ORQA: FAST AND MEMORY-EFFICIENT HYBRID REASONING WITH A QUANTIZED REASONER

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning improves the capability of large language models, but long reasoning traces increase latency and decoding cost. Hybrid reasoning reduces this cost by offloading intermediate reasoning to a lightweight model, which we call an efficient reasoner. Existing methods, however, require extra model parameters and KV cache, increasing memory consumption. Moreover, the capability gap between the reasoner and the target LLM often limits speedup or reduces accuracy. To this end, we propose ORQA, a hybrid reasoning framework that improves speed and accuracy over prior methods without adding memory overhead. Instead of loading an extra reasoner, ORQA uses a quantized counterpart of the target LLM and a shared KV cache pool. We propose weight infusion, which integrates parameters between the target LLM and reasoner, eliminating separate model storage while maintaining lossless inference for both. We further devise Quant-ifier, a routing signal designed for a quantized reasoner that predicts token disagreement using the quantization error distribution. Experiments show that ORQA improves speed and accuracy over prior methods with no additional memory.

1 INTRODUCTION

Reasoning has emerged as an effective form of test-time scaling for large language models (LLMs), enabling stronger performance on tasks that require multi-step computation, planning, or verification (Wei et al., 2022; Snell et al., 2024; Guo et al., 2025; Muennighoff et al., 2025). This improvement, however, comes at a substantial inference cost. Reasoning traces often contain many more tokens than the final answer, and each generated token incurs an autoregressive decoding cost. Therefore, long reasoning increases end-to-end latency, decoding computation, and KV cache memory, making efficient reasoning a critical challenge for practical deployment (Sui et al., 2025; Chen et al., 2024b).

Hybrid reasoning has been explored to reduce this cost (Pan et al., 2025; Yang et al., 2025b; Wang et al., 2026). These methods offload intermediate reasoning to a lightweight model called a *reasoner*, while reserving the full-capability LLM, or *target model*, for steps where its capability is needed. In principle, hybrid reasoning reduces expensive target model decoding while preserving its quality.

Existing hybrid reasoning systems, however, typically use independent small language models as reasoners. This design incurs extra model weights and KV cache, increasing memory overhead. Such additional memory is problematic in practical inference, where accelerator memory constrains the usable batch size, context length, and model configuration (Sheng et al., 2023; Kwon et al., 2023; Pope et al., 2023). Beyond memory, the small reasoner has substantially weaker reasoning capability than the target LLM. At the same time, behavioral differences between the two models make it hard to determine when target intervention is necessary. As a result, prior methods often require more memory, achieve only modest speedups, or fall behind the target model in quality.

These limitations motivate a unique design: the reasoner should be tied to the target model rather than deployed as an independent auxiliary model. A quantized counterpart of the target model is a natural candidate: it enables cheaper decoding than the full-precision target model while remaining better aligned than unrelated small reasoners. To make this design practical and effective, however, the system must avoid storing a separate model and an additional reasoner KV cache, and adapt a routing policy that directly reflects the primary source of token-level disagreement.

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.

Hence, we introduce ORQA (Offloaded Reasoning with Quantized Assistance), a hybrid reasoning framework that uses a quantized counterpart of the target model as the reasoner. ORQA is built on the following techniques: **Weight infusion** proposes an integrated parameter representation that supports lossless decoding for both models, eliminating additional memory of hybrid reasoning. We further devise **Quant-ifier**, which estimates token-level disagreements incurred by the quantization error from the quantized reasoner. Additionally, ORQA improves the quantized reasoner's cost efficiency and quality by **KV transplant**, which reuses the target model's KV cache during reasoner decoding. Our results with ORQA demonstrate that quantized counterparts serve as effective reasoners, offering a favorable balance of accuracy, memory efficiency, and throughput.

Our contributions are summarized as follows:

- We propose **ORQA**, a hybrid reasoning framework that uses a quantized counterpart of the target model as the reasoner, improving speed and accuracy over prior hybrid methods without introducing any additional memory consumption.
- We devise **weight infusion** and **Quant-ifier**. Weight infusion supports target model and reasoner decoding from a single adapted parameter set, while Quant-ifier estimates quantization-induced token disagreements. ORQA further enhances the quality and efficiency of the quantized reasoner through **KV transplant** and extra calibration strategies.
- We empirically demonstrate that ORQA improves accuracy and speed over prior hybrid reasoning methods with no additional model or KV cache memory.

2 BACKGROUND

Autoregressive decoding. Let F_W denote the forward operator of a causal Transformer with weights W. For a prefix $x_{\leq k} = (x_0, \ldots, x_k)$, F_W maps the prefix to next-token logits. We denote the corresponding next-token distribution by $p_W(\cdot \mid x_{\leq k})$. Then, autoregressive decoding is:

$$x_{k+1} = F_W(x_{\leq k}). \tag{1}$$

For efficiency, an L-layer Transformer can maintain KV cache. We denote the cache for prefix $x_{\leq k}$ as $\mathcal{C}_{\leq k}$, which contains key and value tensors. Given $\mathcal{C}_{<k}$, the cached forward operator F_W^{KV} processes only x_k, computing the logits for x_{k+1} with forward operator $F_W^{\mathrm{KV}}(x_k, \mathcal{C}_{<k})$.

Hybrid reasoning is a family of decoding schemes that pair a target LLM with a cheaper auxiliary model (i.e., a reasoner). The reasoner handles low-uncertainty decoding steps, while the target model is invoked when the reasoner is uncertain. Let W_T denote the target LLM and let W_S denote the reasoner. At step k, W_S produces the next-token probability distribution p_{k+1}^S given a prefix $x_{\leq k}$, and a router ρ computes a score with that to decide whether to accept it or invoke the target LLM:

$$score_k = \rho(x_{\leq k}, p_{k+1}^S), \qquad d_k = \begin{cases} S, & score_k \leq \tau_k, \\ T, & score_k > \tau_k, \end{cases} \tag{2}$$

where τ_k is the routing threshold. For example, ρ may compute the entropy of p_{k+1}^S and route to W_T when this entropy exceeds τ_k. The selected model provides the logits used for token selection:

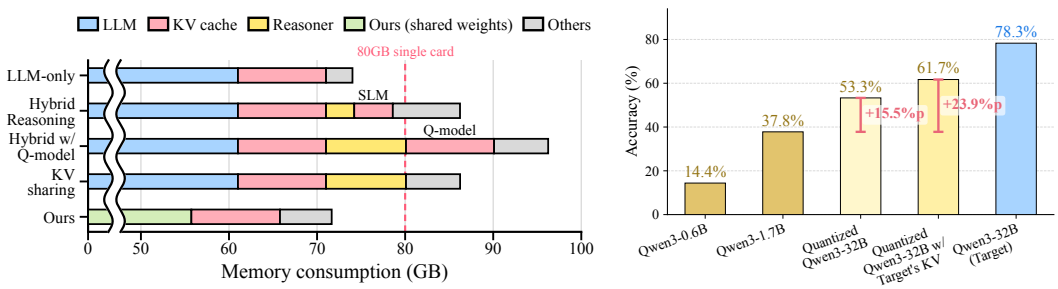
$$x_{k+1} = \begin{cases} x_{k+1}^S, & d_k = S, \\ F_{W_T}(x_{\leq k}), & d_k = T. \end{cases} \tag{3}$$

This token-level notation abstracts a family of hybrid reasoning methods. When model $m \in \{S, T\}$ is evaluated with cached execution, it uses its own KV cache ($F_{W_m}^{\mathrm{KV}}(x_k, \mathcal{C}_{<k}^m)$). The KV caches are model-specific because corresponding model weights produce the key and value tensors. Conventional hybrid reasoning can reduce target model inference, but it introduces additional reasoner weights, separate KV cache states, and switching overhead between models.

Weight quantization compresses an LLM by representing its original weights W with low-bit weights $\widehat{W}$. In group-wise affine quantization, a weight group W_g is represented by low-bit integers q_g and reconstructed as $\widehat{W}_g$ by the dequantization process:

$$q_g = \mathrm{clip}(\lfloor W_g / s_g \rceil + z_g, 0, 2^b - 1), \qquad \widehat{W}_g = s_g(q_g - z_g) \approx W_g, \tag{4}$$

where s_g is the scale, z_g is the zero-point, and b is the bit-width.



(a) Memory consumption at 32K sequence length.

(b) AIME accuracy per model setting.

Figure 1: Bottlenecks in hybrid reasoning. (a) Adding a reasoner and its KV cache can consume substantial memory at long sequence lengths. (b) Conventional reasoner models exhibit limited reasoning capacity, whereas using a quantized counterpart can mitigate such barriers. (Detailed analysis about quantized counterparts with target KV cache is in Section E.1).

3 Challenges & Opportunities

Challenges of hybrid reasoning. Hybrid reasoning reduces inference cost, but introduces critical challenges. First, adding a reasoner introduces additional memory overhead because of its weights and KV cache. As shown in Figure 1a, adding Qwen3-1.7B to Qwen3-32B ('Hybrid Reasoning') at a 32K sequence length increases peak memory from 74.02 GB to as much as 86.24 GB, a 16.5% overhead, which comes from both the model parameters and the separate KV caches. In addition, these SLM-based reasoners often lack enough capability to provide reliable intermediate reasoning. Figure 1b illustrates this barrier: small dense models such as Qwen3-0.6B/1.7B achieve only 14.4% and 37.8% on AIME, far below the target model Qwen3-32B at 78.3%, making them unreliable.

Opportunities from quantized reasoners. Instead of SLM-based reasoners, using a quantized counterpart of the target model ('quantized reasoner' hereafter) is an attractive choice in several aspects. As shown in Figure 1b, the quantized reasoner achieves 53.3% accuracy, far outperforming the SLM-based reasoners. Another advantage is that the quantized model can consume the target model's KV cache due to their identical architecture. Under an idealized assumption where the target model feeds the entire KV cache to the quantized reasoner, the accuracy improves up to 61.7%.

Sharing the KV cache also has benefits for memory consumption, as shown in Figure 1a. In a naive hybrid approach, the quantized reasoner generally consumes more memory than the SLM reasoner, both for model parameters and KV caches ('Hybrid w/ Q-model'). However, by sharing a unified KV cache pool across the models (Zhao et al., 2025a; Tiwari et al., 2025), approximately 10 GB of memory overhead from the reasoner's KV cache can be eliminated ('KV sharing').

Our work builds on this opportunity. Beyond simply using a quantized reasoner, we further propose fusing parameters of the reasoner and the target model to substantially reduce memory consumption ('ours' in Figure 1a). With a novel routing policy that exploits the characteristics of quantization, our approach achieves a favorable trade-off among throughput, memory consumption, and accuracy.

4 Methods

Overview. Figure 2 shows the workflow of ORQA. Rather than reserving more resources to use a separate reasoner model, ORQA realizes a quantized target model as the reasoner, and leverages this relationship to eliminate additional parameters and KV cache storage while enabling accurate target intervention. To this end, **weight infusion** (§ 4.1) supports the target model and reasoner from a unified parameter representation, and **Quant-ifier** (§ 4.2) guides target intervention by estimating whether quantization error alters the generated token decision. On top of this, to use an improved reasoner without additional memory cost (§ 4.3), ORQA adapts **KV transplant** that enables shared use of the target model's KV cache, along with **tailored calibration techniques**. Together, they enable fast and accurate hybrid reasoning with no additional model or KV cache memory compared with target-model-only reasoning.

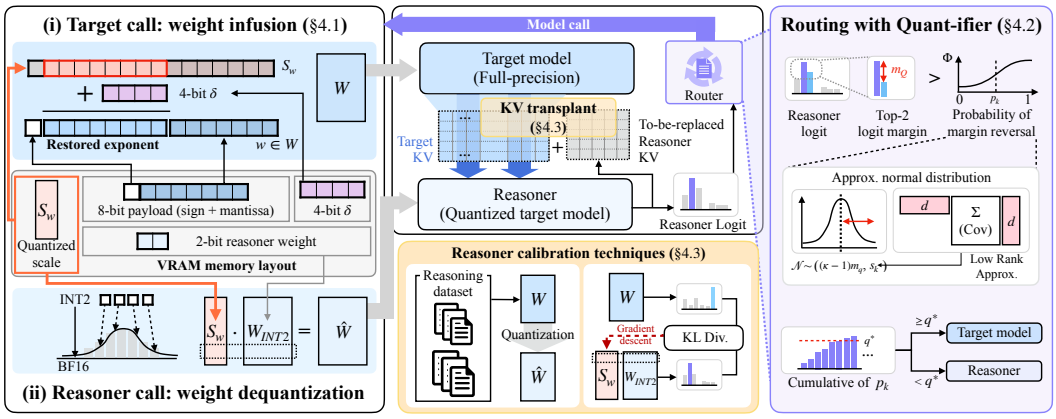


Figure 2: The overall workflow of ORQA.

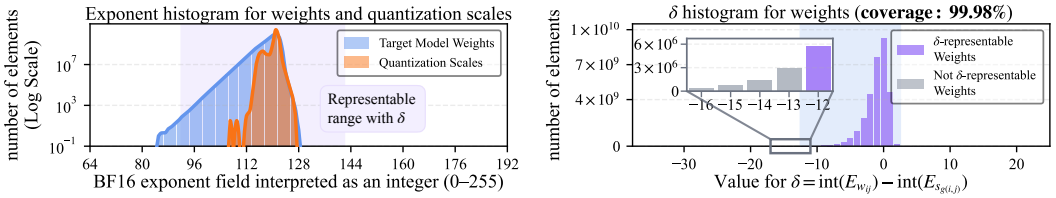


Figure 3: BF16 exponent and δ distributions for Qwen3-32B. The left-hand side compares the bfloat16 exponents of the model weights and their corresponding quantization scales. The right-hand side shows the distributions of element-wise exponent difference, $\delta = \mathrm{int}(E_{w_{ij}}) - \mathrm{int}(E_{s_{g(i,j)}})$.

4.1 Weight Infusion

Weight infusion completely eliminates additional memory requirements for a separate reasoner model's weight parameters. Instead of maintaining separate parameter sets, both models are *infused* into a shared parameter representation. This enables the bit-wise exact recovery of the target and reasoner with substantially lower memory usage. Weight infusion exploits the unique relationship between the models in ORQA: *the reasoner is a quantized counterpart of the target model.*

A bfloat16 value w is represented by a sign bit P_w, an 8-bit exponent E_w, and a 7-bit mantissa M_w:

$$\mathrm{bitfield}(w) = P_w \parallel E_w \parallel M_w. \tag{5}$$

It is known that 8-bit exponents of the bfloat16 LLM weights occupy a narrow range near zero (Fan et al., 2026; Zhang et al., 2025), so that much of the 8-bit exponent range remains unused. Quantization scales exhibit a similar pattern: In Figure 3, the exponent values between the weights and the corresponding quantization scales exhibit substantial overlap. Weight infusion leverages this pattern and performs a lossless recovery of bfloat16 target model weights using quantization scales.

In weight infusion, *the quantization scales are used in both model calls, but for different purposes*: During reasoner inference, they are used for dequantization of the weights. During target model inference, the scale serves as an anchor for each weight exponent, enabling exact recovery from a low-bit offset. This approach eliminates the need to store a full 8-bit exponent for every weight.

As shown on the upper left of Figure 2, each target weight is encoded using a 4-bit offset from its quantization scale s_g and an 8-bit payload of the original sign and mantissa, $S_w \| M_w$. For each weight w_{ij} belonging to group $g(i,j)$ with the quantization scale $s_{g(i,j)}$, we define δ_{ij} as the signed integer difference between the weight exponents and the scale exponent:

$$\delta_{ij} = \mathrm{int}(E_{w_{ij}}) - \mathrm{int}(E_{s_{g(i,j)}}), \tag{6}$$

where each exponent is interpreted as an unsigned 8-bit integer. We encode δ_{ij} using a 4-bit code c_{ij}. Given a delta bound δ^*, the nonzero codes $c_{ij} \in \{1, \ldots, 15\}$ represent 15 consecutive δ_{ij} values:

$$\delta(c_{ij}) = \delta^* + c_{ij} - 15, \qquad \delta^* = \arg\max_{b \in \mathbb{Z}} \sum_{i,j} \mathbf{1}[b - 14 \le \delta_{ij} \le b]\,. \tag{7}$$

A weight value is δ-*representable* when $\delta_{ij} \in [\delta^* - 14, \delta^*]$. We select δ^* as the upper bound of the 15-value δ interval that maximizes weight coverage.

At inference, the original exponent is recovered through a single integer addition, and the reconstructed weight is then formed by concatenating this exponent with the stored sign and mantissa:

$$\widehat{E}_{w_{ij}} = E_{s_{g(i,j)}} + \delta(c_{ij}), \qquad \mathrm{bits}(\widehat{w}_{ij}) = S_{w_{ij}} \| \widehat{E}_{w_{ij}} \| M_{w_{ij}}. \tag{8}$$

Because $\widehat{E}_{w_{ij}} = E_{w_{ij}}$ for every representable δ, the BF16 fields are recovered bit-for-bit. Therefore, the reconstructed target model is identical to the original version at no additional memory costs.

4.2 Routing Policy with Quant-ifier

We propose Quant-ifier, a routing signal designed for quantized reasoners that directly estimates token change from quantization error. Traditional entropy-based signals (Chen et al., 2025) measure the small model's confusion with output entropy. However, the reasoning divergence of quantized models comes from a different source than that of small reasoners. The quantized reasoner shares the same knowledge base as the target LLM, but suffers from arithmetic errors due to limited bitwidths.

We first express the token decision in terms of the logit margin. Let $\hat{a}$ and $\hat{b}$ denote the top-2 tokens predicted by the quantized reasoner Q. We define their logit margin and output-head difference as

$$m^Q = h_{\widehat{W}} \cdot w_{\hat{a}} - h_{\widehat{W}} \cdot w_{\hat{b}} = h_{\widehat{W}} \cdot d > 0, \qquad d = w_{\hat{a}} - w_{\hat{b}}, \tag{9}$$

where h is a token's hidden state at the last Transformer block of Q and w is an output head. For brevity, we omit the subscripts $(\hat{a}, \hat{b})$ from m^Q, m^R, and d throughout the following derivation.

We then inspect how quantization error affects this margin. Let $h_W = h_{\widehat{W}} + h_{\mathrm{err}}$, where h_W and $h_{\widehat{W}}$ are the hidden states of the target model and quantized reasoner, respectively, and h_{err} is a quantization error. The target LLM's margin for the same token pair is

$$m^R = h_W \cdot d = m^Q + h_{\mathrm{err}} \cdot d, \qquad m^R < 0 \iff h_{\mathrm{err}} \cdot d < -m^Q. \tag{10}$$

Thus, whether quantization changes the token depends not only on the magnitude of the quantization error, but specifically on its projection onto the direction d between the competing token heads.

At inference time, m^Q and d are directly available from the quantized reasoner, whereas obtaining h_{err} requires executing the target LLM. We therefore model its projection onto d as

$$h_{\mathrm{err}} \cdot d \sim \mathcal{N}\left((\kappa - 1)m^Q,\, d^\top \Sigma d\right), \tag{11}$$

where κ captures the systematic margin shift induced by quantization and Σ is the covariance of hidden-state quantization error h_{err}. We estimate κ and Σ offline using paired hidden states and margins from the reference and quantized reasoners on a calibration set.

Directly computing $d^\top \Sigma d$ requires storing and applying the full $D \times D$ covariance matrix. To reduce this overhead, we retain only the top-r eigendirections of Σ and approximate the remaining spectrum with a shared tail variance. Accordingly, the projected variance is approximated as

$$s_k^2 = \lambda_{\mathrm{tail}} \|d\|_2^2 + \sum_{j=1}^{r} (\lambda_j - \lambda_{\mathrm{tail}})(v_j^\top d)^2, \tag{12}$$

where (λ_j, v_j) denotes the j-th retained eigenpair. This reduces the per-step computation from $O(D^2)$ to $O(Dr)$. We provide the detailed procedure and construction of λ_{tail} in Section B. From Equations (10) and (11), the probability that the target LLM reverses the current token pair is

$$\widehat{p}_k = \Phi\left(-\frac{\widehat{\kappa} m^Q}{s_k}\right), \tag{13}$$

where Φ is the standard Gaussian cumulative distribution function. This quantity is large when the corrected margin is small and the quantization error strongly favors the competing token.

To account for multiple moderately uncertain decoding steps, we accumulate the crossing probability from the last reference-generated position τ. Under an independence approximation,

$$q_k = 1 - \prod_{i=\tau+1}^{k} (1 - \widehat{p}_i). \tag{14}$$

Quant-ifier routes to the target LLM when $q_k \geq q_\star$, where $q_\star$ is a hyperparameter. After a reference token is generated, the accumulated probability is reset.

Table 1: Experimental results on AIME, GPQA, and LiveCodeBench benchmarks

Target LLM	Reasoner	Routing	Method	AIME			GPQA			LiveCodeBench		
				Acc. ↑	TPS ↑	LLM Ratio ↓	Acc. ↑	TPS ↑	LLM Ratio ↓	Acc. ↑	TPS ↑	LLM Ratio ↓
Qwen3 32B	–	–	Target only	78.3%	18.8	100.0%	59.6%	19.2	100.0%	64.3%	19.5	100.0%
	Qwen3 1.7B	Query Level	BERT	43.3%	134.9	21.7%	41.4%	142.7	23.7%	44.8%	105.8	42.9%
			LLM	70.0%	47.6	81.7%	37.4%	164.5	10.1%	38.6%	125.4	29.5%
			MF	41.7%	144.9	15.0%	38.4%	136.5	27.8%	38.1%	135.6	22.4%
			SW	48.3%	128.7	25.0%	42.4%	135.4	26.3%	42.4%	114.7	35.2%
		Token Level	R-Stitch	66.7%	46.0	23.7%	54.6%	36.2	33.5%	56.2%	34.3	34.1%
			R2R	70.0%	53.6	20.9%	**60.1%**	34.7	34.2%	60.0%	41.0	27.4%
	2-Bit Q.	Token Level	ORQA (Ours)	**75.6%**	55.1	22.9%	59.4%	49.1	36.5%	**62.4%**	55.8	20.5%
Qwen3 8B	–	–	Target only	66.7%	38.5	100.0%	57.4%	39.8	100.0%	54.5%	38.6	100.0%
	Qwen3 0.6B	Query Level	BERT	26.7%	198.5	21.7%	36.2%	243.0	23.7%	28.6%	158.6	42.9%
			LLM	58.3%	75.0	81.7%	32.7%	276.9	9.1%	20.7%	185.7	29.0%
			MF	21.1%	210.7	15.0%	35.4%	233.2	27.8%	22.9%	197.5	21.9%
			SW	33.3%	179.2	26.7%	34.8%	196.2	27.8%	27.6%	159.4	36.2%
		Token Level	R-Stitch	46.7%	72.6	24.2%	39.9%	75.4	28.2%	37.1%	77.1	22.4%
			R2R	58.3%	69.0	26.0%	47.0%	68.6	31.1%	43.8%	70.7	25.3%
	2-Bit Q.	Token Level	ORQA (Ours)	**63.3%**	74.1	18.4%	**53.5%**	68.9	26.5%	**46.7%**	68.4	25.3%

4.3 Further Leveraging Quantized Reasoners

KV transplant eliminates the separate reasoner cache by sharing KV slots between both models, and extends target model intervention from mere token-level correction into KV cache repair. For each layer and position j, the reasoner reads the target model's KV and appends its own without a separate cache. A key invariant of KV transplant is *asymmetric* cache access: the reasoner attends to KV entries written by either model, but the target model never attends to reasoner-written entries. ORQA tracks the target-verified boundary u_k at step k. Then $\mathcal{C}_{\leq u_k}$ contains the target model's KV cache, while later slots contain the reasoner's KV cache. When the reasoner is invoked, the reasoner decodes from the current cache pool and appends its new KV entry to the same cache, forming a reasoner-written suffix at $\mathcal{C}_{>u_k}$. When the target model is invoked, it performs a partial prefill for the slots after u_k, until the latest reasoner-written suffix. The resulting target model KV entries overwrite the corresponding reasoner-written slots in place, and u_k is updated to the refreshed boundary.

Enhanced Quantization Recipes. We quantize the reasoner through *reasoning-oriented calibration* followed by *scale-only distillation*. The two stages adapt the quantized weights to reasoning text and align the reasoner's next-token distribution with the target model's, respectively. (i) Reasoning-oriented calibration data. We combine public reasoning traces from OpenThoughts-114k with general text from pile-10k, focusing calibration on intermediate reasoning while retaining general-text coverage. We apply SignRoundV2 (Cheng et al., 2025) for calibration stage. (ii) Scale-only distillation. We then optimize only the quantization scales $\mathbf{s}$, while keeping the other parameters fixed. We build the calibration data $\mathcal{D}$ by sampling the target model's own reasoning traces. Both models score identical prefixes under teacher forcing, yielding the target model's distribution p^R and the reasoner's distribution p^Q. We minimize the KL divergence between them by updating the scales.

5 Experiments

5.1 Experimental settings

We evaluate ORQA on three long-reasoning benchmark suites: AIME 2024 and 2025 (Ye et al., 2025), GPQA-Diamond (Rein et al., 2024), and LiveCodeBench v5 (Jain et al., 2025). We use Qwen3-32B and Qwen3-8B as the target LLM and report answer accuracy along with tokens per second (TPS) and the LLM ratio, defined as the fraction of generated tokens produced by the full-precision target LLM. We compare ORQA against target-only inference, query-level routing variants of BERT, LLM, MF, SW from RouteLLM (Ong et al., 2025), and token-level hybrid reasoning methods, including R-Stitch (Chen et al., 2025) and R2R (Fu et al., 2025). We describe the runtime decompression paths employed in our implementation of weight infusion in Section A. Additional details about experimental settings are provided in Section C.

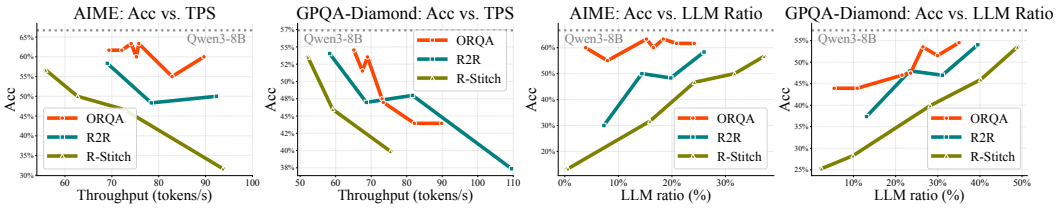


Figure 4: Accuracy–efficiency curves on AIME and GPQA for token-level baselines.

Table 2: Ablation study of main components in ORQA, evaluated on AIME and GPQA, with Qwen3-8B as the target model

Added Component	AIME		GPQA-Diamond		Memory Overhead
	Accuracy	LLM Ratio	Accuracy	LLM Ratio	
Baseline	13.3%	18.2%	36.9%	25.4%	1.000×
+ KV transplant	34.4%	13.0%	48.7%	16.9%	0.824×
+ Calibration recipes	58.9%	11.4%	50.3%	17.3%	0.824×
+ Quant-ifier	66.7%	8.2%	52.5%	13.8%	0.825×
+ Weight infusion (ORQA)	66.7%	8.2%	52.5%	13.8%	0.668×

5.2 Experimental results on long reasoning tasks

Table 1 shows the main results. Overall, target-only inference achieves the strongest accuracy but suffers from low throughput. Query-level routing greatly improves TPS at the cost of substantial accuracy degradation, showing that routing an entire query to a smaller reasoner is too coarse for long-reasoning tasks. Token-level hybrid methods partially close this gap by invoking the target LLM during generation. R2R and R-Stitch maintain relatively stable accuracy at low target-LLM ratios compared to query-level methods, but provide smaller quality gains than ORQA at comparable target-model usage, as the independent reasoner limits reasoning quality while adding its own decoding overhead. In contrast, ORQA preserves near-target-only quality while achieving up to 2.9× higher throughput. This trend holds across target model sizes, showing that ORQA improves throughput while maintaining stronger quality than baselines based on independent reasoners. We additionally evaluate a smaller reasoner setting with the Qwen3-32B in Section E.2 and the DeepSeek family (Guo et al., 2025) in Section E.3, where the same trends remain.

5.3 Accuracy–efficiency trade-off analysis

Figure 4 shows that ORQA forms a consistently stronger accuracy-efficiency trade-off on Qwen3-8B. ORQA preserves higher accuracy at comparable throughput, and also reaches a given accuracy with a lower LLM ratio, indicating more effective use of the target model. R2R approaches the frontier only in a narrow region of GPQA. The separation is especially clear on AIME, where ORQA maintains near 60% accuracy across a broad range while the baselines degrade sharply. Also, we provide experimental results focusing on low LLM ratio regions in Section E.4. Overall, ORQA translates target-model offloading into practical throughput gains with valid accuracy.

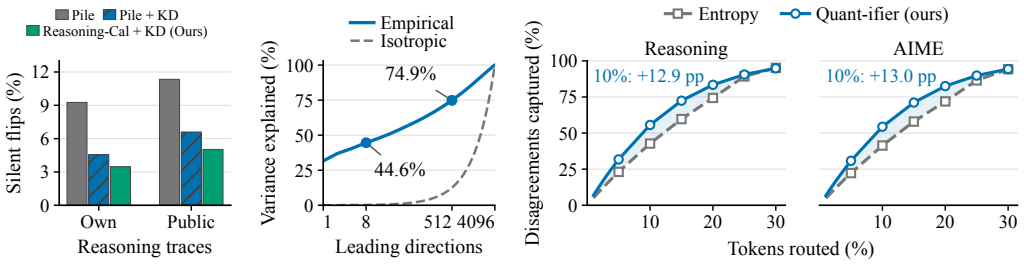
6 Analyses

6.1 Ablation study of ORQA

Table 2 evaluates the cumulative effects of the components comprising ORQA on AIME and GPQA. The baseline naively uses the quantized counterpart as a reasoner without memory optimization, along with an entropy-based router (Chen et al., 2025). Across datasets, KV transplant not only reduces memory cost from $1.00\times$ to $0.82\times$, but also improves reasoner quality by replacing its quantization-degraded KV cache from target model. The suggested two-stage calibration also enhances the capability of the reasoner, leading to higher accuracy. Adding Quant-ifier further reduces the LLM ratio from 11.4% to 8.2% on AIME and from 17.3% to 13.8% on GPQA-Diamond, while improving accuracy to 66.7% and 52.5%, respectively. This suggests that quantization error provides a more effective signal for avoiding unnecessary intervention. Lastly, weight infusion eliminates the

Table 3: Peak GPU memory (GB) usage of ORQA and other reasoning strategies

Target LLM	Reasoner	Method	Target LLM		Reasoner		Others	Total Peak Memory
			Weights	KV Cache	Weights	KV Cache		
Qwen3-32B	–	Target Only	61.02	10.00	0.00	0.00	3.00	**74.02**
	Qwen3-1.7B	R-Stitch	61.02	10.00	3.20	4.38	6.35	84.95
		R2R	61.02	10.00	3.20	4.38	7.63	86.24
	2-Bit Q.	Naive Hybrid	61.02	10.00	9.08	10.00	6.14	96.24
		ORQA	46.65	10.00	9.08	0.00	5.87	**71.60**
Qwen3-8B	–	Target Only	15.26	5.63	0.00	0.00	1.72	**22.60**
	Qwen3-0.6B	R-Stitch	15.26	5.63	1.11	4.38	4.66	31.03
		R2R	15.26	5.63	1.11	4.38	5.40	31.77
	2-Bit Q.	Naive Hybrid	15.26	5.63	4.34	5.63	3.44	34.28
		ORQA	9.91	5.63	4.34	0.00	1.62	**21.50**



(a) Silent flips. (b) Error variance of Σ. (c) Token disagreement capture ratio.

Figure 5: Quantization and routing analyses. (a) Silent-flip rates of reasoners across quantization recipes and reasoning sources. (b) Cumulative h_{err} variance of the quantized reasoner over leading covariance eigendirections. (c) Token disagreements captured when routing the top-scored tokens ranked by entropy or Quant-ifier on fixed reasoning traces. Higher is better.

memory cost of a separate reasoner by integrating the target model weights with quantization parameters. It achieves up to a 33.2% memory reduction over the baseline, improving the cost efficiency.

6.2 Memory Consumption Breakdown

Table 3 provides a breakdown analysis of peak GPU memory consumption.The results show that weight infusion reduces model-weight memory, while KV transplant eliminates the separate reasoner KV cache. For Qwen3-32B, weight infusion reduces the target-model weight footprint from 61.02GB to 46.65GB, which provides sufficient room for the 9.08GB quantized reasoner, and KV transplant removes the separate 10.00GB KV cache. Consequently, ORQA requires 71.60GB in total, which is even less than the 74.02 GB required for target-only inference. The same trend holds for Qwen3-8B. This breakdown shows that ORQA can accommodate a capable quantized reasoner within the original target-model memory budget by reducing both model-weight and KV-cache memory. Additionally, we further analyze weight infusion across δ bitwidths (Section F.2) and verify consistent memory-saving trends across model sizes and quantization methods (Section F.3).

6.3 Quantization & Routing Analyses

Next-token agreement on reasoning traces. We evaluate how the proposed quantization recipe affects next-token agreement between the reasoner and target LLM. 2-bit Qwen3-8B reasoners are compared using Pile calibration, Pile calibration with scale-only distillation (KD), and our full recipe with reasoning-oriented calibration and KD. A *silent flip* denotes a next-token disagreement between the quantized reasoner and target LLM at a low-entropy position that appears confident under an entropy-based criterion. As shown in Figure 5a, KD reduces silent flips, while adding reasoning-oriented calibration further reduces the silent flip, approximately 24% fewer silent flips than KD alone. These results show that our quantization recipe improves token agreement, but some disagreements still occur even when the quantized reasoner has low entropy. This motivates Quant-ifier to detect token changes from quantization error rather than entropy alone.

Quantization-error covariance concentration. The covariance spectrum reveals a strong directional structure in the hidden-state quantization error. For a 2-bit Qwen3-8B reasoner, Figure 5b shows that the leading 8 of 4,096 eigenvectors explain 44.6% of the hidden-state error variance, while the leading 512 explain 74.9%. This strong concentration indicates that token pairs with similar logit margins can have different crossing risks depending on how their direction d aligns with the dominant error directions. It therefore supports using the projected variance $d^\top \widehat{\Sigma} d$ rather than a direction-independent error scale in Quant-ifier. The concentrated spectrum also motivates the low-rank covariance approximation, where the dominant directions capture most of the error structure.

Identifying token disagreements. As shown in Figure 5c, we compare how effectively Quant-ifier and the quantized model's full-vocabulary entropy identify token disagreements between the quantized reasoner and target LLM. The Reasoning traces are from the OpenR1-Math-220k dataset, while the AIME traces are from the AIME-24/25 dataset. At each selection fraction, we rank tokens by each signal and measure the token disagreements captured by the selected subset. Quant-ifier captures more disagreements through 20% selection on both Reasoning and AIME. At 10% selection, it captures 55.6% and 54.4% of disagreements, compared with 42.7% and 41.4% for entropy, respectively. The curves become similar at larger selection fractions, indicating that the main advantage of Quant-ifier lies in prioritizing vulnerable tokens when target-model usage is limited. For further analysis regarding Quant-ifier's probability reliability, please refer to Section F.1.

7 Related work

Efficient reasoning. Long reasoning traces substantially increase autoregressive decoding cost. **Query-level and cascade routing** methods, such as FrugalGPT (Chen et al., 2024a), MoT Cascades (Yue et al., 2024), Synergy-of-Thoughts (Shang et al., 2024), and Think Only When You Need (Jiang et al., 2025), route easy queries or low-conflict thoughts to cheaper computation and invoke a stronger model only when needed. However, these coarse decisions cannot easily repair intermediate errors once a weaker path is selected. **Token-level hybrid reasoning** methods, including R-Stitch (Chen et al., 2025), R2R (Fu et al., 2025), Speculative Thinking (Yang et al., 2025b), SCoT (Wang et al., 2026), and SpecReason (Pan et al., 2025), allow a stronger model to intervene during chain-of-thought generation or verification. These methods still rely on independently loaded reasoners and KV caches. In contrast, ORQA realizes the reasoner as a quantized counterpart of the target model without additional memory overhead, improving the accuracy-TPS trade-off.

Memory efficient inference. A complementary line of work improves LLM inference by compressing model weights or KV caches. Weight quantization methods such as GPTQ (Frantar et al., 2023), SmoothQuant (Xiao et al., 2023), AWQ (Lin et al., 2024), and SignRound (Cheng et al., 2024; 2025) reduce memory consumption and bandwidth through post-training quantization, activation-aware scaling, or optimized rounding. KIVI (Liu et al., 2024) extends quantization to KV caches by exploiting the different statistics of keys and values. Recent speculative decoding methods also study quantization, parameter sharing, and cache reuse. QSpec (Zhao et al., 2025a), QuantSpec (Tiwari et al., 2025), SPEQ (Zhao et al., 2025b), and KVLink (Yang et al., 2025a) reduce storage or redundant cache computation for drafting and verification. However, their primary focus is speculative next-token decoding or generic cache reuse rather than long-form reasoning offloading. ORQA instead uses quantization specifically to construct a target-aligned reasoner, reducing both model state and KV cache overhead while preserving high decoding throughput for long reasoning.

8 Conclusion

We introduced ORQA, a hybrid reasoning framework that uses the target LLM's quantized counterpart as a target-aligned reasoner. Using this shared structure, weight infusion and KV transplant eliminate separate model and KV-cache storage, while Quant-ifier guides target intervention and tailored quantization improves reasoner quality. Across long-reasoning benchmarks, ORQA preserves near-target-only quality with substantially higher throughput and no additional GPU memory, with consistent gains across model settings and low-LLM-budget regimes. These results establish the target LLM's quantized counterpart as an effective reasoner for hybrid reasoning, providing a new alternative to independently trained or loaded small reasoners.

AI USE STATEMENT

In this work, we used generative AI tools to [generate synthetic data sets, implement methods]. We have not used generative AI tools to [formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, interpret results], and to [help develop theoretical models or conceptual frameworks, assist in the writing of proofs] are not applicable to this work. Additionally, we used generative AI tools to [create or modify scientific figures or images, suggest experimental parameters, create or edit software code, creation of artifacts, draft parts of a research paper, brainstorming, sourcing/searching for information, edit a research paper to improve readability]. We have reviewed all AI-assisted work. We checked implemented code and experimental results from AI-written code and manually verified AI-sourced information. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed explanations of each component of ORQA, including the mathematical formulations of weight infusion and Quant-ifier, and the cache-update procedure of KV transplant (Sections 4.1 to 4.3). The appendix further describes the calibration procedure for Quant-ifier, including covariance estimation and its low-rank approximation (Section B). We provide complete experimental settings, including model configurations, hardware, decoding parameters, quantization settings, and calibration and distillation parameters (Section C). Implementation details and configurations for all compared baselines are also specified. Finally, we describe the datasets and evaluation protocols for AIME, GPQA-Diamond, and LiveCodeBench, including the answer extraction and scoring procedures consistently applied to all methods (Section D).

REFERENCES

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024b.

Zhuokun Chen, Zeren Chen, Jiahao He, Lu Sheng, Mingkui Tan, Jianfei Cai, and Bohan Zhuang. R-stitch: Dynamic trajectory stitching for efficient reasoning. *arXiv preprint arXiv:2507.17307*, 2025.

Wenhua Cheng, Weiwei Zhang, Haihao Shen, Yiyang Cai, Xin He, Kaokao Lv, and Yi Liu. Optimize weight rounding via signed gradient descent for the quantization of llms. In *Findings of the Association for Computational Linguistics*, 2024.

Wenhua Cheng, Weiwei Zhang, Heng Guo, and Haihao Shen. Signroundv2: Closing the performance gap in extremely low-bit post-training quantization for llms. *arXiv preprint arXiv:2512.04746*, 2025.

Ruibo Fan, Xiangrui Yu, Xinglin Pan, Zeyu Li, Weile Luo, Qiang Wang, Wei Wang, and Xiaowen Chu. Zipserv: Fast and memory-efficient llm inference with hardware-aware lossless compression. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2026.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Optq: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

Tianyu Fu, Yi Ge, Yichen You, Enshu Liu, Zhihang Yuan, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. R2r: Efficiently navigating divergent reasoning paths with small-large model token routing. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

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

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. Think only when you need with large hybrid-reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, 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. In *Proceedings of machine learning and systems*, 2024.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *International Conference on Machine Learning*. PMLR, 2024.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreason: Fast and accurate inference-time compute via speculative reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 2023.

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

Yu Shang, Yu Li, Fengli Xu, and Yong Li. Synergy-of-thoughts: Eliciting efficient reasoning in hybrid language models. *arXiv preprint arXiv:2402.02563*, 2024.

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

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.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, et al. Stop overthinking: A survey on efficient reasoning for large language models. *Transactions on Machine Learning Research*, 2025.

Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Richard Charles Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Quantspec: Self-speculative decoding with hierarchical quantized kv cache. In *Forty-second International Conference on Machine Learning*, 2025.

Jikai Wang, Juntao Li, Jianye Hou, Bowen Yan, Lijun Wu, and Min Zhang. Efficient reasoning for llms through speculative chain-of-thought. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*. PMLR, 2023.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. Kvlink: Accelerating large language models via efficient kv cache reuse. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a.

Van Yang, Xiang Yue, Vipin Chaudhary, and Xiaotian Han. Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time. In *Second Conference on Language Modeling*, 2025b.

Yixin Ye, Yang Xiao, Tiantian Mi, and Pengfei Liu. Aime-preview: A rigorous and immediate evaluation framework for advanced mathematical reasoning. `https://github.com/GAIR-NLP/AIME-Preview`, 2025. GitHub repository.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thought representations for cost-efficient reasoning. In *The Twelfth International Conference on Learning Representations*, 2024.

Tianyi Zhang, Mohsen Hariri, Shaochen Henry Zhong, Vipin Chaudhary, Yang Sui, Xia Hu, and Anshumali Shrivastava. 70% size, 100% accuracy: Lossless llm compression for efficient gpu inference via dynamic-length float (dfloat11). In *Advances in Neural Information Processing Systems*, 2025.

Juntao Zhao, Wenhao Lu, Sheng Wang, Lingpeng Kong, and Chuan Wu. Qspec: Speculative decoding with complementary quantization schemes. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025a.

Yushu Zhao, Yubin Qin, Yang Wang, Xiaolong Yang, Huiming Han, Shaojun Wei, Yang Hu, and Shouyi Yin. From quarter to all: Accelerating speculative llm decoding via floating-point exponent remapping and parameter sharing. *arXiv preprint arXiv:2510.18525*, 2025b.

A DECOMPRESSION DETAILS IN WEIGHT INFUSION

As we use weight infusion to represent the target model weight exponent fields with a low-bit code, we have to come up with a fast and efficient decompression strategy to load the target model layers at inference. By weight infusion, the inference engine stores each target-model weight in the form of a 4-bit code and an 8-bit field containing the sign and mantissa. Weight parameters that are not δ-representable are retained exactly in a sparse BF16 exception stream.

At runtime, the engine selects one of two reconstruction paths according to the value of M, where M is the number of token activation rows processed together by the current linear operation. Thus, $M = 1$ corresponds to ordinary single-token decoding, while $M > 1$ generally represents prefill or partial-prefill from the target intervention.

For $M \leq 8$, the engine uses a fused small-M product without materializing the target weight matrix. For each regular weight, the kernel reconstructs the BF16 value (Equation (8)), in registers, and immediately multiplies it by the activation. Outlier locations contribute zero to this regular product and are computed in a separate path. The engine computes the products for sparse outliers in parallel, and combines them with the regular product before the final BF16 output conversion. The structure can be expressed as

$$XW^{\mathsf{T}} = XD^{\mathsf{T}} + XS^{\mathsf{T}}, \tag{15}$$

where D is the matrix with only δ-representable weight values, and S is the matrix that only contains the exact outliers as a sparse matrix storage format. This path was inspired by Kim et al. (2023).

For $M > 8$, the engine first reconstructs the complete BF16 weights, then begins the computation. A Triton kernel decodes all regular weights into a shared restoration workspace, initially writing zero at outlier locations. A second kernel patches those locations with their exact stored BF16 values. The resulting dense matrix is then consumed by the standard BF16 linear operation, The workspace is reused sequentially across weights, so the engine does not retain a dense copy of target-model weight.

B CALIBRATION DETAILS OF QUANT-IFIER

For each calibration position i, we evaluate the reference and quantized reasoners on the same prefix and collect the post-normalization hidden-state error

$$h_{\mathrm{err},i} = h_{W,i} - h_{\widehat{W},i}. \tag{16}$$

We estimate the empirical covariance of these errors as

$$\widehat{\Sigma} = \frac{1}{N-1} \sum_{i=1}^{N} (h_{\mathrm{err},i} - \bar{h}_{\mathrm{err}})(h_{\mathrm{err},i} - \bar{h}_{\mathrm{err}})^{\top}. \tag{17}$$

At each position, m_i^Q and m_i^R are evaluated using the same top-2 token pair selected by the quantized reasoner. We fit the systematic margin scaling κ by least squares:

$$\widehat{\kappa} = \frac{\sum_i m_i^Q m_i^R}{\sum_i (m_i^Q)^2}. \tag{18}$$

Hence, the projected error $m_i^R - m_i^Q$ has conditional mean approximately $(\widehat{\kappa} - 1)m_i^Q$.

For efficient inference, let $\{(\lambda_j, v_j)\}_{j=1}^{r}$ be the top-r eigenpairs of $\widehat{\Sigma}$. We approximate all remaining eigenvalues by

$$\lambda_{\mathrm{tail}} = \frac{\mathrm{tr}(\widehat{\Sigma}) - \sum_{j=1}^{r} \lambda_j}{D - r}. \tag{19}$$

This yields the projected-variance approximation in Equation (12).

C Detailed experimental settings

We evaluate ORQA on two long-reasoning benchmark suites: AIME (Ye et al., 2025), GPQA (Rein et al., 2024), and LiveCodeBench (Jain et al., 2025). For AIME, we use the union of AIME 2024 and AIME 2025. For GPQA, we use the 198-question GPQA-Diamond split. For LiveCodeBench, we use the 210-question v5 benchmark (v5, 2024.10-2025.02). Please refer to Section D for the datasets.

We report answer accuracy as the primary quality metric. To characterize inference efficiency, we additionally report tokens per second (TPS) and the LLM ratio, defined as the fraction of generated tokens produced by the full-precision target LLM.

In the main experiments for ORQA, we use Qwen3-8B and Qwen3-32B as the target LLMs. Experiments using Qwen3-32B were conducted on two NVIDIA RTX PRO 6000 GPUs, whereas those using Qwen3-8B were conducted on two NVIDIA RTX A6000 GPUs. For both the target LLM and the reasoner, we use tensor parallelism with TP=2. The maximum context length is set to 32K tokens, and all methods use the decoding configuration with temperature $T = 0.6$, and top-p value $p = 0.95$. Unless otherwise stated, all methods use the same prompt format, decoding rule, and maximum generation length.

We construct the 2-bit reasoner using a fixed two-stage quantization recipe without model-specific hyperparameter tuning. In the first stage, we apply SignRound V2 using symmetric W2A16 quantization with group size 32. All linear layers in each Transformer block are quantized, while the embeddings, output head, and normalization parameters remain in BF16. We use 512 calibration sequences of 2,048 tokens, consisting of 384 OpenThoughts-114k sequences and 128 pile-10k sequences. We do block-wise calibration of 1,000 iterations with batch size 8, using SignSGD and a learning rate of 2×10^{-3} for both the rounding and min-max parameters with linear decay.

In the second stage, we optimize only the group-wise quantization scales by distilling the original BF16 target model into the quantized reasoner on self-generated reasoning traces. Both models process identical prefixes, and we minimize the forward KL divergence over assistant-generated positions with temperature 1 and no auxiliary loss. We use AdamW with learning rate 2×10^{-5}, $\beta = (0.9, 0.999)$, zero weight decay, and 128 optimizer steps without warm-up or gradient clipping. Each step uses 64 sequences of 4,096 tokens, implemented with micro-batch size 1 and 64-step gradient accumulation. We use two NVIDIA A6000 GPUs for 8B-class models and two RTX PRO 6000 GPUs for Qwen3-32B.

For the baselines, we compare ORQA against target-only inference, query-level routing methods from RouteLLM (Ong et al., 2025), and token-level hybrid reasoning methods. The token-level hybrid reasoning baselines include R-Stitch (Chen et al., 2025) and R2R (Fu et al., 2025). For all baselines, we use the available official implementations. Following their intended usage, the baseline hybrid methods use independently loaded smaller reasoners, such as Qwen3-0.6B and Qwen3-1.7B.

For the MF (matrix factorization) and SW-Ranking (similarity-weighted ranking) variants in the router setting of RouteLLM, we adapted the two methods to use an open-source, frozen Qwen3-Embedding-0.6B encoder instead of the API-backed embedding path. We regenerated the reference prompt embeddings from the Arena Human Preference dataset (originally used in RouteLLM), and constructed a new MF model and SW-ranking model in that space.

D Dataset details

This section provides additional details on the datasets and evaluation protocol used in our experiments. We evaluate ORQA on three long-reasoning benchmark suites, AIME, GPQA-Diamond, and LiveCodeBench, following the experimental setting described in the main paper. All datasets are used only for evaluation. Therefore, we do not fine-tune, train, or calibrate model weights on these benchmark answers.

AIME. For mathematical reasoning, we use the union of AIME 2024 and AIME 2025. This results in 60 competition-style mathematical reasoning problems in total. AIME problems are well suited for evaluating long form reasoning because they typically require multi-step symbolic manipulation, arithmetic computation, and careful verification before producing the final answer. Each problem

has a single ground truth integer answer. We keep the original problem statements and evaluate a model prediction by exact matching against the reference integer after answer normalization.

For answer extraction, we first search for standard final-answer patterns such as boxed answers, explicit "final answer" statements, or the last clearly indicated integer answer in the generated response. Intermediate calculations are ignored whenever a final answer marker is present. If multiple valid candidates appear, we use the last valid final answer candidate, since reasoning models often revise intermediate computations before committing to the final response. If no valid integer answer can be extracted, the prediction is marked as incorrect. The same scoring procedure is applied to all methods, including target-only inference, prior hybrid reasoning baselines, and ORQA.

GPQA-Diamond. For scientific reasoning, we use the GPQA-Diamond split, which contains 198 multiple-choice questions. GPQA-Diamond is designed to evaluate challenging graduate-level scientific reasoning across domains such as biology, chemistry, and physics. Compared with AIME, which emphasizes mathematical problem solving with a compact integer answer, GPQA-Diamond evaluates whether the model can reason over specialized scientific concepts and select the correct answer among multiple options.

Each GPQA-Diamond example is formatted as a question with a fixed set of answer choices. All methods are evaluated with the same prompt format and the same option order for a given example. The model is asked to produce its reasoning and then provide a final answer corresponding to one of the answer choices. For scoring, we first extract the final option label from standard final answer patterns. If an option label is not explicitly provided, we attempt to match the normalized final answer text to one of the provided answer choices. A prediction is counted as correct only when it can be uniquely mapped to the ground-truth option. Ambiguous outputs or outputs without an extractable answer are counted as incorrect.

LiveCodeBench. For code reasoning, we use LiveCodeBench to evaluate the model's ability to solve programming problems that require both reasoning and code generation. Unlike AIME and GPQA-Diamond, where correctness is determined from a short final answer, LiveCodeBench requires the model to generate an executable program that satisfies the problem specification. This setting is particularly useful for evaluating long reasoning because a model must understand the problem, derive an appropriate algorithm, and produce a correct implementation.

For each problem, the model generates its reasoning followed by the final code solution using the same prompt format across all methods. We evaluate the generated program using the benchmark test cases and count a prediction as correct only when it passes the required tests. Malformed or non-executable programs, runtime failures, and solutions that fail any required test are counted as incorrect. The same generation, code extraction, and execution procedure is applied to target-only inference, prior hybrid reasoning baselines, and ORQA.

E Additional Experiments

E.1 KV cache quality in quantized reasoning

Table 4 isolates the effect of KV cache quality by letting the 2-bit reasoner generate all tokens while using the target model's KV cache. Using its own KV cache, the 2-bit Qwen3-32B and Qwen3-8B achieve 53.3% and 31.7% on AIME, while using the BF16 KV cache improves them to 61.7% and 51.7%, respectively. This recovery indicates that KV cache quality contributes substantially to the performance gap of quantized reasoning. These results support that KV transplant not only saves memory, but also restores the reasoning quality of the quantized reasoner through higher-quality cached states.

Table 4: Effect of KV sharing between a model and its quantized counterpart, evaluated on AIME

Model Precision	KV-Generating Model Precision	Model	
		Qwen3-32B	Qwen3-8B
W16A16	W16A16	78.3%	66.7%
W2A16	W2A16	53.3%	31.7%
W2A16	W16A16	61.7% (+8.3)	51.7% (+20.0)

Table 5: Experimental results on AIME, GPQA, and LiveCodeBench benchmarks. Reasoner denotes the model that generates offloaded reasoning tokens. BERT, LLM, MF, and SW denote RouteLLM variants. LLM Ratio denotes the fraction of tokens generated by the target LLM.

Target LLM	Reasoner	Routing	Method	AIME			GPQA			LiveCodeBench		
				Acc. ↑	TPS ↑	LLM Ratio ↓	Acc. ↑	TPS ↑	LLM Ratio ↓	Acc. ↑	TPS ↑	LLM Ratio ↓
Qwen3 32B	–	–	Target only	78.3%	18.8	100.0%	59.6%	19.2	100.0%	64.3%	19.5	100.0%
	Qwen3 0.6B	Query Level	BERT	25.0%	176.2	21.7%	33.3%	185.7	23.7%	32.4%	136.0	42.9%
			LLM	68.3%	56.3	81.7%	31.3%	215.1	10.1%	24.3%	163.1	29.5%
			MF	23.3%	190.8	15.0%	32.8%	178.6	27.8%	25.2%	176.4	22.4%
			SW	33.3%	168.1	25.0%	35.4%	169.8	26.3%	30.5%	147.8	35.2%
		Token Level	R-Stitch	63.3%	38.4	31.6%	**62.1%**	28.4	49.3%	35.2%	38.4	24.9%
			R2R	65.0%	48.3	22.5%	51.5%	45.2	37.9%	59.0%	29.1	45.0%
	2-Bit Q.	Token Level	ORQA (Ours)	**75.6%**	55.1	22.9%	59.4%	49.1	36.5%	**62.4%**	55.8	20.5%

Table 6: Experimental results on AIME and GPQA, evaluated by using models from the *DeepSeek-R1-Distill-Qwen* model family

Target LLM	Reasoner	Method	AIME			GPQA		
			Acc. ↑	TPS ↑	LLM Ratio ↓	Acc. ↑	TPS ↑	LLM Ratio ↓
DeepSeek-R1 -Distill-Qwen3 32B	–	Target Only	68.33%	37.7	100.0%	60.86%	39.6	100.0%
	DeepSeek-R1-Distill-Qwen 1.5B	R-Stitch	45.00%	48.4	34.88%	54.04%	47.4	44.13%
		R2R	56.67%	62.5	22.22%	52.02%	61.2	26.76%
	2-Bit Q.	ORQA (Ours)	**61.67%**	73.7	16.22%	**56.06%**	74.9	20.41%

E.2 Evaluation with a smaller reasoner.

Table 5 further evaluates the baselines using Qwen3-0.6B as a smaller independent reasoner. Query-level routing achieves substantially higher throughput, but its accuracy degrades further due to the limited capability of the 0.6B reasoner. R2R and R-Stitch partially recover accuracy through token-level routing, but both achieve lower throughput than ORQA across all three benchmarks. ORQA achieves higher accuracy on AIME and LiveCodeBench, while R-Stitch attains the highest GPQA accuracy at substantially lower throughput (28.4 TPS versus 49.1 TPS for ORQA). Overall, ORQA achieves accuracy close to target-only inference while reaching 49–56 TPS, offering a more favorable balance between reasoning quality and throughput than the smaller independent reasoner.

E.3 Evaluation on other model families

Table 6 evaluates token-level hybrid reasoning methods using models from the *DeepSeek-R1-Distill-Qwen* model family. For the baseline methods, we use DeepSeek-R1-0528-Qwen3-8B as the target model, and DeepSeek-R1-Distill-Qwen-1.5B as the lightweight reasoner. For ORQA, we use the same model as the target model, and use the quantized counterpart as the reasoner. Across both benchmarks, ORQA achieves the highest accuracy and throughput among the hybrid methods while using the lowest target-LLM ratio. This result demonstrates that the benefits of quantized hybrid reasoning, provided by ORQA, generalize to other model families.

E.4 Evaluation under a low LLM budget

Table 7 evaluates token-level hybrid reasoning in the low-LLM-budget regime, where only a small fraction of tokens are generated by the target LLM. On AIME, ORQA achieves 60.0% accuracy with only 3.89% LLM ratio, compared with 30.0% for R2R and 23.3% for R-Stitch at higher LLM ratios of 7.30% and 4.46%, respectively. A similar trend appears on GPQA, where ORQA reaches 43.9% accuracy at a 5.43% LLM ratio, outperforming R2R and R-Stitch by 11.1 and 15.7 percentage points while using no more target-model tokens. Although the baselines achieve higher raw throughput with their smaller 0.6B reasoner, their reasoning quality degrades substantially under such restricted

Table 7: Low-LLM budget experimental results on AIME and GPQA

Target LLM	Reasoner	Method	AIME			GPQA		
			Acc. ↑	TPS ↑	LLM Ratio ↓	Acc. ↑	TPS ↑	LLM Ratio ↓
Qwen3 8B	–	Target Only	66.67%	38.5	100.0%	57.41%	39.8	100.0%
	Qwen3 0.6B	R-Stitch	23.33%	146.1	4.46%	28.28%	134.9	9.91%
		R2R	30.00%	116.2	7.30%	32.83%	142.8	6.60%
	2-Bit Q.	ORQA (Ours)	**60.00%**	89.7	3.89%	**43.94%**	89.9	5.43%

target-model usage. In contrast, ORQA retains much stronger accuracy while operating with less than 6% target-model usage, demonstrating more effective use of a limited LLM budget.

F Additional Analyses

F.1 Per-token probability reliability

We further assess whether the projected-error score provides an interpretable probability scale across reasoning populations. Using the same calibration fit, with $\widehat{\kappa} = 1.690$, we compare $\Phi(-\widehat{\kappa}m/s)$ with the observed top-2 pair-crossing event, where the target hidden state's projection through the shared output head ranks the quantized model's runner-up at least as high as its top-1 token. The analysis retains valid positive-margin predictions and excludes forced-reference cases caused by tied quantized logits.

The reliability curves in Figure 6 use ten equal-count probability bins over 1.65M positions from 324 held-out reasoning traces (*Reasoning*) and 0.91M positions from 120 AIME-derived traces (*AIME*). The predicted probabilities generally follow the observed crossing frequencies, with token-weighted expected calibration errors of 0.83 and 1.01 percentage points, respectively. The similar agreement on AIME without refitting suggests that the estimated probability scale transfers across the two populations. We nevertheless observe a systematic local bias around 24–25% predicted probability, where the observed crossing rate is approximately 20% for both populations. Thus, the score should be interpreted as an approximate crossing probability, particularly at intermediate risk levels.

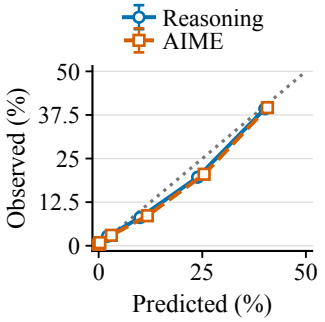


Figure 6: Per-token reliability of Quant-ifier.

The upper limit of 50% follows directly from the model assumption. With $\widehat{\kappa} > 0$, $m > 0$, and $s > 0$, we have $\Phi(-\widehat{\kappa}m/s) < 1/2$. Hence, as the corrected margin becomes small relative to the projected error scale, the predicted crossing probability approaches an even chance but cannot exceed it. Overall, these results support the probability interpretation of the Quant-ifier routing signal on valid fixed-prefix positions, while not implying calibration of accumulated window risk or final-answer error.

F.2 Bitwidth analysis of δ in weight infusion

In Table 8, we show the target model memory reduction when the bitwidth of the δ is adjusted from 2-bit to 6-bit. For each bitwidth, we select the delta interval that maximizes weight coverage. Several configurations (3, 4, 5, and 6-bit) reduce the estimated total weight storage cost relative to the BF16 original target model weights. Smaller bitwidths require fewer bits per element but produce more outliers, whereas larger bitwidths reduce outliers at the expense of increased dense storage.

Across all three evaluated models, 4-bit delta encoding achieves the lowest total storage, covering approximately 99.98% of analyzed weights and reducing total weight storage by 18.5–20.8%, in-

Table 8: Target model weight coverage and storage cost under different delta bit widths

Target Model	Delta Bits	Coverage (%)	Sparse Outliers (%)	Compressed Target (GiB)	Original Target (GiB)	Memory Reduction
Qwen3-32B	2	68.86	31.14	95.34	61.02	1.56×
	3	97.27	2.73	49.44		0.81×
	4	99.98	0.02	48.34		0.79×
	5	99.99999908	9.17×10^{-7}	51.94		0.85×
	6	100.00	0.00	55.57		0.91×
Qwen3-8B	2	67.66	32.34	23.36	15.26	1.53×
	3	97.02	2.98	12.77		0.84×
	4	99.98	0.02	12.43		0.82×
	5	99.99998543	1.46×10^{-5}	13.23		0.87×
	6	100.00	0.00	14.04		0.92×
DeepSeek-R1-0528-Qwen3-8B	2	67.79	32.21	23.31	15.26	1.53×
	3	96.99	3.01	12.79		0.84×
	4	99.98	0.02	12.43		0.81×
	5	99.99998971	1.03×10^{-5}	13.23		0.87×
	6	100.00	0.00	14.04		0.92×

cluding sparse-outlier overhead. We therefore adopt 4 bits as the default delta configuration while retaining bitwidth as a user-configurable parameter.

F.3 Consistency of the observations underlying weight infusion

In this section, we provide the observations that motivate the design of weight infusion across additional models and quantization methods, and demonstrate their consistency across model families, model sizes, and quantization methods. We visualize Figure 7 to Figure 11. For the figures, the left-hand side compares the bfloat16 exponents of the model weights and their corresponding quantization scales. The right-hand side shows the distributions of element-wise exponent differences, $\delta = \mathrm{int}(E_{w_{ij}}) - \mathrm{int}(E_{s_{g(i,j)}})$.

First, we visualize the exponent distributions and δ histograms of the following models: Qwen3-8B (Figure 7) and DeepSeek-R1-0528-Qwen3-8B (Figure 8), and the quantization scale exponent distribution obtained from the proposed quantization recipe in our work. We examine whether their bfloat16 weight exponents exhibit a skewed distribution and whether the corresponding quantization scales follow the same pattern. The substantial overlap between the distributions indicates that this pattern is consistent across model families. Moreover, δ covers 99.98% of all weights, enabling weight infusion to reconstruct nearly the entire parameter set directly.

Second, we visualize the same information of Qwen3-32B model across widely used quantization methods (Frantar et al., 2023; Lin et al., 2024; Cheng et al., 2025) in Figure 9, Figure 10, and Figure 11. Although the weight coverage that δ offers varies by method, we can observe that more than 99.8% of the weight parameters are representable with lightweight δ, allowing weight infusion to consistently reduce memory usage.

Therefore, we show that weight infusion is not tied to model-specific properties or a particular quantization method, supporting its generalizability across models and quantization settings.

G Limitations and future work

In this section, we discuss several limitations of our current work. ORQA is designed for hybrid reasoning scenarios where the target LLM and its quantized counterpart share the same architecture, tokenizer, positional encoding, and KV cache layout. This assumption is natural for our setting, since the reasoner is constructed from the target model itself. However, our empirical validation has so far focused on decoder-only language models, primarily the Qwen3 family models. Diverse experiments for ORQA on a broader set of model families would further clarify the generality of target-aligned quantized reasoning.

Our experiments mainly consider long reasoning benchmarks such as AIME, GPQA, and LiveCodeBench under a controlled GPU setup. These benchmarks are well suited for evaluating the accuracy-speedup tradeoff in reasoning-intensive generation, but they do not cover all possible de-

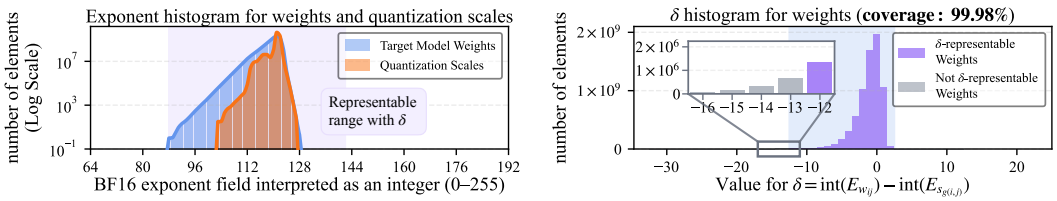


Figure 7: BF16 exponent and δ distributions for Qwen3-8B. Quantization scales were obtained by the *proposed recipes in ORQA*.

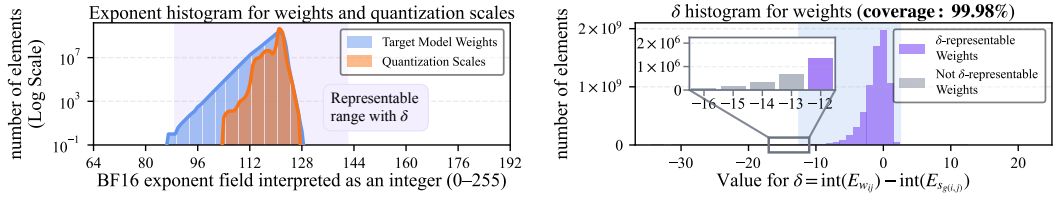


Figure 8: BF16 exponent and δ distributions for DeepSeek-R1-0528-Qwen3-8B. Quantization scales were obtained by the *proposed recipes in ORQA*.

ployment scenarios. For instance, different workloads, context-length distributions, batching strategies, or serving backends may lead to different operating points. Nevertheless, the main memory advantage of ORQA, namely avoiding both an independently stored reasoner and a separate reasoner KV cache, is independent of these deployment details.

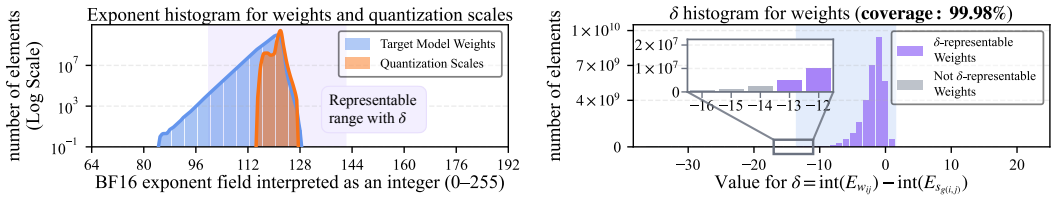


Figure 9: BF16 exponent and δ distributions for Qwen3-32B. Quantization scales were obtained by the method *GPTQ* (Frantar et al., 2023).

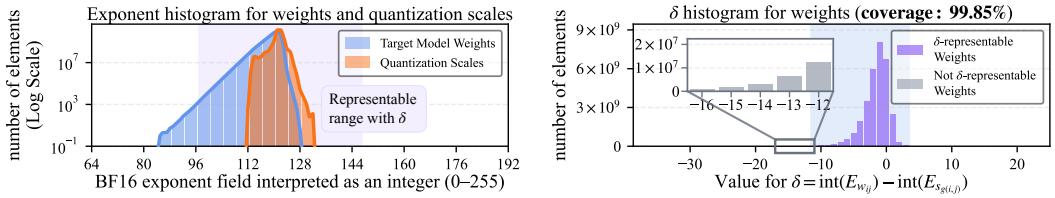


Figure 10: BF16 exponent and δ distributions for Qwen3-32B. Quantization scales were obtained by the method *AWQ* (Lin et al., 2024).

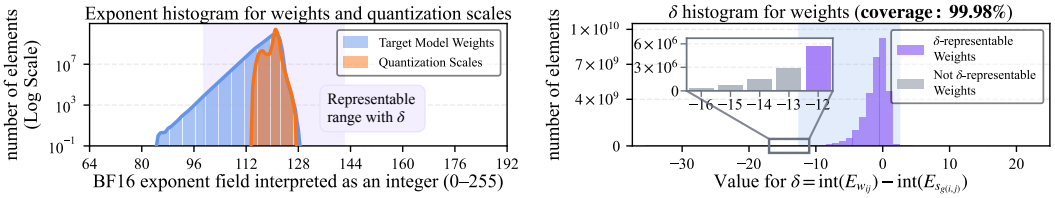


Figure 11: BF16 exponent and δ distributions for Qwen3-32B. Quantization scales were obtained by the method *SignRoundV2* (Cheng et al., 2025).