TTQ: ACTIVATION-AWARE TEST-TIME QUANTIZATION TO ACCELERATE LLM INFERENCE ON THE FLY

Anonymous authors
Paper under double-blind review

ABSTRACT

To tackle the high computational cost of large foundation models, post-training activation-aware quantization methods have been widely used. However, because these methods rely heavily on calibration data, calibration–deployment mismatch may arise on unseen downstream tasks. We propose test-time quantization (TTQ), which constructs prompt-specific low-bit weights online from the current prompt and reuses them during decoding. TTQ removes the need for a predetermined calibration dataset when constructing these weights and targets decode-time weight traffic and integer execution rather than resident model-memory reduction. Experiments across language, vision-language, and vision-language-action models show improved low-bit accuracy over offline baselines in several settings while retaining most of AWQ's decode throughput at a measurable prefill cost.

1 INTRODUCTION

Large foundation models (Touvron et al., 2023; Achiam et al., 2023; Liu et al., 2023b) have shown excellent performance across a variety of tasks (Wei et al., 2022; Katz et al., 2024; Bubeck et al., 2023). Nonetheless, these models, with billions of parameters, demand significant computational resources (Schwartz et al., 2020). To increase the accessibility of large language models (LLMs), a number of compression methods (Xu & McAuley, 2023; Zhu et al., 2024; Bai et al., 2024a) have been introduced: e.g., partial activation (Jiang et al., 2024; Lin et al., 2024a), weight pruning (Frantar & Alistarh, 2023b; Sun et al., 2023; Bai et al., 2024b; Hassibi et al., 1993), quantization (Frantar et al., 2022; Lin et al., 2024b; Wang et al., 2024), knowledge distillation (Hsieh et al., 2023; Hwang et al., 2024), and rank reduction (Yuan et al., 2023; Liu et al., 2024; Hwang et al., 2024; Saxena et al., 2024). Related work is discussed further in Appendix A.

Test-time scaling (Chen et al., 2024; Muennighoff et al., 2025) is a paradigm for improving LLM performance by increasing inference-time computation. In contrast, rather than increasing inference-time computation, *test-time compression* aims to reduce inference cost adaptively for each input on the fly. For instance, test-time pruning can be formulated as a mixture-of-experts (MoE) framework that dynamically selects important modules for each prompt. Recently, μ-MoE (Koike-Akino et al., 2025b) exploits a fast activation-aware weight pruning method at test time to accelerate LLMs. However, unlike quantization or rank reduction, unstructured pruning does not generally improve hardware efficiency. We therefore propose a new **test-time quantization (TTQ)** framework, enabled by fast activation-aware quantization, that can accelerate inference through efficient integer multiplication kernels. The contributions of this paper are summarized as follows:

- We propose TTQ, a per-prompt activation-aware quantization framework that constructs low-bit weights at inference time without a predetermined calibration dataset.
- We theoretically motivate prompt-adaptive quantization and its potential error-bound advantage.
- We integrate low-rank decomposition with TTQ to improve quantization accuracy.
- We demonstrate TTQ's runtime and accuracy across diverse models and benchmarks.

TTQ deliberately reuses AWQ's efficient diagonal quantizer; its contribution is the deployment point at which activation statistics are estimated. Static AWQ commits to calibration-derived weights before serving, whereas TTQ materializes request-conditioned low-bit weights during prefill and reuses

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

them for decoding. This trades extra prefill work and retained full-precision weights for adaptation to each prompt.

2 TEST-TIME QUANTIZATION: TTQ

2.1 GROUPWISE QUANTIZATION

Round-to-nearest (RTN) is a simple quantization method for minimizing approximation error: $\mathcal{L}_0 = \|W - \hat{W}\|^2$, where $W \in \mathbb{R}^{d' \times d}$ is a weight matrix and $\hat{W}$ is its quantized version. RTN uses the groupwise quantization-dequantization (QDQ) operation: $\hat{W} = \mathcal{Q}[W] \triangleq \mathcal{G}^-[\mathcal{G}[W]]$. The quantization $\mathcal{G}[\cdot]$ and dequantization $\mathcal{G}^-[\cdot]$ are defined as:

$$W_{\text{int}} = \mathcal{G}(W) \triangleq \mathsf{round}[\mathsf{clamp}_q[(W - Z) \oslash S]], \qquad \hat{W} = \mathcal{G}^-(W_{\text{int}}) \triangleq W_{\text{int}} \circ S + Z, \quad (1)$$

where $S \in \mathbb{R}^{d' \times d}$ and $Z \in \mathbb{R}^{d' \times d}$ are scale and zero-point parameters. Here, $\oslash$ is element-wise division, $\circ$ is element-wise product, $\mathsf{round}[x]$ gives the closest integer to x, and $\mathsf{clamp}_q[x] = \min(\max(x, 0), 2^q - 1)$ is a q-bit limiting operator. See Appendix B for more details, where pseudocode for RTN with group size g is provided. The recent NVFP format (Egiazarian et al., 2025) uses a microscaling group size of 16 to accelerate GPU computing. Prior work on 4-bit LLM inference kernels (Frantar et al., 2025) reports roughly 2–4-fold speedups.

2.2 AWQ: ACTIVATION-AWARE QUANTIZATION

To improve over naïve RTN quantization, the activation-aware framework (Lin et al., 2023; Frantar et al., 2022; Liu et al., 2025) leverages activation statistics. Let $X \in \mathbb{R}^{d \times T}$ be an input activation with embedding dimension d and token length T. The aim is to minimize the approximation loss:

$$\mathcal{L} \triangleq \mathbb{E}_X\left[\left\|(W - \hat{W})X\right\|^2\right] = \text{tr}[(W - \hat{W})\mathbb{E}_X[XX^\top](W - \hat{W})^\top] = \|(W - \hat{W})C^{1/2}\|^2, \quad (2)$$

where $C \triangleq \mathbb{E}_X[XX^\top] \in \mathbb{R}^{d \times d}$ is the autocorrelation statistic of input X. Because C is unknown at test time, we estimate it from limited calibration data with a shrinkage estimator: $C_\lambda = (1-\lambda)XX^\top + \lambda\eta I$, where λ is a shrinkage parameter (Ledoit & Wolf, 2004), and $\eta = \|X\|^2/d$. GPTQ (Frantar et al., 2022) uses the greedy method inspired by optimal brain surgeon (Hassibi et al., 1993). It requires a Cholesky factorization with at least cubic complexity in d: $\mathcal{O}[d^3 + dd'T]$. Hence, it is not suited for online quantization for inference acceleration.

AWQ (Lin et al., 2023) greatly simplifies the problem by approximating the correlation with a diagonal matrix:

$$C_\lambda^{1/2} \simeq D \triangleq \mathsf{diag}[XX^\top + \lambda I]^\alpha, \quad (3)$$

where $\mathsf{diag}[Y] \triangleq Y \circ I$ forms a diagonal matrix, and α is an auxiliary parameter. The above expression gives the ℓ_2-based diagonal $D_{i,i} = (\|X_{i,:}\|_2^2 + \lambda)^\alpha$, while the original AWQ uses an ℓ_1-based statistic: $D_{i,i} = (\|X_{i,:}\|_1^2 + \lambda)^\alpha$. We therefore use a generalized ℓ_p-norm preconditioner: $D_{i,i} = (\|X_{i,:}\|_p + \lambda)^\alpha$. Given the diagonal preconditioning, the closed-form solution for (2) is given by the scaled QDQ operation:

$$\hat{W} = \mathcal{Q}[WD]D^{-1}. \quad (4)$$

Importantly, AWQ requires only quadratic additional complexity of $\mathcal{O}[d(T + 2d')]$, while achieving performance competitive with GPTQ. See Appendix C for more details, including pseudocode.

2.3 TTQ: TEST-TIME QUANTIZATION WITH ONLINE AWQ

Activation-aware post-training quantization methods such as GPTQ and AWQ have several drawbacks:

1. a calibration phase and extra data are required before deployment;
2. a severe domain shift may arise at test time when the calibration data are mismatched to deployment inputs; and

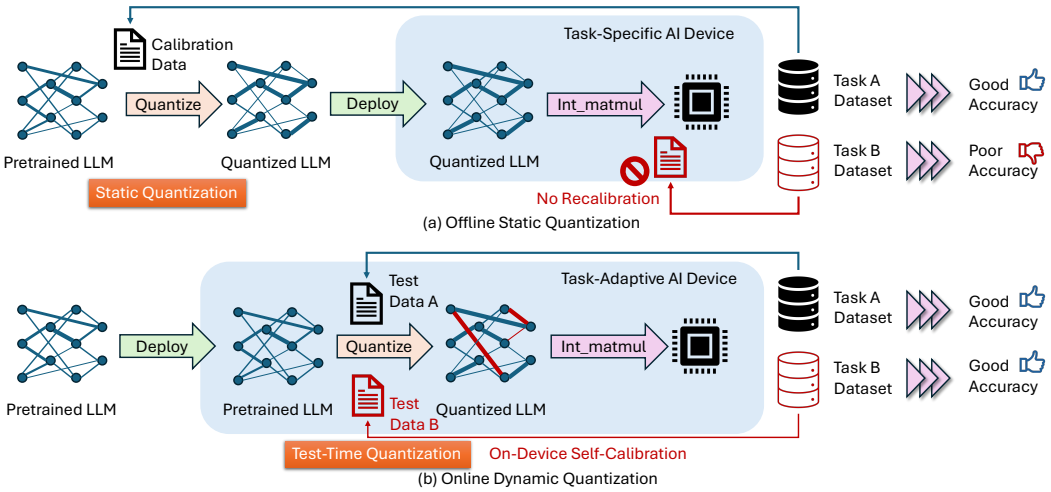


Figure 1: (a) Offline static quantization (e.g., AWQ/GPTQ) uses predetermined calibration data and typically fixes the quantized weights before deployment, creating domain-shift risk. (b) TTQ instead quantizes online from the incoming prompt, avoiding a predetermined calibration dataset for weight construction and enabling prompt-specific on-device calibration.

3. if only the quantized model is deployed, the original full-precision weights are unavailable for on-device recalibration to a new domain.

To solve these issues, we propose a test-time quantization (TTQ) framework shown in Fig. 1, which employs lightweight online AWQ at inference time. Specifically, we compute the diagonal preconditioner D from incoming tokens X on the fly and use it to adapt the scale S and zero-point Z. It therefore requires no predetermined offline calibration dataset to construct prompt-specific quantized weights at deployment; global method hyperparameters may still be selected using offline development experiments (Appendix H). Once the weights are quantized on-device, `int_matmul` kernels [1] can accelerate the linear projection, especially for the decoding phase where weight traffic dominates. Because quantization adapts to each input, TTQ can construct prompt-specific weights for previously unseen downstream tasks without requiring redeployment.

During prefill, TTQ computes the activation-dependent diagonal preconditioner from the current prompt and quantizes the relevant linear weights. During decoding, the resulting quantized weights and scales are reused for subsequent token generation. Thus, TTQ avoids a fixed offline calibration set for constructing request-specific weights while amortizing quantization overhead across decoding steps.

To keep online AWQ fast enough at test time, we fix the hyperparameters (α, λ, p) rather than searching them exhaustively for each prompt. Appendix F discusses the hyperparameter selection. In operation-count terms, the relative additional computation for online quantization vanishes as T and d' grow. Specifically, the overhead relative to the original unquantized projection is:

$$\rho_0 = \frac{\mathcal{O}[dT + 3d'd]}{\mathcal{O}[d'dT]} = \mathcal{O}\left[\frac{1}{d'} + \frac{3}{T}\right] \underset{d',T\gg 1}{\longrightarrow} 0. \tag{5}$$

Here, the original projection for WX requires $\mathcal{O}[d'dT]$, the norm calculation for $D_{ii} = (\|X_{i,:}\| + \lambda)^{\alpha}$ requires $\mathcal{O}[dT]$, scaling as WD requires $\mathcal{O}[d'd]$, quantization $\mathcal{G}[\cdot]$ in (1) needs $\mathcal{O}[d'd]$, and rescaling $\mathcal{Q}[WD]D^{-1}$ uses another $\mathcal{O}[d'd]$. Hence, for sufficiently large d' and prefill length T, the additional operation count vanishes asymptotically; this does not imply negligible finite-prompt prefill latency, which we measure in Appendix J.1.

[1] e.g., `awq_gemm` and `marlin_gemm` CUDA kernels in `https://github.com/vllm-project/vllm`

2.4 THEORETICAL MOTIVATION OF TTQ

We next provide theoretical analyses supporting the motivation for TTQ. Even without domain shift, online quantization can have a theoretical advantage over offline quantization, as formalized below.

Lemma 1. *Let $\mathcal{X}$ be any set of possible input activation tensors and any distribution over that set. Let $\mathcal{W}$ be any admissible class of quantized tensors for a given weight W. Define an oracle per-instance quantized weight $\hat{W}_\star(X)$ which minimizes an activation-aware loss for any instance of $X \in \mathcal{X}$:*

$$\hat{W}_\star(X) = \underset{\hat{W}\in\mathcal{W}}{\operatorname{argmin}} \|WX - \hat{W}X\|^2. \tag{6}$$

Then for any fixed quantization $\hat{W}' \in \mathcal{W}$, we have

$$\mathop{\mathbb{E}}_{X\in\mathcal{X}} \|WX - \hat{W}_\star(X)X\|^2 \le \mathop{\mathbb{E}}_{X\in\mathcal{X}} \|WX - \hat{W}'X\|^2. \tag{7}$$

The inequality is strict if there is a set of positive probability on which $\hat{W}_\star(X)$ is unique and not equal to the constant $\hat{W}'$.

The proof is given in Appendix G, where we also derive the potential advantage for specific quantization formats such as the scaled QDQ: $\hat{W} = \mathcal{Q}[WD]D^{-1}$. Lemma 1 states that an oracle per-instance quantizer cannot have higher expected loss than a fixed quantizer; it does not establish this guarantee for practical TTQ.

The lemma is an oracle result, not a guarantee that practical TTQ outperforms AWQ, but it motivates test-time adaptation. Appendix G develops this motivation using companding theory (Bennett, 1948; Gersho & Gray, 2012): under stated assumptions, the AWQ/TTQ diagonal preconditioner $D = \mathsf{diag}[XX^\top + \lambda I]^\alpha$ minimizes a derived bound, and online TTQ obtains a tighter expected bound than offline AWQ via Jensen's inequality. A Gaussian activation model further shows that this potential advantage vanishes as T grows, exposing a tradeoff: longer prompts reduce the overhead in (5) but also the potential benefit of per-prompt adaptation. The analysis does not guarantee that prompt statistics remain optimal for future generated tokens; strong prompt-to-generation shift is therefore a failure mode rather than a case covered by the bound.

2.5 TTQ WITH LOW-RANK DECOMPOSITION

Generally, very low-bit quantization can cause severe degradation, while QLoRA (Dettmers et al., 2023) compensates for quantization error by adapting low-rank factors. This motivates integrating TTQ with low-rank decomposition. We write the approximation as:

$$\hat{W} = \underbrace{\mathcal{Q}[(W - BA)D]D^{-1}}_{W_{\mathrm{q}}\in\mathbb{R}^{d'\times d}} + BA, \tag{8}$$

where W_{q} denotes the quantized residual after subtracting the unquantized low-rank factors $B \in \mathbb{R}^{d'\times r}$ and $A \in \mathbb{R}^{r\times d}$ of rank r. When $r \ll \min(d, d')$, the low-rank path has complexity $\mathcal{O}[r(d' + d)T]$ for BAX, compared with $\mathcal{O}[d'dT]$ for the full projection WX; hence its relative additional complexity is negligible:

$$\rho_1 = \frac{\mathcal{O}[r(d' + d)T]}{\mathcal{O}[d'dT]} = \mathcal{O}\left[\frac{r}{d} + \frac{r}{d'}\right] \underset{d',d\gg r}{\longrightarrow} 0. \tag{9}$$

A key difference from QLoRA is that TTQ **dynamically** adapts W_{q} depending on X in an online manner, whereas QLoRA uses a **static** W_{q} and adapts only B and A during fine-tuning. We initialize the low-rank factors B and A from a rank-r truncated SVD, while Appendix E discusses alternative initializations. Optionally, we may dynamically adapt B and A, e.g., through online PCA (Feng et al., 2013). Nevertheless, our primary objective is to accelerate LLM inference through dynamic weight quantization, so we focus on static low-rank factors. We later show up to 4.9-fold decode-only projection speedup for Qwen3-32B on RTX 4090 even with the rank-16 residual, while separate full-model measurements expose the corresponding prefill cost.

Table 1: Runtime throughput (k tokens/sec) of the query-projection module for Qwen3 models with 4-bit AWQ and TTQ, on NVIDIA A40 GPU.

Qwen3	0.6B	1.7B	4B	8B	14B	32B
FP16	57.58	41.29	21.38	14.10	10.43	6.86
AWQ (`awq_gemm`)	75.76	71.60	54.61	32.68	25.28	18.28
AWQ (`marlin_gemm`)	73.32	73.70	**73.98**	**38.91**	**37.52**	**30.32**
TTQ ($r = 0$)	**80.63**	**78.68**	69.11	36.52	24.94	19.34
TTQ ($r = 16$)	66.67	59.08	49.67	30.32	23.78	17.43

×2.5

Table 2: Runtime throughput (k tokens/sec) of the query-projection module for Qwen3 models with 4-bit AWQ and TTQ, on NVIDIA RTX 4090 GPU.

Qwen3	0.6B	1.7B	4B	8B	14B	32B
FP16	116.82	77.04	72.45	58.44	46.78	10.76
AWQ (`awq_gemm`)	113.84	90.42	80.19	62.94	51.32	50.45
AWQ (`marlin_gemm`)	**120.62**	**115.20**	**112.61**	**101.00**	**80.45**	**72.34**
TTQ ($r = 0$)	108.00	104.11	103.45	92.23	74.72	67.40
TTQ ($r = 16$)	77.88	73.47	72.55	66.62	58.65	53.17

×4.9

3 Experiments

We evaluate OPT (Zhang et al., 2022), Qwen3 (Yang et al., 2025), and Gemma3 (Gemma Team et al., 2025) on WikiText-2 (WT2) (Merity et al., 2016), PTB (Marcus et al., 1994), and C4 (Raffel et al., 2020); Qwen3-VL (Bai et al., 2025) on TextVQA (Singh et al., 2019) with COCO-Caption (Lin et al., 2014), OK-VQA (Marino et al., 2019), ChartQA (Masry et al., 2022), and TextVQA calibration; and $\pi_{0.5}$ (Intelligence et al., 2025) and X-VLA (Zheng et al., 2025) on LIBERO (Liu et al., 2023a). The $\pi_{0.5}$ model uses PaliGemma/Gemma backbones, whereas X-VLA uses Florence-2 (Xiao et al., 2024). We also evaluate Phi-4-mini-instruct (Abouelenin et al., 2025) on GSM8K (Cobbe et al., 2021) against GPTQ/AWQ and report full-model RTX 5090 throughput to expose TTQ's online cost. Appendices M, N, and J provide model, dataset, and controlled-evaluation details.

3.1 Runtime benchmark

We first show that TTQ can accelerate inference at test time relative to unquantized LLMs, even with the extra overhead of prescaling and low-rank factors. To evaluate TTQ runtime, we use the Marlin kernel (Frantar et al., 2025), one of the best-known `int_matmul` CUDA kernels. Specifically, we use `vllm._custom_ops.marlin_gemm` in the vLLM library[2]. It often outperforms `vllm._custom_ops.awq_gemm`. We use CUDA 12.8, PyTorch 2.9.1, and vLLM 0.15.1. For TTQ, we include prescaling by the diagonal matrix D and low-rank projections using B and A on top of the Marlin kernel. To reduce the overhead of multi-kernel launching, we use `torch.compile` and CUDA graph replay[3] for all cases. We benchmark the projection module since it dominates memory bandwidth during decoding. Because these microbenchmarks omit other model operations, the reported projection throughput is an upper bound on achievable end-to-end throughput. The microbenchmarks therefore characterize the maximum potential contribution of integer projection kernels rather than full-model acceleration.

Tables 1 and 2 show runtime benchmarks for Qwen3 models on NVIDIA A40 and RTX 4090 GPUs, respectively. To highlight the speedup, we focus on the linear query-projection module and benchmark 4-bit TTQ and AWQ. The microbenchmarks show:

- In this decode-only projection benchmark, AWQ reaches up to 6.7-fold speedup at the 32B RTX 4090 point, a large-model memory-bandwidth-bound regime.

[2] `https://github.com/vllm-project/vllm`
[3] `https://docs.vllm.ai/en/stable/design/cuda_graphs/`

Table 4: Effect of calibration length on WT2 perplexity for 3-bit OPT-350M at $g = 32$.

	TTQ		AWQ (C4 Calib)						
Calib Tokens T	$\mathbf{0}_{(r=0)}$	$\mathbf{0}_{(r=16)}$	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
WT2 Perplexity (↓)	24.93	**24.17**	25.73	25.56	25.62	25.55	25.42	25.07	25.02

Table 5: Effect of group size on WT2 perplexity for 3-bit Qwen3-1.7B.

Group size g	8	16	32	64	128	256	512	1024
RTN	33.39	66.61	120.89	114.97	1450.28	15503.10	949478.44	57443.80
AWQ (WT2 Calib)	17.64	19.00	20.35	20.69	24.96	31.41	36.14	68.96
TTQ ($r = 16$)	**17.54**	**17.81**	**18.29**	**19.46**	**22.33**	**24.81**	**31.43**	**48.70**

- TTQ without low-rank factors ($r = 0$) retains most of AWQ's decode-only projection throughput despite its additional prompt-dependent prescaling.
- With $r = 16$, TTQ reaches 4.9-fold decode-only projection speedup for Qwen3-32B on RTX 4090; this best-case point is not representative of all models, GPUs, or workloads.
- The relative benefit increases for larger memory-bandwidth-bound projections because weight traffic increasingly dominates (e.g., Qwen3-32B query projection contains $5{,}120 \times 8{,}192$ FP16 weights, about 84 MB).

AWQ can fuse the prescaling D into preceding modules, whereas TTQ must fuse it into `int_matmul` for comparable decode efficiency. Custom CUDA kernels that fuse TTQ prescaling with integer multiplication could further improve GPU throughput, especially at lower bit widths. Because the projection microbenchmarks exclude TTQ's defining prefill-time requantization, we also measure full-model phase throughput on RTX 5090 in Table 3. TTQ prefill includes prompt processing, activation-statistics collection, preconditioner construction, and online weight quantization. Across Qwen3-0.6B–8B, TTQ reduces prefill throughput by 11–24% relative to AWQ, while decode throughput remains within 0–6.5% of AWQ. Thus Eq. (5) is a scaling explanation rather than evidence that wall-clock overhead is negligible for every prompt length; short prompts are a less favorable regime.

Table 3: Full-model phase throughput on RTX 5090. Prefill is M tokens/s; decode is k tokens/s. TTQ prefill includes online statistics and weight quantization.

Qwen3	AWQ prefill	TTQ prefill	AWQ decode	TTQ decode
0.6B	69.0	52.2	206.1	206.1
1.7B	38.3	32.6	206.8	193.3
4B	16.0	14.2	157.6	155.6
8B	10.1	8.2	142.7	142.2

Additional-GPU and full-model timing results appear in Appendices K and J.1.

3.2 LLM BENCHMARK

We then evaluate OPT, Qwen3, and Gemma3 models on the WT2, PTB, and C4 benchmarks. See details of the experimental setup in Appendix I. We use a group size of $g = 32$ for all experiments unless specified. Table 4 shows AWQ's sensitivity to C4 calibration length for OPT-350M WT2 perplexity. TTQ performs best, whereas AWQ degrades as calibration shrinks.

Table 5 shows the impact of group size g for 3-bit Qwen3-1.7B. As expected, microscaling ($g < 32$) performs well, but RTN still trails AWQ/TTQ. TTQ tolerates roughly twice AWQ's group size, halving scale/zero-point storage.

Table 7 next compares model families and sizes using macro-average perplexity over WT2, PTB, and C4. We observe that few-bit quantization such as $q = 2$ is more feasible for larger LLMs, and 5-bit quantization achieves nearly unquantized performance for most cases. TTQ is best or competitive

in most reported settings, even though AWQ uses a long calibration of $T = 2^{17}$. More importantly, AWQ fluctuates across calibration datasets, indicating the potential risk of domain shift when relying heavily on calibration for such **static** quantization.

Matched ablation and downstream accuracy. A controlled Qwen3-1.7B / WT2 ablation in Appendix J.2 separates the effects of preconditioning, low-rank residuals, and online estimation. Moving AWQ from ℓ_1 to ℓ_2 preconditioning explains a substantial portion of the gain, while the rank-16 residual gives a smaller improvement; after matching both, TTQ still lowers 3-bit perplexity by 1.12–2.23 points across group sizes 16–128. This isolates TTQ's main distinction as prompt-conditioned estimation and reuse rather than a fundamentally different quantization operator.

Perplexity alone does not establish downstream task quality, so Table 6 evaluates Phi-4-mini-instruct on the 1,319-example GSM8K test set using standard `llm-compressor` GPTQ/AWQ baselines. At 4 bits, all differences are within roughly one binomial standard error and are not conclusive. At 3 bits, TTQ reaches 59.36% exact-match accuracy, 7.19 points above GPTQ and 11.44 above AWQ.

Table 6: Phi-4-mini-instruct GSM8K exact-match accuracy (binomial SE in percentage points).

Method	4-bit accuracy (SE)	3-bit accuracy (SE)
BF16	74.98 (1.19)	74.98 (1.19)
RTN	74.35 (1.20)	41.62 (1.36)
GPTQ	73.92 (1.21)	52.17 (1.38)
AWQ	74.83 (1.19)	47.92 (1.38)
TTQ	**75.97 (1.18)**	**59.36 (1.35)**

For Gemma3, the uncompressed PTB perplexity is substantially higher than its WT2/C4 values, so quantization can occasionally produce a slightly lower macro-average perplexity. See Appendix L for full perplexity results and Appendix J.3 for the complete GSM8K comparison and scope.

Table 7: Macro-average WT2/PTB/C4 perplexity (↓) for OPT/Qwen3/Gemma3. $g = 32$; AWQ uses $T = 2^{17}$ calibration tokens. **Bold** and underlined values: best and second-best, respectively; "*": competitive with the uncompressed baseline.

Quantization q	2 bits	3 bits	4 bits	5 bits
OPT-125M (WT2: 27.7, PTB: 39.0, C4: 26.6, Avg: **31.1**)				
RTN	5058.5	56.3	33.5	31.8
AWQ (WT2 Calib)	381.7	37.4	32.3	31.4
AWQ (PTB Calib)	375.3	37.3	32.2	31.3
AWQ (C4 Calib)	451.7	37.7	32.6	31.3
TTQ ($r = 0$)	257.4	36.6	31.9	31.2
TTQ ($r = 16$)	**141.7**	**35.8**	**31.8**	***31.1**
OPT-1.3B (WT2: 14.6, PTB: 20.3, C4: 16.1, Avg: **17.0**)				
RTN	11514.4	27.2	18.1	17.2
AWQ (WT2 Calib)	32.4	18.0	17.3	***17.0**
AWQ (PTB Calib)	32.6	18.1	17.3	***17.0**
AWQ (C4 Calib)	31.7	18.0	17.2	***17.0**
TTQ ($r = 0$)	32.2	18.2	17.2	***17.0**
TTQ ($r = 16$)	**27.2**	**17.9**	**17.1**	***17.0**
OPT-2.7B (WT2: 12.5, PTB: 18.0, C4: 14.3, Avg: **14.9**)				
RTN	6274.5	36.0	15.7	15.0
AWQ (WT2 Calib)	23.1	15.7	15.1	15.0
AWQ (PTB Calib)	23.2	15.7	**15.0**	15.0
AWQ (C4 Calib)	22.9	15.7	**15.0**	15.0
TTQ ($r = 0$)	23.7	15.7	**15.0**	***14.9**
TTQ ($r = 16$)	**21.2**	**15.5**	**15.0**	***14.9**
OPT-6.7B (WT2: 10.9, PTB: 15.8, C4: 12.7, Avg: **13.1**)				
RTN	5716.5	26.2	13.7	13.2
AWQ (WT2 Calib)	17.2	13.5	13.2	***13.1**
AWQ (PTB Calib)	17.1	13.6	13.2	***13.1**
AWQ (C4 Calib)	16.9	13.6	13.2	***13.1**
TTQ ($r = 0$)	17.2	13.6	13.2	***13.1**
TTQ ($r = 16$)	**16.3**	**13.4**	***13.1**	***13.1**

Quantization q	2 bits	3 bits	4 bits	5 bits
Qwen3-0.6B (WT2: 21.0, PTB: 43.8, C4: 30.3, Avg: **31.7**)				
RTN	8.2e6	127.3	38.2	33.5
AWQ (WT2 Calib)	9739.1	49.4	33.5	32.4
AWQ (PTB Calib)	17344.3	47.9	34.1	32.0
AWQ (C4 Calib)	5388.1	48.3	33.0	32.1
TTQ ($r = 0$)	2827.8	44.7	33.0	**31.9**
TTQ ($r = 16$)	**1552.6**	**42.0**	**32.9**	**31.9**
Qwen3-1.7B (WT2: 16.7, PTB: 33.8, C4: 16.1, Avg: **24.2**)				
RTN	1.4e6	162.8	30.6	26.1
AWQ (WT2 Calib)	1864.7	30.0	24.5	24.4
AWQ (PTB Calib)	2309.6	29.8	24.7	24.5
AWQ (C4 Calib)	2364.7	28.2	24.9	24.4
TTQ ($r = 0$)	522.6	27.3	24.4	24.3
TTQ ($r = 16$)	**264.6**	**26.4**	**24.3**	***24.1**
Gemma3-270M (WT2: 70.1, PTB: 698.7, C4: 68.1, Avg: **279.0**)				
RTN	2.6e11	1795.0	391.9	315.0
AWQ (WT2 Calib)	89188.4	517.4	279.6	306.0
AWQ (PTB Calib)	1.9e5	537.7	310.0	293.4
AWQ (C4 Calib)	1.3e5	598.1	326.4	*276.8
TTQ ($r = 0$)	30199.0	387.7	*277.0	*262.5
TTQ ($r = 16$)	**19657.0**	**382.8**	***275.7**	***261.5**
Gemma3-1B (WT2: 27.7, PTB: 212.4, C4: 33.2, Avg: **91.1**)				
RTN	8.6e5	209.4	111.1	96.6
AWQ (WT2 Calib)	4734.7	134.7	91.9	95.9
AWQ (PTB Calib)	9326.6	138.9	99.2	95.5
AWQ (C4 Calib)	5486.9	150.8	93.5	93.6
TTQ ($r = 0$)	1928.5	127.3	***89.9**	***90.2**
TTQ ($r = 16$)	**1804.9**	**114.5**	91.7	*90.3

3.3 VLM BENCHMARK

Table 8 evaluates Qwen3-VL 2B/4B/8B on TextVQA, quantizing all linear modules except the LM head and vision encoder. AWQ uses COCO-Caption, OK-VQA, ChartQA, or TextVQA calibration; TTQ is competitive across most settings. Some quantized scores slightly exceed the unquantized baseline, but without repeated-run uncertainty we do not interpret these small differences as systematic gains.

Table 8: Qwen3-VL TextVQA soft accuracy (↑), $g = 32$. **Bold** and underlined values: best and second-best, respectively; "*": competitive with the uncompressed baseline.

Quantization q	2 bits	3 bits	4 bits	5 bits
Qwen3-VL-2B: Acc 80.35%				
RTN	0.00%	9.23%	73.27%	80.22%
AWQ (COCO-Cap Calib)	1.41%	75.67%	79.04%	79.89%
AWQ (OK-VQA Calib)	1.37%	74.68%	79.47%	79.80%
AWQ (ChartQA Calib)	0.67%	75.52%	79.29%	79.80%
AWQ (TextVQA Calib)	0.82%	72.15%	78.85%	80.21%
TTQ ($r = 0$)	<u>1.42%</u>	**76.59%**	<u>79.59%</u>	<u>80.34%</u>
TTQ ($r = 16$)	**1.93%**	<u>76.01%</u>	**79.67%**	***80.38%**
Qwen3-VL-4B: Acc 81.44%				
RTN	0.03%	74.79%	80.77%	*81.47%
AWQ (COCO-Cap Calib)	0.13%	78.68%	80.49%	81.20%
AWQ (OK-VQA Calib)	0.18%	77.89%	80.58%	80.75%
AWQ (ChartQA Calib)	0.42%	77.37%	80.67%	*81.47%
AWQ (TextVQA Calib)	0.25%	77.49%	80.28%	81.01%
TTQ ($r = 0$)	<u>1.51%</u>	**79.93%**	**81.29%**	*<u>81.48%</u>
TTQ ($r = 16$)	**7.47%**	<u>79.45%</u>	<u>81.22%</u>	***81.49%**
Qwen3-VL-8B: Acc 81.72%				
RTN	0.17%	78.51%	80.63%	*81.73%
AWQ (COCO-Cap Calib)	41.39%	80.37%	81.52%	*81.79%
AWQ (OK-VQA Calib)	31.62%	79.51%	81.01%	81.55%
AWQ (ChartQA Calib)	41.17%	78.65%	81.23%	81.39%
AWQ (TextVQA Calib)	39.60%	78.43%	81.25%	81.69%
TTQ ($r = 0$)	<u>42.56%</u>	<u>80.77%</u>	<u>81.57%</u>	*<u>81.81%</u>
TTQ ($r = 16$)	**48.01%**	**81.04%**	**81.71%**	***81.85%**

3.4 VLA BENCHMARK

Table 9 evaluates $\pi_{0.5}$ on LIBERO Spatial, Object, Goal, and Long, with 200 episodes per suite. We quantize all VLM-backbone linear modules except the LM head and vision encoder; AWQ uses each LIBERO suite as calibration. AWQ substantially improves over RTN but depends on the calibration domain, while TTQ gives the highest average quantized success rate, with its clearest gain on long-horizon LIBERO Long. Thus, TTQ supports 2-bit $\pi_{0.5}$ with limited performance loss.

Another result on X-VLA is listed in Table 10, where we use $q = 2$ bits and group size $g = 128$. Because the original table contains only 800 episodes in total, we expanded this comparison to 8,000 LIBERO episodes across multiple random seeds. The expanded evaluation preserves the ordering: TTQ ($r = 16$) reaches 95.24% success (0.24% binomial SE), versus 94.59% (0.25%) for FP32 and 93.66% (0.27%) for the best AWQ calibration domain. However, the 0.65-point TTQ–FP32 gap is only about 1.9 combined standard errors under an independence approximation, so we treat the FP32-exceeding result as suggestive rather than conclusive and make no causal regularization claim.

Table 9: LIBERO success rate (↑) for $\pi_{0.5}$ at $q = 2$, $g = 64$. **Bold** and underlined values denote the best and second-best results, respectively.

Benchmark	Spatial	Object	Goal	Long	Avg
BF16	97.5%	100.0%	97.0%	96.5%	97.75%
RTN	57.0%	65.0%	27.5%	2.0%	37.88%
AWQ (Spatial Calib)	90.5%	**100.0**%	85.0%	82.0%	89.34%
AWQ (Object Calib)	91.5%	98.5%	92.0%	78.0%	90.00%
AWQ (Goal Calib)	92.5%	**100.0**%	**93.5**%	84.5%	92.63%
AWQ (Long Calib)	94.0%	99.5%	92.5%	76.5%	90.63%
TTQ ($r = 0$)	93.0%	99.5%	**93.5**%	87.0%	93.25%
TTQ ($r = 16$)	**94.5**%	**100.0**%	**93.5**%	**87.5**%	**93.88**%

Table 10: LIBERO success rate (↑) for X-VLA at $q = 2$, $g = 128$. **Bold** and underlined values: best and second-best, respectively; "*": competitive with FP32.

Benchmark	Spatial	Object	Goal	Long	Avg
FP32	95.0%	98.5%	93.5%	87.5%	93.63%
RTN	92.0%	92.0%	87.0%	74.5%	86.38%
AWQ (Spatial Calib)	91.5%	96.0%	*95.0%	86.0%	92.13%
AWQ (Object Calib)	92.5%	96.0%	*95.0%	82.5%	91.50%
AWQ (Goal Calib)	91.0%	95.0%	***96.0**%	84.0%	91.50%
AWQ (Long Calib)	90.0%	96.5%	***96.0**%	85.5%	92.00%
TTQ ($r = 0$)	***98.0**%	97.5%	*93.5%	*88.5%	*94.38%
TTQ ($r = 16$)	***98.0**%	***98.5**%	*94.5%	***89.5**%	***95.13**%

See Appendix J.4 for the uncertainty interpretation and evaluation-scope details.

3.5 Deployment scope and trade-offs

The experiments identify a concrete operating regime for TTQ. Unlike conventional post-training quantization, TTQ retains the full-precision weights so that it can construct a new prompt-specific low-bit instance; it therefore does not reduce resident device memory or enable a larger model to fit on the device. Its practical benefit is during decoding: a 4-bit weight copy moves roughly one quarter of the BF16 weight bytes, excluding scales, zero points, and the retained full-precision copy, so integer kernels can reduce weight traffic when linear projections are memory-bandwidth bound. This explains why the largest decode-only gains occur for large models, while smaller or less bandwidth-limited regimes can show little benefit. The measured prefill slowdown in Table 3 is the price of adaptivity and is least favorable when the one-time cost is amortized over only short generations.

Request-specific quantized weights also conflict with conventional continuous batching, which relies on sharing one static weight tensor across requests. The current method is therefore best suited to single-request, low-batch, or on-device inference. Sharing a preconditioner across requests could recover batching efficiency but would trade away per-prompt adaptivity. Together with the prompt-to-generation shift discussed in Section 2.4, these constraints position TTQ as a bandwidth/adaptivity method rather than a replacement for offline quantization when resident-memory reduction or high-throughput batching is the primary objective.

4 Conclusion

We proposed TTQ, a framework that constructs prompt-specific activation-aware quantized weights at inference time without a predetermined calibration dataset for those weights. Across LLM, VLM, and VLA evaluations, TTQ improves low-bit accuracy over matched offline baselines in several settings, including a 3-bit GSM8K comparison against GPTQ/AWQ, and retains most of AWQ's decode throughput while incurring measurable prefill overhead. TTQ retains full-precision weights in resident memory, so its deployment benefit is reduced decode-time weight traffic and low-bit execution rather than model-capacity reduction. Future work includes custom fused TTQ kernels, prefill acceleration, batching-aware adaptation, and adaptive hyperparameter selection; Appendix H summarizes TTQ's benefits and limitations.

AI USE STATEMENT

OpenAI ChatGPT assisted with manuscript formatting, language editing, proofreading, and drafting. All AI-assisted content was reviewed and revised by the authors, who verified the technical statements, equations, experimental results, citations, and conclusions and take responsibility for the final work.

REFERENCES

Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, et al. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. *arXiv preprint arXiv:2503.01743*, 2025.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Zeyuan Allen-Zhu and Yuanzhi Li. First efficient convergence for streaming k-PCA: a global, gap-free, and near-optimal rate. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 487–492. IEEE, 2017.

Yamato Arai and Yuma Ichikawa. Quantization error propagation: Revisiting layer-wise post-training quantization. *arXiv preprint arXiv:2504.09629*, 2025.

Hicham Badri and Appu Shaji. Half-quadratic quantization of large machine learning models. Mobius Labs technical blog, 2023. Calibration-free HQQ method.

Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, et al. Beyond efficiency: A systematic survey of resource-efficient large language models. *arXiv preprint arXiv:2401.00625*, 2024a.

Guangji Bai, Yijiang Li, Chen Ling, Kibaek Kim, and Liang Zhao. SparseLLM: Towards global pruning for pre-trained language models. *arXiv preprint arXiv:2402.17946*, 2024b.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

William Ralph Bennett. Spectra of quantized signals. *The Bell System Technical Journal*, 27(3): 446–472, 1948.

Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision*, pp. 707–720. Springer, 2002.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023.

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.

Dian Chen, Dequan Wang, Trevor Darrell, and Sayna Ebrahimi. Contrastive test-time adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 295–305, 2022.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 293–302, 2019.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529, 2020.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Vage Egiazarian, Roberto L Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, et al. Bridging the gap between promise and performance for microscaling FP4 quantization. *arXiv preprint arXiv:2509.23202*, 2025.

Mostafa Elhoushi, Zihao Chen, Farhan Shafiq, Ye Henry Tian, and Joey Yiwei Li. DeepShift: Towards multiplication-less neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2359–2368, 2021.

Jiashi Feng, Huan Xu, and Shuicheng Yan. Online robust PCA via stochastic optimization. *Advances in neural information processing systems*, 26, 2013.

Elias Frantar and Dan Alistarh. QMoE: Practical sub-1-bit compression of trillion-parameter models. *arXiv preprint arXiv:2310.16795*, 2023a.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023b.

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.

Elias Frantar, Roberto L Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language models. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pp. 239–251, 2025.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Allen Gersho and Robert M Gray. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012.

Peter M Hall, A David Marshall, and Ralph R Martin. Incremental eigenanalysis for classification. In *BMVC*, volume 98, pp. 286–295, 1998.

Babak Hassibi, David Stork, and Gregory Wolff. Optimal brain surgeon: Extensions and performance comparisons. *Advances in neural information processing systems*, 6, 1993.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

Injoon Hwang, Haewon Park, Youngwan Lee, Jooyoung Yang, and SunJae Maeng. PC-LoRA: Low-rank adaptation for progressive model compression with knowledge distillation. *arXiv preprint arXiv:2406.09117*, 2024.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. GPT-4 passes the bar exam. *Philosophical Transactions of the Royal Society A*, 382(2270):20230254, 2024.

Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

Toshiaki Koike-Akino, Chang Meng, Volkan Cevher, and Giovanni De Micheli. Hardware-efficient quantization for green custom foundation models. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*, 2024.

Toshiaki Koike-Akino, Xiangyu Chen, Jing Liu, Ye Wang, Matthew Brand, et al. LatentLLM: Attention-aware joint tensor compression. *arXiv preprint arXiv:2505.18413*, 2025a.

Toshiaki Koike-Akino, Jing Liu, and Ye Wang. μ-moe: Test-time pruning as micro-grained mixture-of-experts. *arXiv preprint arXiv:2505.18451*, 2025b.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Olivier Ledoit and Michael Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of multivariate analysis*, 88(2):365–411, 2004.

Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025.

Bin Lin, Zhenyu Tang, Yang Ye, Jiaxi Cui, Bin Zhu, Peng Jin, Junwu Zhang, Munan Ning, and Li Yuan. MoE-LlaVa: Mixture of experts for large vision-language models. *arXiv preprint arXiv:2401.15947*, 2024a.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: Activation-aware weight quantization for LLM compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

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, 2024b.

Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Yan Wang, Yongjian Wu, Feiyue Huang, and Chia-Wen Lin. Rotated binary neural network. *Advances in neural information processing systems*, 33:7474–7485, 2020.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *Advances in neural information processing systems*, 30, 2017.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023a.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023b.

Jing Liu, Toshiaki Koike-Akino, Ye Wang, Hassan Mansour, and Matthew Brand. AWP: Activation-aware weight pruning and quantization with projected gradient descent. *arXiv preprint arXiv:2506.10205*, 2025.

Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-Real Net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 722–737, 2018.

Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. OK-VQA: A visual question answering benchmark requiring external knowledge. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pp. 3195–3204, 2019.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the association for computational linguistics: ACL 2022*, pp. 2263–2279, 2022.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

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.

Augustus Odena, Dieterich Lawson, and Christopher Olah. Changing model behavior at test-time using reinforcement learning. *arXiv preprint arXiv:1702.07780*, 2017.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2250–2259, 2020.

Haotong Qin, Mingyuan Zhang, Yifu Ding, Aoyu Li, Zhongang Cai, Ziwei Liu, Fisher Yu, and Xianglong Liu. BiBench: Benchmarking and analyzing network binarization. In *International Conference on Machine Learning*, pp. 28351–28388. PMLR, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea Goldsmith, and Mert Pilanci. Compressing large language models using low rank and low precision decomposition. *Advances in Neural Information Processing Systems*, 37:88981–89018, 2024.

Gabriele Santini, Francesco Paissan, and Elisabetta Farella. A probabilistic framework for dynamic quantization. *arXiv preprint arXiv:2505.10689*, 2025.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for KV cache compression. *arXiv preprint arXiv:2408.05646*, 2024.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

Amanpreet Singh, Vivek Natarjan, Meet Shah, Yu Jiang, Xinlei Chen, Devi Parikh, and Marcus Rohrbach. Towards VQA models that can read. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8317–8326, 2019.

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.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Changyuan Wang, Ziwei Wang, Xiuwei Xu, Yansong Tang, Jie Zhou, and Jiwen Lu. Q-VLM: Post-training quantization for large vision-language models. *arXiv preprint arXiv:2410.08119*, 2024.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

T Wolf. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4818–4829, 2024.

Canwen Xu and Julian McAuley. A survey on model compression and acceleration for pretrained language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 10566–10575, 2023.

Yuhui Xu, Shuai Zhang, Yingyong Qi, Jiaxian Guo, Weiyao Lin, and Hongkai Xiong. DNQ: Dynamic network quantization. *arXiv preprint arXiv:1812.02375*, 2018.

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.

Bin Yang. Projection approximation subspace tracking. *IEEE Transactions on Signal processing*, 43 (1):95–107, 1995.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. HAWQ-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pp. 11875–11886. PMLR, 2021.

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. ASVD: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*, 2023.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. LLM inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

Jinjie Zhang, Yixuan Zhou, and Rayan Saab. Post-training quantization for neural networks with provable guarantees. *SIAM journal on mathematics of data science*, 5(2):373–399, 2023.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng Chen, Deming Chen, and Zhiru Zhang. FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 171–182, 2021.

Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 15–24, 2017.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 196–209, 2024.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577, 2024.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, et al. TTRL: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*, 2025.

A Related work

A.1 Model compression

The field of model compression for LLMs has aimed at mitigating the substantial computation and memory requirements (Zhu et al., 2024; Yuan et al., 2024). Such methods primarily fall into four categories: weight quantization (Lin et al., 2024b; Frantar et al., 2022; Wang et al., 2024), network pruning (LeCun et al., 1989; Hassibi et al., 1993; Frantar & Alistarh, 2023b; Bai et al., 2024b), knowledge distillation (Hsieh et al., 2023; Hwang et al., 2024), and rank reduction (Yuan et al., 2023; Liu et al., 2024; Hwang et al., 2024; Saxena et al., 2024; Saha et al., 2024).

A.2 Dynamic quantization

Dynamic quantization is a strategy for adapting precision at runtime rather than using fixed bit-widths across all layers and inputs. DNQ (Xu et al., 2018) uses a controller that learns per-layer bit-width policies to balance accuracy and compression. Recent work (Santini et al., 2025) uses probabilistic input-adaptive quantization, conditioning parameters on each input with negligible loss relative to static counterparts.

A.3 FEW-BIT QUANTIZATION

Extreme 1-bit quantization (Qin et al., 2023) has shown relatively good performance even with 1-bit quantization, e.g., BNN (Courbariaux et al., 2016; Kim & Smaragdis, 2016), XNOR-Net (Rastegari et al., 2016), DoReFa-Net (Zhou et al., 2016), Bi-Real (Liu et al., 2018), IR-Net (Qin et al., 2020), RBNN (Lin et al., 2020), and ABC-Net (Lin et al., 2017). These methods use a variety of gradient approximations for the non-differentiable binarization operation. For example, BNN uses the straight-through estimator (STE) for the non-differentiable sign function.

There is extensive work on hardware implementations of binarized networks, particularly on field-programmable gate arrays (FPGAs) (Zhao et al., 2017; Zhang et al., 2021). Nevertheless, binarization often causes a substantial loss, and most quantization papers (Frantar et al., 2022; Lin et al., 2023) for foundation models consider at least 3 bits to achieve acceptable performance. By contrast, 2-bit quantization has been shown to be feasible for LLM compression with vector quantization (Chee et al., 2023; Egiazarian et al., 2024).

A.4 ACTIVATION/HARDWARE-AWARE QUANTIZATION

DeepShift (Elhoushi et al., 2021) uses power-of-two weights to eliminate multiplication operations. HAWQ (Dong et al., 2019) uses layer-wise quantization based on optimal brain pruning (LeCun et al., 1989). HAWQv2 (Dong et al., 2020) considers mixed-precision weight and activation. HAWQv3 (Yao et al., 2021) uses integer weight in dyadic format. HEQ (Koike-Akino et al., 2024) jointly minimizes quantization loss and computing energy on custom hardware. Then, GPTQ (Frantar et al., 2022) extends HAWQ using zero-shot calibration, while activation-aware quantization (AWQ) (Lin et al., 2023) uses activation-dependent scaling. GPFQ (Zhang et al., 2023) uses a greedy path-following mechanism to quantize weights with theoretical guarantees. AWP (Liu et al., 2025) uses projected gradient descent for activation-aware quantization and pruning based on compressed sensing framework. QEP (Arai & Ichikawa, 2025) mitigates error propagation across layers. QLoRA (Dettmers et al., 2023) uses a low-rank adapter to compensate for the loss of quantized LLMs. HQQ (Badri & Shaji, 2023) applies calibration-free half-quadratic weight quantization, while Atom (Zhao et al., 2024) co-designs low-bit weight/activation quantization and serving kernels. Our additional downstream-accuracy study in Appendix J includes GPTQ and an implementation-matched AWQ baseline. Without implementation-matched HQQ or Atom results, we make no empirical superiority claim over them.

A.5 TEST-TIME COMPUTING

Prior work has explored improving performance at inference time by increasing computation, adaptation, or interaction beyond a single forward pass. Test-time scaling methods (Chen et al., 2024; Muennighoff et al., 2025; Snell et al., 2024) allocate additional inference-time compute via sampling, search, and reasoning while keeping model parameters fixed, whereas test-time computing dynamically structures inference through adaptive computation, tool use, and external memory. In contrast, test-time update/adaptation methods (Liang et al., 2025; Chen et al., 2022) perform unsupervised or self-supervised parameter updates during deployment, often under distribution shift, with meta-learning explicitly training models for rapid test-time adaptation. Test-time reinforcement learning (Zuo et al., 2025; Odena et al., 2017) further frames inference as an interactive process with online policy improvement driven by reward signals. μ-MoE (Koike-Akino et al., 2025b) is a new test-time pruning framework that applies activation-aware pruning on the fly at inference time.

B RTN: GROUPWISE ROUND-TO-NEAREST QUANTIZATION

Round-to-nearest (RTN) is a simple quantization method for minimizing approximation error:

$$\mathcal{L}_0 = \|W - \hat{W}\|^2, \tag{10}$$

where $W \in \mathbb{R}^{d' \times d}$ is a weight matrix and $\hat{W}$ is its quantized version. RTN uses the groupwise quantization-dequantization (QDQ) operation:

$$\hat{W} = \mathcal{Q}[W] \triangleq \mathcal{G}^{-}[\mathcal{G}[W]], \tag{11}$$

Here, the quantization $\mathcal{G}[\cdot]$ and dequantization $\mathcal{G}^{-}[\cdot]$ are defined as:

$$W_{\text{int}} = \mathcal{G}(W) \triangleq \mathsf{round}[\mathsf{clamp}_q[(W - Z) \oslash S]], \tag{12}$$

$$\hat{W} = \mathcal{G}^{-}(W_{\text{int}}) \triangleq W_{\text{int}} \circ S + Z, \tag{13}$$

where $S \in \mathbb{R}^{d' \times d}$ and $Z \in \mathbb{R}^{d' \times d}$ are scale and zero-point parameters. Here, $\oslash$ is element-wise division, $\circ$ is element-wise product, $\mathsf{round}[x]$ gives the closest integer to x, and $\mathsf{clamp}_q[x] = \min(\max(x, 0), 2^q - 1)$ is a q-bit limiting operator. The straightforward choice of scale and zero-point is

$$S = (W_{\max} - W_{\min})/(2^q - 1), \tag{14}$$

$$Z = W_{\min}, \tag{15}$$

These values can be adjusted slightly to reduce quantization error. QDQ has several variants and nonuniform formats, such as NF4 (Dettmers et al., 2023); see Appendix D. RTN pseudocode for group size g follows:

```
def rtn(W, q, g): # q: bits, g: groupsize
    ddash, d = W.shape # W: (d', d)
    W = W.reshape(-1, g) # grouping (d'*d/g, g)
    Wmax = W.amax(axis=1, keepdim=True) # (d'*d/g, 1)
    Wmin = W.amin(axis=1, keepdim=True) # (d'*d/g, 1)
    S, Z = (Wmax - Wmin) / (2**q - 1), Wmin # scale, zero-point
    Wint = ((W - Z) / S).round().clamp(0, 2**q - 1)
    What = Wint * S + Z # dequantization
    return What.reshape(ddash, d) # reshaping back
```

The representation uses $qd'd$ bits for W_{int} plus $d'd/g$ scale and zero-point parameters. A larger group size reduces scale/zero-point metadata, whereas a smaller group size can reduce quantization error. The recent NVFP format (Egiazarian et al., 2025) uses a microscaling group size of 16 to accelerate GPU computing. Because most GPUs are not fully compatible with arbitrary-bit-width integer operations, the practical benefit of quantization often comes from reduced memory traffic, which can also accelerate GPUs by alleviating memory and cache bottlenecks. Prior work (Lin et al., 2023; Frantar et al., 2022; 2025) reports 2–4-fold speedups with quantized LLMs.

C AWQ: Activation-aware quantization

To improve over naïve RTN quantization, the activation-aware framework (Lin et al., 2023; Frantar et al., 2022; Liu et al., 2025) leverages activation statistics. Let $X \in \mathbb{R}^{d \times T}$ be an input activation with embedding dimension d and token length T. The aim is to minimize the approximation loss:

$$\mathcal{L} \triangleq \mathbb{E}_X \left[\left\| (W - \hat{W})X \right\|^2 \right] \tag{16}$$

$$= \mathrm{tr}[(W - \hat{W}) \underbrace{\mathbb{E}_X[XX^\top]}_{C \in \mathbb{R}^{d \times d}} (W - \hat{W})^\top] \tag{17}$$

$$= \|(W - \hat{W})C^{1/2}\|^2, \tag{18}$$

where $C \triangleq \mathbb{E}_X[XX^\top] \in \mathbb{R}^{d \times d}$ is the autocorrelation statistic of input X. Since C is not known exactly at test time, we estimate it from a small amount of calibration data, e.g., using a shrinkage estimator:

$$C_\lambda = (1 - \lambda)XX^\top + \lambda\eta I, \tag{19}$$

where λ is a shrinkage parameter, and $\eta = \|X\|^2/d$. For large dimension d, the Ledoit–Wolf shrinkage (Ledoit & Wolf, 2004) is reliable. Note that this is equivalent to minimize the weighted errors of activation-aware and weight-only quantization objectives:

$$\mathcal{L}' \triangleq (1 - \lambda)\|(W - \hat{W})X\|^2 + \lambda\eta\|W - \hat{W}\|^2 \tag{20}$$

$$= \|(W - \hat{W})C_\lambda^{1/2}\|^2. \tag{21}$$

Also note that the solution is invariant to any scaling factor to the correlation, e.g., many literature use

$$C'_{\lambda'} \triangleq C_\lambda/(1-\lambda) \tag{22}$$

$$= XX^\top + \lambda' I, \tag{23}$$

where $\lambda' = \lambda\eta/(1-\lambda)$ is a damping factor; we denote any such scaled correlation estimate by C_λ.

When there is no structure in correlation matrix C_λ, it has no closed-form solution to minimize $\mathcal{L}'$, whereas GPTQ (Frantar et al., 2022) uses the greedy method based on the inverse Hessian inspired by optimal brain surgeon (Hassibi et al., 1993). GPTQ requires the Cholesky factorization, whose complexity is at least of cubic order: $\mathcal{O}[d^3 + dd'T]$. In addition, it needs extra computations for off-diagonal error cancellation with the Gaussian elimination. Alternatively, AWP (Liu et al., 2025) uses projected gradient descent to solve it iteratively, i.e., at the kth step:

$$\hat{W}^{(k+1)} = \mathcal{Q}[\hat{W}^{(k)} + \mu(W - \hat{W}^{(k)})C_\lambda], \tag{24}$$

where μ is the stepsize. It requires a cubic complexity of $\mathcal{O}[d'd^2K]$ for K total iterations.

AWQ (Lin et al., 2023) greatly simplifies the problem by approximating the correlation with a diagonal matrix:

$$C_\lambda^{1/2} \simeq D \triangleq \mathsf{diag}[XX^\top + \lambda I]^\alpha, \tag{25}$$

where $\mathsf{diag}[Y] \triangleq Y \circ I$ offers a diagonal matrix zeroing out off-diagonal elements of an argument matrix Y, and α is an auxiliary parameter to compensate for the diagonal approximation loss. The above expression gives the ℓ_2-based diagonal $D_{i,i} = (\|X_{i,:}\|_2^2 + \lambda)^\alpha$, while the original AWQ uses ℓ_1-norm $D_{i,i} = (\|X_{i,:}\|_1^2 + \lambda)^\alpha$. Given a diagonal correlation matrix, the closed-form solution for (21) is given by the scaled QDQ operation:

$$\hat{W} = \mathcal{Q}[WD]D^{-1}. \tag{26}$$

Importantly, the diagonal correlation also reduces the computation for D from $\mathcal{O}[d^2T']$ to $\mathcal{O}[dT']$. AWQ then requires only quadratic additional complexity of $\mathcal{O}[2dd' + dT']$, while achieving performance competitive with GPTQ. Pseudocode for the AWQ concept is given as follows:

```
def awq(X, W, q, g, p, lam, alpha): # q: bits, g: groupsize, p: p-norm
    D = X.norm(p=p, axis=1) # X: (d, T)
    D = (D + lam) ** alpha  # D: (d,)
    What = rtn(W * D[None, :], q, g) # scaled QDQ
    return What * D.reciprocal()[None, :] # scaling back
```

Here, we generalized to any ℓ_p-norm with arbitrary p. Hyperparameters such as α and p can be selected to minimize (21).

D QUANTIZATION-DEQUANTIZATION (QDQ) FORMATS

QDQ operations admit several parameterizations. We use the following form:

$$\mathcal{G}(W) \triangleq \mathsf{round}[\mathsf{clamp}_q[(W - Z) \oslash S]], \tag{27}$$

$$\mathcal{G}^-(W_{\text{int}}) \triangleq W_{\text{int}} \circ S + Z, \tag{28}$$

where $\mathcal{G}[\cdot]$ and $\mathcal{G}^-[\cdot]$ denote quantization and dequantization operations, respectively. An alternative representation is:

$$\mathcal{G}'(W) \triangleq \mathsf{clamp}_q[\mathsf{round}[W \oslash S'] + Z'], \tag{29}$$

$$\mathcal{G}'^-(W_{\text{int}}) \triangleq (W_{\text{int}} - Z') \circ S'. \tag{30}$$

The two representations are equivalent up to parameterization, so we focus on the first. Nonuniform formats such as NF4 (Dettmers et al., 2023) provide another option.

We consider the asymmetric format for the scale and zero-point:

$$S = (W_{\max} - W_{\min})/(2^q - 1), \tag{31}$$

$$Z = W_{\min}. \tag{32}$$

Some methods further adjust the scale and zero point to reduce quantization error. For example, we may use expanded $W'_{\max}$ and $W'_{\min}$ instead of $W_{\max}$ and $W_{\min}$:

$$W'_{\max} \triangleq \frac{1+\nu}{2} W_{\max} + \frac{1-\nu}{2} W_{\min} \tag{33}$$

$$W'_{\min} \triangleq \frac{1-\nu}{2} W_{\max} + \frac{1+\nu}{2} W_{\min} \tag{34}$$

where $\nu \in \mathbb{R}$ is an expansion factor. Note that $\nu = 1$ reduces to the standard scaling. In our experiments, useful expansion factors are often near $\nu \simeq 0.95$.

Alternatively, the symmetric format is often used:

$$S = 2|W|_{\max}/(2^q - 1), \tag{35}$$

$$Z = -|W|_{\max}. \tag{36}$$

As it has fewer degrees of freedom (i.e., Z or S is redundant as $S = -Z/(2^q - 1)$), it generally offers lower accuracy but requires less metadata memory.

TTQ could also be extended to vector-quantized or compressed formats (Egiazarian et al., 2024; Frantar & Alistarh, 2023a).

E Quantization-aware low-rank factorization

Initialization of the low-rank factors can affect quantization performance. We seek factors B and A that reduce the quantization error of the residual. A natural initialization uses the rank-r truncated SVD of W:

$$U_r \Lambda_r V_r = \mathsf{svd}_r[W], \tag{37}$$

$$B = U_r \Lambda_r^{1/2}, \tag{38}$$

$$A = \Lambda_r^{1/2} V_r, \tag{39}$$

where $\mathsf{svd}_r[\cdot]$ denotes the rank-r truncated singular-value decomposition (SVD); $U_r \in \mathbb{R}^{d' \times r}$, $\Lambda_r \in \mathbb{R}^{r \times r}$, and $V_r \in \mathbb{R}^{r \times d}$ are the left singular vectors, diagonal singular-value matrix, and right singular vectors, respectively. If calibration data are available, we can use ASVD (Yuan et al., 2023; Koike-Akino et al., 2025a) instead: i.e., $\mathsf{svd}_r[W C_\lambda^{1/2}] C_\lambda^{-1/2}$.

This truncated-SVD initialization minimizes reconstruction error but does not necessarily minimize the post-quantization residual error. A quantization-aware alternative is to use an alternating update, e.g., at iteration k:

$$B^{(k)} A^{(k)} = \mathsf{svd}_r[W - W_{\mathrm{q}}^{(k)}], \tag{40}$$

$$W_{\mathrm{q}}^{(k+1)} = \mathcal{Q}[W - B^{(k)} A^{(k)}], \tag{41}$$

with a proper initialization. This quantization-aware factorization may reduce quantization error, although it does not guarantee lower inference-time task loss. The alternating solution gave negligible gain in our experiments, so we use truncated-SVD factors for static low-rank initialization.

Low-rank factor quantization The low-rank factors A and/or B can also be quantized online or offline to further accelerate inference. For example, we may consider: (1) quantizing A while leaving B unquantized; (2) quantizing B while leaving A unquantized; or (3) quantizing both A and B. Cases (1) and (2) may be preferable because BA is then not fully quantized.

Low-rank factor pruning Pruning can also be combined with quantization and low-rank factorization. For example, TTQ can be combined with the test-time pruning used in μ-MoE (Koike-Akino et al., 2025b). Because both methods use a similar diagonal correlation statistic, they can share the computation of D. Likewise, low-rank-factor pruning may prune (1) A only, (2) B only, or (3) both factors. Cases (1) and (2) may be preferable because BA is then not fully sparse.

Test-time decomposition Another option is to adapt A and B at inference time using efficient online PCA algorithms:

- **Stochastic gradient methods.** Oja's rule (Oja, 1982) performs stochastic gradient ascent on the Rayleigh quotient to estimate the top principal components from streaming data.
- **Incremental SVD methods.** Deterministic approaches to update low-rank factorizations include incremental eigen-analysis (Hall et al., 1998) and incremental SVD techniques (Brand, 2002).
- **Subspace tracking algorithms.** Recursive least-squares formulations such as PAST (Yang, 1995) provide fast convergence and strong tracking capability under non-stationary data.
- **Online optimization perspectives.** Analyzing online PCA via stochastic optimization frameworks establishes finite-sample convergence and performance guarantees (Allen-Zhu & Li, 2017).

F HYPERPARAMETER SELECTION FOR TTQ

Static AWQ can tune hyperparameters such as α during offline calibration, whereas TTQ recomputes S and Z from C_λ for every incoming activation X; on-the-fly hyperparameter search should therefore be limited.

If post-training development data are available, global hyperparameters and a prior correlation estimate can be selected before deployment. For TTQ, we use fixed global hyperparameters informed by the development sweeps described next.

We evaluated perplexity for the OPT family using uniformly sampled grids of α, λ, and p. We then selected the five best combinations for each OPT model and bit width q. We analyzed histograms of these selected parameter sets, as shown in Figure 2. The histograms suggest several general trends:

- The norm exponent α is typically around 0.5 or 0.75 in this sweep. This suggests tempering the diagonal correlation approximation with $\alpha < 1$ rather than amplifying it with $\alpha > 1$.
- The damping factor λ is often around 0.4, substantially larger than nominal choices such as $\lambda = 0.01$ in prior work. This may reflect the large ratio between dimension d and available token length T, for which stronger shrinkage can improve estimation (Ledoit & Wolf, 2004). Empirically, this range also balances the activation-aware and weight-only terms in (21).
- The best ℓ_p norm is around $p = 2$, consistent with the derivation in (25). More importantly, the ℓ_1 norm performs poorly in our sweep. The original AWQ uses a heuristic metric based on ℓ_1, while prior work has noted that such an approximation may be suboptimal; for example, LatentLLM (Koike-Akino et al., 2025a) compares alternative preconditioning variants, and Wanda (Sun et al., 2023) derives a diagonal approximation using ℓ_2 activation statistics.

In our main perplexity experiments, AWQ uses the ℓ_2 norm rather than the original ℓ_1 norm, with α selected by line search.

We use constant hyperparameters across LLM modules; heterogeneous module-wise values could add flexibility but would require additional offline selection.

G THEORETICAL ANALYSIS

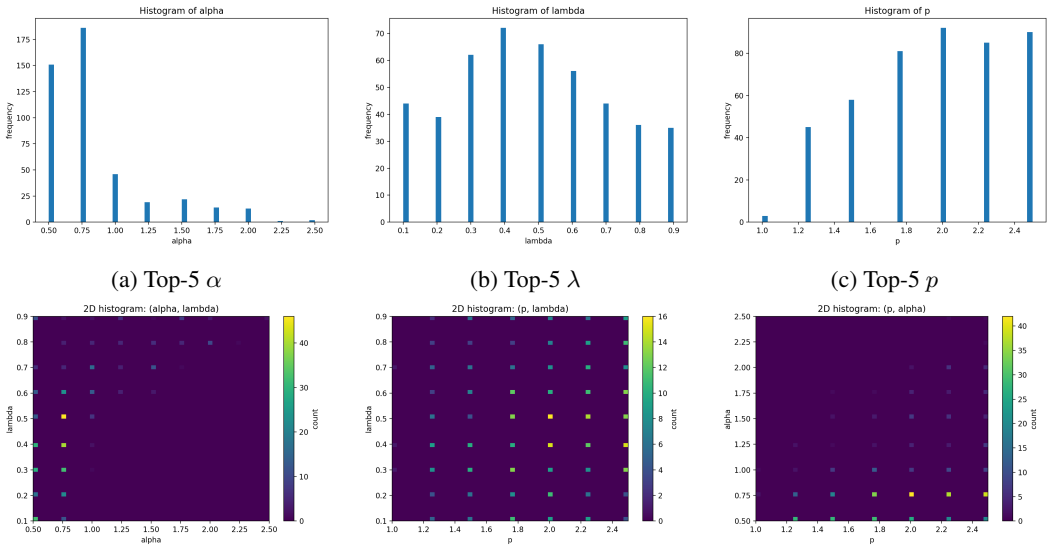
G.1 PROOF OF LEMMA 1

Proof. For every realization $X \in \mathcal{X}$, we have

$$\|WX - \hat{W}_\star(X)X\|^2 \le \|WX - \hat{W}'X\|^2, \tag{42}$$

because $\hat{W}_\star(X)$ minimizes the loss pointwise from the definition. Taking expectation gives

$$\mathop{\mathbb{E}}_{X\in\mathcal{X}} \|WX - \hat{W}_\star(X)X\|^2 \le \mathop{\mathbb{E}}_{X\in\mathcal{X}} \|WX - \hat{W}'X\|^2. \tag{43}$$



(a) Top-5 α (b) Top-5 λ (c) Top-5 p

(d) Top-5 (α, λ) (e) Top-5 (λ, p) (f) Top-5 (p, α)

Figure 2: Top-5 hyperparameter selections (α, λ, p) across OPT models and $q \in \{2, 3, 4, 5\}$.

If for some fixed $\hat{W}' \in \mathcal{W}$ there is a measurable set $\mathcal{A}$ with positive probability $\mathbb{P}(\mathcal{A}) > 0$ such that $\|WX - \hat{W}_\star(X)X\|^2 < \|WX - \hat{W}'X\|^2$ for all $X \in \mathcal{A}$, then the inequality is strict. □

G.2 Online preconditioning benefit

Lemma 1 gives the following corollary.

Corollary 1. *Let $\mathcal{D}$ be any admissible class of preconditioners for quantizing W. Define an oracle per-instance preconditioner $D_\star(X)$ which minimizes the loss for any $X \in \mathcal{X}$:*

$$D_\star(X) = \operatorname*{argmin}_{D \in \mathcal{D}} \|WX - \mathcal{Q}[WD]D^{-1}X\|^2. \tag{44}$$

Then for any fixed preconditioner $D' \in \mathcal{D}$, we have

$$\mathop{\mathbb{E}}_{X \in \mathcal{X}} \|WX - \mathcal{Q}[WD_\star(X)]D_\star(X)^{-1}X\|^2 \leq \mathop{\mathbb{E}}_{X \in \mathcal{X}} \|WX - \mathcal{Q}[WD']D'^{-1}X\|^2. \tag{45}$$

The inequality is strict if there is a set of positive probability on which $D_\star(X)$ is unique and not equal to the constant D'.

Proof. The proof follows directly from Lemma 1 by restricting the admissible quantization class to $\mathcal{W} := \{\hat{W} : \hat{W} = \mathcal{Q}[WD]D^{-1}, D \in \mathcal{D}\}$. □

Corollary 1 states that an oracle per-instance preconditioner cannot have higher expected loss than any fixed preconditioner. Although Corollary 1 provides useful motivation for online quantization, it heavily relies on the idealized oracle preconditioner. Below, we provide more practical derivations to explain why a diagonal preconditioner using (3) can be optimal based on companding theory (Bennett, 1948; Gersho & Gray, 2012).

G.3 Companding perspective

The scaled QDQ (4) used for AWQ/TTQ can be viewed as a companding operation (compressing and expanding) (Bennett, 1948; Gersho & Gray, 2012), which suggests allocating quantization resolution according to a power law of source density and distortion gain. Motivated by the classical companding theory, we consider the following power-law distortion envelope.

Proposition 1. *Let $D(X)$ be a per-instance preconditioner from a class $\mathcal{D}$ of positive diagonal matrices for an instance $X \in \mathcal{X}$. Assume the per-instance loss is bounded by power-law distortion:*

$$\mathcal{L}'(X, D(X)) = \|WX - \mathcal{Q}[WD(X)]D(X)^{-1}X\|^2 \tag{46}$$

$$= \| \underbrace{(WD(X) - \mathcal{Q}[WD(X)])}_{\Delta(D(X))} D(X)^{-1}X\|^2 \tag{47}$$

$$= \sum_{i,j,k} [\Delta(D(X))]_{i,j}^2 \frac{X_{j,k}^2}{D_{j,j}^2(X)} \tag{48}$$

$$\leq \sum_j a \frac{\|X_{j,:}\|_2^2 + \lambda}{D_{j,j}^{\rho}(X)} + bD_{j,j}^{\beta}(X), \tag{49}$$

where $\rho > 0$ models how the post-quantization expander $D^{-1}(X)$ amplifies distortion, $\beta > 0$ models how distortion grows with the pre-quantization compressor $D(X)$, $\lambda > 0$ is a constant, and $a, b > 0$ are coefficients determined by the quantization spectrum associated with $\mathcal{Q}[\cdot]$.

Then for any instance $X \in \mathcal{X}$, the minimizer of the bound is given as:

$$D_{j,j}^{\star}(X) = \Big[\frac{\rho a}{\beta b}(\|X_{j,:}\|_2^2 + \lambda)\Big]^{\frac{1}{\beta+\rho}} \propto (\|X_{j,:}\|_2^2 + \lambda)^{\alpha}, \tag{50}$$

where $\alpha = 1/(\beta + \rho)$.

Proof. The result follows by differentiating the bound (49) with respect to $D_{j,j}(X)$ and finding the root. Specifically, we solve

$$-\rho a \frac{\|X_{j,:}\|_2^2 + \lambda}{D_{j,j}^{\rho+1}(X)} + \beta b D_{j,j}^{\beta-1}(X) = 0. \tag{51}$$

Rearranging gives (50). Substituting (50) into (49) gives the minimized bound

$$\mathcal{L}'(X, D^{\star}(X)) \leq \underbrace{\frac{b(\rho+\beta)}{\rho}\Big(\frac{\rho a}{\beta b}\Big)^{\alpha\beta}}_{\gamma} \sum_j (\|X_{j,:}\|_2^2 + \lambda)^{\alpha\beta} \tag{52}$$

□

Proposition 1 shows that the bound-minimizing exponent is $\alpha = 1/(\beta + \rho)$, with corresponding preconditioner

$$D^{\star}(X) \propto \mathsf{diag}[XX^{\top} + \lambda I]^{\alpha}. \tag{53}$$

This has the same form as the preconditioner used in AWQ and TTQ.

G.4 Ensemble preconditioner

In the static setting considered here, AWQ uses a fixed ensemble preconditioner rather than a per-instance preconditioner like $D^{\star}(X)$: $D' = \mathsf{diag}[\mathbb{E}[XX^{\top}] + \lambda I]^{\alpha}$. We show that this is optimal for the ensemble loss bound below.

Proposition 2. *Under the power-law distortion bound in Proposition 1, an ensemble preconditioner D' such that*

$$D' \propto \mathsf{diag}\big[\mathop{\mathbb{E}}_{X \in \mathcal{X}}[XX^{\top}] + \lambda I\big]^{\alpha} \tag{54}$$

is a minimizer of the ensemble loss bound across any fixed preconditioner $D \in \mathcal{D}$:

$$\mathcal{L} = \mathop{\mathbb{E}}_{X \in \mathcal{X}} \mathcal{L}'(X, D) \tag{55}$$

$$\leq \sum_j a \frac{\mathbb{E}_{X \in \mathcal{X}}[\|X_{j,:}\|_2^2] + \lambda}{D_{j,j}^{\rho}} + bD_{j,j}^{\beta}, \tag{56}$$

Proof. The result follows from the derivation of $D^\star(X)$. The ensemble loss bound in (56) has exactly the same form as the per-instance loss bound in (49). Replacing $\|X_{j,:}\|^2$ with $\mathbb{E}\|X_{j,:}\|^2$ and $D(X)$ with D yields (54). □

Proposition 2 shows that the ensemble preconditioner (54) minimizes the ensemble loss bound under the stated assumptions. Nevertheless, as AWQ uses calibration data for correlation estimation, it remains susceptible to domain shift.

G.5 TTQ vs AWQ

Even without domain shift, the derived upper bound for the adaptive per-instance preconditioner can be tighter than that for the optimal fixed ensemble preconditioner. We formalize this bound comparison below.

Proposition 3. *Under the power-law distortion bound in Proposition 1, TTQ using the adaptive preconditioner $D^\star(X)$ from (53) has average loss bounded by:*

$$\mathcal{L}_{\mathrm{TTQ}} = \mathop{\mathbb{E}}_{X\in\mathcal{X}} \mathcal{L}'(X, D^\star(X)) \tag{57}$$

$$\leq \gamma \sum_j \mathop{\mathbb{E}}_{X\in\mathcal{X}} (\|X_{j,:}\|_2^2 + \lambda)^{\alpha\beta} \triangleq \bar{\mathcal{L}}_{\mathrm{TTQ}}. \tag{58}$$

AWQ using the fixed bound-minimizing preconditioner D' in (54) has average loss bounded by:

$$\mathcal{L}_{\mathrm{AWQ}} = \mathop{\mathbb{E}}_{X\in\mathcal{X}} \mathcal{L}'(X, D') \tag{59}$$

$$\leq \gamma \sum_j (\mathop{\mathbb{E}}_{X\in\mathcal{X}} \|X_{j,:}\|_2^2 + \lambda)^{\alpha\beta} \triangleq \bar{\mathcal{L}}_{\mathrm{AWQ}}. \tag{60}$$

Then the TTQ upper bound is strictly tighter than the AWQ upper bound:

$$\bar{\mathcal{L}}_{\mathrm{TTQ}} < \bar{\mathcal{L}}_{\mathrm{AWQ}}, \tag{61}$$

whenever $\|X_{j,:}\|$ is non-degenerate.

Proof. Since $0 < \alpha\beta = \beta/(\beta+\rho) < 1$, the map $t \mapsto t^{\alpha\beta}$ is strictly concave. By Jensen's inequality for concave functions, we obtain (61). □

Even though Proposition 2 gives an optimal fixed preconditioner for the assumed bound, Proposition 3 gives TTQ a strictly tighter *expected upper bound* under the same distortion model when the activation energy is non-degenerate. This is not a guarantee of lower realized quantization error, nor does it guarantee that prompt statistics remain optimal for subsequently generated tokens.

G.6 Gaussian model

Under a Gaussian activation model, we can quantify the gap between the derived TTQ and AWQ upper bounds.

Proposition 4. *Let $X_{j,k}$ be drawn i.i.d. from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$. Under the power-law distortion bound in Proposition 1, the ratio of the TTQ and AWQ expected upper bounds is:*

$$\frac{\bar{\mathcal{L}}_{\mathrm{TTQ}}}{\bar{\mathcal{L}}_{\mathrm{AWQ}}} = \frac{\mathbb{E}_{\chi_T^2}(\sigma^2\chi_T^2 + \lambda)^{\alpha\beta}}{(\sigma^2\mathbb{E}_{\chi_T^2}[\chi_T^2] + \lambda)^{\alpha\beta}} \tag{62}$$

$$\xrightarrow[\lambda\to 0]{} \frac{\Gamma(T/2 + \alpha\beta)}{(T/2)^{\alpha\beta}\Gamma(T/2)}, \tag{63}$$

where χ_k^2 is a chi-square random variable with k degrees of freedom and $\Gamma(x)$ is the Gamma function.

Proof. The variable $\chi_T^2 = \sum_{i=1}^T X_{i,j}^2/\sigma^2$ follows a chi-square distribution with T degrees of freedom. Equations (58) and (60) give (62). When $\lambda \to 0$, it reduces to a chi-square moment ratio: $\mathbb{E}[(\chi_T^2)^{\alpha\beta}]/\mathbb{E}[\chi_T^2]^{\alpha\beta}$, which gives (63). □

Note that the asymptotic bound ratio in (63) monotonically increases with the token length T. The minimum is thus at $T = 1$: $\bar{\mathcal{L}}_{\mathrm{TTQ}}/\bar{\mathcal{L}}_{\mathrm{AWQ}} \to 2^{\alpha\beta}\Gamma(1/2 + \alpha\beta)/\sqrt{\pi}$. For example, when $\alpha\beta = 1/2$, the ratio is $\sqrt{2/\pi} \simeq 0.798$, corresponding to an approximately 20% smaller derived upper bound for TTQ than for AWQ. As $T \to \infty$, this bound advantage vanishes because (63) approaches 1. Therefore, Proposition 4 illustrates a tradeoff between the potential bound advantage and complexity: shorter prompt lengths T can yield a larger bound gap, while increasing the relative overhead of per-prompt weight quantization in (5).

H Benefits and limitations

TTQ has several advantages over conventional offline quantization:

- TTQ does not require a predetermined calibration dataset to construct the prompt-specific quantized weights; the incoming prompt supplies the activation statistics used for each request.
- Prompt-conditioned calibration can mitigate sensitivity to a mismatched offline calibration set, as observed in our AWQ comparisons across calibration domains.
- Original unquantized weights remain available after deployment, allowing prompt-specific requantization and future on-device recalibration or adaptation.
- In the large-model, memory-bandwidth-bound decode regime, low-bit weights reduce decode-time weight traffic and can exploit fast integer multiplication kernels.
- TTQ offers flexible parameterization through α, λ, p, g, and q, and can incorporate a low-rank residual.
- TTQ applies beyond text-only LLMs to VLMs and VLAs in our experiments.
- Appendix G provides a conditional theoretical motivation for the adaptivity–overhead tradeoff.

TTQ also has important limitations:

- TTQ does *not* reduce the resident model-memory footprint, because the full-precision weights must remain available for prompt-specific requantization. Its intended benefit is reduced decode-time weight traffic and low-bit execution after prefill, not fitting a larger model into device memory.
- Online quantization has a measurable prefill cost. In the additional full-model RTX 5090 measurements in Appendix J.1, TTQ reduces prefill throughput by 11–24% relative to AWQ, while decode throughput is within 0–6.5% of AWQ. Equation (5) is therefore a scaling explanation, not an empirical claim that prefill overhead is always negligible.
- The largest decode-only speedups occur for large, memory-bandwidth-bound projections and should not be interpreted as representative end-to-end speedups for every model size or GPU. TTQ may be slower for small models or compute-rich/large-batch regimes.
- Request-specific quantized weights conflict with conventional continuous batching, which assumes that requests share static weights. The present method is therefore best suited to single-request, low-batch, or on-device inference. Sharing a preconditioner across requests could recover batching efficiency but would reduce per-prompt adaptivity and is not evaluated here.
- Global hyperparameters such as (p, α, λ) may still be selected using offline experiments. Thus, "no calibration data" refers specifically to constructing prompt-specific quantized weights, not to a guarantee that all method-level hyperparameters were selected without offline development data.
- Short prompts provide noisier activation statistics and are also the least favorable regime for amortizing online quantization. We do not yet provide a prompt-length sweep or an isolated measurement of the quantization kernel itself.
- Prompt activations can differ from the statistics of later generated tokens. The theory gives an oracle/bound-based motivation but does not guarantee that a preconditioner estimated during prefill remains optimal throughout decoding; online updating is future work.

- We do not directly measure a distance between the offline AWQ preconditioner and the per-prompt TTQ preconditioner. Consequently, the domain-shift mechanism is supported indirectly through calibration-set sensitivity and downstream performance rather than by a dedicated preconditioner-drift metric.
- Current CUDA kernels support only a limited set of low-bit formats, and TTQ targets linear projections rather than other inference bottlenecks such as attention computation.
- The additional hyperparameters increase optimization complexity, and TTQ offers diminishing gains at higher precision, particularly above 5 bits.

If offline development data are available, global hyperparameters can be selected before deployment, as in AWQ/GPTQ. This does not change the prompt-conditioned construction of quantized weights at inference time, but it should not be conflated with a completely calibration-free development process. For short prompts, the correlation estimate could also be accumulated across request history, although that would change the per-prompt adaptation semantics and is not evaluated in this work.

I Experimental setup

We conduct experiments to evaluate TTQ across language, vision-language, and vision-language-action settings. Our experimental setup follows SparseLLM (Bai et al., 2024b) and its codebase[4]. Following existing work (Sun et al., 2023), we quantize all linear layers in LLM transformers. We implemented TTQ in PyTorch (Paszke et al., 2019) and used the Hugging Face Transformers library (Wolf, 2019) to load models and datasets. Model-quality experiments are conducted on NVIDIA A40 or A100 GPUs. Runtime benchmarks additionally use A40, A100, L40, RTX 3090, and RTX 4090 GPUs as specified in Appendix K; the full-model phase-throughput benchmark in Appendix J.1 uses an RTX 5090 GPU.

For LLM experiments, we first consider the OPT model family (Zhang et al., 2022) as it provides a wide range of model scales from 125M to 175B. We then evaluate more recent LLMs: Qwen3 (Yang et al., 2025) and Gemma3 (Gemma Team et al., 2025). The additional downstream-accuracy experiment uses the 3.8B-parameter Phi-4-mini-instruct model (Abouelenin et al., 2025) on GSM8K (Cobbe et al., 2021). We report results across model sizes to provide a broader view of TTQ performance. We measure perplexity on three common benchmarks: WikiText-2 (WT2) (Merity et al., 2016), the Penn Treebank (PTB) (Marcus et al., 1994), and C4 (Raffel et al., 2020). Details of the LLMs and datasets are provided in Appendices M and N.

J Additional controlled evaluations

The following controlled experiments isolate the main factors in TTQ and quantify runtime and statistical effects not captured by the decode-only microbenchmarks. They are reported separately to make the scope of each comparison explicit.

J.1 Full-model prefill and decode throughput

Table 11 reports full-model phase throughput for Qwen3 on an NVIDIA RTX 5090 GPU. For TTQ, the prefill measurement includes prompt processing, activation-statistics collection, preconditioner construction, and online weight quantization; decode reuses the resulting prompt-conditioned weights. AWQ and TTQ use the same `torch.compile(mode="max-autotune-no-cudagraphs")` and CUDA-graph replay framework.

TTQ reduces prefill throughput by approximately 11–24% relative to AWQ, while decode throughput is within 0–6.5% of AWQ over these models. Thus, online quantization has a real cost concentrated primarily in prefill; our earlier asymptotic argument in Eq. (5) should be interpreted as a scaling explanation rather than as evidence that the measured overhead is negligible at every prompt length. These measurements are full-model phase benchmarks, but they do not isolate the online-quantization

[4] https://github.com/BaiTheBest/SparseLLM

Table 11: Full-model prefill and decode throughput on RTX 5090. Prefill values are in M tokens/s and decode values are in k tokens/s. TTQ prefill includes its online activation-statistics and weight-quantization work.

Qwen3	AWQ prefill	TTQ prefill	AWQ decode	TTQ decode
0.6B	69.0	52.2	206.1	206.1
1.7B	38.3	32.6	206.8	193.3
4B	16.0	14.2	157.6	155.6
8B	10.1	8.2	142.7	142.2

Table 12: Matched ablation on Qwen3-1.7B with WikiText-2 at 3 bits. Lower perplexity is better.

Group size	AWQ ℓ_1	AWQ ℓ_2	AWQ ℓ_1, r=16	AWQ ℓ_2, r=16	TTQ ℓ_2, r=16
16	22.51	19.00	22.30	18.93	**17.81**
32	23.76	20.35	23.09	19.91	**18.29**
64	22.77	20.69	22.52	20.62	**19.46**
128	31.53	24.96	31.52	24.56	**22.33**

kernel itself and do not constitute a prompt-length sweep. Short prompts are therefore a particularly unfavorable regime that remains to be characterized.

J.2 Matched AWQ preconditioner and low-rank ablation

To separate the effects of preconditioning, the low-rank residual, and online estimation, Table 12 reports WikiText-2 perplexity for 3-bit Qwen3-1.7B while matching the AWQ residual architecture to TTQ. The ℓ_2 preconditioner accounts for a substantial portion of the improvement over the original ℓ_1 form, while adding the rank-16 residual to AWQ gives a smaller improvement. After matching both the ℓ_2 preconditioner and rank-16 residual, prompt-conditioned TTQ still lowers perplexity by 1.12–2.23 points over the corresponding offline AWQ baseline across the four group sizes. This ablation therefore isolates TTQ's main distinction as prompt-conditioned estimation and reuse rather than a fundamentally different quantization operator.

J.3 Downstream LLM accuracy and GPTQ comparison

Perplexity alone does not establish downstream task quality, so we additionally evaluate Phi-4-mini-instruct (Abouelenin et al., 2025) on the 1,319-example GSM8K test set (Cobbe et al., 2021). For the offline baselines, we use the standard `llm-compressor` implementations of GPTQ and AWQ; the AWQ recipe includes clipping, ℓ_1 scaling, and exponent search. Table 13 reports exact-match accuracy with binomial standard errors.

At 4 bits, the observed differences are within roughly one binomial standard error and are not statistically conclusive. At 3 bits, TTQ exceeds GPTQ and AWQ by 7.19 and 11.44 accuracy points, respectively, in the observed evaluation. GPTQ follows a different execution path, so this accuracy table does not establish a runtime comparison. We also do not report implementation-matched HQQ (Badri & Shaji, 2023) or Atom (Zhao et al., 2024) experiments and therefore do not claim empirical superiority over those methods.

J.4 Expanded X-VLA evaluation and uncertainty

The original X-VLA table used 800 episodes in total (200 per LIBERO suite), so small differences relative to FP32 could be dominated by evaluation variability. We therefore expanded the evaluation to 8,000 episodes across multiple random seeds, using the same action-inference and evaluation settings for every method. Table 14 reports success rate and the corresponding binomial standard error.

The ordering from the smaller evaluation persists, but the 0.65-point difference between TTQ ($r = 16$) and FP32 is only about 1.9 combined standard errors under an independence approximation. We

Table 13: Phi-4-mini-instruct exact-match accuracy on GSM8K. Parentheses report binomial standard error in percentage points.

Method	4-bit accuracy (SE)	3-bit accuracy (SE)
BF16	74.98 (1.19)	74.98 (1.19)
RTN	74.35 (1.20)	41.62 (1.36)
GPTQ	73.92 (1.21)	52.17 (1.38)
AWQ	74.83 (1.19)	47.92 (1.38)
TTQ	**75.97 (1.18)**	**59.36 (1.35)**

Table 14: Expanded X-VLA evaluation over 8,000 LIBERO episodes across multiple random seeds. The table reports success rate and binomial standard error.

Method	Success rate (%)	Binomial SE (%)
FP32	94.59	0.25
RTN	86.34	0.38
AWQ (best calibration domain)	93.66	0.27
TTQ ($r = 0$)	94.98	0.24
TTQ ($r = 16$)	**95.24**	0.24

therefore regard the FP32-exceeding result as suggestive rather than conclusive and do not attribute it causally to a regularization effect. Quantization can alter model decisions, and finite-sample evaluation variability can also change the observed score.

K Runtime benchmark

We show that TTQ can accelerate inference at test time relative to unquantized LLMs, even with the extra overhead of prescaling and low-rank factors. To evaluate TTQ runtime, we use the Marlin kernel (Frantar et al., 2025), one of the best-known `int_matmul` CUDA kernels. Specifically, we use `vllm._custom_ops.marlin_gemm` in the vLLM library[5]. It often outperforms `vllm._custom_ops.awq_gemm`. We use CUDA 12.8, PyTorch 2.9.1, and vLLM 0.15.1. For TTQ, we include prescaling by the diagonal matrix D and low-rank projections using B and A on top of the Marlin kernel, as shown below:

```
def forward(self, x):
    # quantization params on the fly
    W, S, Z, D = self.find_params(x)
    # low-rank projection
    y0 = (x @ self.A.t().contiguous()) @ self.B.t().contiguous()
    # pre-scaling
    x = x / D
    # scaled int_matmul
    return marlin_gemm(x, y0, W, S, Z)

def find_params(self, x):
    # prefill phase
    if torch.tensor(x.shape[:-1]).prod() > 1:
        # scale
        D = (x.norm(p=self.p, axis=0) + self.lam) ** self.alpha
        # prescale weight
        W = (self.weight * D).reshape(-1, self.groupsize)
        # scale, zeros
        Z = W.amin(axis=1, keepdim=True)
        S = (W.amax(axis=1, keepdim=True) - Z) / self.qmax
        # quantize
        W = ((W - Z) / S).round().clamp(0, self.qmax).to(torch.int32)
```

[5] https://github.com/vllm-project/vllm

Table 15: Runtime throughput (k tokens/sec) of the query-projection module for Qwen3 models with 4-bit AWQ and TTQ, on NVIDIA A40 GPU.

Qwen3	0.6B	1.7B	4B	8B	14B	32B
FP16	57.58	41.29	21.38	14.10	10.43	6.86
AWQ (`awq_gemm`)	75.76	71.60	54.61	32.68	25.28	18.28
AWQ (`marlin_gemm`)	73.32	73.70	**73.98**	**38.91**	**37.52**	**30.32**
TTQ ($r = 0$)	**80.63**	**78.68**	69.11	36.52	24.94	19.34
TTQ ($r = 16$)	66.67	59.08	49.67	30.32	23.78	17.43

×2.5

Table 16: Runtime throughput (k tokens/sec) of the query-projection module for Qwen3 models with 4-bit AWQ and TTQ, on NVIDIA A100 GPU.

Qwen3	0.6B	1.7B	4B	8B	14B	32B
FP16	68.43	**68.91**	42.76	30.69	20.16	15.90
AWQ (`awq_gemm`)	68.10	56.09	48.20	38.28	32.25	24.35
AWQ (`marlin_gemm`)	**69.41**	64.93	**58.41**	**53.13**	**49.07**	**42.28**
TTQ ($r = 0$)	61.85	57.72	52.67	48.58	45.17	39.40
TTQ ($r = 16$)	46.04	43.11	40.01	37.19	34.32	30.46

×1.9

```
        # cache
        self.W, self.S, self.Z, self.D = W, S, Z, D
    return self.W, self.S, self.Z, self.D
```

To reduce the overhead of multi-kernel launching, we use `torch.compile` and CUDA graph replay[6] for all cases. Since most `int_matmul` kernels provide little advantage during prefill, we focus on single-token decoding for runtime analysis to highlight the maximum potential speedup. Because these microbenchmarks omit other model operations, end-to-end acceleration is necessarily smaller. The adaptive preconditioner is computed during prefill.

Tables 15, 16, 17, 18, and 19 show runtime benchmarks for Qwen3 models on NVIDIA A40, A100, L40, RTX 3090 and RTX 4090 GPUs, respectively. To highlight the speedup, we benchmark the linear query-projection module for the original linear layer, offline 4-bit AWQ, and online 4-bit TTQ. The microbenchmarks show:

- Projection throughput generally decreases as model size grows.
- In these decode-only projection microbenchmarks, AWQ reaches up to a 6.7-fold speedup for the 32B model on RTX 4090 with the Marlin kernel.
- In these decode-only projection microbenchmarks, TTQ ($r = 0$) retains most of AWQ's throughput despite its extra prompt-dependent operations.
- Even without a custom TTQ kernel, TTQ with $r = 16$ low-rank projections reaches up to a 4.9-fold decode-only projection speedup for the 32B model on RTX 4090.
- The relative benefit tends to increase for larger LLMs because weight traffic increasingly dominates (e.g., Qwen3-32B query projection contains $5{,}120 \times 8{,}192$ FP16 weights, about 84 MB).

AWQ can fuse the prescaling D into preceding modules, whereas TTQ must fuse it into `int_matmul` for comparable decode efficiency. Most kernels do not yet support such prologue fusion; custom TTQ kernels could therefore improve GPU throughput. In addition, most kernels are not designed for arbitrary bit widths; a custom 2-bit TTQ kernel could further reduce packed-weight traffic.

[6] `https://docs.vllm.ai/en/stable/design/cuda_graphs/`

Table 17: Runtime throughput (k tokens/sec) of the query-projection module for Qwen3 models with 4-bit AWQ and TTQ, on NVIDIA L40 GPU.

Qwen3	0.6B	1.7B	4B	8B	14B	32B
FP16	**91.35**	75.18	74.60	53.31	44.45	34.74
AWQ (`awq_gemm`)	91.01	84.70	76.00	59.99	48.95	47.87
AWQ (`marlin_gemm`)	86.57	91.35	91.28	**90.93**	**77.81**	**68.15**
TTQ ($r = 0$)	90.59	**92.57**	**91.86**	84.19	71.28	63.37
TTQ ($r = 16$)	72.82	69.11	69.84	63.37	55.87	49.62

Table 18: Runtime throughput (k tokens/sec) of the query-projection module for Qwen3 models with 4-bit AWQ and TTQ, on NVIDIA RTX 3090 GPU.

Qwen3	0.6B	1.7B	4B	8B	14B	32B
FP16	59.75	54.38	29.53	20.37	14.15	9.77
AWQ (`awq_gemm`)	62.89	60.54	48.39	33.48	27.92	21.62
AWQ (`marlin_gemm`)	63.13	**67.86**	**65.88**	**46.97**	**36.30**	**26.17**
TTQ ($r = 0$)	**68.07**	67.34	64.43	43.10	34.25	24.96
TTQ ($r = 16$)	57.57	58.72	44.73	33.20	28.45	21.85

L LLM BENCHMARK RESULTS

Tables 20, 21, and 22 report full results for additional OPT, Qwen3, and Gemma3 model sizes. The reported $\pm$ values denote the standard deviation across the three dataset perplexities (WT2/PTB/C4), not repeated-run uncertainty.

M LLM MODELS

OPT family Open Pre-trained Transformer (OPT) (Zhang et al., 2022) is a family of decoder-only pre-trained transformers ranging from 125M to 175B parameters. The OPT paper reports that OPT-175B is comparable to GPT-3 while requiring roughly one-seventh of its development carbon footprint. Table 23 shows model parameters for the OPT open LLM family.

Qwen3 family We use Qwen3 (Yang et al., 2025) dense models, which are decoder-only transformers spanning 0.6B to 32B parameters, built on a consistent architecture with RMSNorm, SwiGLU feed-forward layers, and rotary positional embeddings. All variants employ grouped-query attention (GQA) with a fixed small number of key–value heads while scaling the number of query heads with model width, reducing KV-cache memory cost. Importantly, the hidden size is decoupled from the attention projection width, providing additional flexibility. Parameters are listed in Table 24.

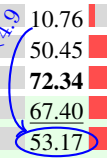
Phi-4-mini family Phi-4-mini-instruct (Abouelenin et al., 2025) is a 3.8B-parameter dense decoder-only Transformer with grouped-query attention and a 128K-token context window. We use `microsoft/Phi-4-mini-instruct` for the additional GSM8K quantization study in Appendix J.3. The model repository is released under the MIT license.

Gemma3 family Gemma3 models (Gemma Team et al., 2025) are decoder-only transformer architectures released across a wide range of scales, from 270M to 27B parameters, and include both text-only and multimodal variants. Similar to Qwen3, all Gemma3 models adopt RMSNorm, SwiGLU feed-forward networks, and rotary positional embeddings, as well as grouped-query attention (GQA). The per-head dimension is fixed at 256 across model sizes. Parameters are listed in Table 25.

LLaVA family LLaVA (Liu et al., 2023b) integrates a vision encoder with an LLM to process both visual and language context through instruction tuning. The vision encoder typically uses a contrastive

Table 19: Runtime throughput (k tokens/sec) of the query-projection module for Qwen3 models with 4-bit AWQ and TTQ, on NVIDIA RTX 4090 GPU.

Qwen3	0.6B	1.7B	4B	8B	14B	32B
FP16	116.82	77.04	72.45	58.44	46.78	10.76
AWQ (`awq_gemm`)	113.84	90.42	80.19	62.94	51.32	50.45
AWQ (`marlin_gemm`)	**120.62**	**115.20**	**112.61**	**101.00**	**80.45**	**72.34**
TTQ ($r = 0$)	108.00	104.11	103.45	92.23	74.72	67.40
TTQ ($r = 16$)	77.88	73.47	72.55	66.62	58.65	53.17



language-image pretraining (CLIP) Vision Transformer (ViT), while the language backbone is based on LLaMA or Vicuna. The LLaVA paper reports that LLaVA-1.6-34B outperforms Gemini Pro on some benchmarks, including MMMU and MathVista.

Qwen3-VL family Qwen3-VL (Bai et al., 2025) is a family of multimodal LLMs that extend the Qwen3 transformer with vision–language capabilities. The models integrate a SigLIP-based vision encoder with the Qwen3 language backbone through a projection module, enabling joint reasoning over images, videos, and text for tasks such as visual question answering, captioning, and document understanding. Parameters are listed in Table 26.

$\pi_{0.5}$ family The $\pi_{0.5}$ model (Intelligence et al., 2025) is a PaliGemma-based VLA transformer with 2.3B parameters that maps visual observations and language instructions directly to continuous robot actions through a flow-matching diffusion policy. The model is trained via distillation from a larger foundation model on diverse robot-interaction data and is designed for generalization across manipulation tasks. Parameters are listed in Table 27.

X-VLA family X-VLA (Cross-modal Vision-Language-Action) (Zheng et al., 2025) is a unified framework that jointly models visual perception, language understanding, and action generation within a shared representation space. X-VLA leverages multimodal pretraining to model compositional and temporal structure across embodied tasks and environments. It uses Florence-2 (Xiao et al., 2024) as a VLM backbone.

Table 20: Perplexity (↓) of OPT models with different quantization methods. Values report the macro average and cross-dataset standard deviation across WT2/PTB/C4. Group size is $g = 32$ for all cases. Calibration token length is $T = 2^{17}$ for AWQ. **Bold** and underlined values denote the best and second-best results, respectively. An asterisk "*" marks performance considered competitive with the original uncompressed LLM.

Quantization q	2 bits	3 bits	4 bits	5 bits
OPT-125M (WT2: 27.7, PTB: 39.0, C4: 26.6, Avg: **31.1**$_{\pm 6.8}$)				
RTN	5058.5$_{\pm 833.7}$	56.3$_{\pm 10.5}$	33.5$_{\pm 7.0}$	31.8$_{\pm 7.0}$
AWQ (WT2 Calib)	381.7$_{\pm 159.8}$	37.4$_{\pm 7.4}$	32.3$_{\pm 6.5}$	31.4$_{\pm 6.1}$
AWQ (PTB Calib)	375.3$_{\pm 84.9}$	37.3$_{\pm 6.9}$	32.2$_{\pm 6.3}$	31.3$_{\pm 6.0}$
AWQ (C4 Calib)	451.7$_{\pm 181.7}$	37.7$_{\pm 7.5}$	32.6$_{\pm 6.6}$	31.3$_{\pm 6.0}$
TTQ ($r = 0$)	257.4$_{\pm 65.3}$	36.6$_{\pm 8.2}$	31.9$_{\pm 7.0}$	31.2$_{\pm 6.8}$
TTQ ($r = 16$)	**141.7**$_{\pm 36.9}$	**35.8**$_{\pm 8.3}$	**31.8**$_{\pm 7.0}$	***31.1**$_{\pm 6.8}$
OPT-350M (WT2: 22.0, PTB: 31.1, C4: 22.6, Avg: **25.2**$_{\pm 5.1}$)				
RTN	25808.8$_{\pm 9601.0}$	55.3$_{\pm 13.8}$	27.6$_{\pm 5.5}$	25.6$_{\pm 5.3}$
AWQ (WT2 Calib)	293.2$_{\pm 103.5}$	28.8$_{\pm 5.1}$	25.8$_{\pm 4.5}$	25.4$_{\pm 4.4}$
AWQ (PTB Calib)	350.7$_{\pm 107.0}$	28.9$_{\pm 5.1}$	25.8$_{\pm 4.5}$	25.4$_{\pm 4.4}$
AWQ (C4 Calib)	322.8$_{\pm 98.6}$	29.0$_{\pm 5.6}$	25.9$_{\pm 4.5}$	25.4$_{\pm 4.4}$
TTQ ($r = 0$)	358.3$_{\pm 123.9}$	28.6$_{\pm 6.0}$	**25.7**$_{\pm 5.3}$	**25.3**$_{\pm 5.1}$
TTQ ($r = 16$)	**129.8**$_{\pm 46.8}$	**27.9**$_{\pm 5.9}$	**25.7**$_{\pm 5.2}$	**25.3**$_{\pm 5.1}$
OPT-1.3B (WT2: 14.6, PTB: 20.3, C4: 16.1, Avg: **17.0**$_{\pm 3.0}$)				
RTN	11514.4$_{\pm 2621.8}$	27.2$_{\pm 6.8}$	18.1$_{\pm 3.4}$	17.2$_{\pm 3.1}$
AWQ (WT2 Calib)	32.4$_{\pm 8.6}$	18.0$_{\pm 2.8}$	17.3$_{\pm 2.6}$	***17.0**$_{\pm 2.5}$
AWQ (PTB Calib)	32.6$_{\pm 6.6}$	18.1$_{\pm 2.7}$	17.3$_{\pm 2.6}$	***17.0**$_{\pm 2.5}$
AWQ (C4 Calib)	31.7$_{\pm 8.1}$	18.0$_{\pm 2.8}$	17.2$_{\pm 2.6}$	***17.0**$_{\pm 2.5}$
TTQ ($r = 0$)	32.2$_{\pm 8.3}$	18.2$_{\pm 3.4}$	17.2$_{\pm 3.0}$	***17.0**$_{\pm 3.0}$
TTQ ($r = 16$)	**27.2**$_{\pm 6.1}$	**17.9**$_{\pm 3.1}$	**17.1**$_{\pm 3.0}$	***17.0**$_{\pm 3.0}$
OPT-2.7B (WT2: 12.5, PTB: 18.0, C4: 14.3, Avg: **14.9**$_{\pm 2.8}$)				
RTN	6274.5$_{\pm 1350.7}$	36.0$_{\pm 11.5}$	15.7$_{\pm 3.1}$	15.0$_{\pm 2.8}$
AWQ (WT2 Calib)	23.1$_{\pm 5.3}$	15.7$_{\pm 3.0}$	15.1$_{\pm 2.8}$	15.0$_{\pm 2.8}$
AWQ (PTB Calib)	23.2$_{\pm 4.0}$	15.7$_{\pm 3.0}$	**15.0**$_{\pm 2.9}$	15.0$_{\pm 2.9}$
AWQ (C4 Calib)	22.9$_{\pm 5.1}$	15.7$_{\pm 3.0}$	**15.0**$_{\pm 2.8}$	15.0$_{\pm 2.8}$
TTQ ($r = 0$)	23.7$_{\pm 5.1}$	15.7$_{\pm 3.0}$	**15.0**$_{\pm 2.8}$	***14.9**$_{\pm 2.8}$
TTQ ($r = 16$)	**21.2**$_{\pm 4.5}$	**15.5**$_{\pm 2.9}$	**15.0**$_{\pm 2.8}$	***14.9**$_{\pm 2.8}$
OPT-6.7B (WT2: 10.9, PTB: 15.8, C4: 12.7, Avg: **13.1**$_{\pm 2.5}$)				
RTN	5716.5$_{\pm 664.0}$	26.2$_{\pm 8.9}$	13.7$_{\pm 2.8}$	13.2$_{\pm 2.5}$
AWQ (WT2 Calib)	17.2$_{\pm 3.6}$	13.5$_{\pm 2.6}$	13.2$_{\pm 2.5}$	***13.1**$_{\pm 2.5}$
AWQ (PTB Calib)	17.1$_{\pm 3.0}$	13.6$_{\pm 2.6}$	13.2$_{\pm 2.5}$	***13.1**$_{\pm 2.5}$
AWQ (C4 Calib)	16.9$_{\pm 3.5}$	13.6$_{\pm 2.6}$	13.2$_{\pm 2.6}$	***13.1**$_{\pm 2.5}$
TTQ ($r = 0$)	17.2$_{\pm 3.5}$	13.6$_{\pm 2.6}$	13.2$_{\pm 2.5}$	***13.1**$_{\pm 2.5}$
TTQ ($r = 16$)	**16.3**$_{\pm 3.4}$	**13.4**$_{\pm 2.6}$	***13.1**$_{\pm 2.5}$	***13.1**$_{\pm 2.5}$
OPT-13B (WT2: 10.1, PTB: 14.5, C4: 12.1, Avg: **12.2**$_{\pm 2.2}$)				
RTN	6413.1$_{\pm 1051.0}$	15.5$_{\pm 3.1}$	12.5$_{\pm 2.4}$	12.3$_{\pm 2.3}$
AWQ (WT2 Calib)	15.3$_{\pm 2.9}$	12.6$_{\pm 2.3}$	**12.3**$_{\pm 2.2}$	***12.2**$_{\pm 2.2}$
AWQ (PTB Calib)	15.4$_{\pm 2.7}$	12.6$_{\pm 2.3}$	**12.3**$_{\pm 2.2}$	***12.2**$_{\pm 2.2}$
AWQ (C4 Calib)	15.2$_{\pm 2.9}$	12.6$_{\pm 2.3}$	**12.3**$_{\pm 2.2}$	***12.2**$_{\pm 2.2}$
TTQ ($r = 0$)	15.0$_{\pm 2.9}$	**12.5**$_{\pm 2.7}$	**12.3**$_{\pm 2.2}$	***12.2**$_{\pm 2.2}$
TTQ ($r = 16$)	**14.8**$_{\pm 2.8}$	**12.5**$_{\pm 2.2}$	**12.3**$_{\pm 2.2}$	***12.2**$_{\pm 2.2}$

Table 21: Perplexity (↓) of Qwen3 models with different quantization methods. Values report the macro average and cross-dataset standard deviation across WT2/PTB/C4. Group size is $g = 32$ for all cases. Calibration token length is $T = 2^{17}$ for AWQ. **Bold** and underlined values denote the best and second-best results, respectively. An asterisk "*" marks performance considered competitive with the original uncompressed LLM.

Quantization q	2 bits	3 bits	4 bits	5 bits
Qwen3-0.6B (WT2: 21.0, PTB: 43.8, C4: 30.3, Avg: **31.7**$_{\pm 11.5}$)				
RTN	8.2e6$_{\pm 4.4e6}$	127.3$_{\pm 54.1}$	38.2$_{\pm 14.7}$	33.5$_{\pm 12.4}$
AWQ (WT2 Calib)	9739.1$_{\pm 2765.7}$	49.4$_{\pm 20.2}$	33.5$_{\pm 11.7}$	32.4$_{\pm 11.8}$
AWQ (PTB Calib)	17344.3$_{\pm 1962.7}$	47.9$_{\pm 17.3}$	34.1$_{\pm 12.5}$	32.0$_{\pm 11.5}$
AWQ (C4 Calib)	5388.1$_{\pm 1862.6}$	48.3$_{\pm 17.3}$	33.0$_{\pm 11.6}$	32.1$_{\pm 11.8}$
TTQ ($r = 0$)	2827.8$_{\pm 883.8}$	44.7$_{\pm 16.9}$	33.0$_{\pm 12.1}$	**31.9**$_{\pm 11.3}$
TTQ ($r = 16$)	**1552.6**$_{\pm 386.3}$	**42.0**$_{\pm 16.0}$	**32.9**$_{\pm 12.2}$	**31.9**$_{\pm 11.3}$
Qwen3-1.7B (WT2: 16.7, PTB: 33.8, C4: 16.1, Avg: **24.2**$_{\pm 8.7}$)				
RTN	1.4e6$_{\pm 2.2e5}$	162.8$_{\pm 107.4}$	30.6$_{\pm 12.2}$	26.1$_{\pm 9.6}$
AWQ (WT2 Calib)	1864.7$_{\pm 2051.9}$	30.0$_{\pm 12.0}$	24.5$_{\pm 8.7}$	24.4$_{\pm 8.6}$
AWQ (PTB Calib)	2309.6$_{\pm 2524.0}$	29.8$_{\pm 11.6}$	24.7$_{\pm 8.3}$	24.5$_{\pm 8.9}$
AWQ (C4 Calib)	2364.7$_{\pm 2878.7}$	28.2$_{\pm 9.9}$	24.9$_{\pm 9.1}$	24.4$_{\pm 8.8}$
TTQ ($r = 0$)	522.6$_{\pm 259.4}$	27.3$_{\pm 9.6}$	24.4$_{\pm 8.5}$	24.3$_{\pm 8.5}$
TTQ ($r = 16$)	**264.6**$_{\pm 90.4}$	**26.4**$_{\pm 9.3}$	**24.3**$_{\pm 8.8}$	***24.1**$_{\pm 8.5}$
Qwen3-4B (WT2: 13.7, PTB: 24.8, C4: 19.9, Avg: **19.4**$_{\pm 5.6}$)				
RTN	5803.3$_{\pm 2179.1}$	28.6$_{\pm 10.1}$	21.4$_{\pm 6.4}$	19.6$_{\pm 5.5}$
AWQ (WT2 Calib)	180.8$_{\pm 65.3}$	21.2$_{\pm 6.4}$	20.1$_{\pm 5.7}$	***19.4**$_{\pm 5.6}$
AWQ (PTB Calib)	199.2$_{\pm 29.7}$	26.6$_{\pm 9.0}$	19.8$_{\pm 5.9}$	***19.4**$_{\pm 5.6}$
AWQ (C4 Calib)	246.9$_{\pm 85.2}$	21.7$_{\pm 6.3}$	19.9$_{\pm 5.5}$	19.6$_{\pm 5.6}$
TTQ ($r = 0$)	120.4$_{\pm 34.3}$	**20.9**$_{\pm 6.0}$	**19.6**$_{\pm 5.5}$	***19.4**$_{\pm 5.5}$
TTQ ($r = 16$)	**78.1**$_{\pm 19.5}$	21.1$_{\pm 6.2}$	**19.6**$_{\pm 5.6}$	***19.4**$_{\pm 5.5}$
Qwen3-8B (WT2: 9.7, PTB: 17.2, C4: 15.4, Avg: **14.1**$_{\pm 3.9}$)				
RTN	5366.8$_{\pm 1985.5}$	18.4$_{\pm 5.7}$	14.9$_{\pm 4.1}$	14.4$_{\pm 3.9}$
AWQ (WT2 Calib)	39.5$_{\pm 12.9}$	15.3$_{\pm 4.2}$	14.4$_{\pm 4.0}$	14.2$_{\pm 4.0}$
AWQ (PTB Calib)	39.4$_{\pm 9.3}$	15.4$_{\pm 4.2}$	14.4$_{\pm 4.0}$	14.2$_{\pm 3.9}$
AWQ (C4 Calib)	37.6$_{\pm 10.9}$	15.2$_{\pm 4.1}$	14.4$_{\pm 4.0}$	***14.1**$_{\pm 3.9}$
TTQ ($r = 0$)	32.3$_{\pm 8.8}$	**15.0**$_{\pm 4.1}$	**14.2**$_{\pm 3.9}$	***14.1**$_{\pm 3.9}$
TTQ ($r = 16$)	**30.2**$_{\pm 8.0}$	**15.0**$_{\pm 4.1}$	14.3$_{\pm 3.9}$	***14.1**$_{\pm 3.9}$
Qwen3-14B (WT2: 8.6, PTB: 15.2, C4: 13.8, Avg: **12.6**$_{\pm 3.5}$)				
RTN	366.3$_{\pm 107.7}$	15.6$_{\pm 4.6}$	12.9$_{\pm 3.7}$	12.8$_{\pm 3.6}$
AWQ (WT2 Calib)	23.0$_{\pm 7.4}$	13.4$_{\pm 3.7}$	12.7$_{\pm 3.5}$	***12.6**$_{\pm 3.4}$
AWQ (PTB Calib)	23.0$_{\pm 6.7}$	13.3$_{\pm 3.7}$	12.7$_{\pm 3.5}$	***12.6**$_{\pm 3.5}$
AWQ (C4 Calib)	23.3$_{\pm 7.5}$	13.4$_{\pm 3.7}$	12.7$_{\pm 3.5}$	***12.6**$_{\pm 3.4}$
TTQ ($r = 0$)	21.1$_{\pm 6.5}$	13.2$_{\pm 3.6}$	12.7$_{\pm 3.5}$	***12.6**$_{\pm 3.5}$
TTQ ($r = 16$)	**20.3**$_{\pm 6.4}$	**13.1**$_{\pm 3.6}$	***12.6**$_{\pm 3.5}$	***12.6**$_{\pm 3.5}$

Table 22: Perplexity (↓) of Gemma3 models with different quantization methods. Values report the macro average and cross-dataset standard deviation across WT2/PTB/C4. Group size is $g = 32$ for all cases. Calibration token length is $T = 2^{17}$ for AWQ. **Bold** and underlined values denote the best and second-best results, respectively. An asterisk "*" marks performance considered competitive with the original uncompressed LLM.

Quantization q	2 bits	3 bits	4 bits	5 bits
Gemma3-270M (WT2: 70.1, PTB: 698.7, C4: 68.1, Avg: **279.0**$_{\pm 353.5}$)				
RTN	2.6e11$_{\pm 1.2e11}$	1795.0$_{\pm 2207.5}$	391.9$_{\pm 505.5}$	315.0$_{\pm 410.9}$
AWQ (WT2 Calib)	89188.4$_{\pm 58307.2}$	517.4$_{\pm 690.0}$	279.6$_{\pm 348.6}$	306.0$_{\pm 405.2}$
AWQ (PTB Calib)	1.9e5$_{\pm 2.0e9}$	537.7$_{\pm 711.8}$	310.0$_{\pm 408.2}$	293.4$_{\pm 386.4}$
AWQ (C4 Calib)	1.3e5$_{\pm 1.6e5}$	598.1$_{\pm 832.9}$	326.4$_{\pm 420.5}$	*276.8$_{\pm 358.0}$
TTQ ($r = 0$)	30199.0$_{\pm 23705.9}$	387.7$_{\pm 489.0}$	*277.0$_{\pm 357.1}$	*262.5$_{\pm 335.2}$
TTQ ($r = 16$)	**19657.0**$_{\pm 17934.2}$	**382.8**$_{\pm 486.2}$	***275.7**$_{\pm 353.6}$	***261.5**$_{\pm 343.1}$
Gemma3-1B (WT2: 27.7, PTB: 212.4, C4: 33.2, Avg: **91.1**$_{\pm 105.1}$)				
RTN	8.6e5$_{\pm 7.7e5}$	209.4$_{\pm 253.2}$	111.1$_{\pm 130.0}$	96.6$_{\pm 112.0}$
AWQ (WT2 Calib)	4734.7$_{\pm 5394.7}$	134.7$_{\pm 159.3}$	91.9$_{\pm 104.1}$	95.9$_{\pm 112.2}$
AWQ (PTB Calib)	9326.6$_{\pm 11932.8}$	138.9$_{\pm 167.5}$	99.2$_{\pm 115.1}$	95.5$_{\pm 112.0}$
AWQ (C4 Calib)	5486.9$_{\pm 6662.0}$	150.8$_{\pm 186.3}$	93.5$_{\pm 106.0}$	93.6$_{\pm 108.4}$
TTQ ($r = 0$)	1928.5$_{\pm 2313.2}$	127.3$_{\pm 150.5}$	***89.9**$_{\pm 100.8}$	***90.2**$_{\pm 103.3}$
TTQ ($r = 16$)	**1804.9**$_{\pm 1850.3}$	**114.5**$_{\pm 130.8}$	91.7$_{\pm 107.4}$	*90.3$_{\pm 103.6}$
Gemma3-4B (WT2: 17.4, PTB: 641.5, C4: 23.3, Avg: **227.4**$_{\pm 358.6}$)				
RTN	8.1e6$_{\pm 1.4e7}$	606.4$_{\pm 997.2}$	251.0$_{\pm 395.1}$	257.8$_{\pm 410.5}$
AWQ (WT2 Calib)	5667.3$_{\pm 9506.0}$	351.7$_{\pm 565.7}$	*225.3$_{\pm 353.5}$	*221.9$_{\pm 348.7}$
AWQ (PTB Calib)	4444.0$_{\pm 7433.4}$	352.9$_{\pm 571.5}$	240.4$_{\pm 380.6}$	240.5$_{\pm 380.9}$
AWQ (C4 Calib)	4938.5$_{\pm 8259.2}$	366.3$_{\pm 593.1}$	245.6$_{\pm 388.9}$	245.8$_{\pm 389.7}$
TTQ ($r = 0$)	**3561.6**$_{\pm 6003.4}$	**239.5**$_{\pm 375.8}$	*222.2$_{\pm 349.4}$	***221.7**$_{\pm 349.1}$
TTQ ($r = 16$)	3974.9$_{\pm 6696.8}$	251.7$_{\pm 396.6}$	***222.0**$_{\pm 349.1}$	*221.9$_{\pm 353.6}$
Gemma3-12B (WT2: 25.1, PTB: 283.8, C4: 35.0, Avg: **114.6**$_{\pm 146.6}$)				
RTN	15467.0$_{\pm 13512.6}$	294.9$_{\pm 390.2}$	146.6$_{\pm 196.1}$	*108.3$_{\pm 136.5}$
AWQ (WT2 Calib)	1700.6$_{\pm 2328.9}$	147.4$_{\pm 196.4}$	125.9$_{\pm 166.1}$	114.7$_{\pm 149.9}$
AWQ (PTB Calib)	4193.8$_{\pm 6142.5}$	128.3$_{\pm 164.6}$	117.3$_{\pm 146.2}$	122.3$_{\pm 157.6}$
AWQ (C4 Calib)	1464.4$_{\pm 2003.8}$	*100.8$_{\pm 122.2}$	146.0$_{\pm 197.0}$	132.0$_{\pm 173.5}$
TTQ ($r = 0$)	**1295.6**$_{\pm 1743.4}$	***84.6**$_{\pm 94.1}$	***86.6**$_{\pm 102.3}$	***92.9**$_{\pm 112.4}$
TTQ ($r = 16$)	1620.9$_{\pm 2248.4}$	*100.7$_{\pm 116.9}$	*91.2$_{\pm 109.4}$	*98.3$_{\pm 121.1}$

Table 23: OPT model parameters (Zhang et al., 2022)

Model	layers	heads	hidden size	head dim	MLP dim	Hugging Face ID
125M	12	12	768	64	3072	facebook/opt-125m
350M	24	16	1024	64	4096	facebook/opt-350m
1.3B	24	32	2048	64	8192	facebook/opt-1.3b
2.7B	32	32	2560	80	10240	facebook/opt-2.7b
6.7B	32	32	4096	128	16384	facebook/opt-6.7b
13B	40	40	5120	128	20480	facebook/opt-13b
30B	48	56	7168	128	28672	facebook/opt-30b
66B	64	72	9216	128	36864	facebook/opt-66b

Table 24: Architecture parameters of Qwen3 dense models (Yang et al., 2025)

Model	layers	heads	KV heads	hidden size	head dim	MLP dim	Hugging Face ID
0.6B	28	16	8	1024	128	3072	Qwen/Qwen3-0.6B
1.7B	28	16	8	2048	128	6144	Qwen/Qwen3-1.7B
4B	36	32	8	2560	128	9728	Qwen/Qwen3-4B
8B	36	32	8	4096	128	12288	Qwen/Qwen3-8B
14B	40	40	8	5120	128	17408	Qwen/Qwen3-14B
32B	64	64	8	5120	128	25600	Qwen/Qwen3-32B

Table 25: Gemma3 instruction-tuned text transformer parameters (Gemma Team et al., 2025)

Model	layers	heads	KV heads	hidden size	head dim	MLP dim	Hugging Face ID
270M	18	4	1	640	256	2048	google/gemma-3-270m-it
1B	26	4	1	1152	256	6912	google/gemma-3-1b-it
4B	34	8	4	2560	256	10240	google/gemma-3-4b-it
12B	48	16	8	3840	256	15360	google/gemma-3-12b-it
27B	62	32	16	5376	256	21504	google/gemma-3-27b-it

Table 26: Qwen3-VL transformer parameters (Bai et al., 2025)

Model	layers	heads	KV heads	hidden size	head dim	MLP dim	Hugging Face ID
2B	24	16	8	2048	128	11008	Qwen/Qwen3-VL-2B-Instruct
4B	32	32	8	4096	128	22016	Qwen/Qwen3-VL-4B-Instruct
8B	36	32	8	4096	128	22016	Qwen/Qwen3-VL-8B-Instruct
32B	64	40	8	5120	128	27392	Qwen/Qwen3-VL-32B-Instruct

Table 27: $\pi_{0.5}$ VLA transformer parameters (Intelligence et al., 2025): Hugging Face ID lerobot/pi05_libero_finetuned

Module	layers	heads	KV heads	hidden size	head dim	MLP dim
LLM: Gemma-2B	18	8	1	2048	256	16384
Vision Encoder: SigLIP ViT-L	24	16	16	1024	64	4096
Diffusion Policy: Gemma-300M	18	8	1	1024	128	4096

N DATASETS

WikiText-2 (WT2) WikiText-2 (Merity et al., 2016) is a language-modeling corpus constructed from verified Good and Featured Wikipedia articles while retaining case, punctuation, and numbers. It contains approximately 2.09M training tokens, with 217,646 validation tokens and 245,569 test tokens; the Hugging Face `wikitext-2-raw-v1` representation contains 36,718, 3,760, and 4,358 text records in the train, validation, and test splits, respectively. We use `https://huggingface.co/datasets/Salesforce/wikitext`, configuration `wikitext-2-raw-v1`. The current repository metadata lists CC BY-SA-3.0 and GFDL licensing.

Penn Treebank (PTB) The Penn Treebank (PTB) corpus (Marcus et al., 1994) is a standard sequence-modeling benchmark derived from 1989 Wall Street Journal material; the text-only release describes roughly one million words, with rare words and numbers normalized to special tokens. We use `https://huggingface.co/datasets/ptb-text-only/ptb_text_only`. The Hugging Face card labels the license as "other" and states that the dataset is provided for research purposes only; the underlying Penn Treebank distribution is governed by the applicable Linguistic Data Consortium user agreement rather than an open-source license.

C4 C4 (Raffel et al., 2020) is the English Colossal Clean Crawled Corpus derived from Common Crawl. We use the "en" configuration from `https://huggingface.co/datasets/allenai/c4`, using the first shard of each split in our experiments. The repository releases C4 under ODC-BY-1.0 and additionally notes that use of the underlying content is subject to the Common Crawl terms of use.

GSM8K GSM8K (Cobbe et al., 2021) contains approximately 8.5k linguistically diverse grade-school mathematics word problems requiring multi-step reasoning, with 7,473 training examples and 1,319 test examples. We use `https://huggingface.co/datasets/openai/gsm8k`; the dataset is released under the MIT license.

TextVQA TextVQA (Singh et al., 2019) evaluates visual question answering that requires reading and reasoning about scene text. It contains 45,336 questions over 28,408 images sourced from Open Images. We use the standard TextVQA release represented at `https://huggingface.co/datasets/facebook/textvqa`. The TextVQA dataset card lists CC BY-4.0; Open Images separately licenses its annotations under CC BY-4.0 and lists source images as CC BY-2.0 while recommending verification of the license associated with each image.

COCO-Caption COCO-Caption is the image-captioning task built on Microsoft COCO (Lin et al., 2014), with multiple human-written captions associated with each image. We use the formatted COCO-Caption2017 evaluation repository at `https://huggingface.co/datasets/lmms-lab/COCO-Caption2017`. Under the COCO terms, the annotations are licensed under CC BY-4.0, whereas the COCO Consortium does not own the image copyrights and use of the images remains subject to the corresponding Flickr terms and image licenses.

OK-VQA OK-VQA (Marino et al., 2019) is a knowledge-based visual question-answering benchmark whose questions require external knowledge beyond the visible image content. The official release provides OK-VQA questions and annotations together with links to MS COCO images; our formatted evaluation set is `https://huggingface.co/datasets/lmms-lab/OK-VQA`, whose `val2014` split contains 5,046 examples. The official OK-VQA download page does not state a separate blanket license for the question/answer annotations, so we do not assign one here; the linked images remain subject to the MS COCO/Flickr terms described above.

ChartQA ChartQA (Masry et al., 2022) evaluates visual and logical question answering over charts using both human-authored and machine-generated questions. We use the formatted test split at `https://huggingface.co/datasets/lmms-lab/ChartQA`, which contains 2,500 examples. The official ChartQA repository is released under GPL-3.0. Because some chart images were collected from external sources, their underlying content may also remain subject to source-specific rights and terms.

LIBERO LIBERO (Liu et al., 2023a) is a robot-learning benchmark designed to study knowledge transfer across manipulation tasks. We evaluate the Spatial, Object, Goal, and Long suites. We use the LeRobot-formatted dataset `https://huggingface.co/datasets/lerobot/libero`, which contains 1,693 episodes, 273,465 frames, and 40 tasks for these demonstrations. This repository lists the Apache-2.0 license.