Not All Bits Are Equal: Scale-Dependent Memory Optimization Strategies for Reasoning Models

Junhyuck Kim[k], **Ethan Ewer**[w,*], **Taehong Moon**[k], **Jongho Park**[b], **Dimitris Papailiopoulos**[w,m]
[k]KRAFTON, [w]University of Wisconsin–Madison, [b]UC Berkeley, [m]Microsoft Research

Abstract

While 4-bit quantization has emerged as a memory-optimal choice for non-reasoning models and zero-shot tasks across scales, we show that this universal prescription fails for reasoning models, where the KV cache rather than model size can dominate memory. Through systematic experiments across 1,700 inference scenarios on AIME25 and GPQA-Diamond, we find a *scale-dependent trade-off*: models with an effective size below 8-bit 4B parameters achieve better accuracy by allocating memory to more weights rather than longer generation, while larger models achieve better accuracy by allocating memory to longer generations. This scale threshold also determines when parallel scaling becomes memory-efficient and whether KV cache eviction outperforms KV quantization. Our findings show that memory optimization for LLMs cannot be scale-agnostic, while providing principled guidelines: for small reasoning models, prioritize model capacity over test-time compute, while for larger ones, maximize test-time compute. Our results suggest that optimizing reasoning models for deployment requires fundamentally different strategies from those established for non-reasoning models.

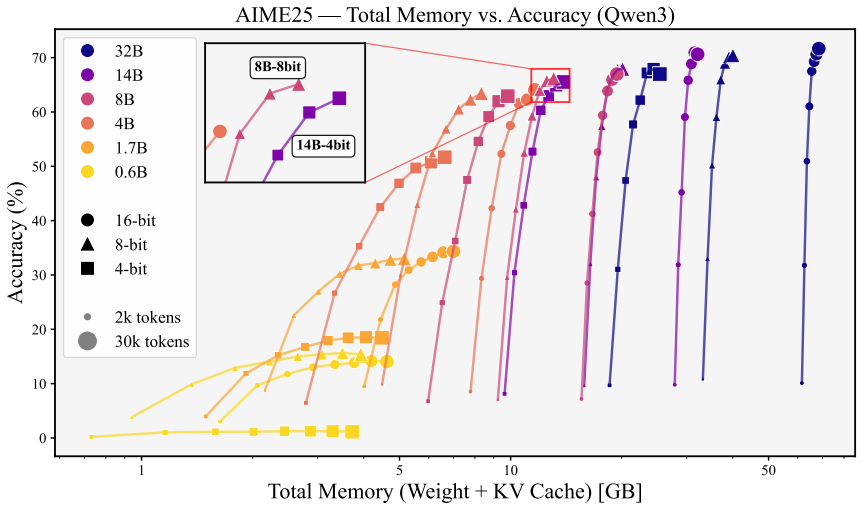

Figure 1: **Memory vs. Accuracy for serial test-time scaling on AIME25.** The plot illustrates the trade-off between pass@1 accuracy and total memory (weights + KV cache) for the Qwen3 family. Model weights are quantized to 4- and 8-bit using GPTQ. Along each curve, the KV cache grows as the generation length increases via budget forcing. For models effectively smaller than an 8-bit 4B, increasing the token budget to saturation is memory-inefficient. Furthermore, for mathematical reasoning, higher weight precision (8- and 16-bit) proves more memory-efficient than 4-bit.

*This work was done during an internship at KRAFTON.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

1 Introduction

Prior memory–performance trade-off studies on Large Language Models (LLMs) for non-reasoning models have focused mostly on compressing model weights, since the model weights generally consume far more GPU memory than the Key-Value (KV) cache [1–3]. Modern reasoning models, however, generate substantially more tokens, causing the proportionally increasing KV cache to become a significant bottleneck. For instance, a Qwen3-4B model with 4-bit weights occupies 2.49 GB, but its KV cache for a 32k-token generation requires 4.42 GB ($\approx 1.8\times$ the weights). This bottleneck is magnified in batched inference: with model weights amortized, the aggregated KV cache becomes the primary memory constraint. With the KV cache becoming a dominant component of memory, it is unclear whether the results established for non-reasoning models still hold for long-generation reasoning tasks.

In this work, we aim to investigate the general principles of memory compression for reasoning models. In addition to the conventional factors of *model size* and *weight precision*, our analysis incorporates three other factors that distinctly affect memory–accuracy trade-offs for reasoning models: *generation length*, *parallel scaling*, and *KV cache compression*. Overall, we ask the question:

Under a fixed memory budget, how should one balance
model size, **weight precision**, **test- and parallel-time compute**, *and* **KV cache compression**
to maximize accuracy on reasoning tasks?

We conduct an empirical study on the Qwen3 model family (0.6B to 32B) [4] across two benchmarks: AIME and GPQA-Diamond. Our investigation spans over **1,700** different scenarios, exploring 4-bit and 8-bit GPTQ weight quantization [2], reasoning token budgets from 2k to 30k, parallel scaling via majority voting with up to 16 samples, and two approaches to KV cache compression: eviction, using R-KV [5] and StreamingLLM [6], and quantization with HQQ [7]. While our findings do not provide specific prescriptions for all tasks or models, we present general principles to consider for memory-efficient reasoning models with minimal loss of accuracy.

Our contributions. In Section 4, we investigate how to allocate memory between model weights and the KV cache under serial test-time scaling. For example, which setting leads to higher accuracy: a 32B 8-bit LLM with less KV cache (*i.e.*, less test-time compute), or a 32B 4-bit LLM with more KV cache? We find that there is *no* optimal strategy that is universal across scale: for models with effective size (parameters $\times$ bits per weight) below 8-bit 4B (≈ 4.2 GB), allocating more memory to model weights yields larger gains, whereas above this threshold, memory is better spent increasing the test-time budget until performance saturates.

We also discover that the choice of weight precision depends on the nature of the task. For knowledge-intensive reasoning, 4-bit weight quantization is broadly memory-optimal, consistent with established findings on the effectiveness of 4-bit or lower precision for zero-shot, non-reasoning models [1, 2, 8]. For mathematical reasoning, however, the higher fidelity of 8- or 16-bit model weights with smaller KV caches often provides stronger performance, suggesting that intricate computational tasks are more sensitive to loss in precision.

Orthogonal to longer generations, increasing the number of generations can yield substantial gains [9], yet its memory efficiency remains underexplored. Parallel scaling via majority voting on top of serial scaling introduces another trade-off: a larger KV cache proportional to group size for higher accuracy, assuming a batched inference setting. This strategy is only more memory-efficient than serial scaling for models with an effective size at or above 8-bit 4B. Interestingly, for such models, the memory-optimal group size also increases with the total memory budget.

In Section 5, we investigate how KV cache compression affects the memory–accuracy trade-off by considering both KV cache eviction and quantization methods. Across model sizes and weight precisions, both eviction and quantization advance their Pareto frontiers beyond the baseline without cache compression. The choice of compression method should be dictated by effective size: eviction offers a better memory trade-off for small models (effective size below 8-bit 4B), while both strategies are competitive for larger models.

Overall, the memory-optimal strategy for reasoning models is *not* universal, but is instead mainly governed by the model's effective size. We summarize our main empirical findings as follows:

1. For models effectively smaller than 8-bit 4B, it is more memory-efficient to allocate memory to more weights than to longer generations, while larger models benefit more from longer generations.
2. 4-bit weights are broadly memory-optimal for the knowledge-intensive task (GPQA-Diamond), while 8-bit or 16-bit weights are more memory-efficient for the mathematical reasoning task (AIME25).
3. Parallel scaling only improves the memory–accuracy trade-off for models effectively larger than 8-bit 4B. The memory-optimal group size increases with the memory budget.
4. Weight quantization alone is not sufficient for memory-optimal reasoning; compressing the KV cache leads to more memory-efficient reasoning.
5. KV cache eviction provides a better memory–accuracy trade-off than KV cache quantization for models with an effective size smaller than an 8-bit 4B model.

2 Background

Weight-only quantization. Weight-only post-training quantization replaces full-precision weights with low-bit representations without retraining, reducing memory usage. Weight-only quantization allows lower bit-widths compared with weight-activation quantization, as it is more robust to quantization error [10]. However, weight-only quantization requires dequantization before multiplying with activations, so it does not reduce computational cost during inference. Any speedup instead comes from reduced memory movement. In this work, we adopt GPTQ [2], a weight-only quantization method that minimizes layer-wise quantization error using a small calibration set and updates weights using inverse-Hessian information.

KV cache quantization. KV cache quantization stores key and value tensors at reduced precision to lower the memory footprint and memory bandwidth during decoding. Unlike weight-only quantization, KV quantization is applied online at inference. During prefill, the KV tensors for the entire input context are quantized and cached in low precision. During decode, the cached tensors are dequantized on the fly for attention computations. Prior work conventionally maintains a small full-precision buffer for the most recent tokens, appending new key and value tensors to this buffer during decoding. In this work, we use per-channel symmetric quantization of both keys and values with an HQQ backend [7], a fast, calibration-free quantization method that is particularly well-suited for online KV cache quantization.

KV cache eviction. On the other hand, KV cache eviction has also emerged as a critical optimization strategy, reducing KV cache size and the cost of attention computation. Specifically for reasoning models, we consider dynamic eviction policies that continuously evict the KV cache during decoding. Early work, such as StreamingLLM [6], employs a sliding-window mechanism that preserves the most recent key and value tensors, in addition to the initial sequence tokens known as the attention sink. More recently, R-KV [5] proposes redundancy-aware selection for reasoning models: it estimates token importance and redundancy during decoding and jointly selects non-redundant, informative tokens to retain, reporting near-baseline accuracy with a small fraction of the KV cache. In Section 5, we study how these eviction policies, together with KV cache quantization, shift the trade-off frontiers.

Test-time scaling. We scope this work to test-time scaling methods that do not rely on external models such as verifiers or process reward models. Reasoning models are typically trained to produce an extended chain-of-thought, continuing generation with planning and reflection to improve performance [11, 12, 4]. We refer to this as serial scaling. Muennighoff et al. [13] introduces budget forcing to scale serial responses beyond the model's natural length for higher accuracy. When the model attempts to stop, a short cue is appended to continue decoding to a specified token budget. Another line of work, parallel scaling, generates multiple independent reasoning trajectories [9]. In its simplest form, without any external model, majority voting selects the final answer as the most frequent among the independently sampled outputs [14]. Further related work is discussed in Appendix A.

3 Experimental Setup

We systematically explore the memory–accuracy trade-offs by measuring how accuracy and memory footprints are affected by five key factors: the number of parameters (N), weight precision (P_W), test-time token budget (T), sampling group size (G, with $G > 1$ indicating multiple samples for majority voting), and KV cache compression strategy (π_{kv}, e.g., eviction or quantization).

The memory cost is given by

$$M \;=\; M_{\text{weights}}(N, P_W) \;+\; M_{\text{kv}}(N, \pi_{\text{kv}}, T, G),$$

where M_{weights} is the memory footprint of the weights, roughly proportional to $N \cdot P_W$. Note that throughout the paper, we use model *size* to refer to the number of parameters N and *effective size* or *scale* to refer to the memory footprint of the weights, M_{weights}. M_{kv} is the KV cache memory, which is roughly proportional to N, G, and T, except when $\pi_{\text{kv}} =$ eviction, where the cost becomes constant beyond a certain token budget. Please refer to Appendix B for the exact memory cost equations and Table 1 for model-specific values.

Table 1: **Memory footprints of evaluated models.**

Model	Model Weight (GB)			KV Cache (GB)			
	4-bit	8-bit	16-bit	2k tokens	18k tokens	30k tokens	30k tokens × 16 samples
Qwen3-0.6B	0.50	0.71	1.40	0.21	1.92	3.20	51.27
Qwen3-1.7B	1.26	1.93	3.78	0.21	1.92	3.20	51.27
Qwen3-4B	2.49	4.19	7.49	0.27	2.47	4.12	65.91
Qwen3-8B	5.68	8.94	15.26	0.27	2.47	4.12	65.91
Qwen3-14B	9.30	15.50	27.51	0.31	2.75	4.58	73.24
Qwen3-32B	18.01	32.66	61.02	0.49	4.39	7.32	117.19

Models. We experiment with the Qwen3 model family [4], which ranges from 0.6B to 32B parameters, offering a wide range of model sizes and thus making it well-suited for a fine-grained systematic study across scales.

Tasks. Experiments are conducted on challenging benchmarks representing complementary difficulty profiles. AIME25 [15] is a competition-level mathematical benchmark that stresses multi-step reasoning. In contrast, GPQA-Diamond [16] emphasizes scientific knowledge and integrated reasoning across domains such as chemistry, biology, and physics [17].

Inference details. Unless otherwise specified, we report accuracy averaged over 32 generations per instance, sampling with temperature 0.6. Following Muennighoff et al. [13], for serial scaling with budget forcing, if generation terminates earlier than the desired token budget, we replace the end-of-sequence token with the prompt `Wait` and continue decoding until the target budget is reached. When the desired budget is met, we inject the prompt `**Final Answer**\n\\boxed{`. We evaluate token budgets from 2k to 30k in 4k increments.

4 Test-Time Scaling with Weight-Only Quantization

When aiming for the best performance under limited memory, how should memory be allocated between model weights and KV cache? Additionally, when allocating space for model weights, is it better to use more parameters at lower precision or fewer parameters at higher precision?

To answer these questions, we study test-time scaling across different model sizes (N) and weight precisions ($P_W \in \{4, 8, 16\}$) by varying the test-time token budget (T). We use GPTQ to quantize models to 4- and 8-bit precision. For this analysis, we fix π_{kv} to keep all cache entries (no eviction, at full precision) and first present results for a sampling group size of $G = 1$. We later discuss parallel scaling with $G > 1$ and other π_{kv} policies.

Figure 1 reveals the Pareto frontier for accuracy versus total memory under serial scaling with a full-precision KV cache. Analyzing the configurations that lie on this frontier provides practical recommendations for optimizing model selection, weight precision, and test-time budgets within fixed memory constraints:

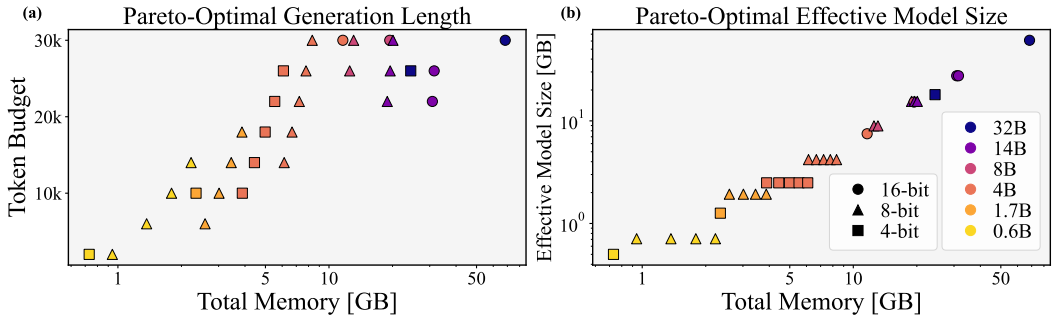

Figure 2: **Composition of Pareto-optimal configurations (AIME25, Qwen3).** The token budget (a) and effective model size (b) are plotted against the total memory budget for configurations on the Pareto frontier from Figure 1. The plots illustrate a strategic shift: at lower memory budgets (<10 GB), increasing effective model size is memory-efficient, whereas at higher budgets, increasing the token budget becomes the dominant strategy for improving performance.

For models effectively smaller than 8-bit 4B, memory is better spent on increasing the effective model size rather than increasing the test-time budget until saturation. While extending the generation budget of a small model is often viewed as a way to trade higher latency for lower memory usage compared to using a large model, our analysis reveals that this is a false economy. In fact, for models effectively smaller than 8-bit 4B, this strategy is often suboptimal in total memory. Figure 2 shows that for memory budgets below 8 GB, the Pareto frontier is advanced primarily by increasing model size, not the token budget. For instance, the 1.7B model in 8-bit with a 6k token budget outperforms the 0.6B model in 8-bit with an 18k token budget. Similarly, the 4B model in 4-bit with a 10k token budget surpasses the 1.7B model in 8-bit with an 18k token budget, demonstrating that choosing a model with a larger effective size is better under a similar memory budget. As our latency analysis confirms (Appendix C), these configurations with larger effective sizes are also faster because end-to-end latency is dominated by the token budget, making the choice to increase the model's effective size strictly dominant.

For large models with an effective size at or above 8-bit 4B, memory is more efficiently used when increasing the test-time budget until performance saturates. In direct contrast to the strategy for small models, extending the generation budget is a more memory-efficient way to improve accuracy for large models. This strategic shift is clearly illustrated in Figure 2, where for memory budgets larger than 10 GB, the best-performing configurations on the Pareto frontier consistently feature token budgets above 20k. In this regime, increasing the token budget becomes the dominant method for improving accuracy.

Finding 1

The memory-efficient allocation strategy between model weights and KV cache is scale-dependent. For models effectively smaller than 8-bit 4B, memory is more efficiently allocated to increasing the effective model size. For models at or above this threshold, it becomes more memory-efficient to increase the test-time budget until performance saturates.

While our analysis mainly assumes a scenario where each inference instance uses the entire model and KV cache, in practice, model weights can be amortized across multiple concurrent generations, fundamentally changing the memory dynamics. Figure 3 examines how memory–accuracy trade-offs change when model weights are shared across multiple concurrent generations. As the theoretical batch size increases, the benefit of smaller model weights diminishes because weight costs are amortized across more generations. We find that the 0.6B model never appears on the Pareto frontier at a theoretical batch size of 16. The 8B and 14B models with 4-bit and 8-bit weight precision and the 4B model with 8-bit and 16-bit precision demonstrate favorable trade-offs in the 1–4 GB memory-per-generation region when the theoretical batch size is 16. Notably, the 8-bit 4B model consistently lies on the Pareto frontier for the 1–2 GB region.

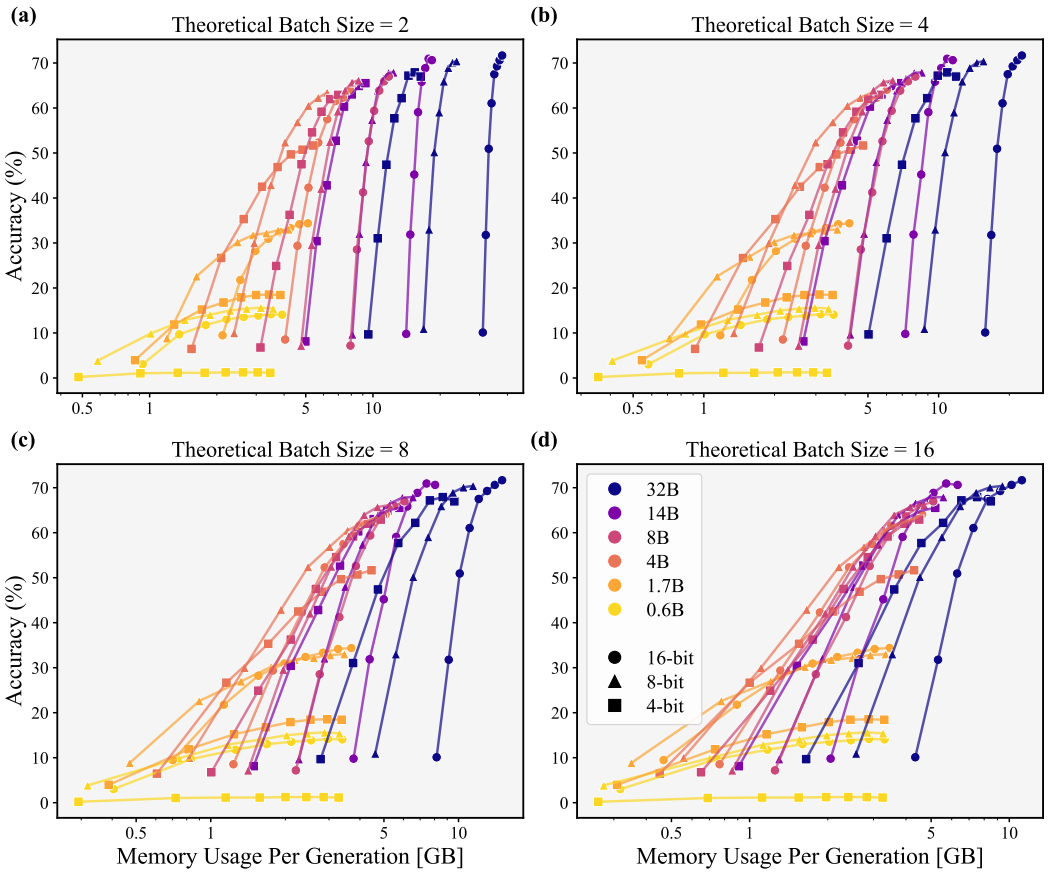

Figure 3: **Memory vs. Accuracy under different theoretical batch sizes (AIME25, Qwen3).** Each subplot shows memory-per-generation vs. accuracy for different theoretical batch sizes, where model weight memory is amortized across concurrent generations. The Pareto frontier shifts as batch size increases, revealing how model weight amortization affects the optimal memory allocation strategy.

The memory-optimal weight precision is task- and size-dependent. Returning to the trade-offs in a single-batch inference setting (Figures 1 and 2), our findings show that for mathematical reasoning tasks, 4-bit weight quantization is consistently memory-inefficient. On the AIME25 benchmark, 8-bit is memory-optimal for small models ($N \in \{0.6\text{B}, 1.7\text{B}\}$), as the performance gains from reallocating memory saved by 4-bit quantization to a larger token budget are insufficient to compensate for the accuracy loss. This inefficiency of 4-bit persists at larger N, where 8-bit and 16-bit configurations achieve higher accuracy at comparable memory. This is shown in Figure 2 (b), where 8- or 16-bit weights are most often memory-optimal along the frontier for memory budgets larger than 6 GB. Notably, the 8B model in 8-bit consistently outperforms the 14B model in 4-bit (Figure 1), and the 32B model in 4-bit is strictly dominated by both the 14B model in 8-bit and the 8B model in 16-bit. Such findings are in direct contrast to Dettmers and Zettlemoyer [1]. However, we do find that for knowledge-intensive tasks, 4-bit quantization is broadly memory-optimal. As shown in Figure 4 for GPQA-Diamond, the frontier shifts to favor lower precision. This suggests that different task types place different demands on model parameters. Mathematical reasoning may rely on numerical precision within the weights, which is damaged by aggressive 4-bit quantization. On the other hand, knowledge-intensive tasks prioritize maximizing the number of parameters to increase knowledge capacity, making large 4-bit models more memory-efficient.

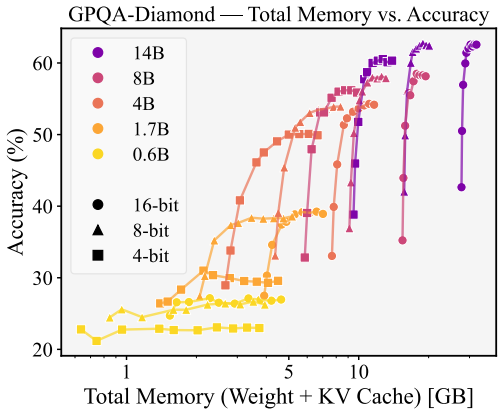

Figure 4: **Memory vs. Accuracy on GPQA-Diamond (Qwen3).** The memory–accuracy trade-off for serial scaling on GPQA-Diamond. Total memory is the sum of model weights and KV cache. Points along each curve represent increasing token budgets. 4-bit weights are broadly memory-optimal for knowledge-intensive tasks.

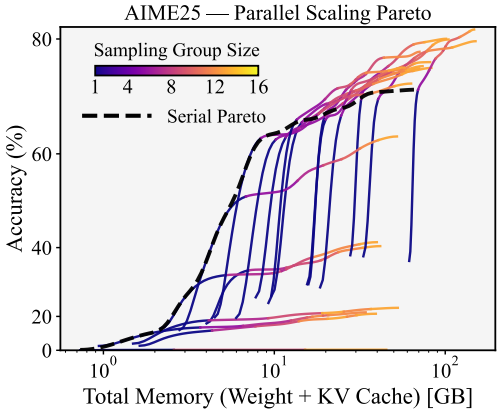

Figure 5: **Effect of parallel scaling on the Pareto frontier (Qwen3).** Each colored curve represents the Pareto frontier for a specific model size and weight precision, obtained by increasing the sampling group size, G. The Pareto frontier for serial scaling ($G = 1$) across all models is shown as a dotted line. Parallel scaling is only effective for large models.

Finding 2

For knowledge-intensive tasks, 4-bit is broadly memory-optimal. For mathematical reasoning tasks, higher precision is required. 8-bit is memory-optimal for small models ($N \in \{0.6\text{B}, 1.7\text{B}\}$), while both 8-bit and 16-bit are competitive at larger numbers of parameters.

In addition to serial scaling by increasing the token budget, we can introduce a parallel scaling axis by increasing the sampling group size (G). Assuming a batched inference setting, the KV cache grows with G, in exchange for higher accuracy. This raises another key question:

When is it more memory-efficient to allocate memory to parallel samples, versus to a larger effective model size or a longer generation length?

The effectiveness of parallel scaling is scale-dependent. For systematic evaluation, we use budget forcing to control the token budget for each of the G parallel samples and use majority voting to select the final answer. Figure 5 shows how parallel scaling affects the memory–accuracy trade-off. The dotted line marks the Pareto frontier from serial scaling alone. Each colored curve represents the frontier for a specific model configuration as the group size, G, is increased (see Appendix D, Figure 10 for a per-model breakdown). For models effectively smaller than 8-bit 4B, parallel scaling is memory-inefficient, as its configurations lie below the frontier established by serial scaling alone. However, for large models, parallel scaling improves the trade-off, and the memory-optimal group size G on the global Pareto frontier increases with the memory budget. While group sizes of $4 \leq G < 8$ are memory-optimal in the 16.4–28.9 GB range; for budgets above 28.9 GB, the frontier is pushed by even larger groups ($G \geq 8$).

Finding 3

For models effectively smaller than 8-bit 4B, serial scaling alone provides a better memory–accuracy trade-off than parallel scaling. For models effectively larger than this threshold, parallel scaling improves the trade-off, and the memory-optimal group size G on the global Pareto frontier increases with the memory budget.

5 Test-Time Scaling with Weight and KV Cache Compression

Our analysis so far shows that while allocating more tokens generally improves accuracy, it is not always memory-efficient, especially for effectively small models where the KV cache can dominate total memory. While compressing the KV cache via quantization or eviction can reduce this footprint, it comes at a potential accuracy cost. This raises the following question:

How do KV cache compression strategies—eviction and quantization—alter the overall memory–accuracy trade-off, and which approach leads to stronger reasoning?

To answer this, we evaluate both compression strategies across model sizes and weight precisions. For eviction, we use R-KV with target KV budgets of 2k, 4k, and 8k tokens. For KV cache quantization, we use symmetric per-channel quantization to 2-, 4-, and 8-bit precisions with a group size of 64 and a full-precision residual buffer of 128 tokens. The results are averaged over 8 generations per instance. We first show that both methods are broadly beneficial and then provide a detailed analysis to determine which strategy is optimal under different conditions.

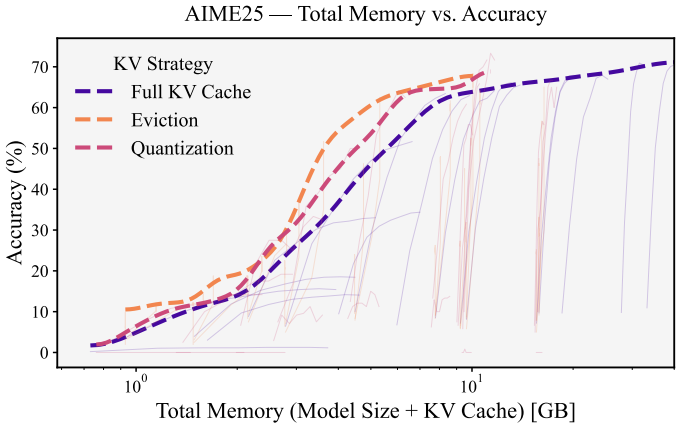

Figure 6: **Memory vs. Accuracy by KV cache compression strategy (AIME25, Qwen3).** The plot shows the Pareto frontiers of KV cache compression across model sizes and weight precisions under serial scaling with budget forcing. Eviction uses R-KV with token budgets of 2k, 4k, and 8k. Quantization is symmetric per-channel (group size 64) at 2-, 4-, and 8-bit. Faint background lines show curves for individual (model size, weight precision, KV strategy) configurations. Both compression strategies consistently improve the memory–accuracy trade-off.

KV cache eviction and quantization consistently advance the Pareto frontier across all tested model sizes and weight precisions. Our first key finding, illustrated in Figure 6, is that the aggregate Pareto frontiers for both quantization and eviction decisively advance beyond a baseline without compression for models with 4-bit, 8-bit, and 16-bit weights. This improvement demonstrates that these strategies enable either higher accuracy at the same memory budget or the same accuracy at a lower memory cost, regardless of the model weight precision. The benefits are especially pronounced in the low-memory regime below 10 GB, where smaller models are most constrained by the KV cache. This indicates that even when model weights are aggressively compressed, the KV cache contains significant redundancies that can be exploited. Our results, therefore, establish KV cache compression as an essential and broadly beneficial strategy for the memory-efficient deployment of reasoning models.

Finding 4

Weight quantization alone is not sufficient for memory-optimal reasoning. KV cache compression advances the memory–accuracy frontier across all weight precisions.

Having established that KV cache compression is broadly beneficial, we now analyze which compression strategy, quantization or eviction, is preferable for a given model size N and weight precision

P_W. Figure 7 shows the resulting memory–accuracy trade-offs, where each strategy shapes the curves differently. Quantization reduces the memory cost per token, shifting the curves leftward, typically with some accuracy degradation. Eviction, in contrast, enforces a fixed memory ceiling for the KV cache, resulting in characteristic vertical curves where accuracy improves while memory usage remains constant.

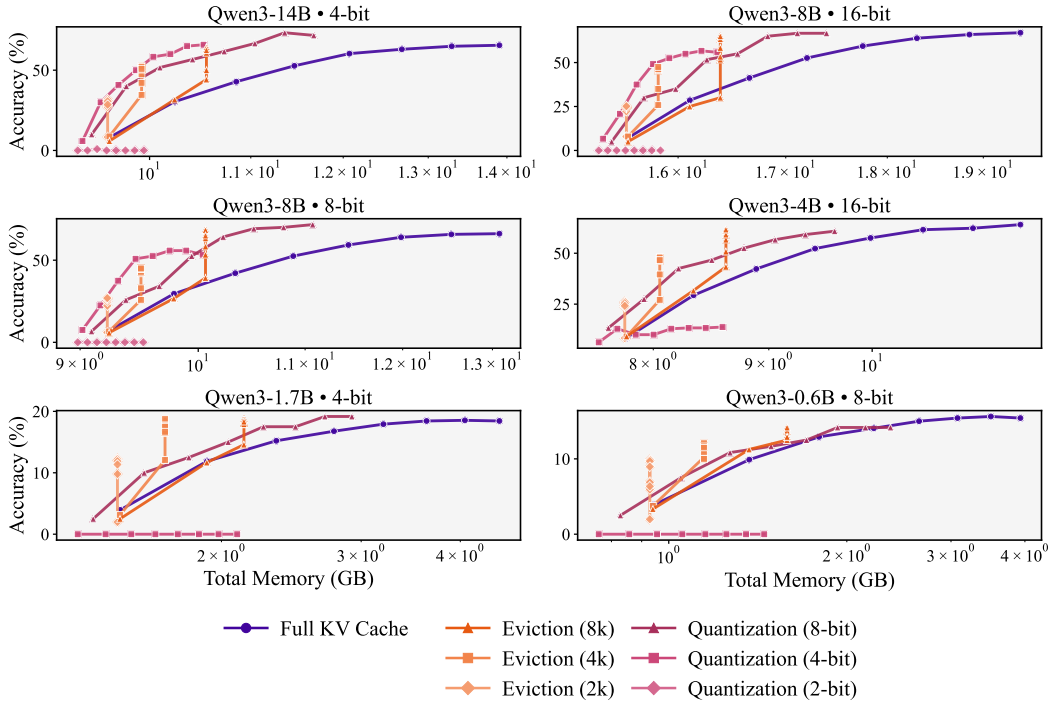

Figure 7: **Per-model Memory vs. Accuracy by KV cache strategy (AIME25).** Each plot illustrates the memory–accuracy trade-off for a single model size and weight precision, comparing a full KV cache baseline against R-KV eviction and symmetric per-channel quantization. Points along each curve represent an increasing number of processed tokens via budget forcing.

Eviction is more effective than quantization for small models. For models with an effective size smaller than an 8-bit 8B model, eviction consistently provides the best memory–accuracy trade-off. As shown in Figure 7 for the full-precision 4B model, eviction with an 8k token budget maintains near-lossless in maximum accuracy while substantially reducing total memory. This observation holds across all weight precisions for the 4B model (see Appendix E, Figure 12 for these results). In contrast, aggressive 4-bit KV cache quantization causes a significant drop in accuracy at these small effective sizes. This suggests that effectively small models are more sensitive to the numerical errors introduced by quantization, whereas eviction preserves the full precision of a smaller, more critical set of tokens. For instance, on the 1.7B model with 4-bit weight precision, eviction achieves the best memory trade-off while maintaining high accuracy, whereas an 8-bit quantized KV cache, while effective, requires significantly more memory to reach a similar performance level.

Quantization becomes competitive with eviction for large models. For models with an effective size larger than an 8-bit 8B model, the clear advantage of eviction diminishes as quantization becomes a highly competitive strategy. On the 8B model with 16-bit weights, for example, quantization and eviction achieve comparable memory–accuracy trade-offs. While 4-bit KV cache quantization is competitive, eviction with smaller budgets (2k or 4k) offers a similar trade-off in low-memory regimes. This suggests that large models, with their greater number of effective parameters, are more robust to the precision loss from quantization. However, we find that more aggressive 2-bit quantization still results in a significant loss of accuracy.

Finding 5

KV cache eviction provides a better memory–accuracy trade-off than KV cache quantization for models with an effective size smaller than an 8-bit 8B model. For models at or above this threshold, quantization becomes an increasingly competitive strategy.

6 Conclusion

Under real-world circumstances with fixed memory budgets, deploying reasoning models is ultimately a problem of *where* to spend bytes, and practitioners are presented with a myriad of choices. Our work reformulates test-time scaling around this constraint. We study the trade-offs in allocating memory across model size, weight precision, KV cache compression, token budget, and sampling group size for reasoning models. We find that the memory-optimal inference strategy for reasoning models *cannot be a one-size-fits-all prescription*: instead, it depends on the model's capacity (determined by effective size) and the nature of the task.

For smaller model sizes (typically models under 8B), prioritizing model weights yields better memory–accuracy trade-offs by using higher-precision 8- or 16-bit weights for mathematical reasoning and favoring KV cache eviction over quantization. For larger models, increasing the token budget until saturation and leveraging parallel scaling become the dominant strategies. Importantly, the inflection point where extra KV cache beats extra model weight may change as models become more sophisticated. However, by shifting the focus from FLOPs-based test-time scaling laws to practical memory constraints, our framework and analysis provide general principles for deploying reasoning models effectively.

7 Limitations and Future Work

Our scope is intentionally focused to keep the search space tractable and inference-only. For test-time scaling, we rely on prompt injection for serial scaling and majority voting for parallel scaling, deliberately excluding methods that require external models. Our analysis also does not include a comparative study of different post-training quantization or KV cache eviction algorithms, including those requiring model retraining, such as quantization-aware training. We evaluate on the Qwen3 family, chosen for its broad size range and fixed architecture, and two challenging benchmarks (AIME25 for mathematical reasoning and GPQA-Diamond for knowledge-intensive reasoning). These choices were necessary to maintain a tractable search space, which already spans more than 1,700 experimental configurations, and to focus on self-contained inference strategies, leaving a broader comparison of methods as a clear avenue for future work.

References

[1] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. In *International Conference on Machine Learning*, pages 7750–7774. PMLR, 2023.

[2] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

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

[4] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[5] Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, et al. R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration. *arXiv preprint arXiv:2505.24133*, 2025.

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

[7] Hicham Badri and Appu Shaji. Half-quadratic quantization of large machine learning models, November 2023. URL `https://mobiusml.github.io/hqq_blog/`.

[8] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36:4396–4429, 2023.

[9] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[10] Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. Zeroquant-v2: Exploring post-training quantization in llms from comprehensive study to low rank compensation. *arXiv preprint arXiv:2303.08302*, 2023.

[11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[12] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[13] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

[14] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[15] AIME. AIME Problems and Solutions. `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`, 2025.

[16] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[17] Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, et al. From system 1 to system 2: A survey of reasoning large language models. *arXiv preprint arXiv:2502.17419*, 2025.

[18] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[19] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

[20] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[21] Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, et al. Language models scale reliably with over-training and on downstream tasks. In *The Thirteenth International Conference on Learning Representations*, 2025.

[22] John X Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G Edward Suh, Alexander M Rush, Kamalika Chaudhuri, and Saeed Mahloujifar. How much do language models memorize? *arXiv preprint arXiv:2505.24832*, 2025.

[23] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405*, 2024.

[24] Tanishq Kumar, Zachary Ankner, Benjamin F Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision. *arXiv preprint arXiv:2411.04330*, 2024.

[25] Guhao Feng, Kai Yang, Yuntian Gu, Xinyue Ai, Shengjie Luo, Jiacheng Sun, Di He, Zhenguo Li, and Liwei Wang. How numerical precision affects arithmetical reasoning capabilities of llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 46–85, 2025.

[26] Anmol Mekala, Anirudh Atmakuru, Yixiao Song, Marzena Karpinska, and Mohit Iyyer. Does quantization affect models' performance on long-context tasks? *arXiv preprint arXiv:2505.20276*, 2025.

[27] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[28] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[29] Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, March 2025. URL `https://qwenlm.github.io/blog/qwq-32b/`.

[30] Kimi Team. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[31] Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

[32] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[33] Ranajoy Sadhukhan, Zhuoming Chen, Haizhong Zheng, Yang Zhou, Emma Strubell, and Beidi Chen. Kinetics: Rethinking test-time scaling laws. *arXiv preprint arXiv:2506.05333*, 2025.

[34] Jian Wang, Boyan Zhu, Chak Tou Leong, Yongqi Li, and Wenjie Li. Scaling over scaling: Exploring test-time scaling pareto in large reasoning models. *arXiv preprint arXiv:2505.20522*, 2025.

[35] James Xu Zhao, Bryan Hooi, and See-Kiong Ng. Test-time scaling in reasoning models is not effective for knowledge-intensive tasks yet. *arXiv preprint arXiv:2509.06861*, 2025.

[36] Ruikang Liu, Yuxuan Sun, Manyi Zhang, Haoli Bai, Xianzhi Yu, Tiezheng Yu, Chun Yuan, and Lu Hou. Quantization hurts reasoning? an empirical study on quantized reasoning models. *arXiv preprint arXiv:2504.04823*, 2025.

[37] Eldar Kurtić, Alexandre Marques, Mark Kurtz, and Dan Alistarh. Deployment-ready reasoning with quantized deepseek-r1 models. `https://developers.redhat.com/articles/2025/03/03/deployment-ready-reasoning-quantized-deepseek-r1-models`, March 2025.

[38] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[39] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

[40] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35:30318–30332, 2022.

[41] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023.

[42] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Lifeng Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit llms: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 1, 2024.

[43] Zechun Liu, Changsheng Zhao, Hanxian Huang, Sijia Chen, Jing Zhang, Jiawei Zhao, Scott Roy, Lisa Jin, Yunyang Xiong, Yangyang Shi, et al. Paretoq: Scaling laws in extremely low-bit llm quantization. *arXiv preprint arXiv:2502.02631*, 2025.

[44] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

[45] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

[46] 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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[47] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

[48] Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

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

[50] Junhyuck Kim, Jongho Park, Jaewoong Cho, and Dimitris Papailiopoulos. Lexico: Extreme kv cache compression via sparse coding over universal dictionaries. In *Forty-second International Conference on Machine Learning*, 2024.

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

[52] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[53] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[54] Liyuan Liu, Feng Yao, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Flashrl: 8bit rollouts, full power rl, August 2025. URL `https://fengyao.notion.site/flash-rl`.

A Related Work

Train-time scaling and knowledge capacity. Foundational scaling studies [18–20] establish power-law relationships between model size, data, and loss, yielding prescriptions for compute-optimal training under fixed compute budgets. While these results provide guidance for allocating parameters and tokens during pre-training, they do not consider inference-time compute and hence require new extrapolations [21]. In parallel, capacity-oriented analyses estimate what models can store, either by modeling knowledge as information per parameter or by measuring memorization versus generalization [22, 23]. These views motivate a budget-centric view but leave precision and inference-time trade-offs under deployment constraints unspecified. Bit-normalized studies examine how performance at different precisions scales with total model bits [1] or the amount of training data [24], particularly in zero-shot or few-shot scenarios. Feng et al. [25], Mekala et al. [26] further show that reduced numerical precision can markedly impair arithmetic reasoning and long-context performance unless compensated by a larger model size, indicating interactions between precision, task structure, and context length.

Inference-time methods and scaling laws. Chain-of-thought prompting elicits intermediate steps, and self-consistency improves performance by sampling diverse rationales and aggregating them via majority voting [27, 14]. Modern reasoning models are trained to generate substantially more tokens, yielding significant gains across benchmarks [14, 27, 9, 13, 11, 4, 28, 12, 29, 30]. Test-time scaling laws study how performance changes with increased FLOPs, tokens, or the number of generations, comparing strategies such as majority voting, best-of-n, and verification-based search [9, 31, 32, 13, 33–35]. However, these studies do not capture the impact of compression techniques such as weight-only quantization, which reduces memory and latency without affecting FLOPs. While concurrent works by Liu et al. [36] and Kurtić et al. [37] study quantization in reasoning models, their focus is on accuracy degradation rather than memory trade-offs. Our work is distinct in its memory-centric view: we analyze the trade-offs in allocating a fixed memory budget between model weights and test-time compute (generation length and parallelism), incorporating the full cost of the KV cache. We also broaden the scope of compression techniques to include KV cache eviction.

Efficient inference. Various strategies have been proposed to address challenges in LLM quantization, particularly for handling outliers [2, 38, 3, 39, 40]. Quantization-aware training extends this idea by training models with quantized weights in the forward pass [41–43]. Post-training KV cache compression techniques can be categorized into eviction and quantization. Eviction methods selectively discard less important entries based on different criteria [5, 6, 44–47], while quantization approaches reduce the precision of cached values [7, 48–51].

B Memory Equations

The total memory cost M is the sum of the memory required for the model weights M_{weights} and the KV cache M_{kv}.

Weight Memory. The total memory footprint for weights is the sum of memory for the quantized and unquantized parameters. The general equation is

$$M_{\text{weights}} \approx \underbrace{\left(N_{\text{quant}} \cdot \frac{P_W}{8} + \frac{N_{\text{quant}}}{g_W} \cdot \frac{P_S + P_Z}{8} \right)}_{\text{Quantized Parameters}} + \underbrace{\left(N_{\text{unquant}} \cdot \frac{P_{\text{native}}}{8} \right)}_{\text{Unquantized Parameters}} \quad \text{[bytes]}$$

where N_{quant} and N_{unquant} are the number of quantized and unquantized parameters, respectively, P_W is the low-bit precision for weights, g_W is the group size, P_S and P_Z are the bit-widths for the scales and zero-points, and P_{native} is the native precision of the unquantized layers.

In our specific setup using GPTQ, the large linear layers are quantized, while components such as the token embedding matrix, normalization layer weights, and the final language model head remain in native BF16 precision. For our experiments, we use a group size $g_W = 128$, a scale precision of $P_S = 16$ (FP16), and symmetric quantization, making the zero-point precision $P_Z = 0$.

KV Cache Memory. Without compression, the KV cache memory is given by

$$M_{\text{kv}} = G \cdot T \cdot n_{\text{layers}} \cdot n_{\text{kv\_heads}} \cdot d_{\text{head}} \cdot 2 \cdot \frac{P_{\text{native}}}{8} \quad \text{[bytes]}$$

where G is the sampling group size, T is the number of tokens, n_{layers} is the number of layers, $n_{\text{kv\_heads}}$ is the number of Key/Value heads, d_{head} is the dimension per head, the factor of 2 accounts for both Key and Value tensors, and P_{native} is the native precision of the cache elements in bits (e.g., 16 for BF16).

The memory cost is modified by different KV cache strategies:

- **Eviction:** This strategy reduces the number of tokens stored. The memory cost is

  $$M_{\text{kv}} = G \cdot \min(T, T_{\text{retain}}) \cdot n_{\text{layers}} \cdot n_{\text{kv\_heads}} \cdot d_{\text{head}} \cdot 2 \cdot \frac{P_{\text{native}}}{8}$$

  where T_{retain} is the maximum number of tokens retained by the policy. In our experiments, we use R-KV and test $T_{\text{retain}} \in \{8192, 4096, 2048\}$.

- **Quantization:** This strategy reduces the precision but introduces overhead for quantization parameters. The cost is

  $$M_{\text{kv}} = (G \cdot T \cdot n_{\text{layers}} \cdot n_{\text{kv\_heads}} \cdot d_{\text{head}} \cdot 2) \cdot \left(\frac{P_{\text{kv}}}{8} + \frac{1}{g_{\text{kv}}} \frac{P_S + P_Z}{8} \right)$$

  where g_{kv} is the group size, and P_S and P_Z are the precisions of the scales and zero-points. For our experiments, we use symmetric quantization ($P_Z = 0$) with $g_{\text{kv}} = 64$, $P_S = 16$, and test $P_{\text{kv}} \in \{8, 4, 2\}$.

Below are the architectural details and per-token KV cache sizes for the evaluated models (Table 2).

Table 2: **Architectural specifications and KV cache memory per token.**

Model	n_{layers}	$n_{\text{kv\_heads}}$	d_{head}	**KV Cache (KB/token)**
Qwen3-0.6B	28	8	128	112
Qwen3-1.7B	28	8	128	112
Qwen3-4B	36	8	128	144
Qwen3-8B	36	8	128	144
Qwen3-14B	40	8	128	160
Qwen3-32B	64	8	128	256

C Latency and Throughput Analysis

While we focus primarily on memory–accuracy trade-offs, latency and throughput can be important practical considerations as well. In this section, we analyze how model size, weight precision, and generation length affect both metrics.

Experimental setup. All measurements are performed on a single NVIDIA A100 80 GB GPU using the vLLM framework [52] with FlashAttention [53] as the attention backend. To measure throughput for a given token budget, we sweep a range of batch sizes and record the highest batch size that completes successfully without out-of-memory errors or KV cache preemption.

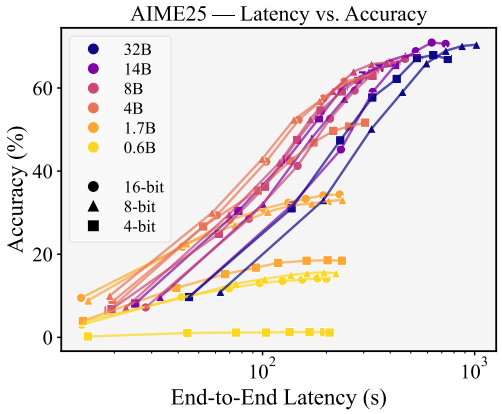

Figure 8: **Latency vs. Accuracy trade-offs (AIME25, Qwen3).** Each curve shows end-to-end latency vs. accuracy for different model sizes and weight precisions with increasing generation length. Generation length emerges as the dominant factor in determining latency, with weight quantization providing more noticeable speedups for large models (14B, 32B).

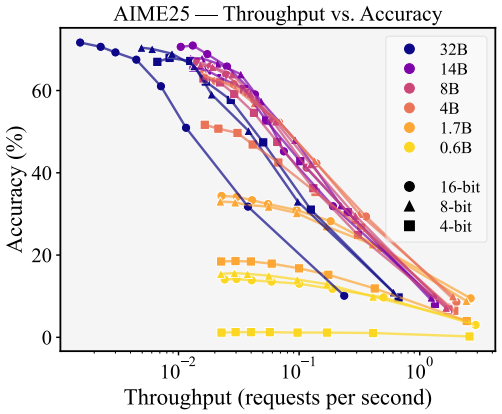

Figure 9: **Throughput vs. Accuracy trade-offs (AIME25, Qwen3).** Each point represents the maximum throughput (requests per second) vs. accuracy under 80 GB VRAM constraints with increasing generation length. While small models can achieve higher batch sizes, the frontier is dominated by configurations that balance model capability with generation efficiency.

We show in Figure 8 that generation length is the dominant factor determining end-to-end latency across all model configurations. The benefit of weight quantization on latency due to reduced memory movement costs is modest for small models (up to 8B) but becomes noticeable for larger models (14B, 32B). For instance, the 14B model takes 137.7 seconds to generate 6k tokens at 16-bit precision, while the 4-bit variant generates 10k tokens in 130.1 seconds. The overall trend for throughput, shown in Figure 9, is similar.

For both latency and throughput, the 4B model with 8-bit and 16-bit precisions consistently demonstrates the strongest speed–accuracy trade-off. Crucially, 4-bit precision is never on the Pareto frontier for any model size, suggesting that for speed-critical applications like reinforcement learning rollouts, higher weight precisions, such as 8-bit, may be the optimal choice [54]. The trade-offs become less favorable at the extremes of the scale: small size models (0.6B, 1.7B) achieve extreme batch sizes up to 160 and 170, respectively, yielding throughput of 2.9 and 2.64 requests per second with 2k-token generations, but their accuracy is fundamentally limited. Conversely, the 32B model performs poorly on throughput due to its slow generation speed and large memory footprint, which restricts batching under an 80 GB VRAM constraint.

Finding 6

For both latency and throughput, 4-bit precision is never on the Pareto frontier, as higher precisions (8-bit and 16-bit) consistently provide a better trade-off between accuracy and speed.

D Detailed Results for Parallel Scaling

Figure 10 presents the per-model plots for the parallel scaling analysis discussed in Section 4.

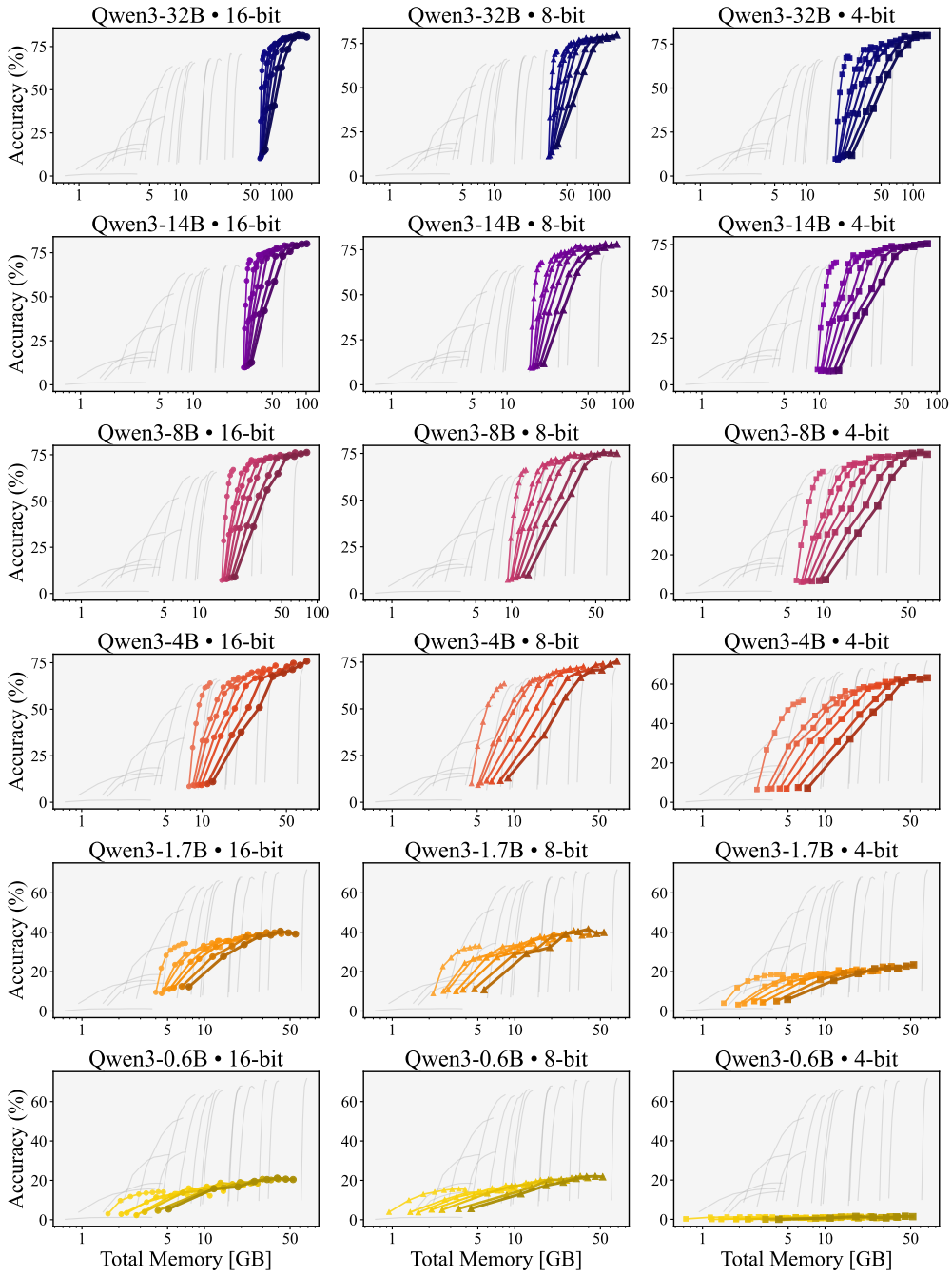

Figure 10: **Per-model Memory vs. Accuracy for parallel scaling (AIME25).** Each plot shows the memory–accuracy trade-off for a single model and weight precision, comparing serial scaling ($G = 1$) with parallel scaling by increasing the sampling group size, $G \in \{1, 3, 4, 6, 8, 12, 16\}$. Points along each curve represent increasing the token budget via budget forcing. Parallel scaling improves the memory–accuracy trade-off only for models effectively larger than 8-bit 4B.

E Detailed Results for KV Cache Compression

Figures 11 and 12 show the per-model results for the KV cache compression analysis discussed in Section 5. For eviction, we also present results for StreamingLLM, where we retain the first $T_{\text{retain}}/2$ tokens and the most recent $T_{\text{retain}}/2$ tokens for a given retention budget T_{retain}.

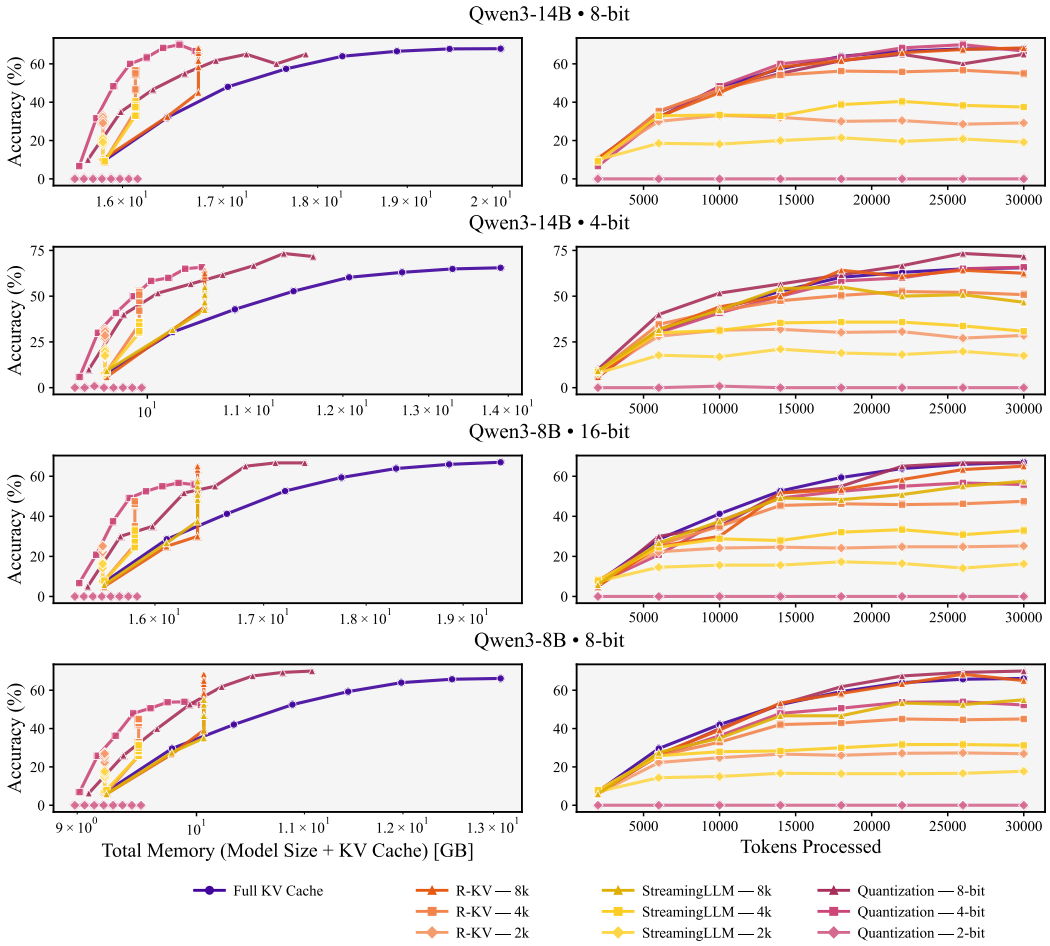

Figure 11: **Per-model Memory vs. Accuracy by KV cache strategy (AIME25, models $>$ 4B).** Each plot shows the memory–accuracy trade-off for a single model and weight precision, comparing KV cache eviction methods (R-KV, StreamingLLM) against KV cache quantization and no compression. Points along each curve represent increasing the number of processed tokens.

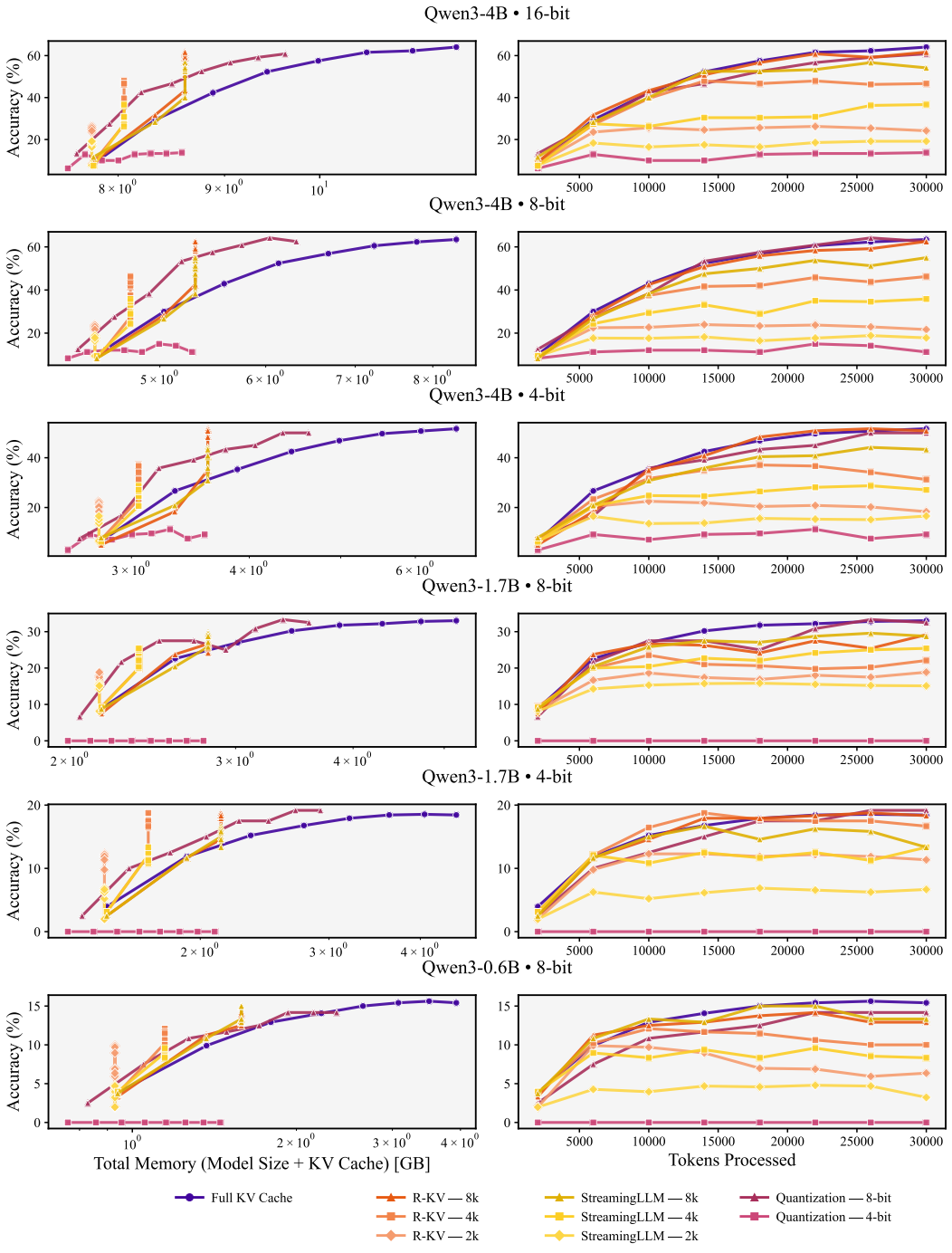

Figure 12: **Per-model Memory vs. Accuracy by KV cache strategy (AIME25, models $\leq$ 4B).** Each plot shows the memory–accuracy trade-off for a single model and weight precision, comparing KV cache eviction methods (R-KV, StreamingLLM) against KV cache quantization and no compression. Points along each curve represent increasing the number of processed tokens.