PAROQUANT: PAIRWISE ROTATION QUANTIZATION FOR EFFICIENT REASONING LLM INFERENCE

Yesheng Liang[3,†] **Haisheng Chen**[3,‡] **Song Han**[1,2] **Zhijian Liu**[1,3]
[1]NVIDIA [2]MIT [3]UC San Diego
[†]Algorithm lead [‡]System lead

ABSTRACT

Weight-only post-training quantization (PTQ) compresses the weights of Large Language Models (LLMs) into low-precision representations to reduce memory footprint and accelerate inference. However, the presence of outliers in weights and activations often lead to large quantization errors and severe accuracy degradation, especially in recent reasoning LLMs where errors accumulate across long chains of thought. Existing PTQ methods either fail to sufficiently suppress outliers or introduce significant overhead during inference. In this paper, we propose **Pairwise Rotation Quantization** (ParoQuant), a weight-only PTQ method that combines hardware-efficient and optimizable *independent Givens rotations* with channel-wise scaling to even out the magnitude across channels and narrow the dynamic range within each quantization group. We further co-design the inference kernel to fully exploit GPU parallelism and keep the rotations and scaling lightweight at runtime. ParoQuant achieves an average **2.4%** accuracy improvement over AWQ on reasoning tasks with less than 10% overhead. This paves the way for more efficient and accurate deployment of reasoning LLMs.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of tasks. However, their massive size and large memory footprint not only incur substantial inference costs but also hinder on-device deployment. To address this, weight-only post-training quantization (PTQ) converts model weights to lower-bit-width representations (*e.g.*, INT4), reducing the memory footprint during inference and thus improving throughput in memory-bound autoregressive decoding.

Nevertheless, both activations and weights in LLMs possess many outliers (Dettmers et al., 2022; Xiao et al., 2023; Lin et al., 2024b), making it challenging to preserve the original precision under low-bit quantization. Most existing PTQ methods (Frantar et al., 2023; Lin et al., 2024b; Wei et al., 2023; Shao et al., 2024; Lee et al., 2024; Ashkboos et al., 2024; Chen et al., 2025; Tseng et al., 2024a;b) try to mitigate the impact of outliers, yet they either incur large quantization errors due to suboptimal outlier elimination or introduce significant overhead from arithmetic-intensive computation. For example, AWQ (Lin et al., 2024b), a widely adopted and fast quantization method, causes a **3.5% accuracy drop** of 4-bit quantized Qwen3-8B (Yang et al., 2025) on MMLU-Pro (Wang et al., 2024). In contrast, QTIP (Tseng et al., 2024b), which achieves state-of-the-art quantization accuracy, is about **30% slower** than AWQ because of the extra overhead introduced to mitigate outliers.

With the advent of reasoning LLMs (Jaech et al., 2024; Guo et al., 2025; Yang et al., 2025), we argue that *both* accuracy and efficiency are critical for practical quantization methods. Reasoning models achieve superior performance by generating a large number of chain-of-thought tokens, presenting unique challenges for quantization. On the one hand, quantization error *accumulates* at each decoding step, which becomes particularly pronounced in long generation. On the other hand, the substantial computational cost of generating long sequences requires that the quantization process itself introduce negligible overhead. Thus, there is a critical need for a quantization method that achieves high fidelity with minimal extra overhead.

In this paper, we propose **Pa**irwise **Ro**tation **Quant**ization (ParoQuant), a weight-only PTQ method that combines high accuracy with minimal computational overhead, making it well-suited to reasoning

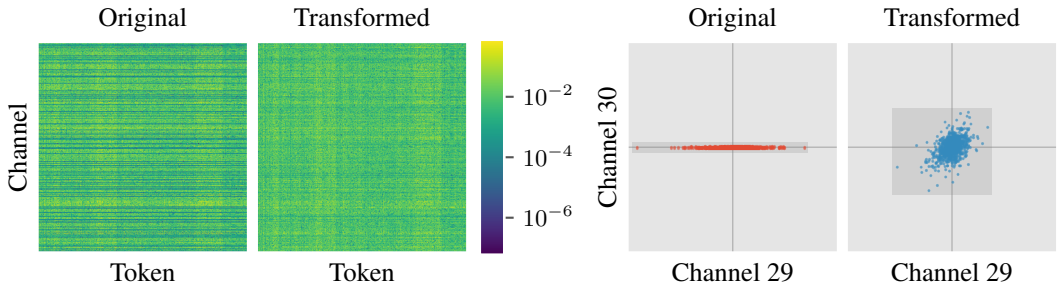


Figure 1: Effect of channel-wise scaling and rotation. **Left**: Magnitude of the `k_proj` weight matrix in the first layer of LLaMA-3-8B (Grattafiori et al., 2024) before and after the transform. The outlier channels have been eliminated effectively. **Right**: Scatter of channels 29 and 30 of the weight matrix before and after the transform. In addition to scaling, which concentrates values of the entire channel, rotations draw values from different channels closer at *each token* (clustering around the $x = y$ line).

LLMs. Our design rests on two key observations: (1) rotations effectively suppress outliers, and (2) a sparsely parameterized rotation can be as effective as a full rotation. Building on these insights, we introduce **scaled pairwise rotation**, a hardware-efficient and optimizable transform composed of *independent Givens rotations* and *channel-wise scaling*. Channel-wise scaling evens out the average magnitude across channels, while the pairwise (*i.e.*, Givens) rotations align the values within each channel pair at every token position, narrowing the dynamic range of each quantization group. As illustrated in Figure 1, our proposed transform makes the weights more quantization-friendly. To fully exploit the massive parallelism of modern GPUs, we further constrain the rotations to be mutually independent, a system-level design choice that keeps the impact on decoding latency minimal. Thanks to our algorithm-system co-design, ParoQuant achieves an average **2.4%** improvement over AWQ on reasoning tasks with less than 10% extra overhead, and matches the accuracy of QTIP while being about **25% faster**. We will release our code to facilitate reproducibility and further research.

2 Background and Related Work

2.1 LLM Quantization

Quantization is the process of converting values from high-precision to low-precision counterparts. The simple Round-to-Nearest (RTN) linear quantization with bit width b can be formulated as:

$$Q(\mathbf{X}) = \text{clamp}\left(\left\lfloor \frac{\mathbf{X}}{s} \right\rceil + z, 0, 2^b - 1\right), \text{where } s = \frac{\max(\mathbf{X}) - \min(\mathbf{X})}{2^b - 1}, z = -\left\lfloor \frac{\min(\mathbf{X})}{s} \right\rceil. \quad (1)$$

In this work, we focus on weight-only post-training quantization (PTQ), *i.e.*, quantizing weights of pre-trained models while keeping the activations in FP16. We follow the best practices proposed by Dettmers & Zettlemoyer (2023) and adopt block-wise quantization with a given group size g, *i.e.*, calculating a separate s and z in Equation (1) for every g consecutive elements along the channel dimension (*i.e.*, input dimension), instead of the whole matrix. Blocking helps to confine outliers within each group and increases overall quantization accuracy, particularly in linear quantization where quantization error is relatively large.

One major challenge of quantizing pre-trained LLMs to low bits is the presence of *outlier channels* across layers (Dettmers et al., 2022; Xiao et al., 2023; Lin et al., 2024b). They occupy the limited dynamic range of low-bit representations and cause precision loss of non-outlier elements, presenting a major challenge to PTQ. Past works have extensively studied the approaches to address the outlier issue, and the solutions can be broadly grouped into three categories: storing the outliers separately in higher precision (Dettmers et al., 2022; Kim et al., 2024; Lee et al., 2024; Zhao et al., 2024), designing quantization algorithms suitable for non-uniform distributions (Frantar et al., 2023; Chee et al., 2023; Tseng et al., 2024a;b), and transforming weights into quantization-friendly counterparts before quantization (Lin et al., 2024b; Wei et al., 2023; Shao et al., 2024; Tseng et al., 2024a; Ashkboos et al., 2024; Lin et al., 2024a; Chee et al., 2023; Liu et al., 2025; Tseng et al., 2024a;b). Yet, it remains a key challenge to balance quantization accuracy and inference speed, as effective outlier elimination often comes at the cost of significant overhead.

2.2 EQUIVALENT WEIGHT TRANSFORM

Among the three outlier handling techniques discussed earlier, *transforming weights before quantization* has been widely adopted by most recent PTQ methods and has proven to be very effective. For a linear layer $\mathbf{Y} = \mathbf{X}\mathbf{W} + \mathbf{b}$, where input $\mathbf{X} \in \mathbb{R}^{T \times D_{\text{in}}}$, weight $\mathbf{W} \in \mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}$, and bias $\mathbf{b} \in \mathbb{R}^{1 \times D_{\text{out}}}$, we can apply an invertible transform $\mathbf{T}$ to the weight $\mathbf{W}$ without affecting the output:

$$\mathbf{Y} = \mathbf{X}\mathbf{W} + \mathbf{b} = (\mathbf{X}\mathbf{T}^{-1})(\mathbf{T}\mathbf{W}) + \mathbf{b}. \tag{2}$$

We then quantize $\mathbf{TW}$ instead of $\mathbf{W}$. An appropriate $\mathbf{T}$ can reduce the outliers in $\mathbf{W}$ and lead to higher quantization accuracy. The inverse transform $\mathbf{T}^{-1}$ can either be applied on the fly during inference or be merged into other operators, depending on the characteristics of the transform.

Two main types of transform have been proposed in previous work: *channel-wise scaling*, where $\mathbf{T}$ is a diagonal matrix (Lin et al., 2024b; Shao et al., 2024; Wei et al., 2023), and *rotation*, where $\mathbf{T}$ is an orthogonal matrix (Chee et al., 2023; Ashkboos et al., 2024; Liu et al., 2025; Lin et al., 2024a; Tseng et al., 2024a;b). Channel-wise scaling scales each channel separately to even out the magnitude across channels and can usually be merged into preceding operators without incurring extra overhead (Lin et al., 2024b). Rotation enables cross-channel interactions that can concentrate values more effectively than channel-wise scaling (Chee et al., 2023; Liu et al., 2025). However, rotations cannot be merged into element-wise operators (*e.g.*, layer normalization) like channel-wise scaling does, so they usually require online computation. This limits the application of rotations in efficient quantization algorithms, as common orthogonal transforms are computationally expensive, and it motivates the design of more efficient yet equally effective alternatives.

3 MOTIVATION

Quantization Error Accumulates in Long Generation. AWQ (Lin et al., 2024b) is a widely used weight-only quantization method and has become the *de facto* approach for INT4 quantization. It employs channel-wise scaling to minimize quantization error and causes only slight performance degradation on most tasks with limited generated tokens, without introducing any extra overhead from the transform. However, we observe that the degradation becomes more severe as the generation length grows, especially on reasoning tasks with reasoning models, where the generation length often exceeds tens of thousands of tokens. For example, the accuracy of Qwen3-8B (Yang et al., 2025) on MMLU-Pro (Wang et al., 2024) drops sharply **from 68.6 to 65.1** after being quantized to 4 bits with AWQ. This degradation occurs because quantization errors accumulate at each decoding step.

Rotations Are Expressive but Expensive. Rotations outperform channel-wise scaling in eliminating outliers and generally lead to lower quantization error when many outliers are present (Figure 2). However, applying arbitrary rotations requires performing matrix multiplications in FP16, which negates the efficiency gains of quantization. There are two main approaches to address this issue. SpinQuant (Liu et al., 2025) proposes to merge the rotation matrix into the weight of the preceding linear layer so that no extra computation is needed during inference. However, in a typical decoder block, the output projection is the only linear layer that can be transformed by such mergeable rotations; other linears are preceded by element-wise operators or residual connections that cannot absorb matrix multiplications. The second approach is to restrict the orthogonal transform to a subset that can be computed efficiently on the fly. Several works adopt the Hadamard transform, a special orthogonal transform that can be computed in $\mathcal{O}(n \log n)$ time for dimension n (Chee et al., 2023; Ashkboos et al., 2024; Liu et al., 2025; Tseng et al., 2024a;b). Yet the Hadamard transform is generated by random vectors, disregarding the unique weight distribution of each linear layer and introducing large variance (Liu et al., 2025). Moreover, it still adds considerable overhead, making Hadamard-based quantization significantly slower (≈30%) than AWQ during inference.

Rotations Have Many Redundant Parameters. An $n \times n$ orthogonal matrix can be decomposed into the product of at most $\frac{1}{2}n(n-1)$ Givens rotations (*i.e.*, rotations in the plane spanned by two axes), which translates to rotating all possible pairs of channels sequentially. Intuitively, rotations between an outlier channel and a normal channel would be more effective at reducing outliers than rotations between two normal channels. We validate this intuition with a simple experiment: for a linear layer with many outliers, optimizing only the top 10% channel pairs with *largest magnitude difference* is

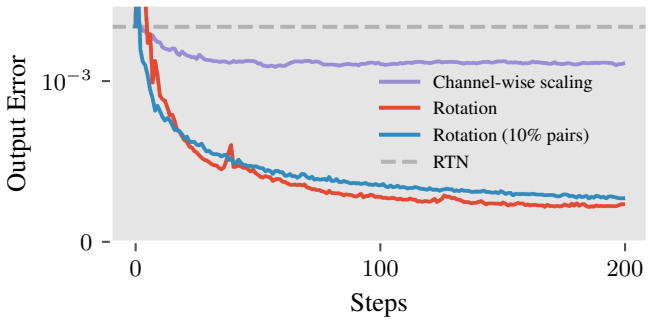


Figure 2: Loss curves from optimizing transforms to minimize quantization-induced output error ($\|\mathbf{X}Q(\mathbf{W}) - \mathbf{X}\mathbf{W}\|$) for the `k_proj` weight matrix in the first layer of LLaMA-3-8B. Rotations can minimize quantization error better than channel-wise scaling, and keeping the 10% most significant pairs is equally expressive as a full rotation. See Section A.2 for more details.

almost as effective at reducing quantization-induced output error as optimizing all pairs (Figure 2). This creates an opportunity for designing parameter-efficient and potentially inference-efficient rotations for addressing the outlier issue: by retaining only the rotations between channel pairs that have large magnitude differences, we can maintain the effectiveness of a full $n \times n$ orthogonal matrix.

4 METHOD

Will be included in preprint.

5 EVALUATION

Models and Tasks. We apply ParoQuant on LLaMA-2 (7B) (Touvron et al., 2023), LLaMA-3 (8B, 70B) & LLaMA-3.1 Instruct (8B) (Grattafiori et al., 2024), DeepSeek-R1-distilled LLaMA-3.1 (8B) (Guo et al., 2025), and Qwen3 (1.7B, 4B, 8B, 14B) (Yang et al., 2025) pre-trained models. We evaluate the quantization quality with three types of evaluation: (1) *Perplexity* on WikiText2 (Merity et al., 2017) and C4 (Dodge et al., 2021); (2) *Reasoning accuracy* on MMLU-Pro (Wang et al., 2024), GSM8K (Cobbe et al., 2021), GPQA Diamond (Rein et al., 2024), and AIME (MAA, 2024); (3) *Non-reasoning accuracy* on BoolQ (Clark et al., 2019), ARC-Challenge, ARC-Easy (Clark et al., 2018), and HellaSwag (Zellers et al., 2019).

Implementation. We focus on 4-bit weight-only linear quantization with a group size of 128. Linear quantization is more efficient and widely supported by existing frameworks. The choice of 4 bits and a 128 group size offers the optimal trade-off between accuracy and bit width for linear quantization (Dettmers & Zettlemoyer, 2023). We apply 8 independent rotations on each 128-channel group, with each rotation consisting of up to 64 pairs. Each layer is optimized for 10 epochs using AdamW (Loshchilov & Hutter, 2019) with a fixed set of hyperparameters for all experiments, except for the 70B model, where we adjust the batch size to accommodate memory constraints. To reduce the risk of overfitting to one dataset, we use a training set of 2048 samples drawn evenly from WikiText2, C4, and RedPajama (Weber et al., 2024), and select the best parameters at each training epoch using 64 validation samples from Pile (Gao et al., 2020). More details are provided in Section A.3.

Baselines. We compare the accuracy and efficiency of ParoQuant with three weight-only PTQ baselines. AWQ (Lin et al., 2024b) optimizes channel-wise scaling with grid search and is the most used 4-bit weight-only quantization method. EfficientQAT (Chen et al., 2025) achieves state-of-the-art linear quantization accuracy with layer-wise fine-tuning of weights and quantization parameters*. QTIP (Tseng et al., 2024b) is the state-of-the-art vector quantization method utilizing randomized Hadamard transform and an advanced trellis quantization algorithm. In addition, we include the perplexity results of QuIP# (Tseng et al., 2024a), a vector-quantization predecessor of QTIP that also adopts the Hadamard transform, and two weight-activation linear quantization methods, OmniQuant (Shao et al., 2024) and SpinQuant (Liu et al., 2025), which are also applicable for weight-only quantization. We apply block-wise quantization with a group size of 128 on all linear quantization methods and the corresponding default settings on vector quantization methods.

5.1 ACCURACY RESULTS

Perplexity. Table 1 shows the perplexity results of 4-bit quantized models ranging in size from 1.7B to 70B. Among linear quantization methods, ParoQuant achieves state-of-the-art quantization

*We only apply the "Block-AP" stage of EfficientQAT, as its "E2E-QP" stage involves supervised fine-tuning, which is out of the scope of PTQ.

accuracy across all sizes, particularly on challenging cases like LLaMA-3-8B and smaller models under 4B. It also delivers strong performance compared with rotation-based methods including QuIP#, QTIP, and SpinQuant. It outperforms QuIP# and matches QTIP on all models, despite the inherently larger error of linear quantization, highlighting the superior effectiveness of our proposed transform over the Hadamard transform (see Section A.2 for detailed analysis). Moreover, ParoQuant provides a decent speedup over these two methods. This underscores the efficiency of our proposed transform.

Method	Type	WikiText2							C4							Speedup
		L3-8	L3-70	L2-7	Q3-1.7	Q3-4	Q3-8	Q3-14	L3-8	L3-70	L2-7	Q3-1.7	Q3-4	Q3-8	Q3-14	
FP16	–	5.54	2.56	5.12	8.32	7.01	6.24	5.70	7.10	5.78	6.63	8.62	7.61	6.97	6.54	1.0×
QuIP#	vector	5.81	2.99	5.19	–	–	–	–	7.32	5.96	6.75	–	–	–	–	1.9×
QTIP	vector	5.67	2.75	5.17	–	–	–	–	7.20	5.83	6.69	–	–	–	–	1.7×
AWQ	linear	5.92	2.96	5.23	8.80	7.36	6.45	5.85	7.42	5.91	6.80	9.01	7.89	7.14	6.65	2.4×
OmniQ	linear	–	–	5.23	–	–	–	–	–	–	6.80	–	–	–	–	2.4×†
SpinQ	linear	5.83	–	5.21	–	–	–	–	7.41	–	6.86	–	–	–	–	2.4×†
E-QAT	linear	5.87	3.33	5.22	8.60	7.19	6.37	5.82	7.36	6.72	6.76	8.84	7.77	7.08	6.63	2.4×†
ParoQ	linear	**5.72**	**2.82**	**5.18**	**8.43**	**7.10**	**6.30**	**5.75**	**7.26**	**5.86**	**6.74**	**8.75**	**7.70**	**7.04**	**6.60**	2.2×

† Uses results of AWQ as a reference as the method does not incur significant overhead from the transform.

Table 1: Perplexity (↓) results of 4-bit models. The context length is 8192 for LLaMA-3 and Qwen3 (base models), and 4096 for LLaMA-2. The best results among linear quantization methods are in **bold**. Speedup over FP16 models is reported as the geometric mean across Q3-1.7, Q3-4, L3-8, and Q3-14, measured on an RTX A6000 with a batch size of 1 during decoding.

Reasoning Tasks. Table 2 shows the accuracy results of five reasoning benchmarks: MMLU-Pro (12k samples), GSM8K (1.3k samples), GPQA Diamond (198 samples), and AIME24/25 (30 samples each). On the larger MMLU-Pro benchmark, ParoQuant consistently outperforms all baselines. While results on the smaller GSM8K, GPQA and AIME benchmarks exhibit more randomness due to the limited number of samples, ParoQuant's performance never degrades by more than 1.7%, showcasing a level of stability not seen in other baselines. Overall, ParoQuant preserves the reasoning capabilities of the original models and achieves **2.4%** and **2.9%** improvements over AWQ and EfficientQAT, respectively. This demonstrates ParoQuant's superior quantization accuracy in long generation.

Method	Type	R1-Distill-Llama-8B			Qwen3-8B				Qwen3-14B				Avg.
		GSM8K	GPQA	AIME	MMLU	GSM8K	GPQA	AIME	MMLU	GSM8K	GPQA	AIME	
FP16	–	72.9	39.4	38.3	68.6	73.5	44.4	70.0	72.2	88.2	53.0	76.7	63.4
QTIP	vector	71.0	37.9	40.0	–	–	–	–	–	–	–	–	–
AWQ	linear	70.3	40.4	35.0	65.1	71.4	**48.5**	**71.7**	70.9	87.2	52.0	73.3	62.3
E-QAT	linear	69.5	40.4	**43.3**	61.1	**79.8**	47.2	55.0	71.0	91.2	51.8	70.0	61.8
ParoQ	linear	**71.4**	**42.4**	**43.3**	**67.8**	75.9	47.5	**71.7**	**71.5**	**92.3**	**52.5**	**75.0**	**64.7**

Table 2: Accuracy (↑) on reasoning tasks. We report 5-shot accuracy for GSM8K and zero-shot accuracy for the other benchmarks. DeepSeek-R1-Distill-Llama-8B fails to produce reasonable results on MMLU within the set token limits. More details are provided in Section A.6.

Non-Reasoning Tasks. Table 3 shows the zero-shot accuracy on commonsense benchmarks with thinking mode disabled. ParoQuant maintains near-lossless performance, outperforming AWQ and EfficientQAT by 1% and 0.7%, respectively. The accuracy gap is smaller than in reasoning tasks because these benchmarks evaluate only a few generated tokens, so error accumulation is minimal.

5.2 Efficiency Results

Table 4 shows the decoding throughput on an RTX A6000. To ensure a fair comparison, we implement all methods on top of the Transformers library (Wolf et al., 2020), modifying only the weight transform and dequantization code (details and more results are in Section A.4). ParoQuant is only about 10% slower than AWQ while providing a significant accuracy improvement, and it matches the accuracy of QTIP while being 15%-30% faster. For the training efficiency, see Section A.5 for more details.

Method	Type	LLaMA-3.1-8B-Instruct				Qwen3-4B				Qwen3-8B				Qwen3-14B				Avg.
		BoolQ	ARC-C	ARC-E	HSwag	BoolQ	ARC-C	ARC-E	HSwag	BoolQ	ARC-C	ARC-E	HSwag	BoolQ	ARC-C	ARC-E	HSwag	
FP16	-	84.1	51.7	81.8	59.1	85.1	50.7	80.5	52.3	86.6	55.8	83.6	57.1	89.3	58.6	84.1	61.0	70.1
QTIP	vector	**84.3**	51.2	**81.7**	**58.8**	–	–	–	–	–	–	–	–	–	–	–	–	–
AWQ	linear	83.5	51.5	80.6	58.4	85.1	47.4	77.9	51.3	86.2	53.8	82.2	56.3	89.0	57.9	83.2	60.3	69.0
E-QAT	linear	83.8	**51.8**	81.6	58.4	**85.2**	47.2	78.4	51.2	**86.8**	54.7	82.7	**56.5**	88.8	58.1	83.7	60.3	69.3
PAROQ	linear	83.8	51.3	81.3	**58.8**	84.8	**51.0**	**80.4**	**51.5**	**86.8**	**56.3**	**84.1**	56.4	**89.6**	**58.6**	**84.3**	**60.7**	**70.0**

Table 3: Zero-shot accuracy (↑) on non-reasoning tasks.

Method	Qwen3-1.7B		Qwen3-4B		LLaMA-3-8B		Qwen3-14B	
	Throughput	W2 PPL	Throughput	W2 PPL	Throughput	W2 PPL	Throughput	W2 PPL
FP16	170 (1.0×)	8.32	78 (1.0×)	7.01	45 (1.0×)	5.54	25 (1.0×)	5.70
AWQ	320 (1.9×)	8.80	176 (2.3×)	7.36	120 (2.7×)	5.92	70 (2.8×)	5.85
QTIP	209 (1.2×)	–	117 (1.5×)	–	95 (2.1×)	5.67	55 (2.2×)	–
PAROQ	278 (1.6×)	8.43	160 (2.1×)	7.10	112 (2.5×)	5.72	65 (2.6×)	5.75

Table 4: Decoding (with batch size of 1) throughput (tokens/s).

5.3 Ablation Study

Will be included in preprint.

6 Conclusion

In this paper, we proposed ParoQuant, an efficient weight-only PTQ method that achieves state-of-the-art quantization accuracy with minimal overhead. Based on the insight that a sparsely parameterized rotation can effectively suppress weight outliers, we designed scaled pairwise rotation, which combines hardware-friendly independent Givens rotations with channel-wise scaling. ParoQuant matches the accuracy of the best existing quantization methods while running much faster, and it consistently outperforms prior efficient quantization methods, especially on reasoning tasks where quantization errors accumulate over long chains of thought. We hope that our method will inspire future research on high-fidelity, low-overhead quantization techniques for next-generation reasoning LLMs.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024. 1, 2, 3

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-Bit Quantization of Large Language Models with Guarantees. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023. 2, 3

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. EfficientQAT: Efficient Quantization-Aware Training for Large Language Models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. 1, 6, 7

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019. 7

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*, 2018. 7

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021. 7

Tri Dao. Fast Hadamard Transform in CUDA, with A PyTorch Interface. https://github.com/Dao-AILab/fast-hadamard-transform, 2024. 7

Tim Dettmers and Luke Zettlemoyer. The Case for 4-bit Precision: K-bit Inference Scaling Laws. In *International Conference on Machine Learning (ICML)*, 2023. 2, 7

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022. 1, 2

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting Large Webtext Corpora: A Case Study on The Colossal Clean Crawled Corpus. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021. 7

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. In *International Conference on Learning Representations (ICLR)*, 2023. 1, 2, 16

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv preprint arXiv:2101.00027*, 2020. 7

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, et al. The Language Model Evaluation Harness, 2024. URL https://zenodo.org/records/12608602. 17

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 2, 7

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025. 1, 7

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI O1 System Card. *arXiv preprint arXiv:2412.16720*, 2024. 1

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-Sparse Quantization. In *International Conference on Machine Learning (ICML)*, 2024. 2

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 *ACM Symposium on Operating Systems Principles (SOSP)*, 2023. 4

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. OWQ: Outlier-Aware Weight Quantization for Efficient Fine-Tuning and Inference of Large Language Models. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2024. 1, 2

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. Datasets: A Community Library for Natural Language Processing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021. 15

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. DuQuant: Distributing Outliers via Dual Transformation Makes Stronger Quantized LLMs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024a. 2, 3

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-Aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Conference on Machine Learning and Systems (MLSys)*, 2024b. 1, 2, 3, 4, 7

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. SpinQuant: LLM Quantization with Learned Rotations. In *International Conference on Learning Representations (ICLR)*, 2025. 2, 3, 7

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations (ICLR)*, 2019. 7

MAA. American Invitational Mathematics Examination - AIME, 2024. URL https://maa.org/math-competitions/american-invitational-mathematics-examination-aime. 7

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations (ICLR)*, 2017. 7

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. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 15

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 *Conference on Language Modeling (COLM)*, 2024. 7

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2024. 1, 2, 3, 7

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. 7

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks. In *International Conference on Machine Learning (ICML)*, 2024a. 1, 2, 3, 7

Albert Tseng, Qingyao Sun, David Hou, and Christopher M De Sa. QTIP: Quantization with Trellises and Incoherence Processing. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024b. 1, 2, 3, 7

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024. 1, 3, 7

Maurice Weber, Dan Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, et al. RedPajama: an Open Dataset for Training Large Language Models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024. 7

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier Suppression+: Accurate Quantization of Large Language Models by Equivalent and Optimal Shifting and Scaling. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. 1, 2, 3

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-Art Natural Language Processing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020. 8, 15

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 (ICML)*, 2023. 1, 2

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. 1, 3, 7

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019. 7

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 *Conference on Machine Learning and Systems (MLSys)*, 2024. 2

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. SGLang: Efficient Execution of Structured Language Model Programs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024. 4

A.1 CORE ALGORITHMS

Will be included in preprint.

A.2 EFFECTIVENESS ANALYSIS

To study the effectiveness of different transforms at minimizing quantization-induced output error, we optimize each linear layer individually with each transform applied to the input dimension of its weight $\mathbf{W}$ and obtain the loss curves of the optimization process. We compare the effectiveness of 5 transforms:

Figure A1: Loss curves from optimizing transforms to minimize quantization-induced output error of linear layers in LLaMA-3-8B.

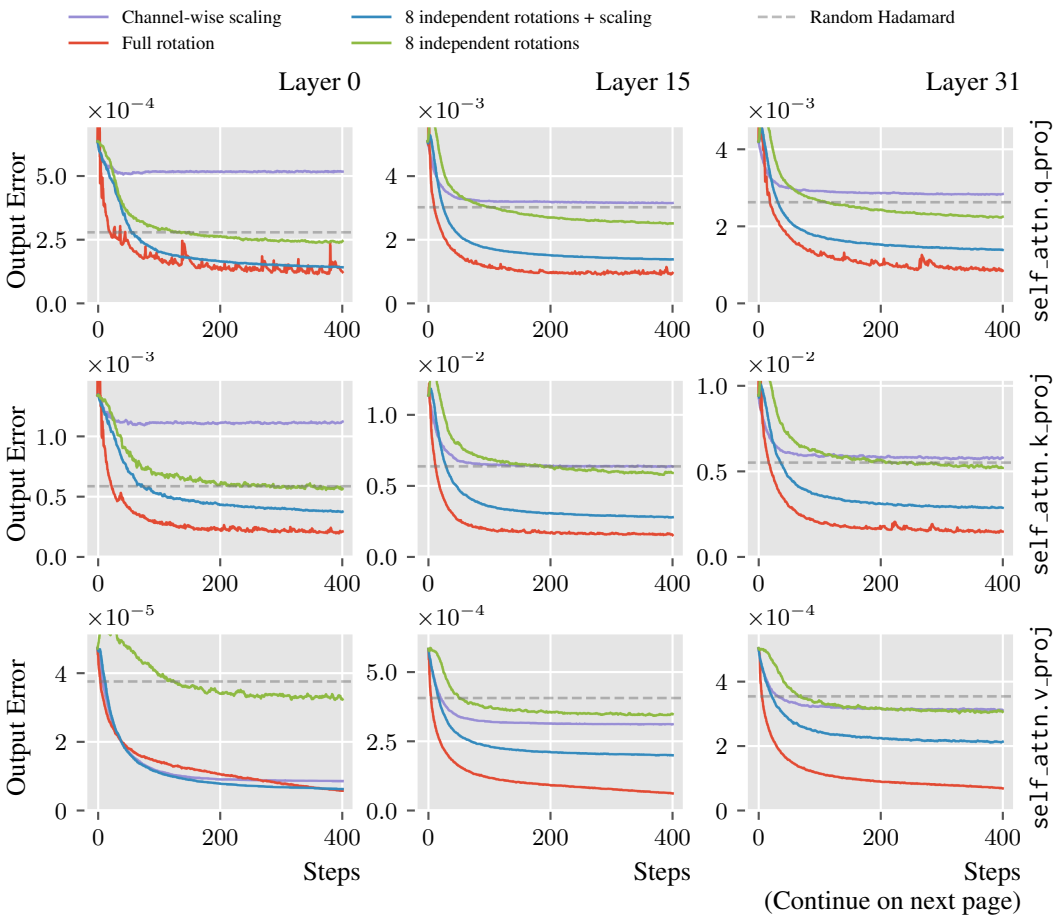
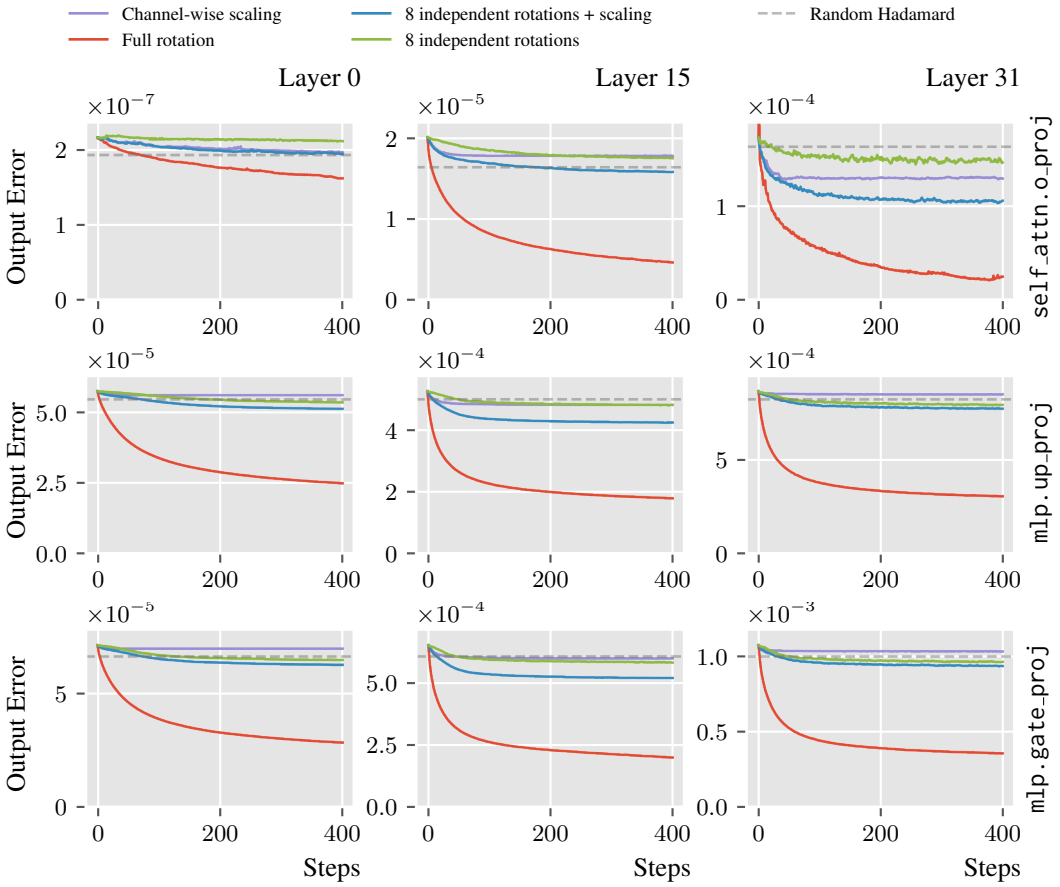


Figure A1: Loss curves from optimizing transforms to minimize quantization-induced output error of linear layers in LLaMA-3-8B (continued).



A.3 Implementation Details

Quantization and Transform. We apply block-wise linear quantization with a group size of 128. We apply channel-wise scaling and 8 independent rotations on each 128-channel group of the weights, with each rotation consisting of up to 64 pairs. The independent rotations are applied sequentially after channel-wise scaling.

Libraries. Our implementation is built with PyTorch `2.8.0` (Paszke et al., 2019), Transformers `4.55.2` (Wolf et al., 2020), and Datasets `3.6.0` (Lhoest et al., 2021).

Optimization. We optimize all ParoQuant-quantized models in Section 5 with a single NVIDIA H200 GPU. We sample a training set of size 2048 evenly from WikiText2, C4, and RedPajama, and use 64 samples from Pile as the validation set to select the best parameters at each epoch. The training set and the validation set are shuffled with a fixed seed of 0. The sequence length of each sample is 2048. Except for the LLaMA-3-70B model, we set the batch size as 16, and apply a learning rate of 0.05 for rotation angles and channel-wise scaling, 10^{-5} for weights, and 10^{-6} for scales and zero points. The batch size and the learning rates are halved for the 70B model due to memory constraints. The rotation angles are initialized to 0, the channel-wise scaling factors are initialized to 1, and the scales and zero points are initialized using Equation (1). We use AdamW to optimize the parameters for 10 epochs at each stage (see Algorithm A2 for the two stages) with a cosine learning rate scheduler, which gradually decays the learning rate to $\frac{1}{20}$ of the original value. The

hyperparameters `weight_decay`, `betas`, and `eps` of the AdamW optimizer are set to 0.01, (0.9, 0.95), and 10^{-10}, respectively. We use `SmoothL1Loss` from PyTorch as the loss function.

A.4 Decoding Throughput

Table A2 shows the decoding throughput of the original FP16 models and 4-bit models quantized with AWQ, QTIP, and ParoQuant. To ensure a fair comparison, we only replace the original linear layers in the original models with the implementation provided by the official open-sourced repository of each baseline. For ParoQuant, we adopt the W4A16 kernels from the AWQ repository together with our transform kernel. All throughput results were obtained with PyTorch `2.6.0` using `torch.compile` in `max-autotune` mode and with CUDA Graphs enabled.

Table A2: Decoding (batch size = 1) throughput (tokens/s).

RTX A6000	Bits	Q3-1.7	Q3-4	L2-7	L3-8	Q3-8	L2-13	Q3-14	Q3-32	L3-70
FP16	16	170	78	50	45	44	26	25	OOM	OOM
AWQ	4	320	176	140	120	113	78	70	34	17
QTIP	4	209	117	106	95	91	62	55	28	15
PAROQ	4	278	160	130	112	106	74	65	33	16
RTX 6000 Ada	Bits	Q3-1.7	Q3-4	L2-7	L3-8	Q3-8	L2-13	Q3-14	Q3-32	L3-70
FP16	16	213	99	63	56	55	33	31	OOM	OOM
AWQ	4	394	230	176	153	147	100	89	44	21
QTIP	4	270	166	138	125	118	83	80	40	20
PAROQ	4	341	206	163	142	136	94	84	42	21
RTX 4090	Bits	Q3-1.7	Q3-4	L2-7	L3-8	Q3-8	L2-13	Q3-14	Q3-32	L3-70
FP16	16	233	109	69	62	61	OOM	OOM	OOM	OOM
AWQ	4	433	251	192	167	159	109	98	48	OOM
QTIP	4	286	172	149	138	138	91	82	42	OOM
PAROQ	4	372	224	177	155	147	102	93	46	OOM

A.5 Training Efficiency

Table A3 shows the calibration size and GPU time for quantizing LLaMA-3-8B on an NVIDIA H200 GPU. Although ParoQuant is slower than EfficientQAT due to an extra tuning stage and the additional computation graph nodes from independent rotations, it's still significantly faster than QTIP, which is slowed down by two extra steps in addition to layer-wise fine-tuning: generating Hessian matrices and end-to-end fine-tuning.

Table A3: Calibration data (# samples $\times$ sequence length) and GPU time for quantizing LLaMA-3-8B on an NVIDIA H200 GPU.

	AWQ	E-QAT	QTIP	PAROQ
Calibration Data	128 $\times$ 512	4096 $\times$ 2048	4096 $\times$ 8192	2048 $\times$ 2048
GPU Time	minutes	$\approx$ 3 hours	$\approx$ 20 hours	$\approx$ 9 hours

A.6 Evaluation Settings

Perplexity Evaluation. We use the test split in GPTQ (Frantar et al., 2023) to measure the perplexity on WikiText2 and C4. The sequence length is 8192 for LLaMA-3 and Qwen3 models, and 4096 for LLaMA-2 models. Note that the perplexity results of Qwen3 family in Table 1 are from the pre-trained "Base" models (*e.g.*, `Qwen/Qwen3-8B-Base` from huggingface), not the models after post-training (*e.g.*, `Qwen/Qwen3-8B`).

Non-Reasoning Task Evaluation. We use the Language Model Evaluation Harness library (`lm_eval`) version `0.4.9.1` (Gao et al., 2024) to evaluate the tasks in Table 3 with the default settings of the library.

Reasoning Task Evaluation. Like non-reasoning tasks, we use the same version of `lm_eval` to evaluate the reasoning tasks in Table 2. However, the library does not have good support for reasoning models. For example, in the settings for the MMLU-Pro benchmark, the maximum generated tokens is set as 2048, which is insufficient for reasoning models; the QA prompt includes "Answer" immediately after the question, which contradicts the reasoning mechanism. Thus, we override some default settings for the benchmarks in Table 3. We set the limit of maximum generated tokens to 32768 for AIME24/25 and 8192 for other tasks. The task-specific modifications are listed in Table A4.

Benchmark	Modification	Details
MMLU-Pro	Prompt	Change "Answer: Let's think step by step." to "Please put your answer in \boxed{}."
	Answer Extraction	Use regular expression to extract "\boxed{[A-J]}".
GSM8K	Prompt	Remove "Answer:" in the prompt.
GPQA	Answer Extraction	Use regular expression to extract "\boxed{[A-D]}" or "([A-D])".
AIME24/25	Prompt	Remove "Answer:" in the prompt.

Table A4: Modifications to the `lm_eval` library for reasoning tasks.