

Accuracy, Memory, and Decoding Performance of Quantized Language Models for Local Deployment

Maximiliano Moyano-Fontana*, Santiago Ferreiros-Cabrera and Maria Florencia Pollo-Cattaneo

Universidad Tecnológica Nacional, Buenos Aires, Argentina

Abstract

Running a language model locally keeps prompts and data on the user's machine, but a consumer GPU offers 8 to 24 GB of memory, so deployment depends on quantization. Model cards report full-precision accuracy, which describes neither the memory a quantized build needs nor its speed on a given card. This study measures accuracy, memory fit, and decoding performance together for three four-bit checkpoints: Gemma-4 12B and Gemma-4 26B, both quantization-aware, and Qwen-3.6 35B, post-training quantized. Accuracy is measured on GPQA-Diamond, AIME 2026, and LiveCodeBench v6 through an RTX Pro 6000 server, and decoding is done on a desktop with a 16 GiB Radeon RX 6800 under llama.cpp. In that order, the configurations score 70.2%, 74.7%, and 84.8% on GPQA-Diamond, and average 78.9%, 81.1%, and 92.2% on AIME 2026. Restricting LiveCodeBench to the 182 problems released after the Gemma-4 training cutoff lowers every score by more than eleven points, and the drop persists within difficulty bands. Memory fit affects local decoding performance. At a 4,096-token context, both Gemma checkpoints fit entirely in VRAM, with Gemma-4 12B decoding at 53.8 tokens per second and Gemma-4 26B at 103.3, while Qwen-3.6 fits only 25 of its 65 layers on the GPU and runs the rest on the CPU at 32.3 tokens per second. Holding that short-context rate constant, its median mathematics trace of about 30,100 tokens would need roughly 15.5 minutes, an estimate rather than a measurement. Full-precision baselines were not reproduced, so differences against published references cannot be attributed to quantization alone.

Keywords

Local AI, Language model quantization, On-device inference, Model benchmarking, Consumer hardware

1. Introduction

Large language models have become widely adopted in recent years, reaching a 53% population adoption rate within three years of launch [1], with ChatGPT alone serving more than 900 million weekly users by February 2026 [2]. One of the main ways users access these tools is through hosted services, which provide access to state-of-the-art models without requiring users to run the models themselves.

This approach, however, requires prompts and data to be processed on external infrastructure, which is not suitable for many applications. Prompts that contain proprietary source code or medical records leave the user's environment and are processed on infrastructure the user does not control. Latency requirements can be tighter than a network round trip allows. Operating cost is harder to predict when it is metered per token and set by a provider, particularly while much hosted inference is priced below what it costs to serve [3, 20].

Local deployment addresses these cases by keeping data and computation on the user's own machine. However, it introduces new constraints. A consumer GPU typically offers between 8 and 24 GB of memory, far less than the memory required to run these leading models at full precision. As a result, local deployment generally depends on compression, most often through quantization. Prior evaluations report that this process can result in accuracy loss, especially on the complex reasoning tasks that motivate running capable models in the first place [4]. Response time is another important factor. A

ICAIW 2026: Workshops at the 9th International Conference on Applied Informatics 2026, October 13–17, 2026, Orlando, United States

*Corresponding author.

✉ mmoyanofontana@gmail.com (M. Moyano-Fontana); santiago.ferreiros@gmail.com (S. Ferreiros-Cabrera); flo.pollo@gmail.com (M. F. Pollo-Cattaneo)

🆔 0009-0009-2924-1944 (M. Moyano-Fontana); 0009-0008-3098-7340 (S. Ferreiros-Cabrera); 0000-0003-4197-3880 (M. F. Pollo-Cattaneo)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

model that fits within the available memory may still decode too slowly to be usable, which is even more relevant for reasoning models that can generate thousands of tokens per response.

This study evaluates three four-bit quantized checkpoints from the Gemma-4 and Qwen-3.6 families, recent open-weight releases available at sizes relevant to consumer-hardware deployment. It addresses three research questions.

- RQ1.** How do the evaluated four-bit quantized checkpoints perform across scientific reasoning, competition mathematics, and code generation?
- RQ2.** How do memory requirements and model architecture affect decoding performance on consumer-grade hardware?
- RQ3.** How do generation length and truncation interact with decoding performance and benchmark accuracy?

This study compares three deployed four-bit checkpoints rather than isolating the effect of a quantization method. The configurations differ in model family, architecture, scale, and compression strategy, so the differences reported here should be read at the checkpoint level.

The main contributions are: (1) an empirical evaluation of three four-bit quantized configurations across science, mathematics, and code generation; (2) a comparison of server-side and consumer-hardware decoding performance for the same checkpoints; and (3) an analysis of generation length, truncation, memory fit, and CPU offloading as deployment factors.

2. Related work

2.1. Quantization for local deployment

Post-training quantization (PTQ) compresses the weights of a trained model, and optionally its activations and KV cache, by converting its parameters from high-precision to lower-precision formats [4, 5]. This reduces memory usage and improves inference latency, allowing larger models to be deployed on resource-constrained devices.

Quantization-aware training (QAT) takes a different approach. It folds simulated quantization into training, so the weights adapt to low-precision arithmetic [6]. QAT usually recovers more accuracy than PTQ at the same bit width, but at the cost of increased training compute and access to the model.

Recent evaluation work shows that quantization does not affect all tasks equally. The authors suggest that quantization errors may accumulate along chain-of-thought reasoning, making tasks that require complex reasoning more exposed [4].

2.2. Model origin and scale

Two models derived from the same base model can show different levels of tolerance to compression depending on how they are post-trained, while models from different families show distinct sensitivities under the same quantization settings [4].

Model size matters as well. Quantization methods that perform well at larger scales can result in greater performance degradation when applied to smaller models [7].

2.3. Local execution with llama.cpp

Inference engines such as llama.cpp [8] made local execution practical by combining efficient model formats such as GGUF with hardware-accelerated backends for consumer GPUs. A recent evaluation compares the quantization formats supported by llama.cpp on a single model, measuring task performance, perplexity, throughput, model size, and compression [9]. In contrast, the present study fixes the quantization format at four bits and varies the model checkpoint, which enables a practical comparison across distinct deployed checkpoints while holding the four-bit format constant.

Table 1

Coverage of the closest prior evaluations against the present study. Contamination control is marked partial here because the date filter is post-cutoff for Gemma-4 only (§3.2.1).

Study	Models	Bit widths	Consumer HW	Memory/latency	Contam. control
Liu et al. [4]	7 (1.5-70B)	3-8	no	no	no
Wang et al. [7]	2 (0.14-0.5B)	4-8	no	yes	no
Kurt [9]	1 (8B)	3-8	yes	yes	no
Present study	3 (12-35B)	4	yes	yes	partial

2.4. Memory limits and CPU offloading

On consumer hardware, model size is limited by the memory available on the GPU (VRAM). When a model exceeds the available VRAM, it cannot be loaded entirely onto the GPU. One solution to this problem is partial offloading, where some layers remain in VRAM while the remainder are stored in system memory and executed on the CPU. This allows models that exceed the GPU’s memory capacity to run, but degrades performance due to the transfer of data between system memory and VRAM. A recent comparison reports that offloading reduces throughput by more than 90% relative to full-GPU execution for models with 70B parameters and above [10].

2.5. Contamination in benchmarks

The benchmarks used to test the models are often publicly available, so their problems and solutions can be included in the training data of models released afterwards. A score obtained on a contaminated benchmark may reflect memorization instead of the model’s ability to reason.

These concerns have increased as LLMs have become more prevalent and are trained on large amounts of data collected from the Internet. To address this, evaluation has evolved from static benchmarks that relied on test sets that remained unchanged throughout time to dynamic evaluation settings that constantly incorporate recently released problems to reduce the likelihood that evaluation data has appeared in a model’s training corpus [11].

Existing studies have extensively analyzed accuracy degradation under quantization, but fewer evaluations jointly consider reasoning performance, memory fit, decoding throughput, output length, and partial CPU offloading on consumer hardware. Table 1 sets the closest prior evaluations against the present study on those dimensions. This study addresses that practical combination, while recognizing that its limited model coverage does not support general conclusions about quantization methods.

3. Methodology

This section describes the experimental methodology used to benchmark the language models. The models and the rationale for the selection come first (§3.1), followed by the benchmark suite (§3.2), the serving and hardware setup (§3.3), the local deployment setup (§3.4), the evaluation harness (§3.5), the decoding configuration (§3.6), the seeds and reproducibility setup (§3.7), the metrics (§3.8), and the fixed experimental conditions (§3.9).

3.1. Models

Two open-weight families are evaluated, Gemma-4 [12] and Qwen-3.6 [13]. The selected checkpoints span different parameter scales and include both QAT- and PTQ-based quantization. All three checkpoints are the Unsloth Dynamic UD-Q4_K_XL GGUF releases [14]. For Gemma-4, these builds start from the official QAT checkpoints, so the weights were trained with four-bit arithmetic in mind. Qwen-3.6 has no QAT release and is therefore evaluated as a PTQ checkpoint that keeps its multi-token prediction (MTP) module.

Table 2

Evaluated model configurations. All checkpoints are Unsloth UD-Q4_K_XL GGUF builds: gemma-4-26B-A4B-it-qat, gemma-4-12B-it-qat, and Qwen3.6-35B-A3B-MTP. Memory figures are the approximate unified memory requirements published by Unsloth.

Configuration	Family	Parameters	Method	Unified memory required
Gemma-4 26B QAT	Gemma-4	26B (A4B, MoE)	QAT	about 15 GB
Gemma-4 12B QAT	Gemma-4	12B (dense)	QAT	about 7 GB
Qwen-3.6 35B PTQ	Qwen-3.6	35B (A3B, MoE)	PTQ	about 23 GB

Two of the checkpoints are mixture-of-experts models, which the publishers label with an active-parameter suffix. In that notation, A4B and A3B denote four and three billion active parameters, respectively, meaning the parameters through which any single token is routed, against 26 and 35 billion total parameters that are stored in memory even when not used.

Table 2 lists the three configurations together with the approximate unified memory each one needs according to the publisher. The Gemma checkpoints fit on common consumer GPUs, while Qwen-3.6 needs about 23 GB and therefore a larger memory pool or partial CPU offloading on many desktops.

3.2. Benchmark tasks

The suite contains three benchmarks, each chosen to evaluate a different field, namely science, code, and mathematics.

GPQA-Diamond [15] probes graduate-level science. The Diamond subset holds the hardest multiple-choice questions in physics, chemistry, and biology, written so that non-specialists cannot answer them through search. It is evaluated zero-shot with chain-of-thought prompting and letter-answer extraction.

LiveCodeBench v6 [16] probes code generation and reasoning. Problems are drawn from programming contests, and performance is reported as pass@1 against the hidden test suites.

AIME 2026 [17] probes competition mathematics. Answers are integers, so scoring is exact match on the final value. The set holds only 30 problems, so a single problem moves the score by 3.3 points. For this reason, AIME 2026 is run three times per model and the mean and standard deviation across runs are reported.

3.2.1. Contamination control

As discussed in §2.5, a benchmark problem published before a model was trained may already sit in that model’s training corpus, so the possibility of contamination has to be acknowledged when interpreting results.

AIME 2026 needs no control. Its problems were released in February 2026, after every checkpoint evaluated here was trained according to the publishers’ model cards.

GPQA-Diamond questions were published in 2023. While questions were validated to ensure that non-specialists with web access still perform poorly, they could still appear in the models’ training data. Therefore, GPQA-Diamond results should be interpreted with this limitation in mind.

LiveCodeBench supports date filtering, because every problem carries the release date of the contest it came from. The latest dataset contains 1,055 problems from contests held between May 2023 and April 2025. Although publishers report LiveCodeBench v6 results, they do not state whether those scores exclude problems that may have appeared in training data.

The control is applied by matching each problem to its contest date and recomputing accuracy over the problems released after January 2025, which leaves 182 of the original 1,055. Both the full-dataset and the filtered scores are reported. January 2025 is the Gemma-4 training cutoff, so it works as a contamination control for the two Gemma-4 checkpoints. Qwen-3.6 has a later cutoff; therefore, the filtered subset is provided only for comparison purposes and should not be interpreted as a contamination-controlled evaluation for Qwen-3.6.

3.3. Serving and hardware

For the benchmark evaluation, the models are served through the llama.cpp server [8], built with the CUDA backend and deployed on Lightning AI using an NVIDIA RTX Pro 6000 GPU, from the `ghcr.io/ggml-org/llama.cpp:server-cuda` container image. This setup enables higher throughput and reduces the time required to complete the evaluation.

3.4. Local deployment performance

Local decoding performance is measured on a desktop equipped with an AMD Radeon RX 6800 GPU with 16 GiB of VRAM and 31.1 GiB of usable system RAM. The models are executed on Windows using the Vulkan backend of llama.cpp (build `ac2557cb2`, release `b10054`), and throughput is measured with the llama-bench utility, which repeats each test five times and reports the mean and standard deviation. GPU and system memory are sampled during the runs to record how each model is distributed between VRAM and RAM when it cannot fit entirely in GPU memory.

Accuracy and decoding speed are therefore measured on different machines. Accuracy is reported as the observed benchmark performance of each checkpoint under the server-side evaluation configuration, and decoding performance as the rate each checkpoint reaches on the tested desktop, so the two are not combined into a single claim about either machine.

3.5. Evaluation harness

Evaluation is orchestrated with EvalScope [18], which communicates with the llama.cpp server through the OpenAI-compatible API, records per-sample predictions, applies task-specific metrics, and generates structured reports with built-in performance statistics. EvalScope was chosen because it can resume interrupted evaluations from the last completed checkpoint, which makes long-running benchmarks robust to failures. It includes a broad collection of benchmark suites, and it records the configuration of each run alongside the results.

3.6. Decoding and sampling

Each model is evaluated using the sampling parameters recommended by its publisher, so that the measured behavior corresponds to the configuration specified in the model card. Both model families use a temperature of 1.0 and a top-p of 0.95, while top-k is set to 64 for Gemma-4 and 20 for Qwen-3.6.

3.7. Seeds and reproducibility

Every run is executed with an explicit generation seed, recorded in the configuration file the harness writes alongside its results. GPQA-Diamond and LiveCodeBench use seed 42 for all three models. The three AIME 2026 repetitions use seeds 42, 1234, and 123.

Despite the use of fixed seeds, the runs are not guaranteed to be bit-for-bit reproducible. The accuracy experiments are served by llama.cpp under concurrent requests, where request arrival order and dynamic batching can affect execution.

3.8. Metrics

Accuracy. GPQA-Diamond and AIME 2026 use exact-match extraction on the answer letter and the final integer, respectively. LiveCodeBench v6 uses `pass@1` against the hidden tests. GPQA-Diamond and LiveCodeBench figures come from a single run per model, while AIME 2026 figures are the mean of three runs (§3.2).

Pairwise comparison. Differences on GPQA-Diamond and LiveCodeBench are evaluated using the exact McNemar test, based on problems where one of the two models is correct. AIME 2026 is not tested this way because the evaluation set is too small to provide sufficient statistical power.

Full-precision references. Published full-precision results are included only as external references [12, 13]. Since these values were obtained using different evaluation harnesses, prompts, and decoding configurations, the observed differences cannot be attributed exclusively to quantization.

Throughput. Mean output tokens generated per second across a run, as recorded by the evaluation harness on the serving machine. Concurrency inflates these values relative to single-user decoding, so they order the configurations rather than promise local speed.

Token usage. Mean input and output token counts per sample, with the median and interquartile range reported for the per-problem distributions.

Response latency. The wall-clock time of each request on the serving machine, summarized by the median and the 90th and 99th percentiles. Requests were issued without streaming, so the recorded time covers the whole response and includes queueing. These values were measured under concurrency and therefore describe the loaded server, not single-user local decoding.

3.9. Fixed conditions

The remaining conditions are held constant. Each model is served with the chat template packaged in its own GGUF file, and the special tokens and stop sequences are the ones declared in the file metadata.

Every request carries a single user turn. The chain-of-thought instruction and the answer format come from the prompt template EvalScope ships for each benchmark. Dataset splits are the standard partitions distributed by each benchmark. The maximum generation length is fixed at 65,536 tokens for AIME 2026 and GPQA-Diamond and 128,000 tokens for LiveCodeBench.

For every generation, the harness records the reason the server stopped, which is either the end-of-sequence token or the generation cap. Degenerate repetition is detected separately on generations longer than 4,000 tokens, by flagging text where a single line accounts for at least 30% of non-empty lines, or where a repeating cycle of at most eight characters fills at least 30% of the final 4,000 characters.

The full specification of the experiments, including the configuration files, the resolved prompts, and the per-sample predictions, is published in the accompanying repository.

4. Results

This section reports the measured values. Possible causes and their implications are treated separately in §5.

4.1. Accuracy

On GPQA-Diamond (Figure 1), Gemma-4 12B scores 70.2% (139 of 198, 95% CI 63.5-76.1), Gemma-4 26B 74.7% (148 of 198, 95% CI 68.3-80.3), and Qwen-3.6 35B 84.8% (168 of 198, 95% CI 79.2-89.2). Qwen-3.6 35B beats Gemma-4 12B on 37 problems and loses on 8 ($p < 0.001$), and beats Gemma-4 26B on 26 problems while losing on 6 ($p = 0.0005$). The two Gemma checkpoints are not distinguishable, with 23 problems in favor of the 26B model and 14 of the 12B model ($p = 0.19$). All three configurations score lower than their publisher-reported references, under different evaluation conditions: Gemma-4 12B by 8.6 percentage points against its 78.8% reference, Gemma-4 26B by 7.5 points against 82.3%, and Qwen-3.6 35B by 1.1 points against 86.0%.

Table 3 and Figure 2 show the individual runs as correct answers out of 30 for AIME 2026. Gemma-4 12B averages 78.9%, Gemma-4 26B 81.1%, and Qwen-3.6 35B 92.2%, each with a standard deviation of 3.8 points across three runs. Repeated runs of one checkpoint differ by up to two problems. Gemma-4 12B sits 1.4 points above its 77.5% figure, Qwen-3.6 35B 0.5 points below its 92.7% figure, and Gemma-4 26B 7.2 points below its 88.3% figure.

On the full 1,055-problem LiveCodeBench set, every configuration scores above the reference published for its model. Gemma-4 12B reaches 84.7% (95% CI 82.6-86.9), Gemma-4 26B 87.1% (85.1-89.1), and Qwen-3.6 35B 85.6% (83.4-87.6). Restricting scoring to the 182 date-filtered problems lowers all three by more than eleven points, to 72.5% (65.9-78.6), 75.8% (69.8-81.9), and 73.1% (66.5-79.7) respectively.

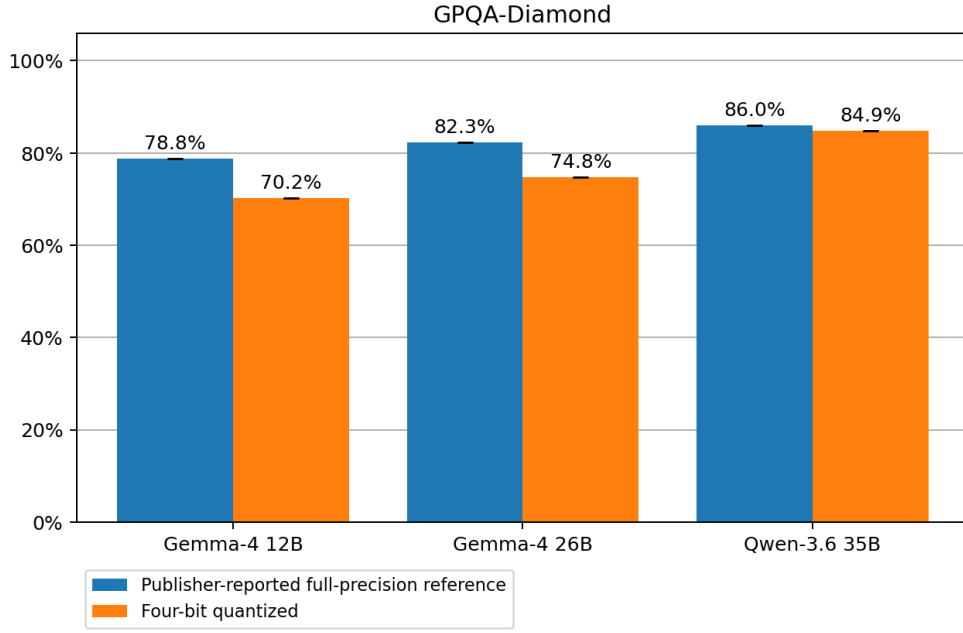


Figure 1: GPQA-Diamond accuracy of each four-bit configuration next to the publisher-reported full-precision reference for the same model.

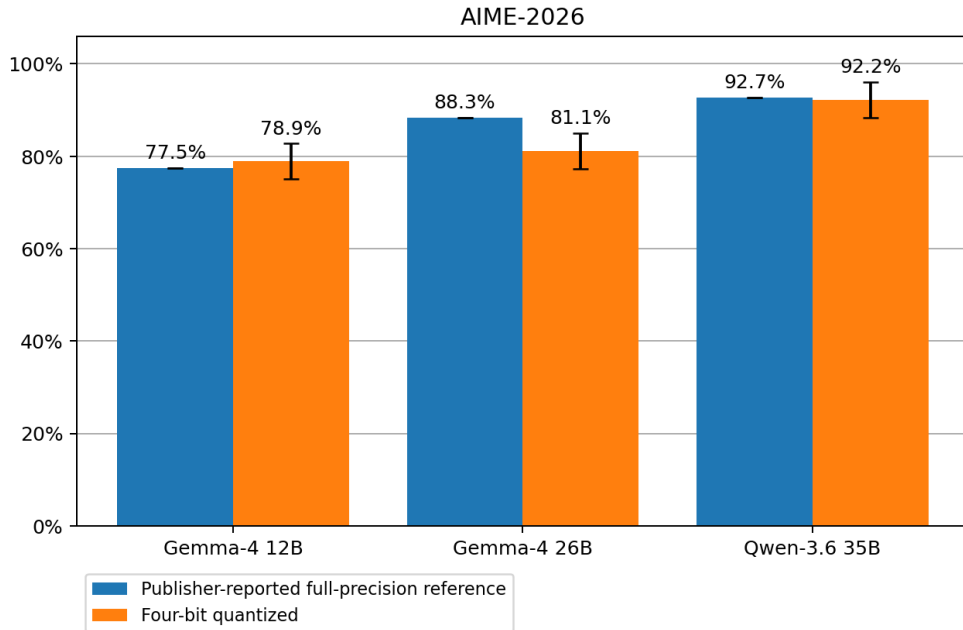


Figure 2: AIME 2026 accuracy. Quantized bars are the mean of three runs of 30 problems and error bars span one standard deviation. Reference bars are publisher-reported full-precision figures.

On the filtered subset, no pair shows a statistically significant difference, with p values of 0.26, 1.00, and 0.44. On the full set, only the comparison between Gemma-4 26B and Gemma-4 12B is statistically significant ($p = 0.006$).

4.2. Composition of the date-filtered subset

The date-filtered subset is not equivalent in difficulty to the original dataset. Problems labeled hard make up 45.1% of the 182 filtered problems, compared with 33.3% of the full 1,055. Easy problems fall from 30.5% to 24.7%. The source platforms also differ slightly, with AtCoder rising from 57.1% to 61.5%

Table 3

AIME 2026 results by run, as correct answers out of 30 problems. Percentages are the mean across the three runs.

Configuration	Run 1	Run 2	Run 3	Mean	SD
Gemma-4 12B QAT	25/30	23/30	23/30	78.9%	3.8
Gemma-4 26B QAT	25/30	23/30	25/30	81.1%	3.8
Qwen-3.6 35B PTQ	29/30	27/30	27/30	92.2%	3.8

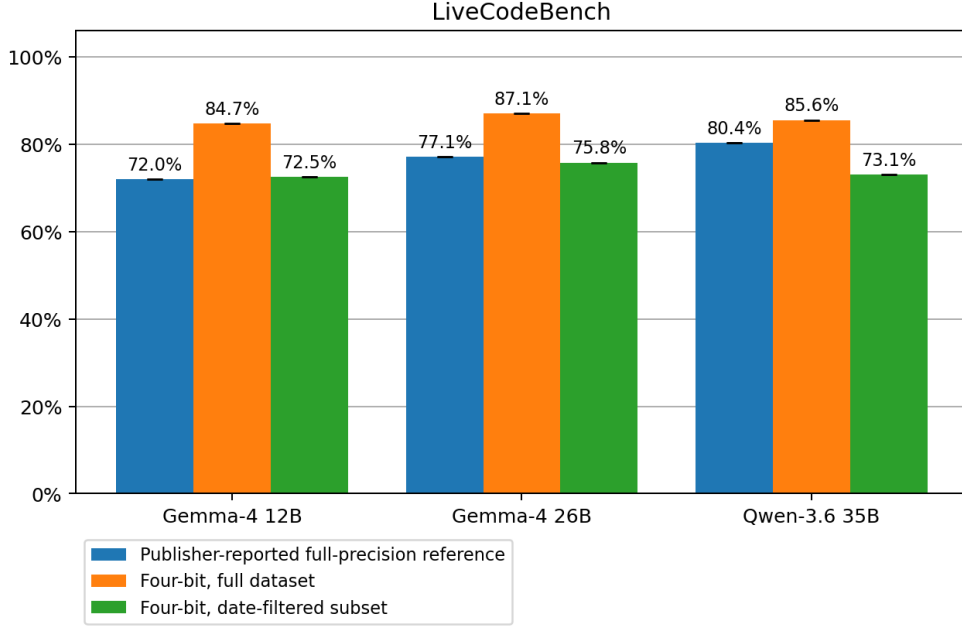


Figure 3: LiveCodeBench pass@1, showing the publisher-reported reference, the quantized models’ scores on the full dataset, and the quantized models’ scores on the subset released after January 2025.

Table 4

LiveCodeBench pass@1 by difficulty band, full dataset against the date-filtered subset, in percent with the 95% Wilson interval in brackets

Configuration	Set	Easy	Medium	Hard
Gemma-4 12B QAT	full	99.1 [97.3, 99.7]	91.4 [88.1, 93.8]	64.4 [59.2, 69.2]
	filtered	100.0 [92.1, 100]	80.0 [67.6, 88.4]	52.4 [41.8, 62.9]
Gemma-4 26B QAT	full	99.4 [97.8, 99.8]	93.5 [90.5, 95.5]	68.9 [63.9, 73.6]
	filtered	100.0 [92.1, 100]	87.3 [76.0, 93.7]	54.9 [44.1, 65.2]
Qwen-3.6 35B PTQ	full	98.4 [96.4, 99.3]	91.4 [88.1, 93.8]	67.5 [62.5, 72.2]
	filtered	97.8 [88.4, 99.6]	80.0 [67.6, 88.4]	54.9 [44.1, 65.2]

and the Codeforces examples disappearing.

Table 4 therefore reports accuracy within each difficulty band, with the interval for each cell. On hard problems, the three configurations fall by 12.0, 14.0, and 12.6 points respectively. On medium problems, they fall by 11.4, 6.2, and 11.4 points. Easy problems stay at or near the ceiling for all three.

4.3. Server-side throughput

Figure 4 reports throughput as average output tokens per second on the serving machine, under concurrent requests. Qwen-3.6 35B is the fastest configuration on every benchmark, at 185 tokens per second on AIME 2026, 137 on GPQA-Diamond, and 155 on LiveCodeBench. Gemma-4 26B runs faster than the dense 12B model on every benchmark, at 122 against 89 tokens per second on GPQA-Diamond. These values do not describe single-user local decoding, which §4.7 reports separately.

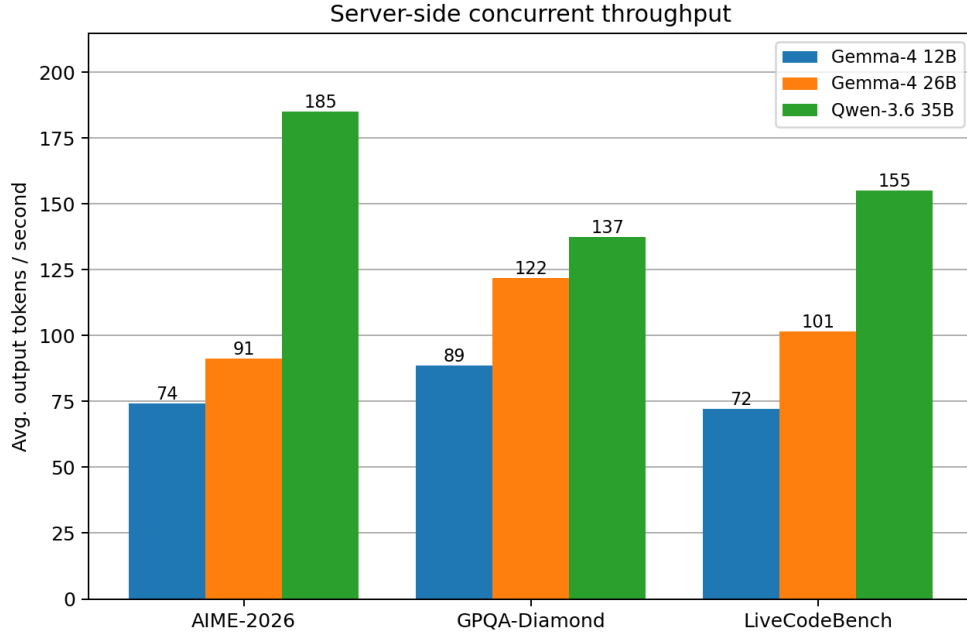


Figure 4: Server-side concurrent throughput in average output tokens per second.

Table 5

Per-request latency on the serving machine, in seconds, under concurrent load. AIME 2026 figures are averaged over the three runs, and the maximum is the slowest request across them.

Model	Benchmark	p50	p90	p99	Max
Gemma-4 12B	AIME 2026	186.0	461.2	813.5	1009.8
	GPQA-Diamond	122.7	261.9	455.0	610.9
	LiveCodeBench	137.0	354.0	1258.3	8296.8
Gemma-4 26B	AIME 2026	112.9	596.1	824.7	1011.8
	GPQA-Diamond	94.9	190.4	359.3	572.3
	LiveCodeBench	99.7	257.4	914.4	1645.1
Qwen-3.6 35B	AIME 2026	165.9	287.9	404.0	600.5
	GPQA-Diamond	39.8	180.5	400.7	417.4
	LiveCodeBench	53.4	322.3	1306.9	2217.9

4.4. Response latency

Table 5 reports per-request latency on the serving machine. Qwen-3.6 35B has a median GPQA-Diamond latency of 39.8 seconds, compared with 122.7 for Gemma-4 12B. Its LiveCodeBench p99 reaches 1,306.9 seconds, with the slowest request taking 2,217.9 seconds. Gemma-4 12B shows the widest spread of all, with a median LiveCodeBench request at 137.0 seconds and a slowest request of 8,296.8 seconds.

Median latency does not always reflect the behavior of the slowest requests. Gemma-4 26B has the lowest AIME 2026 median at 112.9 seconds, but the highest p90 at 596.1 seconds, because a small number of requests produce much longer outputs. Latency therefore depends both on decoding speed and generation length. Since requests were processed concurrently on the serving machine, the measured latency also includes queueing time.

4.5. Stopping behavior

Table 6 reports how generations terminate, following the procedure in §3.9. Across the three benchmarks, between 93.3% and 100% of generations end on the end-of-sequence token. Gemma-4 12B never reaches the generation cap, while that limit is reached in 5.6% of Gemma-4 26B AIME generations, 6.7% of Qwen-3.6 AIME generations, 0.5% of Gemma-4 26B GPQA generations, and 0.3% of Qwen-3.6 LiveCodeBench

Table 6

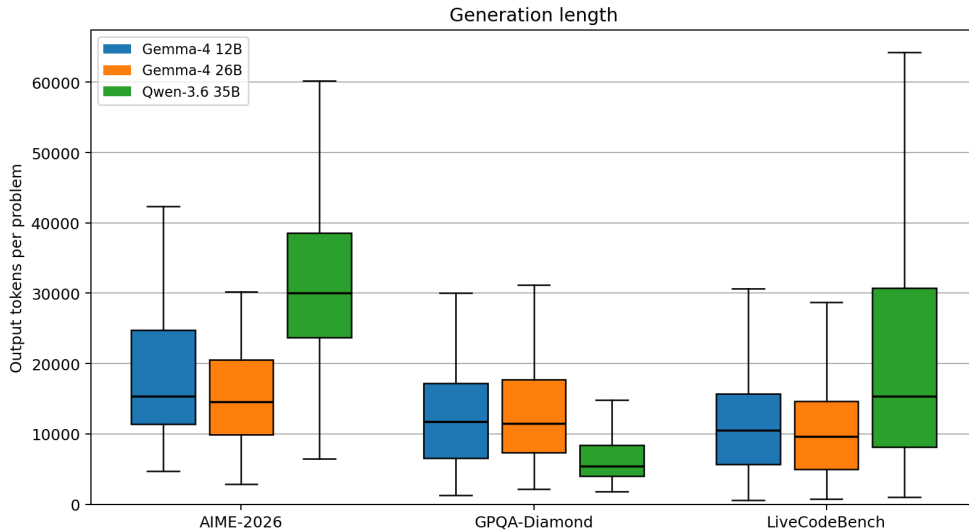
Termination of generations by model and benchmark, pooled across runs.

Configuration	Benchmark	Generations	EOS	Cap	Degenerate
Gemma-4 12B QAT	GPQA-Diamond	198	100.0%	0.0%	0.0%
Gemma-4 12B QAT	AIME 2026	90	100.0%	0.0%	0.0%
Gemma-4 12B QAT	LiveCodeBench	1,055	100.0%	0.0%	0.0%
Gemma-4 26B QAT	GPQA-Diamond	198	99.5%	0.5%	0.0%
Gemma-4 26B QAT	AIME 2026	90	94.4%	5.6%	4.4%
Gemma-4 26B QAT	LiveCodeBench	1,055	100.0%	0.0%	0.0%
Qwen-3.6 35B PTQ	GPQA-Diamond	198	100.0%	0.0%	0.0%
Qwen-3.6 35B PTQ	AIME 2026	90	93.3%	6.7%	0.0%
Qwen-3.6 35B PTQ	LiveCodeBench	1,055	99.7%	0.3%	0.1%

generations. Degenerate repetition appears in 4.4% of Gemma-4 26B AIME generations and 0.1% of Qwen-3.6 LiveCodeBench generations.

4.6. Generation length

Figure 5 shows per-problem output length as box plots, with medians and interquartile ranges pooled over all runs. AIME 2026 produces the longest generations from every model. Qwen-3.6 35B is the extreme case, with a median near 30,100 tokens per problem. On GPQA-Diamond, Qwen-3.6 is the most concise, with a median of about 5,500 tokens, less than half of the 11,700 both Gemma variants generate. Its LiveCodeBench distribution is the widest in the study, with an interquartile range from 8,100 to 30,800 tokens.

**Figure 5:** Boxplot of tokens generated per problem, pooled across runs.

Inspection of the fifteen truncated generations shows two patterns. Eight end in degenerate repetition, generating a single line repeated up to 210 times, an unbroken digit stream, or a two-character cycle. One repeats the correct phrase “Final Answer: 156” 209 times without closing the reasoning. The remaining seven stay coherent to the final token, and they are still evaluating distinct mathematical cases when the generation limit is reached. Six of those come from Qwen-3.6 on AIME problems, where the correct answer is not among the candidates under consideration at the point of truncation.

Table 7

Local performance on the RX 6800 desktop at a 4,096-token context, in tokens per second. ppX is prompt processing of X tokens, tgY is generation of Y tokens, and each pp512+tgN row is the overall rate for a 512-token prompt followed by N generated tokens.

	Gemma-4 26B QAT	Gemma-4 12B QAT	Qwen-3.6 35B PTQ
GPU layers	all (63/63)	all (49/49)	25/65
Layers on GPU	100%	100%	38%
VRAM	15.1 GiB	8.3 GiB	13.7 GiB
System RAM peak	24.8 GiB	16.8 GiB	30.3 GiB
GPU utilization	98%	98%	30%
pp512	1294.0 $\pm$ 4.1	681.9 $\pm$ 0.5	354.6 $\pm$ 12.8
tg128	103.3 $\pm$ 0.2	53.8 $\pm$ 0.3	32.3 $\pm$ 1.2
pp512+tg128	372.3 $\pm$ 1.6	197.7 $\pm$ 0.6	124.8 $\pm$ 1.2
pp512+tg256	252.8 $\pm$ 0.8	133.2 $\pm$ 0.5	84.6 $\pm$ 0.9
pp512+tg512	172.4 $\pm$ 0.5	86.7 $\pm$ 4.4	59.6 $\pm$ 0.2
pp512+tg1024	129.0 $\pm$ 0.3	65.9 $\pm$ 0.3	46.4 $\pm$ 0.3
pp512+tg2048	106.1 $\pm$ 0.4	51.4 $\pm$ 0.1	38.3 $\pm$ 0.8

4.7. Local decoding performance

Table 7 reports decoding speed on the consumer desktop described in §3.4, along with the memory each configuration used while running.

The two Gemma checkpoints load every layer onto the 16 GiB card. Gemma-4 26B uses 15.1 GiB of VRAM and decodes 103.3 tokens per second, while the dense 12B model occupies 8.3 GiB and decodes at 53.8 tokens per second. The mixture-of-experts checkpoint therefore decodes about twice as fast as the dense model, while using about 1.8 times the VRAM. Qwen-3.6 35B does not fit entirely in VRAM, so 25 of its 65 layers are placed on the GPU and the rest run on the CPU. System memory in use rises from 9.6 GiB idle to a peak of 30.3 GiB of the 31.1 GiB available. Decoding reaches 32.3 tokens per second, and prompt processing 354.6 tokens per second, against 103.3 and 1294.0 for Gemma-4 26B.

Combining the measured decoding rates with the trace lengths in Figure 5 gives estimated response times. At 32.3 tokens per second, the median Qwen-3.6 AIME 2026 trace of about 30,100 tokens would take about 15.5 minutes. Gemma-4 26B would generate its median trace of about 15,000 tokens in about 2.4 minutes. On GPQA-Diamond, the median Qwen-3.6 trace of about 5,500 tokens gives about 2.8 minutes, compared with about 1.9 minutes for Gemma-4 26B at 11,700 tokens.

The estimates assume that the decoding rate remains constant, and therefore, they describe the response time on a machine with sufficient memory, rather than achievable times on the tested desktop. The pp512+tgN data in Table 7 also shows that the overall rate decreases as output length increases, so the estimates become less representative for longer generations.

The memory figures above were measured at a 4,096-token context, which is the setting used by llama-bench in Table 7. The traces reported in §4.6 run from about 5,500 to about 30,100 tokens, and a context of that size needs a larger key-value cache than the one measured here. Full GPU residency at 4,096 tokens does not imply that the same checkpoint stays fully resident at 16,000 or 30,000 tokens. Therefore, the residency results should be read as a short-context measurement.

4.8. Summary for deployment

Table 8 summarizes the main results for deploying these configurations on a 16 GiB consumer GPU.

5. Discussion

All three configurations use a four-bit quantized format, but their performance relative to the publisher-reported references varies across benchmarks. On GPQA-Diamond, all three score lower than their references. On AIME 2026, Gemma-4 12B scores higher than its reference. The post-training quantized

Table 8

Deployment summary for the tested 16 GiB RX 6800. Code shows LiveCodeBench pass@1 on the date-filtered subset, GPQA shows GPQA-Diamond accuracy, AIME shows the mean of three AIME 2026 runs, and pp512+tg2048 shows the overall rate for a 512-token prompt followed by 2,048 generated tokens.

Configuration	Fits 16 GiB	Code	GPQA	AIME	pp512+tg2048	Suited to
Gemma-4 12B QAT	yes	72.5%	70.2%	78.9%	51.4 tok/s	smaller GPUs
Gemma-4 26B QAT	yes	75.8%	74.7%	81.1%	106.1 tok/s	the tested GPU
Qwen-3.6 35B PTQ	no	73.1%	84.8%	92.2%	38.3 tok/s	GPUs fitting layers

checkpoint is the closest to its reference on all three benchmarks. These checkpoints, however, differ in model family, architecture, scale, and compression strategy. Therefore, the quantization method cannot be isolated as the only cause of these differences. This aligns with prior findings that the effects of quantization are different for each model and configuration [4].

While every configuration scores above its reference on the full LiveCodeBench, they fall by more than eleven points on the date-filtered subset. As the publisher references were produced under unknown conditions, the difference may come from the evaluation setting, the training-data contamination, or both. When breaking down the filtered subset by difficulty, accuracy also drops in the medium and hard bands from 6.2 to 14.0 points. This could be explained by training-data contamination in the older problems, or by a difficulty gradient that makes the newer hard problems harder than the older ones. Identifying the reason would need a model whose training cutoff precedes both periods.

Memory fit works as a threshold on the tested desktop. Both Gemma checkpoints fit entirely in VRAM, and the mixture-of-experts checkpoint is the faster one despite being the larger model, which is consistent with its lower active parameter count of about four billion. On the other hand, Qwen-3.6 does not fit, so only 25 of its 65 layers run on the GPU and the rest run on the CPU, resulting in much lower GPU utilization and lower decoding speed. This design cannot isolate the threshold, because the checkpoint that exceeds the card is also the one from a different family with a different total parameter count and architecture; kernel implementation and memory-access patterns may contribute as well. Separating them needs a 35B checkpoint that fits. Consumer GPUs commonly ship with 8 to 24 GB, so a buyer needs to know which side of their own memory limit a release falls on. Model cards that listed active parameter counts and quantized memory footprints next to accuracy would answer that question directly.

Qwen-3.6 35B at 32.3 tokens per second is not slow in itself, and it returns a 512-token answer in 15.9 seconds. But partial CPU offloading and long reasoning traces make it unsuitable for low-latency local use on the tested desktop. The median AIME 2026 response would take about 15.5 minutes, at a rate measured over 128 generated tokens rather than over the full trace. On GPQA-Diamond the same model is among the fastest to a finished answer, because there it spends the fewest tokens of the three.

The long generations are probably not caused by the inference server, since a broken template or a wrong stop token would raise cap rates on every benchmark instead of concentrating them in AIME 2026 (Table 6). These results make a systematic stopping-token failure unlikely and suggest that most long traces reflect model behavior under the evaluated prompting and decoding configuration.

5.1. Threats to validity

Internal validity. The full-precision figures are publisher-reported and were not reproduced here. GPQA-Diamond and LiveCodeBench use a single run per model at a temperature of 1.0, so their generation is stochastic and was not repeated. Their intervals and the paired tests built on them therefore cover the sampling error of the problem set, not the variance of repeated generation. Contamination is controlled only where the benchmark allows it, since GPQA-Diamond has no date-based control and the LiveCodeBench filter applies only to Gemma-4. Accuracy and decoding speed come from different machines, which differ in GPU, backend, and execution mode. Local memory and speed were measured at a 4,096-token context, while the accuracy runs produced traces an order of magnitude longer, so the reported residency does not describe the memory footprint of those longer generations.

Construct validity. Three benchmarks do not represent reasoning in general, and benchmark accuracy does not measure usefulness in production. Tokens per second alone does not describe practical inference performance.

External validity. The local measurements come from one specific setup so they are not representative of general consumer hardware. Three checkpoints from two families, at one quantization format each, do not support conclusions about quantization methods or about other model sizes.

Conclusion validity. AIME 2026 has 30 problems and three runs, which limits any statistical conclusion. The exact McNemar tests on GPQA-Diamond and LiveCodeBench compare single runs, so they measure a difference between two particular generations and not a difference that repeated sampling would reproduce. The three filtered LiveCodeBench comparisons return no separation, so the ranking on that subset is a ranking of point estimates only.

6. Future lines

The first line is reproducing the full-precision baselines under the same conditions used here. Running each checkpoint in BF16 or FP16 on the same server, with the same prompts, template, harness, and sampling settings, would allow the publisher-reported values used in this study to be replaced with controlled measurements to directly quantify the accuracy loss associated with four-bit quantization.

A second priority is to compare quantization levels within a single model. Comparing FP16, eight-bit, six-bit, five-bit, and four-bit builds, along with the QAT and PTQ releases, would allow the effect of quantization format to be separated from differences between families.

A third priority is to repeat each experiment across multiple seeds and hardware configurations. Three to five runs per model and benchmark would replace the reported sampling error with estimates that account for generation variance as well, and would let the paired tests reported here be applied to a mean rather than to a single generation.

A fourth is measuring local decoding at the context lengths these models actually reach. The sweep here stops at 2,048 generated tokens, while the median AIME 2026 trace of Qwen-3.6 35B runs past 30,000. Extending llama-bench to tg8192 and beyond, or timing a sample of complete AIME generations on the desktop, would replace the projected response times of §4.7 with measured wall-clock figures.

Finally, prompting techniques could be evaluated to determine how they affect performance. Prompt repetition, which duplicates the query, improves non-reasoning models without adding output tokens or latency [19]. Whether this technique can recover part of the accuracy difference observed under quantization remains open.

7. Conclusions

The evaluated four-bit configurations show that local deployment can be viable for selected checkpoints and hardware, with differences that are uneven across tasks. On GPQA-Diamond paired tests separate Qwen-3.6 35B from both Gemma checkpoints and leave the two Gemma checkpoints indistinguishable, though that comparison rests on one generation per checkpoint and does not establish a difference that repeated sampling would reproduce. AIME 2026 gives Qwen-3.6 35B the highest mean across three runs, on a set too small for stronger claims. On LiveCodeBench, the 182 date-filtered problems lower every score by more than eleven points. The drop survives stratification by difficulty label, even though the filtered subset is the harder of the two. A code benchmark score should therefore be interpreted cautiously when the evaluated problem window crosses a model’s training cutoff, unless a temporal control is applied.

On the evaluated 16 GiB desktop, memory fit separated the fastest fully GPU-resident configuration from the partially offloaded one. Qwen-3.6 35B scores highest on science, but it exceeds the card. It runs 25 of its 65 layers on the GPU and the rest on the CPU, decoding at 32.3 tokens per second, which projects its median AIME 2026 trace to about 15.5 minutes. Gemma-4 26B fits at 15.1 GiB under a 4,096-token context, decodes at 103.3 tokens per second, and reaches the highest date-filtered code

score, although the three filtered scores are not separable at 182 problems. Speed and memory fit make it the most practical of the three on this hardware. In these three configurations, active parameter count is more consistent with the observed decoding-speed ordering than total file size is, since the 26B mixture-of-experts checkpoint decodes about twice as fast as the dense 12B.

Model cards often provide insufficient information for local deployment decisions. Full-precision benchmark scores alone do not describe the memory a quantized build needs or the latency it delivers on a given consumer GPU. Whether a deployment is practical depends on both, and this study measures them together for three current checkpoints. Conclusions about quantization itself still need three things: full-precision reproduction under matched conditions, wider coverage of quantization levels, and repeated runs.

Acknowledgments

The authors acknowledge Cloudgenia for their technical assistance in this work. The research was carried out within the GEMIS.BA research group at Universidad Tecnológica Nacional, Facultad Regional Buenos Aires, Argentina. The first author acknowledges the support of a BINID scholarship granted by Universidad Tecnológica Nacional.

Declaration on Generative AI

During the preparation of this work, the authors used Claude for text translation, paraphrasing and rewording, Grammar and spelling checking, and citation management. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

- [1] Stanford Institute for Human-Centered Artificial Intelligence, The 2026 AI index report, <https://hai.stanford.edu/ai-index/2026-ai-index-report>, 2026. Accessed 2026-08-15.
- [2] OpenAI, Scaling AI for everyone, <https://openai.com/index/scaling-ai-for-everyone/>, 2026. Accessed 2026-08-15.
- [3] C. Weinberg, OpenAI projections imply losses tripling to \$14 billion in 2026, The Information. <https://www.theinformation.com/articles/openai-projections-imply-losses-tripling-to-14-billion-in-2026>, 2025. Accessed 2026-08-15.
- [4] R. Liu, Y. Sun, M. Zhang, H. Bai, X. Yu, T. Yu, C. Yuan, L. Hou, Quantization hurts reasoning? an empirical study on quantized reasoning models, in: Proceedings of the Conference on Language Modeling (COLM), 2025. URL: <https://arxiv.org/abs/2504.04823>. arXiv:2504.04823.
- [5] E. Frantar, S. Ashkboos, T. Hoefler, D. Alistarh, GPTQ: Accurate post-training quantization for generative pre-trained transformers, in: Proceedings of the 11th International Conference on Learning Representations (ICLR), 2023. URL: <https://arxiv.org/abs/2210.17323>. arXiv:2210.17323.
- [6] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2704–2713.
- [7] J. Wang, Y. Zeng, J. Guo, Y. Ma, A. Liu, X. Liu, SLMQuant: Benchmarking small language model quantization for practical deployment, in: Proceedings of the 3rd International Workshop on Rich Media with Generative AI, 2025. doi:10.1145/3746262.3761973.
- [8] G. Gerganov, et al., llama.cpp: LLM inference in C/C++, <https://github.com/ggml-org/llama.cpp>, 2023. Local measurements use build ac2557cb2 with the Vulkan backend. Accessed 2026-08-15.
- [9] U. Kurt, Which quantization should I use? A unified evaluation of llama.cpp quantization on Llama-3.1-8B-Instruct, <https://arxiv.org/abs/2601.14277>, 2026. arXiv:2601.14277, accessed 2026-08-15.

- [10] A. Javat, A. Kazakov, Silicon showdown: Performance, efficiency, and ecosystem barriers in consumer-grade LLM inference, <https://arxiv.org/abs/2605.00519>, 2026. `arXiv:2605.00519`, accessed 2026-08-15.
- [11] S. Chen, Y. Chen, Z. Li, Y. Jiang, Z. Wan, Y. He, D. Ran, T. Gu, H. Li, T. Xie, B. Ray, Recent advances in large language model benchmarks against data contamination: From static to dynamic evaluation, <https://arxiv.org/abs/2502.17521>, 2025. `arXiv:2502.17521`, accessed 2026-08-15.
- [12] Gemma Team, Gemma 4 technical report, 2026. URL: <https://arxiv.org/abs/2607.02770>. `arXiv:2607.02770`, accessed 2026-08-15.
- [13] Qwen Team, Qwen3.6-35B-A3B: Agentic coding power, now open to all, <https://qwen.ai/blog?id=qwen3.6-35b-a3b>, 2026. Accessed 2026-08-15.
- [14] Unsloth AI, Unsloth dynamic GGUF quantized model releases, <https://huggingface.co/unsloth/gemma-4-26B-A4B-it-qat-GGUF>, 2026. Evaluated checkpoints, all UD-Q4_K_XL: `unsloth/gemma-4-26B-A4B-it-qat-GGUF`, `unsloth/gemma-4-12B-it-qat-GGUF`, `unsloth/Qwen3.6-35B-A3B-MTP-GGUF`. Accessed 2026-08-15.
- [15] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, S. R. Bowman, GPQA: A graduate-level google-proof Q&A benchmark, in: Proceedings of the Conference on Language Modeling (COLM), 2024. URL: <https://arxiv.org/abs/2311.12022>. `arXiv:2311.12022`.
- [16] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, I. Stoica, LiveCodeBench: Holistic and contamination free evaluation of large language models for code, in: Proceedings of the 13th International Conference on Learning Representations (ICLR), 2025. URL: <https://arxiv.org/abs/2403.07974>. `arXiv:2403.07974`.
- [17] Mathematical Association of America, American invitational mathematics examination (AIME) 2026, <https://maa.org/maa-invitational-competitions/>, 2026. Competition held February 2026. Problems evaluated through the `evalscope/aime26` dataset, 30 problems. Accessed 2026-08-15.
- [18] ModelScope Community, EvalScope: A streamlined and customizable framework for efficient large model evaluation and performance benchmarking, <https://github.com/modelscope/evalscope>, 2024. Version 1.7.0. Accessed 2026-08-15.
- [19] Y. Leviathan, M. Kalman, Y. Matias, Prompt repetition improves non-reasoning LLMs, 2025. `arXiv:2512.14982`.
- [20] J. Hernandez, M. F. Pollo-Cattaneo, D. H. Peluffo-Ordóñez, and H. Florez, Cognitive Architecture for Learning in Autonomous Agents: A Study in Discrete Navigation, in *Communications in Computer and Information Science*, pp. 3–19, Springer, 2025, doi: 10.1007/978-3-032-07175-0_1.