KV Pareto: Systems-Level Optimization of KV Cache and Model Compression for Long Context Inference

Sai Gokhale [*] [†]
Georgia Institute of Technology

Devleena Das [†] **Rajeev Patwari** [†] **Ashish Sirasao** **Elliott Delaye**
Advanced Micro Devices (AMD)

Abstract

Long-context Large Language Models (LLMs) face significant memory bottlenecks during inference due to the linear growth of key-value (KV) cache with sequence length. While individual optimization techniques like KV cache quantization, chunked prefill, and model weight quantization have shown promise, their joint effects and optimal configurations for edge deployment remain underexplored. We introduce KV Pareto, a systems-level framework that systematically maps the trade-off frontier between total memory consumption and task accuracy across these three complementary optimization techniques. Our framework evaluates multiple LLM architectures (Qwen, Llama, Mistral) with varying KV quantization schemes (int2/4/8, mixed-precision), granularities (per-token, per-tensor, per-block), and 4-bit weight quantization via AWQ. Our framework identifies model-specific Pareto-optimal configurations that achieve 68-78% total memory reduction with minimal (1-3%) accuracy degradation on long-context tasks. We additionally verify the selected frontiers on additional benchmarks of Needle-in-a-Haystack, GSM8k and MMLU as well as extended context lengths of up to 128k to demonstrate the practical need of joint optimization for efficient LLM inference.

1 Introduction

Large Language Models (LLMs) have become useful in many applications, such as code generation (Jiang et al., 2024b), long question-answering (Liu et al., 2025) and retrieval-augmented generation (RAG) (Arslan et al., 2024). These tasks increasingly demand longer-context capabilities, pushing models like Qwen (Bai et al., 2023), Mistral (Jiang et al., 2024a) and Llama (Grattafiori et al., 2024b) to support long context lengths.

[*]This work was done during her internship with Advanced Micro Devices (AMD)

[†]Equal Contribution

The bottleneck for efficient inference arises from the transformer architecture which operates primarily in two phases: prefill and decode (Raiaan et al., 2024). During prefill, the input context is processed and stored in the key-value (KV) cache. During decode, outputs are generated autoregressively by repeatedly accessing the KV cache. Importantly, the KV Cache size grows linearly with sequence length (Patwari et al., 2025) and increases time-to-first-token (TTFT) during prefill, and time-per-output-token (TPOT) during decode. The resulting increased latency is a bottleneck for practical deployment of long-context models on edge devices.

To reduce the latency and scalability challenges, several optimization techniques have been proposed for long-context inference, including KV Quantization (Hooper et al., 2024; Li et al., 2025; Liu et al., 2024b), token eviction (Xiao et al., 2023; Corallo and Papotti, 2024), and chunked prefill (Agrawal et al., 2023). These methods are typically evaluated in isolation of one another in the context of accuracy degradation (Li et al., 2025; Liu et al., 2024b; Corallo and Papotti, 2024). This creates a practical question for deployment: *which memory optimizations, together, offer the best-trade offs between memory savings and task accuracy?*

To this end, we introduce **KV Pareto**, a framework for evaluating and understanding the trade-offs between KV memory compression and task performance in long-context LLMs. KV Pareto focuses on studying the impact of two widely accessible optimization techniques for long context, KV quantization, and chunked prefill in conjunction with 4-bit model weight quantization. Prior work evaluates these optimization techniques in isolation (Li et al., 2025; Liu et al., 2024b; Corallo and Papotti, 2024; Lin et al., 2024; Agrawal et al., 2023). Instead, our KV Pareto provides a joint assessment of optimization techniques, considering total memory savings and accuracy degradation. This enables practitioners to identify the Pareto-optimal configurations for edge deployment.

Our KV Pareto spans multiple models (Mistral, Qwen, LLaMA), KV cache quantization granularities (per-token, per-tensor, per-block), group sizes (32, 64, 128), precision formats (int2, int4, int8), as well 4-bit weight quantization via AWQ (Lin et al., 2024). We benchmark across long context evaluations including LongBench (Bai et al., 2024b), Needle-in-a-Haystack (NIAH) (Kamradt, 2023) , and traditional tasks such as GSM8k (Cobbe et al., 2021) and MMLU (Hendrycks et al., 2020), measuring total memory through peak activation memory, KV memory, and model memory, as well as task accuracy. Our contributions are:

1. **KV Pareto Framework:** A KV optimization Pareto framework that systematically maps the trade-off search space between total memory savings and task accuracy.

2. **Joint Optimization Study:** A comprehensive evaluation of chunked prefill, KV cache quantization and AWQ weight-only quantization across multiple KV cache quantization granularities and precisions, identifying Pareto-optimal configurations using LongBench.

3. **KV Pareto Validation:** Validation our framework's selected frontiers on NIAH, showing strong task performance even at 20-32k context lengths, as well as MMLU, GSM8k.

2 Related Works

2.1 KV Quantization

KV quantization reduces the precision of stored key and value tensors, thereby lowering memory usage (Li et al., 2024). For example, KIVI (Liu et al., 2024b) and KVQuant (Hooper et al., 2024) introduce tuning-free asymmetric quantization schemes that apply per-channel and per-token quantization, achieving up to 2-bit compression. More recently, KVTuner (Li et al., 2025) proposes an adaptive framework that searches for the optimal layer-wise KV quantization precision pairs and demonstrates near lossless 3.25 bit mixed precision KV quantization for mathematical reasoning tasks. Inspired by KVTuner (Li et al., 2025), our KV Pareto framework also considers mixed-precision quantization schemes. While KVTuner (Li et al., 2025) focuses on layerwise adaptability, our KV Pareto framework focuses on additional important quantization hyperparameters such as, quantization scheme (blockwise, per tensor, per token), as well as system-level interactions with other optimizations such as, PC and model quantization.

2.2 Prefill Chunking

Prefill chunking (PC) reduces the peak memory consumption by dividing input prompts into equal-sized segments that are processed sequentially (Agrawal et al., 2023). PC is adopted in inference systems like vLLM (Kwon et al., 2023). Additionally, follow on works such as WiM (Russak et al., 2024) leverage the concept of smaller chunks to improve model reasoning. However, the benefits and effects of PC has not yet been studied in conjunction with model quantization as well as KV cache quantization. We address this gap by analyzing PC at a systems-level, understanding its tradeoffs when combined with KV cache and model quantization.

2.3 Model Quantization

Model quantization algorithms are popular for efficient inference as they significantly compress model size. Popular methods include, GPTQ (Frantar et al., 2023), and AWQ (Lin et al., 2024). GPTQ (Frantar et al., 2023) uses second-order information derived from the Hessian matrix to enable 3-4 bit quantization. AWQ (Lin et al., 2024) preserves accuracy in 4-bit quantization by using activation metrics to identify the most important weight channels, which are then scaled prior to quantization to reduce error. In our framework, we study model memory savings via AWQ (Lin et al., 2024) to provide practical insights for edge deployment on how model compression, with KV quantization and PC, can provide the most memory savings with the least task performance degradation.

3 Background

The KV cache is a crucial component for LLM inference, storing intermediate representations that are used for autoregressive generation. KV cache memory can be represented as the follows: KV Cache Memory $= B \times H \times N \times D \times L \times s$, where B is batch size, H is number of attention heads, N is number of tokens stored in the cache, L is number of layers, D is head dimension and s is the size per element.

At a systems level, KV cache optimizations for edge deployment remain largely unexplored when considering its interactions with other memory-savings optimizations such as weight-only quantization and prefill chunking. Therefore, our work focuses specifically on the *joint* interactions among KV cache quantization, prefill chunking, and model weight quantization, on accuracy and total memory.

3.1 KV Quantization Optimization

KV Quantization reduces the element size s by using lower-precision formats (e.g. int8, int4, int2), allowing memory savings: $M_{KV}^{quant} << M_{KV}^{bf16}$. Additionally, quantizing the KV cache reduces memory bandwidth, improving TPOT during decode. However, KV cache quantization also introduces approximation error $\epsilon_{Q_{KV}}$, which can degrade attention quality and therefore task accuracy.

3.2 Prefill Chunking Optimization

Standard prefill involves processing the entire input M in one pass, which leads to higher peak memory consumption due to the size of the attention computation. The attention weights are computed as: $softmax(\left(\frac{QK^\top}{\sqrt{d_k}}\right))$, where Q, K are the query and key matrices, and d_k is dimensionality of the key matrix (Vaswani et al., 2017). The larger the M, the larger the query matrix Q, and therefore larger the attention computation and associated peak memory. Thus, peak memory during prefill can loosely be approximated as: $M_{peak} \approx M_{atten}$.

PC reduces the peak memory by dividing M into smaller chunks sizes of k « M, and processing each chunk sequentially. This limits the number of queries computed at once, thereby reducing the size of the attention computation: $M_{peak}^{chunked} \approx \max_i(M_{atten}(k_i))$, where k_i represents the number of tokens in chunk i.

3.3 Model Weight Optimization

While KV cache optimization is crucial, model weight quantization is often needed for deploying larger models on edge devices with constrained memory. Therefore, we also consider AWQ (Lin et al., 2024), a SoTA weight-only quantization technique which further reduces the total memory. However, the KV cache quantization error $\epsilon_{Q_{KV}}$ and AWQ weight quantization ϵ_{Q_W} can compound errors, further degrading task performance.

Each type of optimization presents its own set of hyperparameters, and optimizing across these require a systematic framework. Our KV Pareto empirically identifies the Pareto frontier for a given model, characterizing tradeoffs between total memory and accuracy for long-context inference.

4 Methodology

Our KV Pareto Framework finds the Pareto frontiers, per model, considering the trade-offs between total memory savings and task accuracy. KV Pareto is designed to find the frontier, from a systems level, considering not only KV cache quantization schemes, but additionally further memory savings from PC and model-weight quantization. As shown in Figure 1, PC is enabled in the prefill phase where a prompt of length M is segmented into C smaller chunks, such that the size of the Query matrix Q into the multi-head-attention block (MHA) is of size $c_i \in C$, lowering peak memory consumption. KV Cache quantization is enabled both in the prefill and decode phase, and simulated by a quantization-dequantization (QDQ) process to insert quantization error into both the key K and value V states. Lastly, weight quantization can be applied via AWQ for both the prefill and decode phase on all linear layers.

4.1 Prefill Chunking

PC partitions the input into equal-sized chunks, and the KV cache is filled iteratively. As shown in Figure 1, when the prompt is divided into C chunks, the KV cache undergoes C updates. For each chunk $i \in \{1, \ldots, C\}$, the update can be expressed as: $\text{KV}_i \leftarrow \text{KV}_{i-1} + \{K_i, V_i\}$. As mentioned in Section 3.2, PC lowers the size of the attention computation, thereby reducing peak memory consumption. We ran ablations to understand the impact of different chunk sizes, $\{64, 128, 256, 512, 1024\}$, on task accuracy. Appendix B shows minimal accuracy changes from PC, and for consistent comparisons within our framework, we arbitrarily set the chunk size to 256 for all KV Pareto experiments.

4.2 KV Quantization

KV quantization compresses the K and V matrices into a lower bit width. We consider int8, int4 and int2 quantization with mixed-precision variants: $\{k8v8, k8v4, k8v2, k4v4, k4v2, k2v2\}$ and apply signed, asymmetric round-to-nearest (RTN) quantization. Appendix C provides details on RTN quantization using 3 techniques, **per-token groupwise, per-sequence groupwise and per-tensor**. We also apply k-smoothing (Zhang et al., 2025) to improve quantization error. Appendix D shows our ablations on varying group sizes, showing larger models perform best at per-token, group size 64, and smaller models at, per token, group size 32.

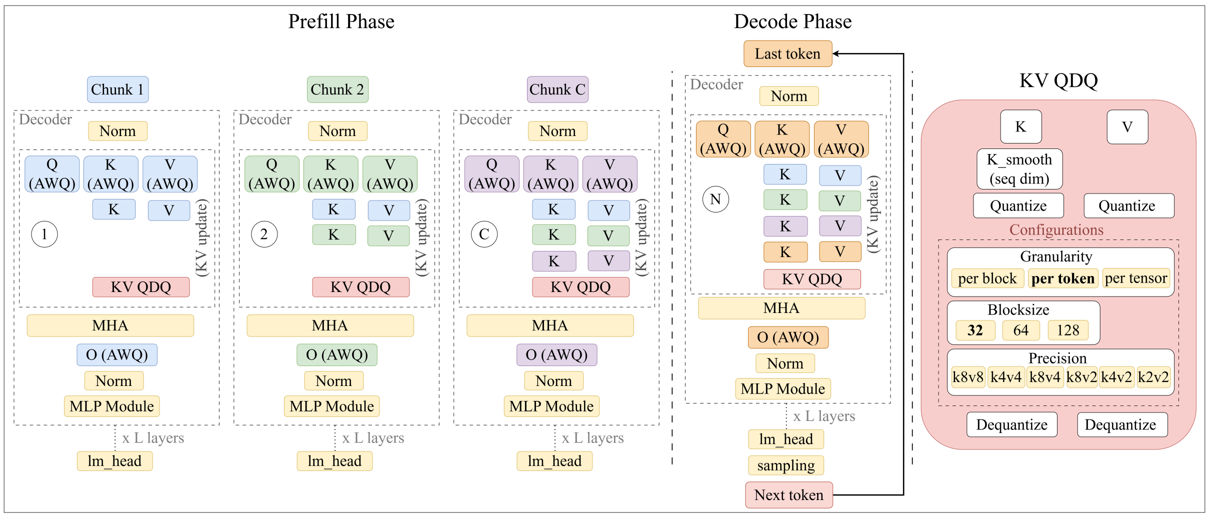


Figure 1: Our KV Pareto Framework, showcasing the integration of prefill chunking (PC), KV cache quantization and model quantization for prefill and decode phases.

K-smoothing Inspired by SageAttention (Zhang et al., 2025), we apply mean smoothing to the K tensor, prior to quantization, to mitigate uneven distributions in K. Appendix E details the K-smoothing process and our ablations reveal k4v4 significantly benefits from K-smoothing.

4.3 Model Quantization

KV Pareto also considers the benefit of model weight compression via weight-only quantization to reduce total memory utilization. Given its SoTA performance, we apply AWQ (Lin et al., 2024), which selectively protects important weight channels based on activation statistics calculated from calibration data. We leverage a robust configuration of AWQ: 4-bit unsigned, asymmetric quantization with group size 128 along the channel dimension.

5 Experimental Design

All experiments are performed on AMD MI-210 and MI-325 GPUs.

Datasets. We evaluate long context performance with Hotpotqa (Yang et al., 2018) and Qasper (Dasigi et al., 2021) from LongBench (Bai et al., 2024a). To ensure KV Pareto does not degrade shorter-context tasks, we also evaluate on GSM8k (Cobbe et al., 2021) and MMLU (Hendrycks et al., 2020). Dataset details are in Appendix F.

Models. We evaluate across diverse LLM architectures, including Qwen2.5-3b and Qwen2.5-7b instruct (Qwen et al., 2025), Llama3.2-3b and Llama3.1-8b instruct (Grattafiori et al., 2024a), and Mistral-7b-instruct-v0.3 (Jiang et al., 2023).

5.1 KV Pareto Frontier Metrics

In our context, a configuration is Pareto dominated if there exists another configuration that achieves equal or better task performance with lesser total memory utilization. Our metrics are:

1. **Total Memory Consumption** We approximate total memory utilization to include *peak memory, KV cache memory, and model memory*. Appendix I provides details.

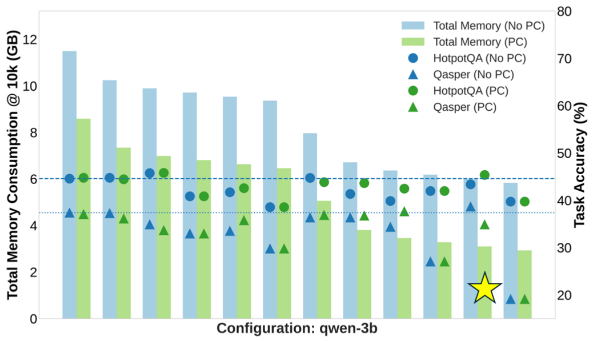
2. **Task Accuracy** We measure how accurate each Pareto configuration is on the LongBench (Bai et al., 2024b) tasks to analyze impact of joint optimizations on task performance.

KV Pareto Validation. We validate our selected pareto-optimal configurations using NIAH (Kamradt, 2023), GSM8k(Cobbe et al., 2021) and MMLU(Hendrycks et al., 2020) to ensure robustness at higher context lengths and non-long context tasks. (Yang et al., 2018).

6 Results

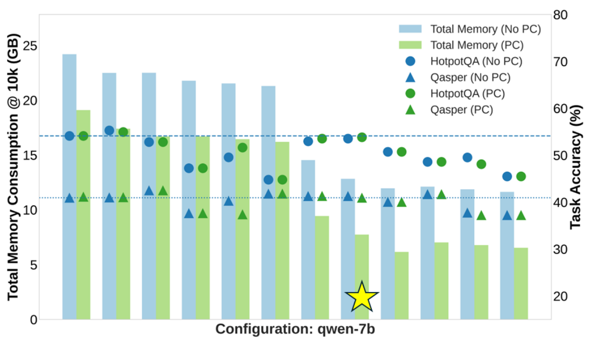
6.1 KV Pareto Frontiers

Figure 2 shows the Pareto-optimal configurations from our framework, measuring total memory consumption at a 10k context length alongside accuracies of two long-context tasks. The pareto frontiers yield 68-78% memory savings with marginal (1-3%) task accuracy drop. The frontier for Qwen2.5-3b-instruct, Mistral-v0.3-7b-instruct and Llama-3.2-3b-instruct (See Appendix G) is w4a16-k4v4,



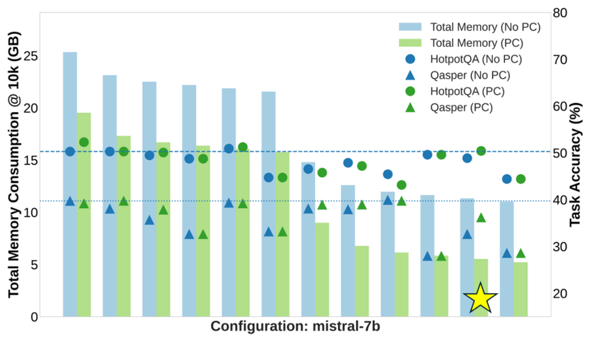
W	16	16	16	16	16	16	4	4	4	4	4	4
A	16	16	16	16	16	16	16	16	16	16	16	16
K	16	8	8	8	4	4	16	8	8	8	4	4
V	16	8	4	2	4	2	16	8	4	2	4	2

(a) Qwen-2.5-3B-I's pareto frontier is w4a16_k4v4



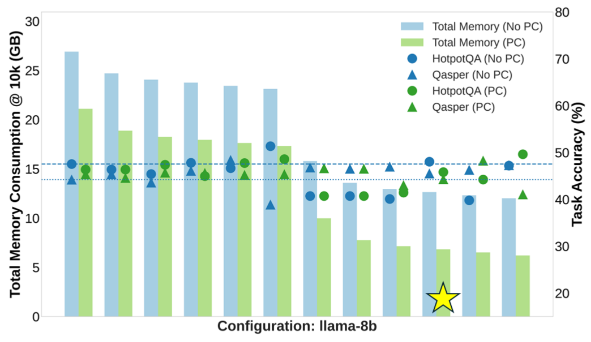
W	16	16	16	16	16	16	4	4	4	4	4	4
A	16	16	16	16	16	16	16	16	16	16	16	16
K	16	8	8	8	4	4	16	8	8	8	4	4
V	16	8	4	2	4	2	16	8	4	2	4	2

(b) Qwen-2.5-7B-I's pareto frontier is w4a16_k8v8



W	16	16	16	16	16	16	4	4	4	4	4	4
A	16	16	16	16	16	16	16	16	16	16	16	16
K	16	8	8	8	4	4	16	8	8	8	4	4
V	16	8	4	2	4	2	16	8	4	2	4	2

(c) Mistral-v0.3-7B-I's pareto frontier is w4a16_k4v4



W	16	16	16	16	16	16	4	4	4	4	4	4
A	16	16	16	16	16	16	16	16	16	16	16	16
K	16	8	8	8	4	4	16	8	8	8	4	4
V	16	8	4	2	4	2	16	8	4	2	4	2

(d) Llama-3.2-8B-I's pareto frontier is w4a16_k8v2

Figure 2: Pareto curves for five models that show the tradeoff between task accuracy and memory consumption, with frontiers shown with a star, and horizontal lines showing baseline (w16a16_k16v16) accuracy.

while for Qwen2.5-7b-instruct it is w4a16-k8v8, and for Llama3.2-8b-instruct it is w4a16-k8v2.

Benefit of Joint Study From Figure 2, we see that PC yields the most reduction in peak memory with minimal changes to task accuracy and AWQ further reduces memory consumption. While AWQ generally causes task accuracy loss, there are instances where it benefits task accuracy. For example, pairing 4-bit weight quantization with k4v4 improves HotpotQA accuracy compared to k8v4. Similarly, combining PC with KV quantization yields higher-than-baseline task accuracies on Qasper, while reducing memory footprint (w16a16-k16v16 vs w16a16-k8v8). We hypothesize this improvement stems from k-smoothing. Our findings stress the importance of our framework, and considering Pareto-optimal configurations at a systems-level for edge deployment to maximize tradeoffs.

6.2 Validation of KV Pareto Frontiers

We validate the efficacy of our selected frontiers by further evaluating them on the following:

GSM8k & MMLU Evaluations Table 1 shows the task accuracy for each pareto frontier on GSM8k and MMLU. Overall, GSM8k shows a greater performance drop compared to MMLU, with AWQ weight quantization having a stronger impact on GSM8k. In general, these results confirm the efficacy of our pareto-optimal configurations for complex, shorter context generation (GSM8k), and standard non-generation (MMLU) tasks, with 1-10% degradation, depending on the model.

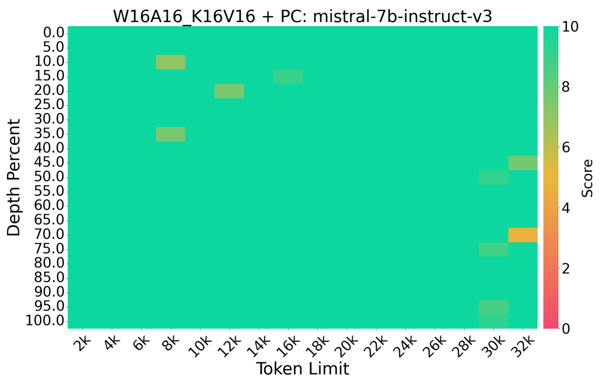
NIAH Evaluations Figure 3 shows the retrieval scores for each depth (y axis) within a given document length (x-axis) for Mistral-v0.3-7b. The w4a16-k4v4 frontier maintains stable performance up to 20k tokens. These results suggest that beyond 20k, additional finetuning may be required to recover task accuracy while preserving memory savings. See Appendix H for more NIAH results.

6.3 Memory Savings Benefit Beyond 30k

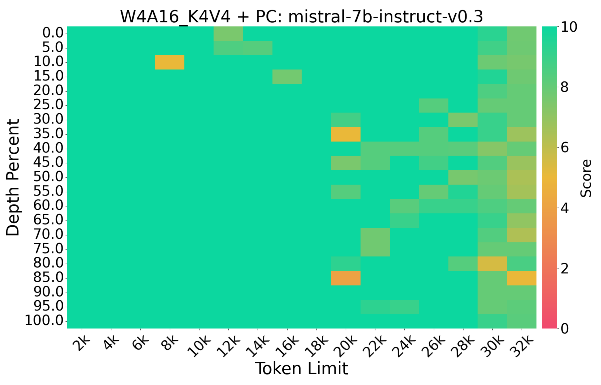
Many real world applications, such as coding (Jiang et al., 2024b) and RAG (Arslan et al., 2024),

PC	AWQ	Qwen2.5-3B-I			Qwen2.5-7B-I			Llama-3.2-3B-I			Llama-3.1-8B-I			Mistral-v0.3-7B-I		
		K/V	gsm8k	mmlu	K/V	gsm8k	mmlu	K/V	gsm8k	mmlu	K/V	gsm8k	mmlu	K/V	gsm8k	mmlu
no	no	16/16	60.95	66.91	16/16	77.48	70.00	16/16	68.76	58.45	16/16	78.92	64.52	16/16	50.79	60.94
yes	no	16/16	61.48	66.87	16/16	77.17	70.07	16/16	68.84	58.66	16/16	76.50	64.50	16/16	50.03	60.98
yes	no	4/4	56.63	65.95	8/8	77.28	69.96	4/4	67.55	57.33	8/2	66.00	60.40	4/4	50.64	60.17
yes	yes	16/16	60.12	61.92	16/16	71.03	69.64	16/16	51.78	56.07	16/16	75.74	64.30	16/16	48.30	60.49
yes	yes	**4/4**	59.21	61.33	**8/8**	71.72	69.64	**4/4**	61.03	57.51	**8/2**	66.00	60.28	**4/4**	43.66	58.77

Table 1: Performance comparison PC, AWQ and selected pareto optimal configurations (bolded).



(a) W16A16_K16V16 retrieval scores for Mistral-v0.3-7B-I



(b) W4A16_K4V4 retreival scores for Mistral-v0.3-7B-I

Figure 3: NIAH performance on baseline (a) and pareto-optimal configurations (b).

require even larger context lengths. To address these practical scenarios, we analyze the benefit of our selected frontiers at extended context lengths, such as 128k tokens. Figure 4 explains the importance of taking a systems-level approach for selecting the pareto frontier, as each additional optimization provides a significant memory savings. For example, a smaller chunk size of 1k saves 23% memory consumption with W4A16-K8V8. Similarly, a smaller KV cache provides an additional 15% memory savings from w4a16-k4v4. Furthermore, for real world deployment, we see the compounded benefit of adding optimized kernels, such as FlashAttention (Dao et al., 2022), resulting an additional 6% memory savings from w4a16-k4v4-Flash. Note, it is imperative to evaluate the extent of task performance degradation under these Pareto-optimal configurations, even at greater context lengths. Given the application dependency, we leave such evaluations for 128k and beyond context lengths for future work.

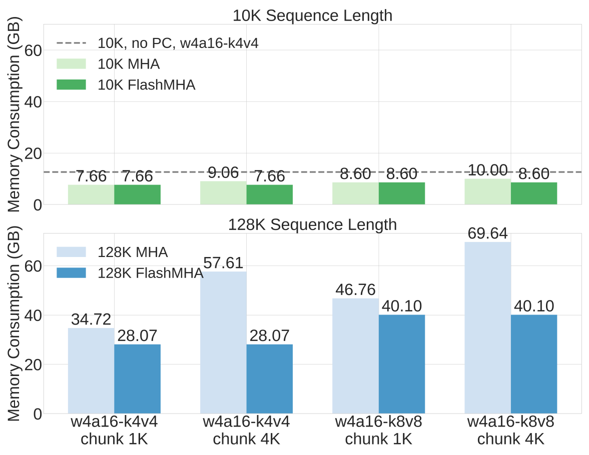


Figure 4: Peak memory consumption on 10k vs. 128k context lengths, comparing SDPA and Flash MHA.

7 Conclusion

We introduce KV Pareto, a systems-level framework for evaluating memory-accuracy tradeoffs in long-context LLMs. KV Pareto jointly considers prefill chunking, 4-bit weight quantization, and KV cache quantization across multiple precision levels, enabling practitioners to identify the pareto-optimal configurations for edge deployment scenarios, where maximum memory savings are needed for efficient inference. We specifically focus on optimization techniques that are lightweight and do not require out-of-box training for scalability to diverse LLMs. Our results highlight that pareto-optimal configurations are model dependent, and that our framework's chosen configurations work well in long context scenarios, as well as shorter context scenarios. Overall, KV Pareto finds optimal configurations with a 68-78% total memory savings with 1-3% long-context task accuracy loss.

Limitations

Our Pareto-optimal configurations currently use a fixed chunk size, focusing on the impact of enabling prefill chunking, varying KV cache quantization and weight quantization. At 128k context length, our results show that chunk size plays a critical role in performance, suggesting that future work should explore dynamic chunk sizing within the KV Pareto frontier search. Additionally, future work should consider improving the robustness of KV cache quantization, beyond using RTN quantization. Specifically, future work should consider the inclusion of Hessian rotations, similar to QuaRot (Ashkboos et al., 2024), and SpinQuant (Liu et al., 2024a), to improve KV cache quantization and push the frontier of KV Pareto. Also, while we evaluate int8, int4 and int2 KV quantization (including mixed-precision variants), future work should expand to other quantization schemes that are adaptive and layer-specific (Zhang et al., 2024; Duanmu et al., 2024). Additionally, while prefill chunking reduces peak memory consumption, it can introduce additional latency due to repeated KV cache writes, compared to a single-pass prefill. Future work should add latency as an additional optimization criteria in KV Pareto and analyze the frontiers' latency tradeoffs. Lastly, future work should consider the generalizability and applicability of KV Pareto to mixed model architectures such as Granite (Granite Team, 2024) or LFM2[1].

References

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. 2023. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *Preprint*, arXiv:2308.16369.

Muhammad Arslan, Hussam Ghanem, Saba Munawar, and Christophe Cruz. 2024. A survey on rag with llms. *Procedia computer science*, 246:3781–3790.

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

[1]https://www.liquid.ai/models

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

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. 2024a. Longbench: A bilingual, multitask benchmark for long context understanding. *Preprint*, arXiv:2308.14508.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024b. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Giulio Corallo and Paolo Papotti. 2024. Finch: Prompt-guided key-value cache compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1517–1532.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *Preprint*, arXiv:2105.03011.

Haojie Duanmu, Zhihang Yuan, Xiuhong Li, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. 2024. Skvq: Sliding-window key and value cache quantization for large language models. *Preprint*, arXiv:2405.06219.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers. *Preprint*, arXiv:2210.17323.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

IBM Granite Team. 2024. Granite 3.0 language models. *URL: https://github. com/ibm-granite/granite-3.0-language-models*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024a. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024b. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *CoRR*, abs/2009.03300.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

AQ Jiang, A Sablayrolles, A Mensch, C Bamford, DS Chaplot, Ddl Casas, F Bressand, G Lengyel, G Lample, L Saulnier, and 1 others. 2024a. Mistral 7b. arxiv 2023. *arXiv preprint arXiv:2310.06825*.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024b. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Gregory Kamradt. 2023. Needle in a haystack - pressure testing llms.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. 2024. A survey on large language model acceleration based on kv cache management. *arXiv preprint arXiv:2412.19442*.

Xing Li, Zeyu Xing, Yiming Li, Linping Qu, Hui-Ling Zhen, Wulong Liu, Yiwu Yao, Sinno Jialin Pan, and Mingxuan Yuan. 2025. Kvtuner: Sensitivity-aware layer-wise mixed-precision kv cache quantization for efficient and nearly lossless llm inference. *arXiv preprint arXiv:2502.04420*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, and 1 others. 2025. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024a. Spinquant: Llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.

Rajeev Patwari, Ashish Sirasao, and Devleena Das. 2025. Forecasting llm inference performance via hardware-agnostic analytical modeling. *arXiv preprint arXiv:2508.00904*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Mohaimenul Azam Khan Raiaan, Md Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. 2024. A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE access*, 12:26839–26874.

Melisa Russak, Umar Jamil, Christopher Bryant, Kiran Kamble, Axel Magnuson, Mateusz Russak, and Waseem AlShikh. 2024. Writing in the margins: Better inference pattern for long context retrieval. *arXiv preprint arXiv:2408.14906*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Preprint*, arXiv:1809.09600.

Jiebin Zhang, Dawei Zhu, Yifan Song, Wenhao Wu, Chuqiao Kuang, Xiaoguang Li, Lifeng Shang, Qun Liu, and Sujian Li. 2024. More tokens, lower precision: Towards the optimal token-precision trade-off in kv cache compression. *arXiv preprint arXiv:2412.12706*.

Jintao Zhang, Jia Wei, Pengle Zhang, Jun Zhu, and Jianfei Chen. 2025. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *International Conference on Learning Representations (ICLR)*.

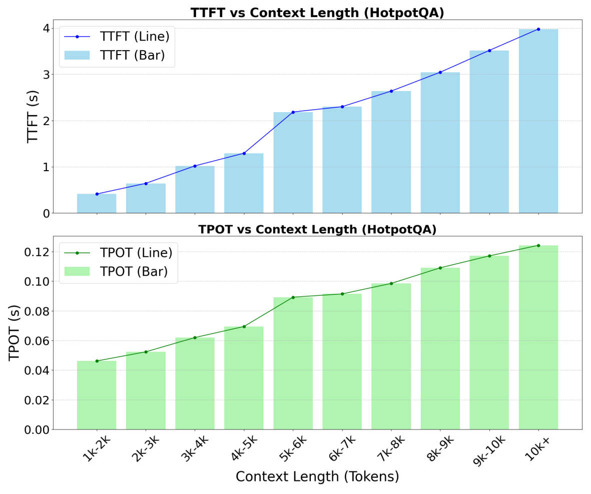


Figure 5: TPOT and TTFT curves on the the HotpotQA dataset, showcasing the bottleneck of a growing KV cache at longer contexts.

Appendix

A KV Cache Growth

Figure 5 show how KV cache growth increases TTFT (time to first token) and TPOT (time per output token) as the context length increases, on a long context task (HotpotQA) (Yang et al., 2018), ultimately increasing inference latency. These increases arise because the prefill phase in LLMs is compute bound and incurs peak memory usage due to KV cache initialization, while the decode phase is memory-bound due to repeated KV cache access (Patwari et al., 2025).

B Prefill Chunking Ablations

To study the effect of variation in chunksize on task performance, we evaluated the long context performance on chunksizes ranging from 64, 128, 256, 512, 1024. Overall, we notice that variation in chunksize shows no impact on performance. In this table, we show ablations using the $w16a16\_k16v16$ configuration. From these results, we select a chunksize of 256 for all further experiments.

C RTN Quantization Details

Round-To-Nearest Quantization (RTN) can be defined with the following. Let the K and V tensors have shape: (B, H, N, D) where B is batch size, H is number of attention heads, N is sequence length, and D is head dimension. A quantized

longbench Mistral v0.2 instruct		
prefill	**hotpotqa**	**qasper**
64	36.62	29.68
128	37.10	29.30
256	36.62	29.42
512	36.76	29.51
1024	36.61	29.21

Table 2: HotpotQA and Qasper scores for different chunksizes for chunked prefill. Variation in chunksize does not affect task accuracy.

tensor q_T via RTN quantization can be defined as:

$$q_T = round\left\lfloor \frac{T}{s} \right\rceil + z \tag{1}$$

where, scale s and zero point z are defined follows, where q_{min} and q_{max} are the integer range of the target quantization:

$$s = \frac{\max(T) - \min(T)}{q_{\max} - q_{\min}} \tag{2}$$

and

$$z = round\left\lfloor q_{\min} - \frac{\min(T_q)}{s} \right\rceil \tag{3}$$

Similarly, for the QDQ process, de-quantization is performed as follows:

$$T \approx \lfloor q_T - z \rceil * s \tag{4}$$

Per-token group-wise Each token's representation is quantized independently across the heads. For each token $t \in [1, N]$, the head dimension D is divided into G groups of equal size and $T \in R^{B \times H \times N \times \frac{D}{G} \times G}$. Scales and zero points are calculated for each group $g \in G$.

Per-sequence group wise Tokens within the same sequence group share quantization parameters. Specifically, the entire sequence dimension N is broken into G groups of equal size and scales and zero points are calculated for each group $g \in G$.

Per-tensor This represents the coarsest granularity where the entire tensor is globally quantized. Specifically, a single scale and zero point is calculated for the entire tensor $T \in R^{B \times H \times N \times D}$.

Figure 6 provides a diagramatic explanation of the aforementioned granularities.

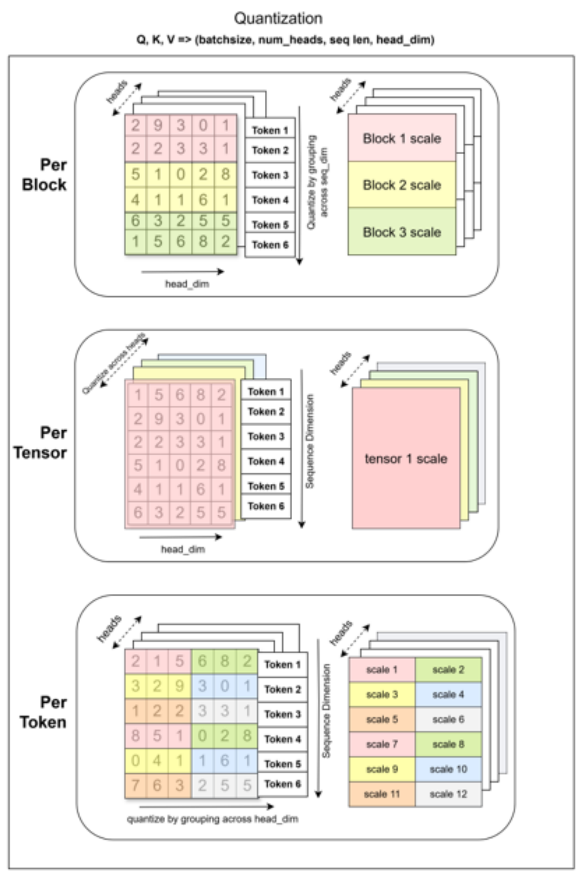


Figure 6: Illustration of KV quantization granularities.

D KV Cache Quantization Ablations

We evaluated multiple KV cache quantization granularities, as outlined in Table 3, including per-token, per-block, and per tensor quantization, considering int4 and int8 precision, to isolate the effect of granularity on task accuracy. For per-token and per-block quantizations, we further ablated group sizes in the range of $\{32, 64, 128\}$. Table 3, shows that per-token quantization yields the best performance compared to per-tensor and per-block. Additionally, a group size of 32 yields the best task performance for Qwen 3B, while a group size of 64 yields the best performance for Mistral 7B. Using this information, we leverage per-token quantization with group size 32 for the the smaller models in KV Pareto (Qwen 3B and Llama3.2 3B), and per token quantization with group size 64 for the larger models (Mistral 7B and Llama3.2 8B).

E K-smoothing Method

K-smoothing is inspired from SageAttention (Zhang et al., 2025) where mean-smoothing is applied to the K tensor. Specifically, we apply the

Granularity	blocksize	KV bits	LongBench		KV bits	LongBench	
			hotpotqa	qasper		hotpotqa	qasper
Mistral 7b							
-	-	bf16	50.28	39.73	bf16	50.28	39.73
Per tensor	-	int8	44.57	36.52	int4	40.05	29.50
Per block	32	int8	44.81	40.74	int4	47.36	38.01
Per block	64	int8	45.21	38.87	int4	54.01	31.95
Per block	128	int8	47.21	39.75	int4	48.03	32.47
Per token	32	int8	47.21	38.94	int4	50.42	36.21
Per token	**64**	int8	**46.57**	**39.53**	int4	**48.21**	**37.03**
Per token	128	int8	46.57	39.53	int4	48.21	37.03
Qwen 3b							
-	-	bf16	44.59	37.38	bf16	44.59	37.38
Per tensor	-	int8	42.19	34.69	int4	33.96	13.92
Per block	32	int8	45.68	35.58	int4	42.45	25.18
Per block	64	int8	41.99	36.36	int4	39.87	26.96
Per block	128	int8	43.49	35.88	int4	31.19	25.01
Per token	**32**	**int8**	**43.63**	**36.75**	int4	**45.37**	**34.91**
Per token	64	int8	41.63	36.24	int4	37.96	27.56
Per token	128	int8	41.39	36.01	int4	40.01	33.26

Table 3: Granularity-wise KV precision and LongBench scores for Mistral 7b and Qwen 3b.

following:

$$\tilde{K}_{b,i,d} = K_{b,i,d} - \frac{1}{L}\sum_{j=1}^{L} Kk_{b,j,d} \qquad (5)$$

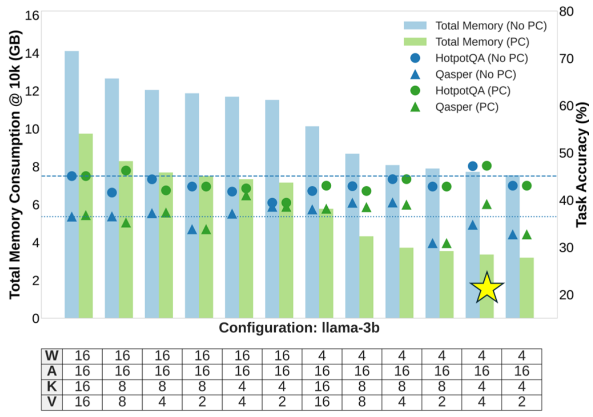
where $K \in bR^{B\times L\times D}$ is the original tensor, $\tilde{K}$ is the mean-centered tensor, B is the batch size, L is the sequence length (dimension over which mean is computed), D is the feature or head dimension, b indexes the batch, i indexes the sequence position, d indexes the feature dimension.

E.1 K-smoothing Ablations

Our ablation studies show that subtracting K_{mean} (averaging along sequence dimension) from K for per-token quantization gives the best configuration for smoothing. Overall, this shows the necessity of K smoothing for lower precision (int4) support.

K smoothing: qwen3b		
precision	averaging across	hotpotqa
int8	No smoothing	44.77
int8	*head_dim*	44.46
int8	*seq_len*	**46.15**
int4	No smoothing	gibberish
int4	*head_dim*	gibberish
int4	*seq_len*	**41.69**

Table 4: Results for K smoothing by subtracting mean across various dimensions, for int4 and int8. Including K smoothing improves results significantly.



W	16	16	16	16	16	16	4	4	4	4	4	4
A	16	16	16	16	16	16	16	16	16	16	16	16
K	16	8	8	8	4	4	16	8	8	8	4	4
V	16	8	4	2	4	2	16	8	4	2	4	2

Figure 7: Llama-3.2-3B-Instruct's pareto optimal search.

F Evaluation Dataset Details

We evaluate KV Pareto on long context datasets from LongBench (Bai et al., 2024b), specifically HotpotQA (Yang et al., 2018) and Qasper (Dasigi et al., 2021). Both Qasper and HotpotQA evaluate multi-document QA and single-document QA using F1 scores. The average prompt length in HotpotQA is 9k, whereas the average prompt length in Qasper is 4k. We additionally evaluate on the Needle-in-a-haystack (NIAH) (Kamradt, 2023) which evaluates text retrieval (needle), in large document scenarios. The NIAH benchmark supports up to 32k context length.

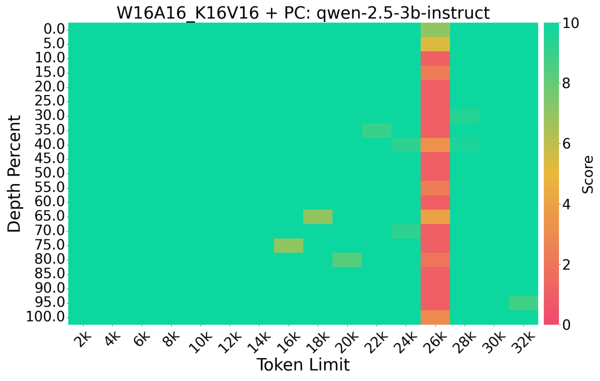
We also evaluate on GSM8k (Cobbe et al., 2021) and MMLU (Hendrycks et al., 2020) tasks which are not considered long context tasks to ensure minimal task performance degradation on these standard evaluation tasks. For both GSM8k (Cobbe et al., 2021) and MMLU (Hendrycks et al., 2020) evaluations, we leveraged LM-Eval-Harness (Gao et al., 2024), and specifically set the evaluation sample size to 50 across all subjects for MMLU.

G Longbench Pareto curves

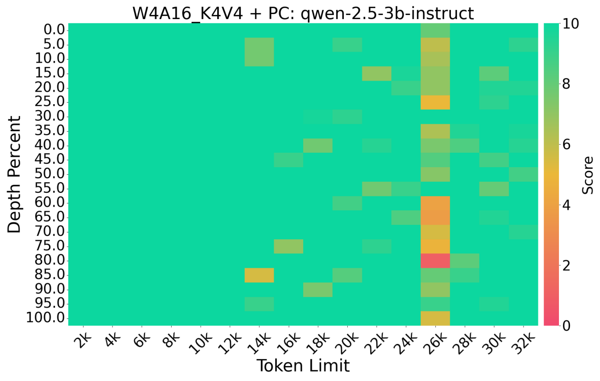
Figure 7 shows an additional pareto search from our KV Pareto framework for the llama-3.2-3b-instruct model. T pareto-optimal solution is at W4A16_K4V4 configuration.

H NIAH results

We also assess information retrieval performance using the Needle in a Haystack benchmark on the Qwen-2.5-3B-Instruct model. Figure 8 illustrates retrieval accuracy up to a 32k context length. Notably, the baseline results show poor performance at



(a) W16A16_K16V16 retrieval scores for Qwen-2.5-3B-I



(b) W4A16_K4V4 retrieval scores for Qwen-2.5-3B-I

Figure 8: NIAH performance on selected configurations.

26k tokens, which may be due to model-specific behavior. The $w4a16\_k4v4$ configuration maintains acceptable performance up to 14k context length.

I Memory Consumption Approximation Details

The memory consumed by the model is a sum of model parameters, KV cache size and peak activation memory. Model parameters are calculated by counting the parameters and corresponding datatypes. KV cache is calculated as described in Section 3. The peak activation memory is dominated by either lm_head layer or by the MHA depending on the operating conditions.

$$\text{mem}_{\text{MHA}} = \begin{cases} n_h M^2 & \text{if SDPA,} \\ n_h c M & \text{if SDPA, PC,} \\ b_q^2 + 2b_{kv}^2 + \Delta & \text{if Flash-MHA.} \end{cases} \tag{6}$$

$$mem_{lm\_head} = M \times vocab\_size \tag{7}$$

$$mem_{peak} = max(mem_{MHA}, mem_{lm\_head}) \tag{8}$$

Model Name	**bf16 baseline memory (GB)**	**Pareto Optimality**	**Memory @ Optimality (GB)**	**Memory reduction %**
Qwen 2.5 3B Instruct	11.49	W4A16_K4V4 + PC	3.10	**73%**
Llama 3.2 3B Instruct	14.10	W4A16_K4V4 + PC	3.36	**76%**
Qwen 2.5 7B Instruct	24.90	W4A16_K8V8 + PC	7.74	**68%**
Llama 3.1 8B Instruct	26.91	W4A16_K8V2 + PC	6.83	**75%**
Mistral v0.3 7B Instruct	24.34	W4A16_K4V4 + PC	5.52	**78%**

Table 5: Pareto-optimal memory configurations for different LLMs.

where n_h denotes number of attention heads, b_q and b_{kv} are block sizes in Flash Attention kernel, M is the total sequence length and c is the chunk size.