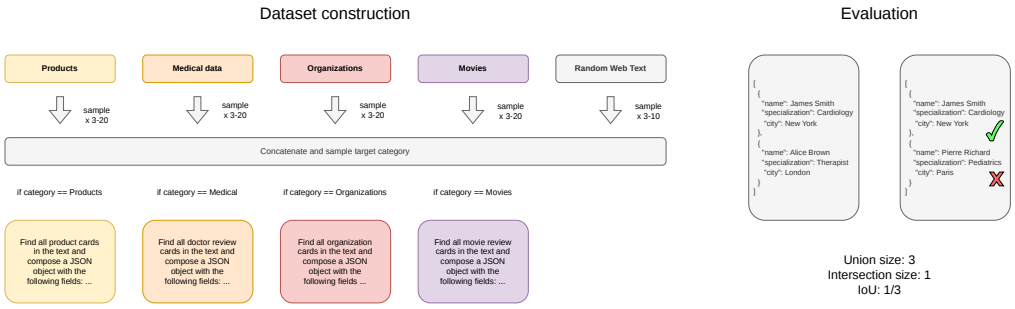


Figure 1: A high-level summary of Text2JSON dataset composition and evaluation.

To preserve privacy, the real user data is replaced with LLM-generated facts that mimic real-world structure (with privacy moderation). The problems vary between 10.0K and 63.5K tokens (gpt2) with an average length of 20.1K. We deliberately chose long but not extreme context lengths so we decouple context intensity from raw length. The ground-truth JSON answer contains 3-20 entries (info dictionaries), each with multiple keys and values. To facilitate reproducibility, we avoid LLM-as-a-Judge (Zheng et al., 2023): instead, we compare each individual item via exact match and report IoU (Intersection over Union) accuracy. Details on dataset curation and evaluation are in Appendix C.

3.2 Takeaways and Yet Another KV Offloading

In this work, we systematically analyze KV offloading algorithms on Text2JSON as well as three other context-intensive subtasks from above: MultiNeedle (from NeedleBench v2 Li et al., 2025a), HTML to TSV 8K (from LongProc Ye et al., 2025) and the Loong benchmark (Wang et al., 2024). To make our approach easier to follow, we summarize our main takeaways here and provide detailed results in Sections 4.1 through 4.4. To make our approach easier to follow, we summarize our main takeaways here and provide detailed results in Sections 4.1–4.3.

Takeaway A: previously sufficient key compression becomes inaccurate. Modern KV offloading methods use low-rank key compression. ShadowKV (Sun et al., 2025) compresses attention keys using truncated SVD (recommended rank 160 layer-wide), which was sufficient for low context-intensity tasks they evaluate on. However, when this is applied to context-intensive tasks, we found that the LLM can no longer reliably select which information it needs, causing significant accuracy drop (see Section 4.1). In turn, InfiniGen (Lee et al., 2024) and LRQK (Li et al., 2025b) keep the original keys, but propose specialized decompositions to select which keys are loaded: this leads to missing the necessary keys on context-intensive tasks.

Takeaway B: Inaccurate group landmarks. ShadowKV (Sun et al., 2025) and ArkVale (Chen et al., 2024) split KV vectors into groups of 8-32 consecutive tokens, similarly to how it is done in GPU sparse attention. Each group is represented with a "landmark" — either the channel-wise average of keys in ShadowKV or a cuboid digest approximation in ArkVale. During inference, the algorithm keeps landmarks on GPU and uses them to determine which groups should be loaded from RAM. Similarly to above, this approximation proved sufficient for NIAH-like tasks. However, as we analyze higher context-intensity tasks, this strategy caused many false positives, resulting in unnecessary token loads and reduced accuracy.

YAKV Offloading. To evaluate the practical impact of our analysis, we combine both takeaways into a simple KV offloading algorithm. YAKV starts from ShadowKV (Sun et al., 2025) and simplifies it:

- **No key SVD:** we use data-free HIGGS quantization (Malinovskii et al., 2025) (4-bit, d=2,n=256).
- **No grouping / landmarks:** we avoid key groups altogether. Instead, we keep extremely quantized key approximations (HIGGS 1-2bit) on GPU to select which KV pairs are loaded from RAM.
- **No outliers:** YAKV does not need to store ShadowKV-like outliers since it has no landmarks.
- **Streamlined offloading:** we offload both keys and values into system memory without prefetching.

YAKV is designed as a simple baseline that combines takeaways A and B with popular industry practices (e.g. quantization instead of SVD). We deliberately keep YAKV minimal to demonstrate that simple KV offloading with our takeaways outperforms more sophisticated methods on context-intensive tasks. That said, we offer simple strategies to further improve it in Appendices F & G.

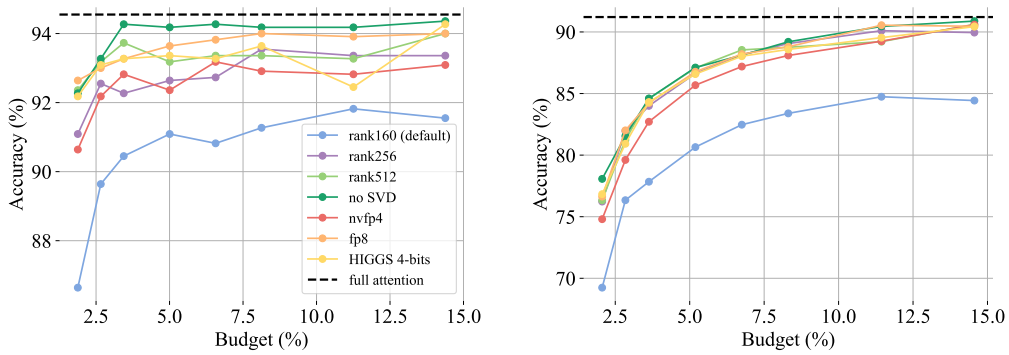


Figure 2: Evaluation of ShadowKV offloading with different KV compression strategies for Llama-3.1-8B-Instruct on MultiNeedle (left) and Qwen3-4B-Instruct-2507 on Text2JSON (right). The X axis denotes the total percentage of tokens loaded (sparse, outlier and local tokens), see Section 4.1.

4 Analysis & Evaluation

Modern KV offloading systems (Lee et al., 2024; Sun et al., 2025; Chen et al., 2024; Li et al., 2025b) have multiple interdependent components: landmarks, compression, decomposition, separate buffers for outliers and recent tokens, prefetching, and others. Furthermore, their behavior can vary between model and task pairs. To keep our results organized, we analyze individual components in Sections 4.1—4.3, then generalize to all algorithm-benchmark combinations in Section 4.4 and evaluate practical inference throughput in Section 4.5.

4.1 Analyzing Key Compression & Alternatives

KV offloading algorithms often include key and/or value compression to speed up host-to-device transfer (Sheng et al., 2023; Sun et al., 2025; Yang et al., 2026). Notably, ShadowKV uses an SVD-based decomposition to compress attention keys across heads within one layer & token, which leads to almost no accuracy degradation on low context-intensity tasks (Sun et al., 2025). Our preliminary experiments (Appendix B) suggest that this compression degrades performance on context-intensive tasks. To analyze this further, we evaluate two models from popular families: Llama-3.1-8B-Instruct (Dubey et al., 2024) and Qwen3-4B-Instruct-2507 (Yang et al., 2025a), with additional models in Appendix D. We use two datasets: the established NeedleBench V2 benchmark (Li et al., 2025a) in the MultiNeedle Retrieval 128K setting, and our Text2JSON dataset introduced in Section 3.1. We use the following KV-cache compression schemes on top of ShadowKV[2]:

- **Truncated SVD:** the original ShadowKV uses rank 160; we additionally report ranks 256 & 512.
- **Uncompressed:** removing key compression from ShadowKV, at the cost of slower inference.
- **Quantization:** replacing SVD with KV-cache quantization using FP8, NVFP4, or HIGGS-4bit.

For quantization, we consider three popular KV-cache compression schemes. FP8 and NVFP4 are compute-oriented quantization formats that are already widely used in deep learning (Micikevicius et al., 2022) and have also been applied to KV-cache compression (Qiao et al., 2025; NVIDIA, 2025). For FP8, we use the E4M3 format. For NVFP4, we follow the protocol of (Egiazarian et al., 2026); note that NVFP4 uses micro-scales and averages 4.5 bits per value. HIGGS, by contrast, is a memory-oriented scheme (Malinovskii et al., 2025; Shutova et al., 2025) that combines vector quantization with random Hadamard transform; we use a grid with $d = 2$ and $n = 256$, which averages 4.02 bits per value. Note that certain methods (e.g. LRQK Li et al., 2025b) use alternative key decompositions for key selection, but keep the original keys for evaluation. We analyze these selection methods in Section 4.3.

The results in Figure 2 clearly show that **the default SVD setting is insufficient** to match the performance of full attention, even when we load $10\times$ as many tokens. We attribute this to the greater retrieval difficulty of our context-intensive tasks, where the model must repeatedly attend to correct tokens from the prompt. In this regime, coarse key compression is more likely to introduce retrieval errors. This is not a fundamental limitation of SVD: higher ranks (e.g., 512) perform better. However, at that point, compressing $8\times$ 128-dimensional keys (1024 total dimensions) using two projection matrices yields a worse compression ratio than FP8. HIGGS provides a better memory–accuracy trade-off than low-rank SVD, further suggesting that quantization is better suited than aggressive low-rank compression for preserving retrieval quality in these tasks. We provide additional evaluations in Appendix D. Based on this observation, we disable SVD in subsequent analysis (Sections 4.2—4.3).

[2]We use the official ShadowKV codebase `https://github.com/ByteDance-Seed/ShadowKV`

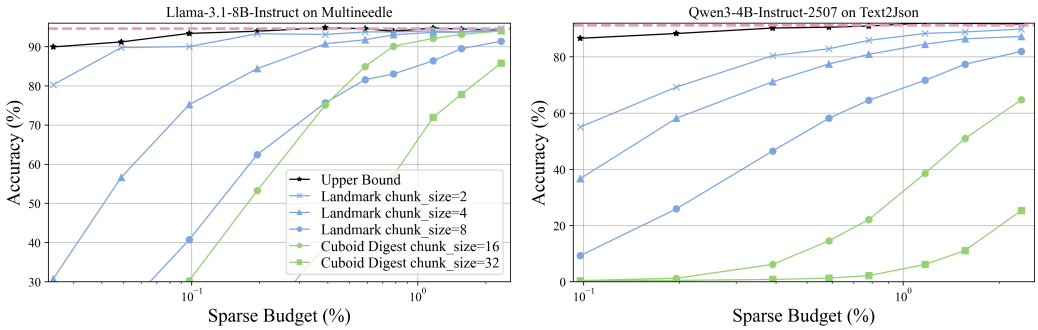


Figure 3: Evaluation of ShadowKV (w/o SVD) landmarks and ArkVale digests for Section 4.2: **(left)** Llama-3.1-8B-Instruct on MultiNeedle-128K, and **(right)** Qwen3-4B-Instruct-2507 on Text2JSON.

4.2 Analyzing Key Grouping, Landmarks & Budgets

Without SVD compression, the models can achieve near-lossless accuracy after loading enough tokens from system memory. However, they need to load significantly more tokens to match full attention accuracy: 5-10% in Figure 2, compared to the ShadowKV default budget of 1.56% (Sun et al., 2025). There are two possible explanations: either **A)** the model fundamentally needs to load more KVs per query or **B)** the offloading algorithms cannot determine which KVs need to be loaded.

To determine which is the case, we analyze how existing methods select KVs for loading and compare them against an "oracle" that always selects the keys with the highest true dot product. Note that the "oracle" is not an efficient KV selection algorithm: we use it exclusively as an upper bound on KV selection. Both ShadowKV and ArkVale use a similar procedure to select which KVs are loaded from RAM: they segment their KV entries into groups (or pages) and compute one "landmark" (or digest) per group to determine when it is needed. ShadowKV uses groups of 8 consecutive tokens and summarizes them with a "landmark" — a channel-wise average of keys in that chunk, stored on GPU. The algorithm then computes the dot product between queries and landmarks and loads the highest scoring chunks to GPU. ArkVale uses a slightly more complex bounding-cuboid approximation (digest) that captures how much the keys differ in each consecutive group.

We compare KV selection landmarks from ShadowKV (default group size 8) and ArkVale (default group size 16-32) on top of the same configuration as in Section 4.1. The results in Figure 3 demonstrate that both types of landmarks are inaccurate compared to the "oracle" KV selection and need significantly more KVs to match full attention accuracy, even with reduced group sizes (2-4). Similarly to Section 4.1, this problem does not affect low context-intensity tasks nearly as much.

To better isolate the source of this inaccuracy, we test two related components: group outliers and local buffer. In ShadowKV, tokens that do not fit their groups well are called "outliers" stored separately (e.g. attention sinks (Xiao et al., 2024)). Additionally, most KV offloading methods treat several most recent tokens this way ("local budget"). In Figure 4, we test if increasing either of those budgets can make up for landmark-based KV selection and find that they cannot: even doubling the number of outliers or the local budget (stored on GPU) leaves a significant performance gap compared to full attention. We verify this for additional model-dataset pairs in Appendix E. In summary, **group-based KV selection is inaccurate on context-intensive tasks**, and this is not easily mitigated with outliers.

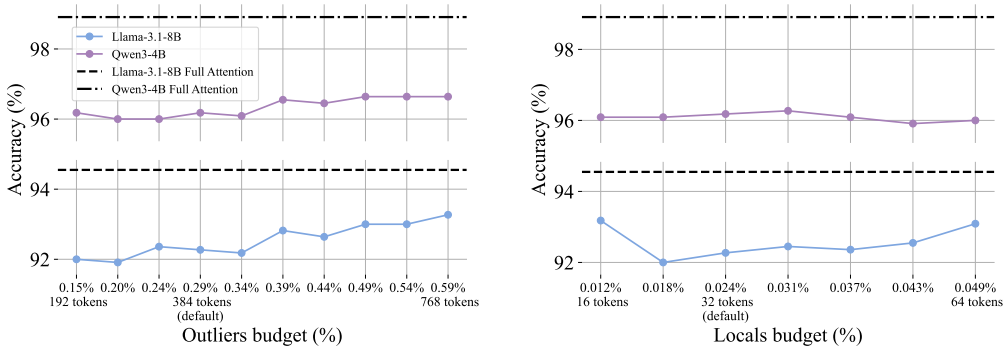


Figure 4: Evaluation of ShadowKV (w/o SVD compression) offloading for Section 4.2 with varying outlier budget (left) and local window (right) for Llama-3.1-8B-Instruct and Qwen3-4B-Instruct-2507 on MultiNeedle Retrieval 128K dataset from NeedleBench v2, all other parameters set to default.

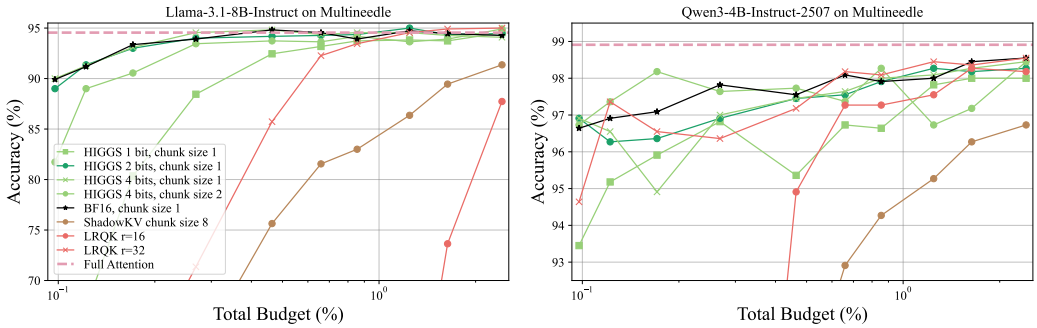


Figure 5: Evaluation of ShadowKV offloading (w/o key SVD) with different KV selection types on MultiNeedle-128K for **(left)** Llama-3.1-8B-Instruct and **(right)** Qwen3-4B-Instruct-2507.

4.3 Analyzing & Improving KV Selection Without Grouping

Our previous findings indicate that chunk-based KV selection is inaccurate on context-intensive tasks, even with reduced group size. However, decreasing chunk size to 1 would mean storing all keys on GPU and only offloading values, which would not be practical in real-world offloading setups. This raises the question of how to balance model quality against the amount of data loaded into memory.

Instead of selecting keys as groups, a more straightforward approach is to select individual keys based on compact representations. InfiniGen (Lee et al., 2024) uses compact SVD-based key representations to select which KV entries are loaded from system memory. In subsequent work, LRQK (Li et al., 2025b) uses a more specialized low-rank approximation to also select which top-k keys are needed at a given step. We consider a simpler alternative: extreme quantization. We use the same HIGGS quantizer, but with 1-2 bit grids (d=8, n=256 for 1.02 bit, d=4,n=256 for 2.02 bit).

In Figures 5 and 6, we compare ShadowKV group landmarks, LRQK individual low-rank proxies and HIGGS data-free quantization. We compare quantization setups that use the same amount of GPU memory as ShadowKV landmarks. Specifically, the configurations 16-bit/chunk8, 4-bit/chunk2, and 2-bit/chunk1 all require, on average, 2 bits per key on GPU. In turn, BF16/chunk1 serves as an "oracle" configuration, providing an upper bound on landmark selection accuracy. Overall, 2-bit HIGGS approximation with chunk size 1 substantially outperforms both ShadowKV (chunk 8, equal GPU memory budget) and LRQK (rank=32) that takes up more GPU memory. In turn, 4-bit HIGGS matches the upper bound performance at similar budgets to LRQK rank 32. These results suggest that simple data-free quantization allows for more accurate KV selection than both grouping and specialized low-rank representations on context-intensive tasks. Furthermore, not using group landmarks means that KV offloading does not need group outliers (which is why YAKV disables outliers as discussed in Section 3.2), simplifying code and reducing GPU footprint.

Further improvements: we found two more strategies that further improve KV selection:

1. **Residual Landmark Quantization:** we can achieve accurate KV selection at ≈1.5 bits per value with group-wise RVQ (Jégou et al., 2011), i.e. keeping one 4-bit "landmark" per 8 keys and 1-bit "error correction" for each key in the group. We describe and evaluate it in Appendix F.
2. **Adaptive "Top-P" budget:** instead of loading a fixed number of KV vectors per head, we partition KV budget dynamically between heads to maximize total coverage, evaluated in Appendix G.

Note that **we do not include either in YAKV** to keep our method simple and easy to implement. While they reduce memory usage and improve accuracy, adding them would confound our analysis.

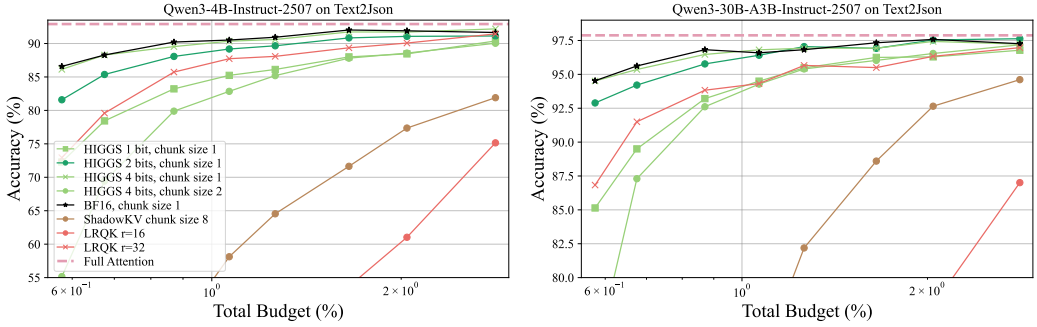


Figure 6: Evaluation of ShadowKV offloading (w/o key SVD) with different KV selection type on Text2JSON for **(left)** Qwen3-4B-Instruct-2507 and **(right)** Qwen3-30B-A3B-Instruct-2507.

Table 2: Evaluation of KV Offloading for Qwen3-Instruct-2507 4B and 30B-A3B on context-intensive tasks, compared in terms of accuracy and average host-to-device transfer per decoding step.

Method	Text2JSON		MultiNeedle		LongProc		Loong		Average	
	Acc	GiB	Acc	GiB	Acc	GiB	Acc	GiB	Acc	GiB
Model: `Qwen/Qwen3-4B-Instruct-2507`										
Original	91.25	2.74	98.91	18.00	50.5	5.22	46.81	11.53	71.87	9.37
ShadowKV	69.38	0.02	92.64	0.14	15.75	0.04	44.56	0.09	55.58	0.07
ArkVale	22.10	0.02	96.64	0.14	3.37	0.04	-	0.09	-	-
LRQK	87.99	0.02	98.64	0.14	8.75	0.04	44.70	0.09	60.02	0.07
YAKV (ours)	90.83	0.02	98.45	0.14	33.37	0.04	47.34	0.09	**67.50**	0.07
Model: `Qwen/Qwen3-30B-A3B-Instruct-2507`										
Original	97.88	1.84	100.00	12.00	59.7	3.48	56.29	7.69	78.47	6.25
ShadowKV	80.39	0.014	98.82	0.09	27.00	0.03	51.53	0.06	64.44	0.05
ArkVale	46.05	0.014	99.91	0.09	14.06	0.03	-	0.06	-	-
LRQK	95.66	0.014	99.82	0.09	31.75	0.03	55.50	0.06	70.68	0.05
YAKV (ours)	96.74	0.014	99.82	0.09	38.00	0.03	57.58	0.06	**73.04**	0.05

4.4 Combined Evaluations

In this section, we evaluate our previous observations across a broader range of models and context-intensive tasks. We use the four benchmarks identified earlier: Text2JSON, MultiNeedle-128K, LongProc HTML to TSV 8K, and Loong. We evaluate four LLMs from two popular families: Qwen3-4B-Instruct-2507, Qwen3-30B-A3B-Instruct-2507 (Yang et al., 2025a), Llama 3.1 8B, and Llama 3.2 3B (Touvron et al., 2023; Dubey et al., 2024). We compare the following methods:

- **YAKV** (ours): offloading with 4-bit HIGGS compression and 2-bit HIGGS for selection.
- **ShadowKV** (Sun et al., 2025) with SVD compression, group-mean landmarks and outliers.
- **ArkVale** (Chen et al., 2024) with group-based KV selection using cuboid-mean approximations.
- **LRQK** (Li et al., 2025b): specialized low-rank decomposition for uncompressed key selection.
- **InfiniGen** (Lee et al., 2024): offloading with SVD-based KV selection, does not support Qwen3.

For fair comparison, we evaluate these methods using the same total PCIe budget per token (GiB). We use the recommended ShadowKV parameters as our reference point and tune YAKV, LRQK and InfiniGen to fit into the budget, using recommended parameters where possible. One notable exception is that InfiniGen does not natively support Group Query Attention (GQA Ainslie et al., 2023). To address this, we modified the official implementation to aggregate key scores within the same query group, similar to ShadowKV. We provide detailed configurations in Appendix H.

The results in Tables 2 & 3 align with our earlier findings with a few caveats. For Qwen3, YAKV consistently achieves near-lossless accuracy and outperforms baselines in most cases. LRQK scores slightly (0.09%) higher on MultiNeedle, but drops significantly on other benchmarks. Llama 3 shows similar results, except that all methods score poorly on Text2JSON and LongProc. Upon closer inspection, we found that this is caused by the model getting "stuck" and generating invalid output structures. We provide additional setups in Appendices I, J & K, including hybrid Qwen 3.5 models.

Table 3: Evaluation of KV Offloading for Llama 3.1-8B and 3.2-3B (both Instruct) on context-intensive tasks, compared in terms of accuracy and average host-to-device transfer per decoding step.

Method	Text2JSON		MultiNeedle		LongProc		Loong		Average	
	Acc	GiB	Acc	GiB	Acc	GiB	Acc	GiB	Acc	GiB
Model: `meta-llama/Llama-3.1-8B-Instruct`										
Original	46.75	2.44	94.55	16.00	22.71	4.64	37.28	9.64	50.32	8.18
ShadowKV	10.46	0.02	85.73	0.12	4.59	0.04	31.25	0.08	33.01	0.06
InfiniGen	2.64	0.02	84.91	0.12	0.00	0.04	-	0.08	-	0.06
ArkVale	1.19	0.02	90.09	0.12	0.06	0.04	-	0.08	-	0.06
LRQK	3.30	0.02	93.18	0.12	0.85	0.04	30.49	0.08	31.96	0.06
YAKV (ours)	30.60	0.02	93.36	0.12	14.02	0.04	33.53	0.08	**42.88**	0.06
Model: `meta-llama/Llama-3.2-3B-Instruct`										
Original	27.69	2.14	85.45	14.00	0.99	4.06	20.77	8.44	33.73	7.16
ShadowKV	9.46	0.02	72.91	0.11	0.15	0.03	19.74	0.07	25.57	0.06
InfiniGen	5.47	0.02	84.73	0.11	0.01	0.03	-	0.07	-	0.06
ArkVale	0.45	0.02	78.82	0.11	0.01	0.03	-	0.07	-	0.06
LRQK	2.05	0.02	84.64	0.11	0.01	0.03	18.31	0.07	26.25	0.06
YAKV (ours)	25.70	0.02	80.82	0.11	0.24	0.03	19.84	0.07	**31.65**	0.06

Table 4: Inference throughput on one H200 GPU, including prefill with continuous batching. Synthetic inputs use 65K prompts with forced decoding. TPOT is in milliseconds; throughput is in tokens/s.

Qwen3 model	Data	Method	Batch	TPOT	Throughput	Speedup
30B-A3B Instruct	Synthetic	Baseline	1	54	18.5	1.00
		YAKV	16	380	**42.1**	**2.28**
32B (YARN)	Synthetic	Baseline	1	37	27.0	1.00
		YAKV	16	370	**43.2**	**1.60**
30B-A3B Instruct	Real	Baseline	1	38	26.32	1.00
		YAKV	32	420	**76.2**	**2.89**
30B-A3B Thinking	Real	Baseline	1	39	25.64	1.00
		YAKV	32	252	**127.0**	**4.95**

4.5 GPU Inference Throughput

In this section, we evaluate real-world GPU inference throughput to verify that our KV selection techniques are practical. We implement a minimal version of YAKV offloading on top of mini SGLang (Zheng et al., 2024) codebase with modified HIGGS kernels (Guo et al., 2024; Malinovskii et al., 2025) (see Appendix M for details) and evaluate inference throughput in two setups: synthetic 65K prompts (forced decoding w/o EOS) and shuffled Text2JSON+MultiNeedle inputs (realistic decoding). For a realistic evaluation scenario, we continuously add new client requests as the model finishes processing existing ones. We use SGLang continuous batching: the new prompts are encoded and added to the current batch on the fly.

We report the optimal batch size that fits in GPU memory, individual token generation latency (TPOT, time per output token), and the overall inference throughput (tokens per second). Both Throughput and TPOT include the prefill time which is unaffected by offloading. For YAKV, we use the same budget parameters as in Section 4.4. For synthetic inputs, we evaluate Qwen3-30B-A3B-Instruct-2507 (MoE) and dense Qwen3-32B models. The 32B model natively supports sequence length up to 32,768, so we use the recommended YARN (Peng et al., 2024) configuration to support longer sequences[3]. For real data, we compare Qwen3-30B-A3B Instruct and Thinking variants: these have the same number of parameters, but differ in how many *output* tokens they generate per prompt. The Thinking model generates more tokens, making it a decoding-heavy scenario, while the Instruct version generates less and spends a larger portion of time on prefill (increasing average TPOT).

The results in Table 4 demonstrate that the offloaded inference can achieve $1.5-4.9\times$ higher throughput than full attention by processing larger input batches that would otherwise not fit into memory. This is consistent with previous works on KV offloading, showing that YAKV can maintain high inference throughput while being more accurate on context-intensive tasks (see Section 4.4). We provide more detailed configurations and evaluation results (e.g. TTFT) in Appendix M.

5 Discussion

In this work, we tested the limits of sparse KV offloading and found that simply increasing the amount of required information makes the problem harder for offloading. We gathered a dataset of context-intensive tasks and combine it with context-intensive subsets of existing benchmarks to test modern offloading methods. Our findings demonstrate that modern KV offloading algorithms become lossy or require significantly higher PCIe budgets.

However, unlike traditional KV eviction, *offloading is fundamentally capable of solving context-intensive tasks.* The failures we observed can be attributed to 1) overly aggressive key compression that worked on easier problems and 2) inaccurate token selection heuristics based on landmarks. These are not conceptual problems with the offloading itself, but technical limitations that can be circumvented with better compression and key selection. Furthermore, our YAKV evaluations demonstrate that addressing these two failure modes can significantly improve KV offloading accuracy while retaining high inference throughput. We hope that our findings and evaluation sets will help inform the design of future KV offloading methods .

Limitations. The main focus of our work is to better understand the effectiveness of KV offloading, not design the highest performance algorithm. As such, we designed YAKV as a minimal working KV offloading system without complex technical features such as KV prefetch, adaptive budgeting or better quantization that could improve performance (see Appendix F & G). Our analysis is also limited to pure language models: in future work, it would be interesting consider multimodal LLMs.

[3]Context size and YARN from model card, `https://huggingface.co/Qwen/Qwen3-32B`.

AI USE STATEMENT

In this work, we used generative AI tools for generating, cleaning, and reformatting a synthetic dataset. More specifically, we used GPT-5.2 to generate the synthetic entries in Text2JSON, as described in Appendix C, and GPT-4-Turbo as the LLM judge for Loong evaluation, as described in Appendix B. We have not used generative AI tools for developing theoretical models or conceptual frameworks, implementing methods, proposing hypotheses, assisting with translation, feedback on research methodology, supporting qualitative analysis, or interpreting results. Formulating mathematical claims or providing ingredients for these claims are not applicable to this work. Additionally, we used generative AI tools to help search for additional related works and draft empty templates for certain tables and plots. We have reviewed all AI-assisted work: we checked the validity of evaluation scenarios and read the generated table and plot templates before plugging in our evaluation results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Dataset construction and scoring are detailed in Appendix C. Baseline configurations, dataset versions and licenses, inference implementation, and hardware/runtime details are provided in Appendices H, M, and N. Our (anonymous) supplementary materials include the Text2JSON dataset, the reference implementation of YAKV inference and evaluation scripts, to be released upon publication.

REFERENCES

567-labs. Instructor: Structured outputs for llms, 2026. URL `https://github.com/567-labs/instructor`. Version information varies; actively maintained repository.

AI21 Labs. Introducing jamba2: The open source model family for enterprise reliability and efficiency, 2026. URL `https://www.ai21.com/blog/introducing-jamba2/`.

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, 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, 2023.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du 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*, 2022.

Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14388–14411, 2024.

Samhruth Ananthanarayanan, Ayan Sengupta, and Tanmoy Chakraborty. Understanding the physics of key-value cache compression for llms through attention dynamics, 2026. URL `https://arxiv.org/abs/2603.01426`.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. 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)*, pp. 3639–3664, 2025.

Owen Bianchi, Mathew J. Koretsky, Maya Willey, Chelsea X. Alvarado, Tanay Nayak, Adi Asija, Nicole Kuznetsov, Mike A. Nalls, Faraz Faghri, and Daniel Khashabi. Hidden in the haystack: Smaller needles are more difficult for llms to find. *arXiv preprint arXiv:2505.18148*, abs/2505.18148, 2025. URL https://arxiv.org/abs/2505.18148.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ayi7qezU87.

Alex Chen, Renato Geh, Aditya Grover, Guy Van Den Broeck, and Daniel Mingyi Israel. The pitfalls of KV cache compression. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 41530–41553, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1926. URL https://aclanthology.org/2026.acl-long.1926/.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. Arkvale: Efficient generative llm inference with recallable key-value eviction. In *Advances in Neural Information Processing Systems 37*, 2024.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. FinQA: A dataset of numerical reasoning over financial data. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3697–3711, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.300. URL https://aclanthology.org/2021.emnlp-main.300/.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. Magicpig: Lsh sampling for efficient llm generation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 44169–44190, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/6d50d824ae819d5a961c1d8edc15e833-Paper-Conference.pdf.

Lei Cui, Furu Wei, and Ming Zhou. Neural open information extraction. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 407–413, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2065. URL https://aclanthology.org/P18-2065/.

John Dagdelen, Alexander Dunn, Sanghoon Lee, Nicholas Walker, Andrew S. Rosen, Gerbrand Ceder, Kristin A. Persson, and Anubhav Jain. Structured information extraction from scientific text with large language models. *Nature Communications*, 15(1):1418, 2024. ISSN 2041-1723. doi: 10.1038/s41467-024-45563-x. URL https://doi.org/10.1038/s41467-024-45563-x.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Zican Dong, Tianyi Tang, Junyi Li, Wayne Xin Zhao, and Ji-Rong Wen. Bamboo: A comprehensive benchmark for evaluating long text modeling capacities of large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 2086–2099, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Vage Egiazarian, Roberto Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, et al. Bridging the gap between promise and performance for microscaling fp4 quantization. In *International Conference on Learning Representations*, volume 2026, pp. 113529–113563, 2026.

Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference of Empirical Methods in Natural Language Processing (EMNLP '11)*, Edinburgh, Scotland, UK, July 27-31 2011.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *Advances in Neural Information Processing Systems*, 38:113152–113188, 2025.

Team GLM, :, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Jingyu Sun, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024. URL https://arxiv.org/abs/2406.12793.

Akshay Goel. Langextract, 2026. URL https://github.com/google/langextract.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric Xing, and Yoon Kim. Fast matrix multiplications for lookup table-quantized llms. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 12419–12433, 2024.

Ziyuan He, Yuxuan Wang, Jiaqi Li, Kexin Liang, and Muhan Zhang. Loogle v2: Are llms ready for real world long dependency challenges? *Advances in Neural Information Processing Systems*, 38, 2025.

Amey Hengle, Prasoon Bajpai, Soham Dan, and Tanmoy Chakraborty. Multilingual needle in a haystack: Investigating long-context behavior of multilingual large language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5165–5180, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.267. URL https://aclanthology.org/2025.naacl-long.267/.

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. In *Advances in Neural Information Processing Systems 37*, 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? In *Proceedings of the First Conference on Language Modeling (COLM)*, 2024.

Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 861–864. IEEE, 2011.

Simon Jegou and Maximilian Jeblick. Kvzap: Fast, adaptive, and faithful kv cache pruning. *arXiv preprint arXiv:2601.07891*, 2026.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. In *Advances in Neural Information Processing Systems 37*, 2024.

Martin Josifoski, Nicola De Cao, Maxime Peyrard, Fabio Petroni, and Robert West. Genie: Generative information extraction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4626–4643, 2022.

Greg Kamradt. Needle in a haystack - pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`, 2023. GitHub repository.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. *Advances in Neural Information Processing Systems*, 38:167563–167591, 2025.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2024.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long software tasks. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 92213–92266. Curran Associates, Inc., 2025. doi: 10.52202/085713-3086. URL `https://proceedings.neurips.cc/paper_files/paper/2025/file/85069585133c4c168c865e65d72e9775-Paper-Conference.pdf`.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. Infinigen: Efficient generative inference of large language models with dynamic kv cache management. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can context length of open-source LLMs truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023. URL `https://openreview.net/forum?id=LywifFNXV5`.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. Loogle: Can long-context language models understand long contexts? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16304–16333, 2024a.

Jonathan Li, Nasim Farahini, Evgenii Iuliugin, Magnus Vesterlund, Christian Häggström, Guangtao Wang, Shubhangi Upasani, Ayush Sachdeva, Rui Li, Faline Fu, Chen Wu, Ayesha Siddiqua, John Long, Tuowen Zhao, Matheen Mussadiq, Håkan Zeffer, Yun Du, Mingran Wang, Qinghua Li, Bo Li, Urmish Thakker, and Raghu Prabhakar. Snapstream: Efficient long sequence decoding on dataflow accelerators. In *ISC High Performance 2026 Research Paper Proceedings (41st International Conference)*, pp. 1–14, 2026. doi: 10.23919/ISC.2026.11520481.

Mo Li, Songyang Zhang, Taolin Zhang, Haodong Duan, Yunxin Liu, and Kai Chen. Needlebench: Evaluating LLM retrieval and reasoning across varying information densities. *Transactions on Machine Learning Research*, 2025a. ISSN 2835-8856. URL `https://openreview.net/forum?id=cEvmIKsRw0`.

Tenghui Li, Guoxu Zhou, Xuyang Zhao, Yuning Qiu, and Qibin Zhao. Efficient low rank attention for long-context inference in large language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-1284. URL `https://papers.nips.cc/paper_files/paper/2025/hash/36db9d16a9f80ede9e69d5d174cfd6ea-Abstract-Conference.html`.

Xiang Li, Xiangyu Zhou, Rui Dong, Yihong Zhang, and Xinyu Wang. Efficient bottom-up synthesis for programs with local variables. *Proceedings of the ACM on Programming Languages*, 8(POPL), January 2024b. doi: 10.1145/3632894. URL https://doi.org/10.1145/3632894.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, et al. Scbench: A kv cache-centric analysis of long-context methods. In *International Conference on Learning Representations*, volume 2025, pp. 66063–66093, 2025c.

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. In *Advances in Neural Information Processing Systems 37*, 2024c.

Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning. *Advances in Neural Information Processing Systems*, 38:119406–119432, 2025.

Hongzhan Lin, Zhiqi Bai, Xinmiao Zhang, Siran Yang, Jiamang Wang, Yunlong Xu, Jiaheng Liu, Yongchi Zhao, Xiang Li, Yuchi Xu, et al. Reconstructing kv caches with cross-layer fusion for enhanced transformers. In *International Conference on Learning Representations*, volume 2026, pp. 87101–87124, 2026.

Liquid AI. Introducing lfm2.5: The next generation of on-device ai, 2026. URL https://www.liquid.ai/blog/introducing-lfm2-5-the-next-generation-of-on-device-ai.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. Retrievalattention: Accelerating long-context llm inference via vector retrieval. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 54358–54385. Curran Associates, Inc., 2025. doi: 10.52202/085713-1816. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4e36d4049fb0fea195a8267c8dcd0824-Paper-Conference.pdf.

Liu Liu, Zheng Qu, Zhaodong Chen, Yufei Ding, and Yuan Xie. Transformer acceleration with dynamic sparse attention. *arXiv preprint arXiv:2110.11299*, 2021. URL https://arxiv.org/abs/2110.11299.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations*, volume 2024, pp. 47832–47850, 2024a.

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. In *Proceedings of the 41st International Conference on Machine Learning*, 2024b.

Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, et al. Repoagent: An llm-powered open-source framework for repository-level code documentation generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 436–464, 2024.

Vladimir Malinovskii, Andrei Panferov, Ivan Ilin, Han Guo, Peter Richtárik, and Dan Alistarh. Higgs: Pushing the limits of large language model quantization via the linearity theorem. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10857–10886, 2025.

William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, et al. Olmo hybrid: From theory to practice and back, 2026. URL https://arxiv.org/abs/2604.03444.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. Fp8 formats for deep learning, 2022. URL https://arxiv.org/abs/2209.05433.

Amirkeivan Mohtashami and Martin Jaggi. Random-access infinite context length for transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=7eHn64w0Vy.

NVIDIA. Nvidia, arm, and intel publish fp8 specification for standardization as an interchange format for ai. https://developer.nvidia.com/blog/nvidia-arm-and-intel-publish-fp8-specification-for-standardization-as-an-interchange-format-for-ai/, 2022.

NVIDIA. Optimizing inference for long context and large batch sizes with nvfp4 kv cache. https://developer.nvidia.com/blog/optimizing-inference-for-long-context-and-large-batch-sizes-with-nvfp4-kv-cache/, 2025.

NVIDIA. Quantization in tensorrt llm. https://nvidia.github.io/TensorRT-LLM/features/quantization.html, 2026. Accessed: 2026-04-08.

NVIDIA Corporation. Speed up inference with sota quantization techniques in tensorrt-llm. https://nvidia.github.io/TensorRT-LLM/blogs/quantization-in-TRT-LLM.html, 2026. Describes post-training quantization (FP8, INT8, INT4), performance/accuracy trade-offs, and KV-cache quantization in TensorRT-LLM. Accessed: 2026-04-08.

Leonid Pekelis, Michael Feil, Forrest Moret, Mark Huang, and Tiffany Peng. Llama 3 gradient: A series of long context models. https://gradient.ai/blog/scaling-rotational-embeddings-for-long-context-language-models, 2024. Gradient AI blog post.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 31932–31951, 2024.

Aurick Qiao, Zhewei Yao, Samyam Rajbhandari, and Yuxiong He. SwiftKV: Fast prefill-optimized inference with knowledge-preserving model transformation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25734–25753, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1306. URL https://aclanthology.org/2025.emnlp-main.1306/.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. Cake: Cascading and adaptive kv cache eviction with layer preferences. In *International Conference on Learning Representations*, volume 2025, pp. 89945–89977, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026. URL https://qwen.ai/blog?id=qwen3.5.

sgl-project. Mini-sglang, 2026. URL https://github.com/sgl-project/mini-sglang. Version information varies; actively maintained repository.

Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, and Omer Levy. SCROLLS: Standardized CompaRison over long language sequences. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 12007–12021, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.823. URL https://aclanthology.org/2022.emnlp-main.823/.

Sergii Shcherbak. Contextgem: Effortless llm extraction from documents, 2025. URL https://github.com/shcherbak-ai/contextgem. Built in Oslo, Norway; actively developed open-source project.

Yiqun Shen, Song Yuan, Zhengze Zhang, Xiaoliang Wang, Daxin Jiang, and Nguyen Cam-Tu. Lava: Layer-wise kv cache eviction with dynamic budget allocation, 2025. URL https://arxiv.org/abs/2509.09754.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Miaojing Shi, Teddy Furon, and Hervé Jégou. A group testing framework for similarity search in high-dimensional spaces. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, pp. 407–416, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330633. doi: 10.1145/2647868.2654895. URL https://doi.org/10.1145/2647868.2654895.

Alina Shutova, Vladimir Malinovskii, Vage Egiazarian, Denis Kuznedelev, Denis Mazur, Nikita Surkov, Ivan Ermakov, and Dan Alistarh. Cache me if you must: Adaptive key-value quantization for large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 55451–55473. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/shutova25a.html.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *Advances in Neural Information Processing Systems*, 37: 16692–16723, 2024.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

vLLM Project. Quantized KV cache. https://docs.vllm.ai/en/latest/features/quantization/quantized_kvcache/, 2026. Accessed: 2026-04-08.

Minzheng Wang, Longze Chen, Fu Cheng, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, et al. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5627–5646, 2024.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents, 2025. URL https://arxiv.org/abs/2504.12516.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 86809–86836, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/d813d324dbf0598bbdc9c8e79740ed01-Paper-Conference.pdf.

You Wu, Haoyi Wu, and Kewei Tu. A systematic study of cross-layer kv sharing for efficient llm inference. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 396–403, 2025b.

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

Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng, Yang Wang, and Enhong Chen. Large language models for generative information extraction: a survey. *Frontiers of Computer Science*, 18(6):186357, 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40555-y. URL https://doi.org/10.1007/s11704-024-40555-y.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. Qwen2.5-1m technical report, 2025b. URL https://arxiv.org/abs/2501.15383.

Dongquan Yang, Yifan Yang, Xiaotian Yu, Xianbiao Qi, and Rong Xiao. Hcattention: Extreme kv cache compression via heterogeneous attention computing for llms. *Neurocomputing*, pp. 134247, 2026.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025c. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/4904fad153f6434a7bcf04465d4be2cc-Paper-Conference.pdf.

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. Kvsharer: Efficient inference via layer-wise dissimilar kv cache sharing. *arXiv preprint arXiv:2410.18517*, 2024.

Alexander Yates, Michele Banko, Matthew Broadhead, Michael Cafarella, Oren Etzioni, and Stephen Soderland. TextRunner: Open information extraction on the web. In Bob Carpenter, Amanda Stent, and Jason D. Williams (eds.), *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pp. 25–26, Rochester, New York, USA, April 2007. Association for Computational Linguistics. URL https://aclanthology.org/N07-4013/.

Xi Ye, Fangcong Yin, Yinghui He, Joie Zhang, Howard Yen, Tianyu Gao, Greg Durrett, and Danqi Chen. Longproc: Benchmarking long-context language models on long procedural generation. In *Conference on Language Modeling*, 2025.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational*

Linguistics (Volume 1: Long Papers), pp. 23078–23097, Vienna, Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1126. URL https://aclanthology.org/2025.acl-long.1126/.

Tao Yuan, Xuefei Ning, Dong Zhou, Zhijie Yang, Shiyao Li, Minghui Zhuang, Zheyue Tan, Zhuyu Yao, Dahua Lin, Boxun Li, Guohao Dai, Shengen Yan, and Yu Wang. LV-eval: A balanced long-context benchmark with 5 length levels up to 256k. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=r0AXK5Cnhr.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *Proc. ACM Manag. Data*, 3(3), June 2025a. doi: 10.1145/3725338. URL https://doi.org/10.1145/3725338.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattention: accurate and training-free sparse attention accelerating any model inference. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025b.

Lei Zhang, Yunshui Li, Ziqiang Liu, Jiaxi Yang, Junhao Liu, Longze Chen, Run Luo, and Min Yang. Marathon: A race through the realm of long context with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5201–5217, 2024a.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.814.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710. Curran Associates, Inc., 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623. Curran Associates, Inc., 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Qihui Zhou, Peiqi Yin, Pengfei Zuo, and James Cheng. Progressive sparse attention: Algorithm and system co-design for efficient attention in llm serving, 2025. URL https://arxiv.org/abs/2503.00392.

A Extended Background

Transformer-based LLMs typically store task-specific information in a key-value (KV) cache. This cache contains token-level vector representations that are consumed by the attention layers at each inference step (Vaswani et al., 2017). In modern LLMs, the cache is usually limited to $\approx 10^{4-6}$ tokens, with more extreme cases exceeding one million tokens (GLM et al., 2024; Yang et al., 2025b). Though this limit can be extended (Peng et al., 2024; Li et al., 2023; Pekelis et al., 2024), large KV caches quickly exhaust accelerator memory[4] and reduce inference throughput.

To mitigate this, several lines of work propose KV-cache quantization (Liu et al., 2024b; Hooper et al., 2024; Ashkboos et al., 2024), eviction (pruning) (Xiao et al., 2024; Zhang et al., 2023; Li et al., 2024c), cross-layer sharing (Wu et al., 2025b; Yang et al., 2024; Lin et al., 2026), and other related techniques. These approaches are not mutually exclusive: an inference server may evict a subset of KV entries and quantize the remainder to further reduce memory usage (Li et al., 2024c; Shutova et al., 2025). However, while quantization is already widely used in industrial deployments (vLLM Project, 2026; NVIDIA Corporation, 2026), KV eviction has seen slower adoption. A key reason is that pruning KV entries can significantly degrade performance on certain problem types (Li et al., 2026; Chen et al., 2026; Ananthanarayanan et al., 2026) where it is hard to guess which entries can be evicted, or if the problem needs the entire prompt (e.g. translation).

A more recent line of work proposes an alternative: instead of permanently removing KV-cache entries, these methods offload KV vectors to a larger but slower system memory (Aminabadi et al., 2022; Sheng et al., 2023; Mohtashami & Jaggi, 2023). To avoid reloading the entire cache, they estimate which tokens are most relevant at a given step (Lee et al., 2024; Chen et al., 2024; Sun et al., 2025; Liu et al., 2025; Mohtashami & Jaggi, 2023), using mechanisms closely related to sparse attention (Tang et al., 2024; Liu et al., 2021; Yuan et al., 2025a; Jiang et al., 2024). For instance, InfiniGen (Lee et al., 2024) uses SVD approximation to determine important keys and load them from system memory concurrently with previous layer computation. ShadowKV (Sun et al., 2025) further splits KV entries into groups of 8 consecutive tokens and uses group-average keys ("landmarks") to determine which KV entries should be loaded at a given inference step. Subsequent works extend this idea further with better "landmarks" (Chen et al., 2024), specialized key decomposition (Li et al., 2025b) and technical optimizations (Yang et al., 2026). Another line of work (Chen et al., 2025; Zhang et al., 2025a) proposes low-bit representations for important token identification. Unlike permanent pruning, offloading preserves the possibility of accessing the full KV cache over time, and is therefore, in principle, capable of solving problems that pruning would struggle with. In this work, we find that this promise does not always hold in practice and propose a way to fix it.

B Preliminary Benchmark Exploration & Configurations

Before our primary investigation in Sections 3 & 4, we ran preliminary experiments on a range of popular benchmarks with KV offloading. Notably, the main results of ShadowKV reproduce easily and perfectly from the official code: Llama 3.1 8B consistently scores near-losslessly on RULER and select LongBench subtasks. We then apply the same code to evaluate on more context-intensive benchmarks: Loong (Wang et al., 2024) and MultiNeedle. We use the official Loong codebase[5] using `gpt-4-turbo` for LLM-as-a-judge. The results for Loong are summarized in Table 5: the context-intensive benchmark ("Leave No Document Behind") shows significant accuracy drawdowns across all levels with default ShadowKV hyperparameters. This aligns with our observations and makes Loong an important example of context-intensive multi-document QA. The only reason why we prefer MultiNeedle for Sections 4.1—4.3 is that evaluating on Loong incurs significant API costs for LLM-as-a-Judge, making it prohibitive to run extensive budget sweeps necessary for our analysis.

Our MultiNeedle evaluations use the MultiNeedle Retrieval 128K configuration from NeedleBench v2 (Li et al., 2025a) using OpenCompass (commit id `1246210`). This setup hides 11 synthetic "needles" among 128K prompt tokens and scores how many of those the model finds correctly (exact match accuracy). We use `needlebench_v2_multi_retrieval_128k` configuration. For

[4] For example, storing 1M tokens for Qwen2.5-7B-Instruct-1M in `bfloat16` requires over 180 GiB of GPU memory.

[5] `https://github.com/mozerwang/loong`

Table 5: Comparison of Full Attention and ShadowKV on Loong for Llama-3.1-8B-Instruct. Level 1: Spotlight Locating, level 2: Comparison, Level 3: Clustering, Level 4: Chain of Reasoning

	Level 1		Level 2		Level 3		Level 4	
Method	Score	Perfect	Score	Perfect	Score	Perfect	Score	Perfect
Full Attention	63.24	0.53	37.20	0.19	25.63	0.01	37.27	0.18
ShadowKV	46.15	0.37	25.81	0.12	15.70	0.01	24.75	0.06

experimental consistency, we use the same sample of 100 english samples instead of drawing random needles for each experiment.

Table 6: Generation and extraction prompts used for each data type in Text2JSON.

Subset	**Doctors**	**Movies**	**Organizations**	**Products**
Generation	Generate X entries, each starting with a newline, in the following format: `Name Surname, Doctor Specialization, City`	Suggest X unique movies, each starting with a newline, in the following format: `Movie title, Country of Production, Year of Production`	Generate 200 organization cards, each starting with a newline, in the following format: `The name of the organization, address, site`	Generate 10 product cards in the following format: `Product name: <Product name>` `* Color: <Color>` `* Material: <Material>` `* Length: <Length>` `* Category: <Category>`
Extraction	Find all doctor review cards in the text and compose a JSON object with the following fields: `name` — doctor's name; `specialization` — specialization; `city` — city. There is no need to reproduce the reviews. Output only JSON. Do not skip cards and do not produce duplicates.	Find all movie review cards in the text and compose a JSON object with the following fields: `name` — movie title; `country` — country of production; `year` — year of release. There is no need to reproduce the reviews. Output only JSON. Do not skip cards and do not produce duplicates.	Find all organization cards in the text and compose a JSON object with the following fields: `name` — the name of the organization (exactly as written in the card); `address` — the address; `site` — the website. There is no need to reproduce the reviews. Output only JSON. Do not skip cards and do not produce duplicates.	Find all product cards in the text and compose a JSON object with the following fields: `name` — product name (exactly as written in the card); `material` — material; `color` — color. There is no need to reproduce the descriptions. Output only JSON. Do not skip cards and do not produce duplicates.

C TEXT2JSON CONSTRUCTION AND EVALUATION

Text2JSON is constructed from four types of entries: organization cards, doctor cards, movie review cards, and product cards. All entries are generated by GPT-5.2 using predefined prompts. Importantly, the benchmark contains no personal data. To build each benchmark instance, we randomly sample between 3 and 20 entries from each category and between 3 and 10 passages from FineWeb-Edu (Penedo et al., 2024) and moderate the results manually. These segments are then concatenated using `\n\n` as a separator. We provide specific prompts for each type in Table 6.

Given such an input, the model is required to produce a valid JSON object containing all entries of the target type, in arbitrary order. We avoid LLM-as-a-Judge and use a deterministic, name-anchored metric. Each predicted/gold record is a dictionary with a unique name key and two additional fields. Evaluation first aligns prediction and gold by exact name match. Unmatched predicted names are treated as false positives, and unmatched gold names are false negatives. Each matched entry receives a score in $0, 1$: it gets 1.0 when all required fields are present and all values are correct, and it is reduced when fields are missing or values are incorrect. These per-entry scores are summed and then normalized by the total number of matched entries plus false positives and false negatives. This is a soft IoU-style entity metric: the denominator penalizes missing/spurious entities (set overlap behavior), while the numerator gives partial credit for attribute correctness on matched entities.

D ADDITIONAL EXPERIMENTS FOR SECTION 4.1

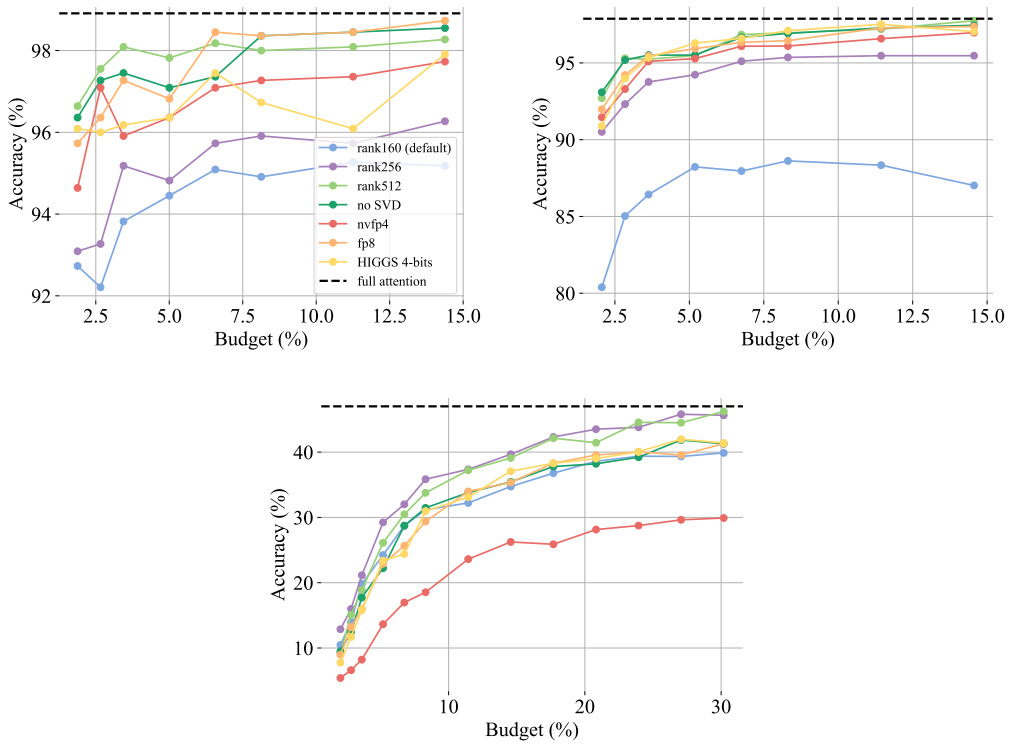


Figure 7: Evaluation of ShadowKV offloading with different KV compression strategies for Qwen3-4B-Instruct-2507 on MultiNeedle (left), Qwen3-30B-A3B-Instruct-2507 on Text2JSON (right) and Llama3.1-8B on Text2JSON (bottom). The X axis denotes the total percentage of tokens loaded (sparse, outlier and local tokens), see Section 4.1.

E Additional Experiments for Section 4.2

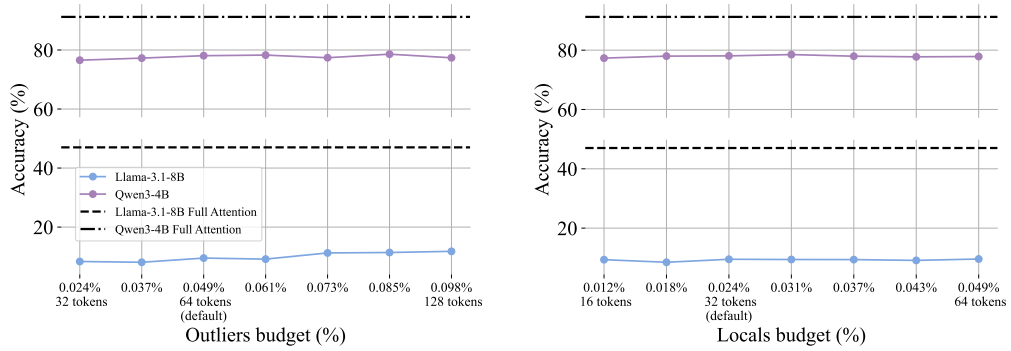


Figure 8: Evaluation of ShadowKV (w/o SVD compression) offloading for Section 4.2 with varying outlier budget (left) and local window (right) for Llama-3.1-8B-Instruct and Qwen3-4B-Instruct-2507 on Text2Json dataset.

F Residual Quantization of Landmarks

Low-bit landmark compression can noticeably degrade performance, as shown in Figures 5 and 6 for the 1-bit setting. To achieve a better trade-off between the substantial performance degradation of 1-bit quantization and the near-lossless behavior of higher-precision representations, we propose *residual quantization* for landmarks. Specifically, we first construct landmarks at higher precision (e.g., 4-bit HIGGS) using a larger chunk size (the default chunk size of 8 used in ShadowKV), and then quantize the residual between the original keys and the quantized landmarks to a low bit width (1 bit). This configuration has the same memory footprint as 1.5-bit quantized landmarks with a chunk size of 1. We report the evaluation results of residual-quantized landmarks and vanilla HIGGS in Figure 9. The results show that residual quantization achieves performance close to that of 2-bit landmarks while incurring 25% less memory overhead.

Note that residual quantization does not require reconstructing the landmarks in order to compute the highest-scoring dot products. Let $\hat{K} = \texttt{repeat}(L) + R$, where L denotes the chunk-average landmarks and R the low-bit residuals. Then the query–key dot products can be written as

$$Q\hat{K} = Q(\texttt{repeat}(L) + R) = \texttt{repeat}(QL) + QR.$$

That is, the dot products can be computed by first multiplying the query by the quantized landmarks, then repeating the resulting scores for all vectors within each chunk, and finally adding the dot product between the query and the residuals. For HIGGS-quantized landmarks, this procedure can be implemented using existing inference kernels (Malinovskii et al., 2025).

An even faster *approximate* top-k procedure can be derived from residual quantization methods used in nearest-neighbor search (Jégou et al., 2011), although related ideas had appeared earlier in other forms (Shi et al., 2014). In our setting, the algorithm first selects a subset of landmarks with high dot products for the current query and then computes residual contributions only for keys belonging to the chunks associated with those selected landmarks.

G Adaptive Budgets with Top-P and Top-KP Loading

In both the KV-cache offloading and sparse attention fields, there is growing interest in switching from conventional top-k token selection to promising top-p selection, which allows retrieving a dynamic number of tokens, potentially avoiding transfer for unnecessary tokens and utilizing more tokens upon request. Papers such as Twilight (Lin et al., 2025) and Progressive Sparse Attention (PSA) (Zhou et al., 2025) report improvements of their proposed top-p methods over top-k counterparts. Yet, under our evaluation protocol, switching from standard top-k to top-p showed little to no gain in terms

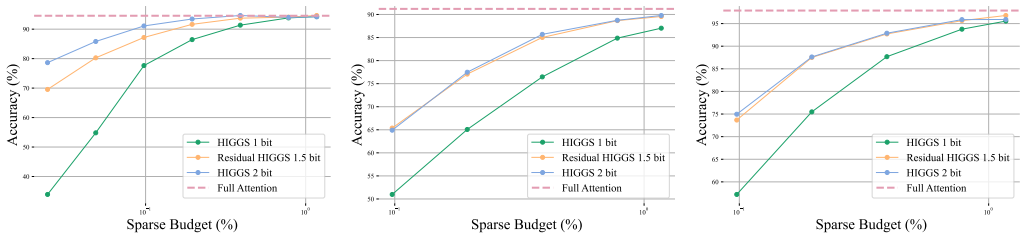


Figure 9: Comparison of 1.5-bit residual landmark quantization with 1-bit and 2-bit HIGGS. Results are shown for Llama-3.1-8B-Instruct on MultiNeedle (left), and for Qwen-3-4B-Instruct-2507 (middle) and Qwen3-30B-A3B-Instruct-2507 (right) on Text2JSON.

of both generation quality and the number of transferred tokens. Figure 10 summarizes our results. For grouped query attention (GQA) (Lin et al., 2025), token selection is ambiguous because each key head is shared by multiple query heads and therefore receives multiple attention scores. We consider two ways to aggregate these scores. In "GQA mean", we average the attention scores within each query-head group and then apply top-p selection. In "GQA any", we apply top-p selection separately for each query head and load a token if it is selected by at least one query head in the group.

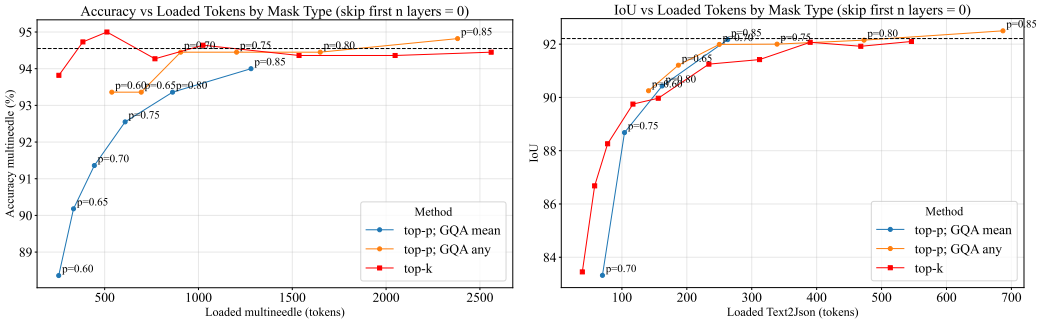


Figure 10: Trade-off between number of loaded tokens and generation quality for top-k and top-p methods. Results are shown for Llama-3.1-8B-Instruct on MultiNeedle (left), and for Qwen-3-4B-Instruct-2507 on Text2JSON (right).

Another approach to dynamic token selection, extensively studied in KV-cache pruning—e.g., AdaKV (Feng et al., 2025), CAKE (Qin et al., 2025), and LAVa (Shen et al., 2025)—is to use a shared budget K for all attention heads within a layer, rather than allocating a fixed budget of $K/\texttt{kv_heads}$ to each head. Since different heads may require different numbers of tokens, reallocating the budget across heads can increase the total attention mass covered by the layer while keeping the overall token budget fixed. This can potentially improve generation quality without increasing the number of selected tokens. Having accurate attention scores during prefill, it is easy to determine which heads benefit more from receiving additional tokens. In pruning methods, such budget allocation is performed only once, during the prefill stage, because the remaining tokens are permanently evicted.

In offloading, unlike pruning, a more optimal budget allocation could in principle be performed at every decoding step. However, this remains challenging because, unlike during prefill, offloading methods do not have access to ground-truth attention scores during decoding. Instead, their only source of information is the dot product between the current query and the "landmarks". To the best of our knowledge, dynamic budget allocation across heads has not been explored before in the context of KV-cache offloading. This may be partly because query–landmark dot products are unnormalized, making it difficult to compare token importance across heads. Moreover, as shown in Sections 4.2 and 4.3, existing methods struggle to identify important tokens and often load unimportant ones due to key grouping. Since our proposed method excels at important tokens selection, we further experiment with dynamic head budgets. We find that this consistently improves generation quality under a fixed token budget; see Figure 11. We refer to this approach as top-kp: it uses the same

fixed budget K as top-k methods, and performs selection based on attention scores rather than raw dot-product logits as in top-p methods. Attention score is computed based on query and HIGGS 2-bit landmarks, and the K tokens with the highest scores across all attention heads are selected. However, due to the lack of a low-level implementation that supports attention computation with different heads attending to different numbers of tokens, we do not include this mechanism in the final YAKV method.

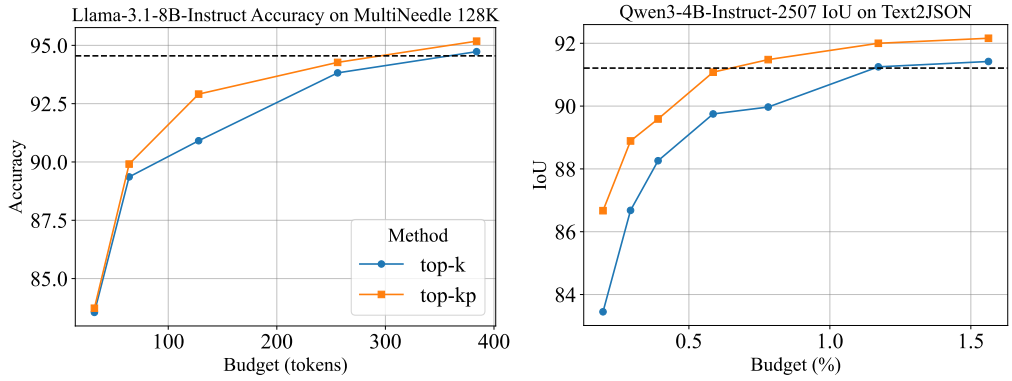


Figure 11: Generation quality for top-k and top-kp methods. Results are shown for Llama-3.1-8B-Instruct on MultiNeedle (left), and for Qwen-3-4B-Instruct-2507 on Text2JSON (right).

H Detailed Configurations for Section 4.4

We provide detailed configurations for KV offloading methods used in Section 4.4. We obtained these configurations as follows: we started with ShadowKV inference parameters from the original paper (Sun et al., 2025), then constructed equivalent budgets for other algorithms by taking the official parameters and varying the offloading budget to equalize the amount of data loaded from RAM per forward:

- **ShadowKV:** SVD rank 160, outlier budget 384, local budget 32, chunk size 8 (for landmarks), sparse budget = 1.5625% of input length (loading Vs, keys on device).
- **InfiniGen:** alpha 99.0, capacity 1.0, partial weight ratio 0.3, budget = 0.78125% of total KVs (see below).
- **ArkVale:** page size 16 (cuboid-mean digest), sink tokens 32, window tokens = 64, unlimited layers = 0, pages loaded: 0.78125% of total KV pages (rounded up).
- **LRQK:** rank = 32, recent = 64, budget = 0.78125% of total KVs.
- **YAKV:** KV: 4.02 bit, Landmark: 2.02 bit, recent 64, sparse budget = 3.125% input KVs.

For InfiniGen, the original implementation[6] does not support grouped query attention (Ainslie et al., 2023) — we modify it to implement GQA-aware KV scoring using the same methodology as in ShadowKV. Note also that the original implementation can adaptively load less than the maximum number of KVs based on the score threshold (alpha). However, since our evaluations in Section 4.4 use a fixed inference budget, it is advantageous to always load the maximum allowed number of KVs. For fair comparison, we set alpha=99, which enforces this behavior and slightly improves accuracy.

However, unlike ShadowKV, InfiniGen stores original 16-bit precision keys in system memory (ShadowKV uses key SVD and only loads values). For this reason, InfiniGen loads half as many KVs to match the PCIe data transfer of ShadowKV.

Finally, original Infinigen implementation materializes full attention scores to select top-k keys for each query, which is not suitable for long context prefill. Thus we only perform selection during subsequent decoding steps to obtain upper bound on Infinigen accuracy.

[6] `https://github.com/snu-comparch/InfiniGen`

For YAKV, we use standard 4- and 2-bit HIGGS (Malinovskii et al., 2025) grids as stated earlier: d=2, n=256 d=4, n=256 and adjust the budget accordingly.

H.1 Dataset Sources, Versions and Licenses

- **MultiNeedle:** using OpenCompass `https://github.com/open-compass/opencompass`, version 0.5.1.post1, Apache-2.0 license.
- **Loong:** `https://github.com/mozerwang/loong`, commit ID `6d2115b`, Apache-2.0 license.
- **LongProc:** `https://github.com/princeton-pli/LongProc`, commit ID `673ec4c`, Apache-2.0 license.
- Llama 3.x is under `https://www.llama.com/llama3/license/`.
- Qwen 3.x uses Apache-2.0 license, `https://huggingface.co/Qwen/Qwen3-30B-A3B-Instruct-2507/blob/main/LICENSE`.

I Alternative Quantization Experiments for YAKV

To avoid costly transfers between the CPU and GPU, different offloading methods propose various techniques for reducing the PCIe burden. For instance, ShadowKV (Sun et al., 2025) retrieves only values from the CPU, while reconstructing keys from a compressed representation. InfiniGen (Lee et al., 2024) proposes overlapping the costly PCIe transfer of the keys and values for the next layer with the computation of the current layer. In contrast, we show that compressing both keys and values to as low as 4 bits reduces the number of bytes transferred from the CPU by a factor of 4, while maintaining generation quality on par with that obtained using bfloat16 keys and values. Before drawing this conclusion, we ablated various quantization methods for both keys and values, as shown in Figure 12. Our results show that 4-bit quantization of keys and values onto the HIGGS grid preserves accuracy, whereas quantizing keys to NVFP4 may degrade quality.

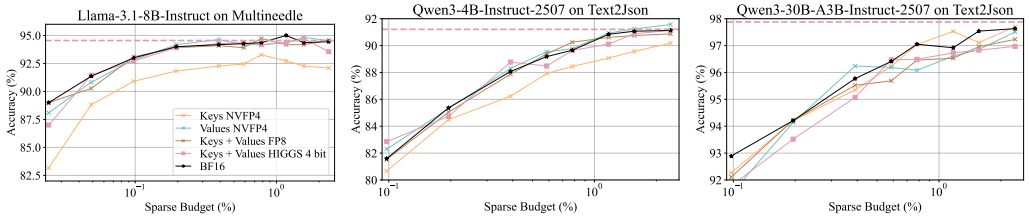


Figure 12: Comparison of different quantization methods. Results are shown for Llama-3.1-8B-Instruct on MultiNeedle (left), and for Qwen-3-4B-Instruct-2507 (middle) and Qwen3-30B-A3B-Instruct-2507 (right) on Text2JSON.

J Performance of hybrid architectures on context-intenstive tasks

Recently, subquadratic alternatives to the attention mechanism have gained increasing popularity in LLM architectures. Replacing a large fraction of attention layers with modules such as Mamba (Gu & Dao, 2024; Dao & Gu, 2024) or GatedDeltaNet (Yang et al., 2025c) has been shown to achieve performance competitive with full-attention architectures while offering better scalability with increasing sequence length. However, the use of a fixed-size hidden state may potentially limit a model's ability to process and retain information over long contexts.

To investigate this hypothesis, we evaluated several models in which a subset—or all—of the attention layers were replaced with subquadratic alternatives. The results are presented in Table 7 and Table 8.

Notably, some hybrid architectures (Liquid AI, 2026; Merrill et al., 2026; AI21 Labs, 2026) exhibit substantially degraded performance. Inspection of the generated responses revealed that most failures

Table 7: Performance of Linear & Hybrid attention models on context-intensive tasks.

Model	MultiNeedle	Text2JSON
Jamba-2-3B	57.45	13.34
LFM2.5-1.2B-Instruct	0.00	3.68
Olmo-Hybrid-Instruct-SFT-7B	0.18	0.00

Table 8: Comparison of Accuracy (%) across different models and benchmarks. Average is the mean of the four Acc columns.

Model	MultiNeedle	Text2JSON	Loong	LongProc	Average
Qwen3-4B	98.91	91.25	46.81	50.50	71.87
Qwen3-30B-A3B	100.00	97.88	56.29	59.70	78.47
Qwen-3.5-4B	99.91	83.98	56.60	46.14	71.66
Qwen-3.5-9B	100.00	96.10	60.61	55.67	78.10
Qwen3.5-27B	100.00	97.96	63.61	70.26	82.96
Qwen3.5-35B-A3B	100.00	98.02	60.92	68.34	81.82

stem from either entering repetitive generation loops or violating the required output format. In contrast, the Qwen3.5 (Qwen Team, 2026) family achieves performance comparable to, and in some cases exceeding, that of the corresponding Qwen3 models employing full attention throughout the network.

These observations suggest that a relatively small number of self-attention layers may be sufficient to support context-intensive reasoning tasks, while the remaining layers can be replaced with more computationally efficient subquadratic alternatives without a significant loss in performance.

K Comparison with PQCache and MagicPIG

Although low-rank compression proxies are the most widely used approach (Sun et al., 2025; Chen et al., 2024; Li et al., 2025b), another line of work employs quantized representations to determine which KV pairs should be retrieved from the GPU. PQCache (Zhang et al., 2025a) stores PQ codes and centroids on the GPU to identify important tokens, then retrieves the corresponding KV pairs from CPU memory. MagicPIG (Chen et al., 2025) uses LSH signatures—effectively 1-bit projections—for token selection while offloading the KV cache to CPU memory.

Tables 9 and 10 compare PQCache, MagicPIG, and YaKV on the same task suite used in the main text. PQCache substantially underperforms YaKV, whereas MagicPIG is competitive with YaKV and, in some cases, outperforms it. However, several properties of MagicPIG complicate deployment and limit end-to-end efficiency:

- **CPU-side attention computation.** Because LSH lookup tables do not map naturally to tensor operations, MagicPIG performs attention computation on the CPU. Consequently, its low PCIe bandwidth does not by itself imply low end-to-end latency: execution must also wait for CPU-side computation. This makes its results not directly comparable under the balanced-compute criterion used in Section 4.3.
- **Large system-memory overhead.** MagicPIG's hash tables require several times more system memory than the BF16 KV cache itself. For example, for Llama-3.1-8B-Instruct with a 96K-token context budget, the hash tables consume an additional 28 GB of memory, compared with 12 GB for the BF16 KV cache. YaKV requires less than 4 GB.
- **No fixed retrieval budget.** Owing to the nature of LSH, MagicPIG processes a variable number of keys for each query. This variability can worsen tail latency, particularly Q99 latency, and is another reason that its selection computation is performed on the CPU.

In contrast, YaKV is designed to be simple to implement and straightforward to integrate into efficient serving frameworks.

Table 9: Comparison of PQCache, MagicPIG, and YAKV on context-intensive tasks for Qwen3-Instruct-2507 models. Values are accuracy (%).

Method	Text2JSON	MultiNeedle	LongProc	Loong	Average
Model: `Qwen/Qwen3-4B-Instruct-2507`					
Original	91.25	98.91	50.50	46.81	71.87
PQCache	37.81	82.64	2.68	41.67	41.20
MagicPIG	89.08	**98.55**	**40.67**	47.14	**68.86**
YAKV (ours)	**90.83**	98.45	33.37	**47.34**	67.50
Model: `Qwen/Qwen3-30B-A3B-Instruct-2507`					
Original	97.88	100.00	59.70	56.29	78.47
PQCache	66.34	98.36	12.35	50.61	56.92
MagicPIG	**97.34**	**100.00**	**56.78**	56.18	**77.58**
YAKV (ours)	96.74	99.82	38.00	**57.58**	73.04

Table 10: Comparison of PQCache, MagicPIG, and YAKV on context-intensive tasks for Llama 3.x Instruct models. Values are accuracy (%).

Method	Text2JSON	MultiNeedle	LongProc	Loong	Average
Model: `meta-llama/Llama-3.1-8B-Instruct`					
Original	46.75	94.55	22.71	37.28	50.32
PQCache	0.12	5.73	0.00	25.94	7.95
MagicPIG	**31.43**	91.45	8.54	**34.81**	41.56
YAKV (ours)	30.60	**93.36**	**14.02**	33.53	**42.88**
Model: `meta-llama/Llama-3.2-3B-Instruct`					
Original	27.69	85.45	0.99	20.77	33.73
PQCache	0.24	23.18	0.00	16.79	10.05
MagicPIG	16.28	68.09	**0.33**	**20.13**	26.21
YAKV (ours)	**25.70**	**80.82**	0.24	19.84	**31.65**

L Comparison of pre-RoPE and post-RoPE

Our main YAKV setup and inference experiments in Section 4.5 employ post-RoPE quantization to enable efficient inference by avoiding the need to rotate past keys at each step. However, some prior work on KV compression suggests that pre-RoPE quantization may yield higher accuracy.

In this section, we conduct additional experiments to compare post-RoPE YAKV with a pre-RoPE alternative. We evaluate YAKV using both pre- and post-RoPE quantization, keeping the same default hyperparameters as in Section 4.4. Table 11 presents the comparison between pre- and post-RoPE YAKV. In the pre-RoPE variant, keys are quantized before RoPE and rotated after retrieval. Note that landmarks are always quantized post-RoPE to maintain inference efficiency.

The results indicate that pre-RoPE key quantization does not significantly improve model accuracy for YAKV. We attribute this to the randomized Hadamard transformation used in the HIGGS quantizer (Malinovskii et al., 2025).

Table 11: Comparison of pre- and post-RoPE quantization in YAKV for Llama-3.1-8B-Instruct on MultiNeedle and Qwen3-4B-Instruct-2507 on Text2JSON.

Setup	Llama 3.1 8B MultiNeedle	Qwen3 4B Text2JSON
Pre-RoPE	93.36	90.83
Post-RoPE	93.73	91.05

M Inference Implementation Details for Section 4.5

We implement our inference code on top of Mini-SGLang (sgl-project, 2026), a lightweight version of SGLang (Zheng et al., 2024) designed for rapid research prototyping while retaining the high performance of the full-scale framework. Unlike standard setups, we allocate the full KV cache in CPU memory. In addition, we allocate the necessary workspace buffers, including an intermediate KV buffer for the selected KVs transferred from CPU memory to GPU memory. This introduces minimal memory overhead because these buffers are allocated only once and shared across all model layers. The intermediate KV buffer uses a contiguous layout rather than a paged layout, since paging provides no benefit in this context. Since each key-value head has it's own set of tokens, we use per-head per-token quantization layout for HIGGS. This provides a substantial memory footprint from quantization scales, precisely $\frac{2}{head_dim}$ from memory needed by full precision 16-bit KV-cache. We could reduce this overhead by concatenating each key-value entry into one vector or use a single scale for multiple consecutive tokens, but we implement neither of those to keep implementation as close as possible to described method.

To support these operations, we develop several custom kernels. The HIGGS quantization, scoring and CPU-to-GPU KV-copying kernels are implemented in CUDA, while the Top-K kernel is written in Triton. We fuse landmark dequantization and scoring kernels to effectively overlap computation and codebook lookup. To accelerate memory transfers, we employ a memory-mapping strategy. Instead of relying on explicit `cudaMemcpy` calls, we map the full CPU-resident KV tensor directly into the GPU address space, allowing GPU memory reads and writes to be translated into PCIe transactions. This approach enables KV cache offloading without requiring substantial modifications to the core inference code. Furthermore, it allows the GPU to fetch data without expensive per-layer CPU-GPU synchronization, which is a critical optimization because the tokens selected for loading are determined directly on the GPU.

In future work, we intend to further optimize the inference kernels. We also plan to integrate widely used inference features, such as chunked prefill and a paged layout for the full KV cache. Importantly, the current absence of these features does not compromise the validity or relevance of our performance benchmarks.

We measure inference speed using combined inputs from Text2JSON and MultiNeedle, with an average input length of approximately 38K tokens and a maximum input length of approximately 126K tokens. With the bfloat16 Qwen3-30B-A3B model, the available VRAM is just sufficient to support batch size 1 in the full-attention baseline setup. To simulate a real-world serving scenario, we

continuously maintain the workload and keep the batch full. We evaluate two setups: a prefill-heavy setting using the Instruct model and a decode-heavy setting using the Thinking model. In Table 12, we define RPS as responses per second, TTFT as time to first token, i.e., the time between sending a request and receiving the first token, and TPOT as time per output token.

Table 12: Detailed runtime performance evaluations on real data.

Method	Batch	RPS	TTFT (ms)	TPOT (ms)	Output Length (tok)	Throughput (tok/s)
`Qwen3-30B-A3B-Instruct-2507`						
Baseline	1	0.06	3100	38	359	26.3
YAKV	4	0.07	3319	124	349	**32.3**
YAKV	8	0.13	3401	163	367	**49.1**
YAKV	16	0.17	3828	247	366	**64.8**
YAKV	32	0.2	6164	420	371	**76.2**
`Qwen3-30B-A3B-Thinking-2507`						
Baseline	1	0.011	3036	39	2289	25.6
YAKV	4	0.015	3985	108	2502	**37.0**
YAKV	8	0.026	6079	122	2560	**65.6**
YAKV	16	0.043	6659	150	2586	**106.7**
YAKV	32	0.044	10343	252	2381	**127.0**
`Qwen3-32B`						
Baseline	1	0.02	9049	30	1416	33.3
YAKV	4	0.024	9088	97	1656	**41.2**
YAKV	8	0.033	9725	157	1539	**51.0**
YAKV	16	0.045	14341	214	1671	**74.8**

N Hardware and Runtime Specifications

The specific hardware configurations necessary to reproduce our experiments vary by model size and experiment. However, all of the experiments reported in Sections 4.1, 4.2, 4.3, and 4.4 can be reproduced on a server with a single A100-80G GPU with 512 GiB RAM and 16 average performance (v)CPU cores. The runtime and throughput evaluations in Section 4.5 and Appendix M require a single H200 GPU at PCIe gen. 5 x16 host-device interconnect and similar CPU / RAM requirements. They can be run with a different GPU (e.g. multiple weaker GPUs or a single B200), but the runtime and optimal batch sizes may be different.

In total, we estimate that our evaluations, preliminary experiments, development and re-runs (e.g. due to hardware failures) took $\approx$8500 A100-hours and fewer than 24 B200-hours. However, partial replication on a subset of models and/or benchmarks will take significantly less than that. Individual results for Qwen3-4B on Text2JSON, LongProc and MultiNeedle take 0.5-2 GPU-hours and can be run on 24-48GiB GPUs, depending on the KV offloading method.

Additionally, our evaluations on Loong use LLM-as-a-Judge verification. Individual evaluations in the canonical setting cost $\$10-30$, with total expenses within $\$1200$ across all main and preliminary experiments.