QTEA: Ternary LLMs with Sparse Residual Salient Weight and By-Column Optimization

Anonymous ACL submission

Abstract

Weight-only post-training quantization (PTQ) can alleviate the computational burden of serving large language models (LLMs) at scale. However, existing PTQ methods often fail to generalize across models and suffer severe accuracy loss below 2 bits. Many leverage unstructured sparsity to mitigate this loss, but at the cost of regularity and GPU-friendly execution. We present QTEA, a sub-2-bit PTQ framework that quantizes weights into ternary values and uses salient weights as residual error compensators. To maintain hardware efficiency, residuals are assigned to selected columns with semi-structured 1:4 sparsity within the salient columns. We further add column-wise rescale refinement to GPTQ-style column-by-column quantization, alternately updating per-column scales and ternary assignments to reduce reconstruction error. We also identify order-dependent error propagation in GPTQ and introduce error decay to attenuate late-stage error accumulation. On Qwen3-14B, QTEA compresses all weights to an effective 1.7 bits per weight while improving average accuracy over the strongest ternary PTQ baseline by 16.7%. It also achieves 1.40× and 2.61× lower perplexity on WikiText and C4 respectively. This trend holds on Llama3-8B, where QTEA obtains a 6.6% accuracy gain and 1.34×/1.95× lower perplexity on the same datasets. Finally, we develop a lookup-table based kernel that achieves 7.2× faster per-token generation over an FP16 baseline.

1 Introduction

Large language models (LLMs) have demonstrated strong capabilities across natural language understanding, generation, and reasoning tasks (Touvron et al., 2023; Meta, 2024; Yang et al., 2025). However, deploying these models remains expensive because inference requires repeatedly loading and storing billions of parameters. A 70-billion-parameter model requires roughly 140 GB for BF16 weights alone, before accounting for KV cache and runtime overhead. This cost is especially significant for low-batch autoregressive decoding, where each generated token is dominated by low-arithmetic-intensity GEMV operations that repeatedly fetch model weights from HBM to the compute cores, making inference memory-bandwidth bound. Therefore, weight-only post-training quantization (PTQ) is a direct path toward lower memory footprint and faster low-batch inference.

Recent PTQ methods have delivered 2-bit LLMs (Chee et al., 2024; You et al., 2024; Guo et al., 2024) and even sub-2-bit techniques (Wang et al., 2023; Ma et al., 2024), substantially improving intelligence density. At such extremely low precision, uniformly quantizing all weights often severely degrades accuracy, making it necessary to preserve a small subset of salient weights with higher fidelity. However, preserving salient weights well is itself costly, and existing approaches differ mainly in which cost they pay. AWQ (Lin et al., 2024) rescales salient channels but degrades in the ternary regime. Unstructured salient-weight preservation (Yuan et al., 2024) improves accuracy but introduces irregular memory access(Hoefler et al., 2021; Gale et al., 2020), while column-wise preservation (Huang et al., 2024; Zhao et al., 2025) is more regular but wastes bit budget by storing and processing many non-salient weights together with salient ones. Each design therefore gives up either accuracy or hardware efficiency.

We observe that, in ultra-low-bit PTQ, salient weights can serve as error compensators rather than simply bypassing quantization. Instead of excluding important weights from the low-bit representation, we first quantize all weights into a compact ternary base and then allocate a bit budget to columns where ternary quantization is most harmful. We further combine column-level saliency with semi-structured sparsity, into a scheme we call *column-semi sparse* salient weight: residuals are as-

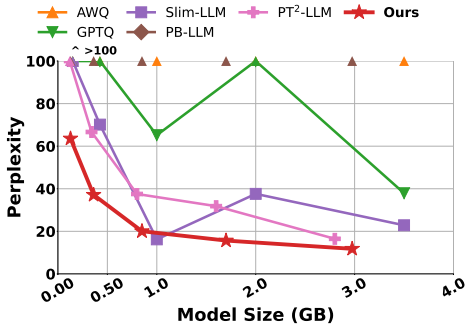

Figure 1: WikiText2 perplexity comparison across Qwen3 model sizes. Results with perplexity greater than 100 are omitted.

signed only to important columns, and within these columns only a sparse subset of high-impact entries is retained. This preserves critical weight corrections under an extremely tight bit budget while maintaining a regular structure suitable for efficient kernels.

Building on this representation, we incorporate column-wise rescale refinement into GPTQ-style column-by-column quantization. Rather than fixing the ternary scale after the initial parameter estimation, we refine a per-column rescale factor during quantization so that the ternary base better matches the local weight distribution after Hessian-aware error propagation.

In addition, we observe that GPTQ error propagation introduces an order-dependent imbalance. Earlier columns can distribute their quantization errors over many subsequent columns, while later columns can only propagate errors to a short remaining suffix. Thus, applying the same propagation strength throughout the block may make late-stage error compensation overly aggressive and less reliable, since fewer columns are available to absorb the update at each step. To mitigate this effect, we introduce *error decay*, which assigns position-dependent propagation strengths during GPTQ-style quantization. By gradually attenuating error propagation along the column order, error decay reduces late-stage error accumulation and stabilizes ultra-low-bit quantization.

We integrate these observations and optimizations into a unified ultra-low-bit PTQ framework, termed **Quantized Ternary Error Adaptation (QTEA)**. On Qwen3-14B, QTEA substantially outperforms the current state-of-the-art ternary PTQ method, improving average downstream-task accuracy by 16.7% while achieving 1.40× and 2.61× lower perplexity on WikiText and C4, respectively. The same trend holds on Llama3-8B, where QTEA improves average downstream-task accuracy by 6.6% and reduces perplexity by 1.34× on WikiText and 1.95× on C4. Figure 1 demonstrates that QTEA achieves the best trade-off between model size and perplexity on Qwen3.

In addition, we develop efficient computation kernels for QTEA, translating the compact ternary representation into practical inference speedups. Combined with CUDA Graphs, QTEA reduces runtime overhead and achieves a 7.2× generation speedup over the FP16 baseline with CUDA Graphs; compared with FP16 inference without CUDA Graphs, the speedup reaches 13.3×.

Our contributions can be summarized as follows:

- We propose **QTEA**, a Ternary PTQ framework that quantizes all weights into a ternary base and uses column-semi sparse salient weights as residual compensators rather than unquantized bypasses.
- We incorporate column-wise rescale refinement into GPTQ-style column-by-column quantization and alternately optimize the rescale factor and ternary assignment to improve the expressiveness of the ternary base.
- We identify the order-dependent imbalance in GPTQ error propagation and propose an error decay mechanism to balance compensation strength across the quantization order.
- We have developed CUDA kernels enabling QTEA to achieve a per-token generation speedup of up to 7.2× over FP16 with CUDA Graphs and 13.3× over FP16 without CUDA Graphs.

2 Related Work

Post-training quantization for LLMs. Post-training quantization (PTQ) compresses pre-trained LLMs without retraining, and weight-only PTQ directly reduces inference memory footprint and bandwidth pressure. GPTQ (Frantar et al., 2023) minimizes layer-wise reconstruction error with approximate second-order information. AWQ (Lin et al., 2024) protects salient channels via activation-aware scaling, while OmniQuant (Shao et al., 2024) learns clipping ranges and equivalence transformations.

Ultra-low-bit LLMs QAT. Training-aware ultra-low-bit methods improve quantizability through

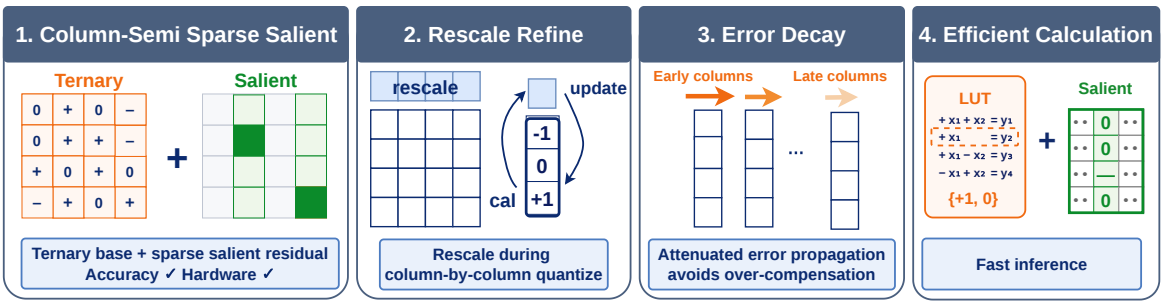

Figure 2: Overview of QTEA. We employ Column-Semi Sparse Salient to enhance quantization robustness while maintaining equivalent overhead; utilize Rescale Refinement to boost the representational power of ternary weights; apply Error Decay to mitigate the issue of uneven error propagation during column-wise quantization; and finally, construct an efficient computational kernel to accelerate inference.

fine-tuning or retraining. QuIP# (Tseng et al., 2024) combines hardware-friendly rotations with fine-tuning for 2-bit quantization, BitNet (Wang et al., 2023; Ma et al., 2024) trains LLMs with binary or ternary weights, and Sherry (Huang et al., 2026) introduces hardware-efficient 1.25-bit ternary QAT with fine-grained 3:4 sparsity. In contrast, QTEA is a PTQ method that recovers accuracy through adaptive ternary error compensation without any retraining.

Ultra-low-bit LLMs PTQ. Sub-2-bit PTQ requires additional mechanisms to avoid severe accuracy degradation. QuIP (Chee et al., 2024) improves quantizability with rotations, PB-LLM (Yuan et al., 2024) preserves unstructured salient weights during partial binarization, and SliM-LLM (Huang et al., 2025) assigns mixed precision by layer sensitivity. BiLLM (Huang et al., 2024) explores binary PTQ with magnitude-based grouping, while PT^2-LLM (Yan et al., 2026) shows that post-training ternarization can provide a practical sub-2-bpw regime. However, many ultra-low-bit PTQ methods rely on unstructured designs (Huang et al., 2024; Yuan et al., 2024; Li et al., 2026; Gu et al., 2026; Yan et al., 2025), which require extra mask bits and are less friendly to efficient implementation.

3 Quantized Ternary Error Adaptation

3.1 Ternary Weight Quantization

Ternary quantization approximates each full-precision weight with a ternary value, a scale, and a bias:

$$\hat{w} = \alpha t + \beta, \quad t \in \{-1, 0, +1\}, \tag{1}$$

where α controls the magnitude of non-zero entries and β shifts the quantization center. Following PT^2-LLM (Yan et al., 2026), we estimate α and β for each quantization group and assign each weight to the nearest point in $\{\beta-\alpha, \beta, \beta+\alpha\}$. Equivalently, the ternary value is determined by

$$T_{ij} = \begin{cases} +1, & W_{ij} - \beta_i > \Delta, \\ -1, & W_{ij} - \beta_i < -\Delta, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where $\Delta = \alpha_i/2$ with $\alpha_i > 0$. Thus, T provides the compact ternary base representation used by QTEA.

3.2 Column-Semi Sparse Salient Weights

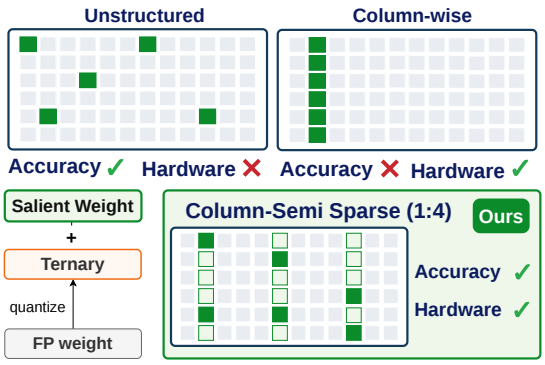

Figure 3: Column-semi sparse salient weights balance accuracy and hardware friendliness.

Preserving a small fraction of salient weights in higher precision can mitigate the accuracy loss of ultra-low-bit quantization (Lin et al., 2024). Existing methods usually preserve them either with unstructured sparsity (Yuan et al., 2024), which is hard to accelerate, or by selecting salient columns and quantizing them separately at higher precision (Huang et al., 2024; Zhao et al., 2025), which wastes bit budget on non-salient entries.

We first study how to select salient columns. Prior methods use metrics such as $\mathrm{diag}(H)$ (Frantar et al., 2023) or $\mathrm{mean}(W^2 \cdot \mathrm{diag}(H)^2)$ (Huang et al., 2024), where $H = \frac{2}{N} X X^\top$ is the approximate Hessian, here N represents the number of samples. Empirically, we find that selecting columns by their maximum individual weight impact is more effective than Hessian-diagonal or average-importance metrics, as shown in Table 3. Specifically, we score each column by

$$s_j = \max_i \left(W_{ij}^2 \cdot \mathrm{diag}(H)_j^2 \right). \tag{3}$$

Intuitively, $W_{ij}^2 \cdot \mathrm{diag}(H)_j$ is the standard second-order estimate of the output error from perturbing W_{ij}, and the additional $\mathrm{diag}(H)_j$ factor further favors columns driven by high-energy activations. Taking the maximum over rows ranks each column by its single most damaging entry rather than its average importance, consistent with observations on SuperWeights (Yu et al., 2025). Because the score is used only to order columns for residual allocation, it needs no normalization, since any per-layer rescaling preserves the ranking.

Based on this observation, QTEA uses salient weights as sparse residual compensators rather than high-precision bypasses. We first quantize all weights into the ternary base, and then add an independent sparse residual matrix on selected salient columns to correct the most harmful ternary quantization errors. This preserves a ternary approximation for every weight, avoiding the degeneration caused by directly keeping sparse high-precision weights while setting the rest to zero. We further impose a 1:4 semi-sparse pattern on the residual matrix, reducing overhead and making the structure more hardware-friendly than fully unstructured sparsity.

3.3 Column-Wise Rescale Refinement

A straightforward ternary quantizer uses the group-wise scale α for all columns in a group. However, in GPTQ-style column-by-column quantization, each column may have been modified by Hessian-aware error propagation from previous columns, making the initial group-wise scale suboptimal for the current column.

To address this, we introduce column-wise rescale refinement. For the j-th column, we keep the group-wise bias β and scale α, and add a lightweight column-wise rescale factor v_j:

$$\hat{W}_{ij} = \beta_i + v_j \alpha_i T_{ij}, \quad T_{ij} \in \{-1, 0, +1\}. \tag{4}$$

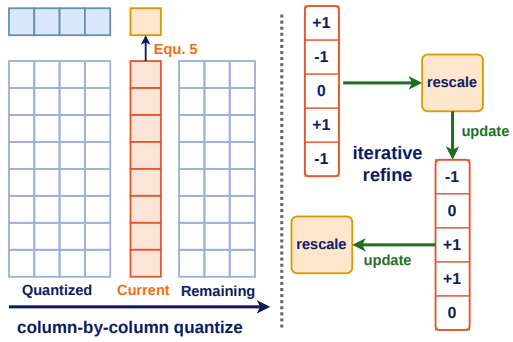

Figure 4: Column-wise rescale refinement during GPTQ-style quantization.

Here, α_i and β_i remain shared within the quantization group, while v_j is refined for each column before the GPTQ error update.

Intuitively, GPTQ error propagation changes the magnitude distribution of each remaining column as quantization proceeds. A static group-wise scale therefore becomes mismatched to the current column, especially when ternary assignments are highly sensitive to scale. The rescale factor v_j corrects this local magnitude drift, and alternating it with ternary reassignment allows the scale and discrete ternary codes to adapt to each other. Given the current ternary assignment, we estimate v_j from the non-zero ternary entries:

$$v_j = \mathrm{mean}_{i:T_{ij} \neq 0} \left(\frac{|W_{ij} - \beta_i|}{\alpha_i} \right). \tag{5}$$

We then reassign each weight to the nearest centered ternary value in $\{\beta_i - v_j\alpha_i, \beta_i, \beta_i + v_j\alpha_i\}$, and repeat this rescale-ternary update for a small number of iterations. This refinement introduces only one scalar per column, but makes the ternary base better adapt to the column-wise distribution after error propagation. During inference, v_j forms a length-d_{in} vector and can be applied by element-wise scaling of the input, leaving the computation scheme unchanged.

3.4 Error Decay

QTEA follows GPTQ-style column-by-column quantization. For a weight block W, after quantizing the i-th column from w_i to q_i, GPTQ computes

$$e_i = \frac{w_i - q_i}{[H^{-1}]_{ii}}, \tag{6}$$

and propagates the error to the remaining columns:

$$W_{:,i+1:} \leftarrow W_{:,i+1:} - e_i [H^{-1}]_{i,i+1:}. \tag{7}$$

This allows later columns to absorb errors from earlier quantized columns, but also creates an order-dependent imbalance. Early columns can spread their errors over a long suffix, whereas late columns can only update a few remaining columns. This imbalance is especially harmful in ultra-low-bit quantization, where ternary weights have limited capacity to absorb propagated errors. Thus, using the same propagation strength throughout the block can make late-stage compensation overly aggressive, even with column reordering.

To address this, we introduce *error decay*, a capacity-aware damping mechanism that attenuates GPTQ propagation according to the column position. For the i-th column in a block of size B, we apply

$$\gamma_i = \exp\left(-\lambda_{\text{decay}} \cdot \frac{[H^{-1}]_{ii}}{\text{mean}_k [H^{-1}]_{kk}} \cdot \frac{i}{B}\right), \tag{8}$$

where λ_{decay} controls the decay strength. The term i/B reflects the shrinking capacity of the remaining suffix, while the normalized inverse-Hessian diagonal makes the damping curvature-aware and comparable across layers. The GPTQ update is then modified as

$$W_{:,i+1:} \leftarrow W_{:,i+1:} - \gamma_i e_i [H^{-1}]_{i,i+1:}. \tag{9}$$

When $\lambda_{\text{decay}} = 0$, this reduces to the original GPTQ update. With positive λ_{decay}, propagation is gradually weakened along the quantization order, reducing late-stage over-compensation and stabilizing ultra-low-bit quantization.

3.5 Efficient Ternary Computation

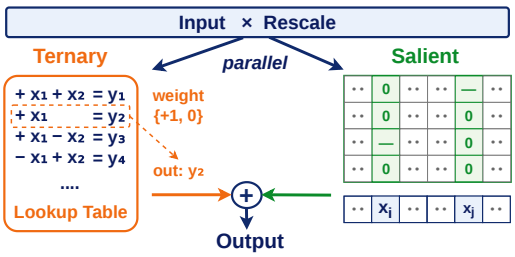

Figure 5: LUT-based computation kernel for QTEA.

To translate QTEA into inference speedup, we design a CUDA kernel for efficient ternary GEMV. For each quantization group, the output is

$$y_r^{(g)} = \alpha_{r,g} \sum_{j \in g} T_{rj} v_j x_j + \beta_{r,g} \sum_{j \in g} x_j, \tag{10}$$

where $T_{rj} \in \{-1, 0, +1\}$, v_j is the column-wise rescale factor, and $\alpha_{r,g}$, $\beta_{r,g}$ are the group-wise scale and bias.

Instead of explicitly decoding ternary weights and performing multiply-accumulate operations, we adopt lookup-table-based computation (Park et al., 2024; Wang et al., 2025). Since $3^5 < 2^8$, we pack five ternary values into one byte and precompute dot products between all ternary patterns and the scaled activations $v_j x_j$. During GEMV, each packed code indexes the LUT, turning most ternary computation into lightweight lookup and accumulation. We further exploit sign symmetry: the dot product of pattern $-p$ is the negative of that of p, so only half of the LUT entries need to be explicitly constructed.

The column-semi sparse residual path is accumulated in parallel:

$$y_r = \sum_g y_r^{(g)} + \sum_{j \in \mathcal{S}} R_{rj} x_j, \tag{11}$$

where R is the sparse residual matrix and $\mathcal{S}$ denotes selected salient columns. Because the residual path is sparse, it adds limited overhead while compensating the most important ternary errors. Overall, the kernel avoids full dequantization, performs most computation through LUT access, and accumulates sparse residuals alongside the ternary output.

4 Experiments

4.1 Experimental Setup

Setting. Quantization parameters are optimized using 256 calibration samples drawn from WikiText-2(Merity et al., 2016) training subset. We maintain an FP8 (E4M3) salient weight of 1.25% of the total weight, achieved through 5% column selection and 1:4 semi-sparse residuals. λ_{decay} is set to 1, and rescale refinement is iterated twice. Kernel and end-to-end speeds have been optimized and tested on the NVIDIA H200. All models are quantized with group size of 128. The resulting storage cost is 1.7 bits per weight (bpw): 1.6 bpw for ternary weights plus $5\% \times 8bit/4 = 0.1$ bpw for the salient weights. ***Baseline.*** We consider well-known methods such as GPTQ(Frantar et al., 2023) and AWQ(Lin et al., 2024), as well as methods optimized specifically for ultra-low-bit quantization including Slim-LLM (Huang et al., 2025), PB-LLM(Yuan et al., 2024) and PT2-LLM(Yan et al., 2026). ***Metrics.*** Quantization quality is then measured as perplexity on the WikiText-2 test

Table 1: Perplexity and Zero-shot accuracy for low-bit PTQ methods across Llama and Qwen3-Base backbones. All quantized models use block size 128. Best sub-2-bit result per row group is in **bold**; second-best is underlined. Baseline results are taken from PT^2-LLM (Yan et al., 2026).

Model	Method	#W	WikiText2(↓)	C4(↓)	PiQA	Arc E	Arc C	Hella.	Wino.	OBQA	BoolQ	Avg(↑)
Q-0.6B	FP16	16	12.70	17.10	70.00	65.60	33.90	41.10	58.50	24.60	69.70	51.91
	AWQ	2	6.42e7	9.85e7	53.20	24.80	21.90	25.70	50.80	5.00	46.30	32.53
	GPTQ	2	5.37e2	3.57e3	51.58	24.96	19.62	25.34	49.96	15.20	42.32	32.71
	Slim-LLM	2	282.12	1190.46	54.03	27.95	21.25	26.03	49.64	**16.80**	**39.82**	33.65
	PB-LLM	1.7	1.25e6	1.68e6	51.14	24.75	11.25	25.29	51.10	12.60	34.80	30.13
	PT^2-LLM	1.6	145.57	617.11	53.21	**30.77**	18.77	26.33	51.10	13.80	37.83	33.13
	Ours	**1.7**	**63.58**	**172.67**	**55.17**	27.60	**21.80**	**27.60**	**51.54**	16.30	38.92	**34.13**
Q-1.7B	FP16	16	9.39	13.40	75.70	73.20	41.50	49.20	64.20	30.20	79.20	59.03
	AWQ	2	1.13e7	1.02e7	52.40	24.00	**21.70**	25.60	49.40	10.00	48.50	33.09
	GPTQ	2	101.69	375.74	52.12	27.23	18.94	26.29	**52.96**	**17.80**	44.98	34.33
	Slim-LLM	2	70.14	280.59	54.20	31.69	19.45	25.09	50.43	12.60	35.66	32.73
	PB-LLM	1.7	6.85e5	4.41e5	52.23	24.58	20.99	25.52	50.36	13.60	41.87	32.74
	PT^2-LLM	1.6	66.67	338.19	54.68	31.90	20.22	26.79	51.62	11.20	38.81	33.60
	Ours	**1.7**	**37.13**	**112.48**	**57.83**	**39.94**	21.58	**28.84**	51.54	13.60	**58.69**	**38.86**
Q-4B	FP16	16	7.90	11.60	78.10	79.00	48.40	54.60	70.00	31.40	82.90	63.49
	AWQ	2	7.53e6	5.94e6	53.40	24.70	22.20	25.80	47.50	11.10	46.80	33.07
	GPTQ	2	65.17	193.61	52.61	27.53	20.73	27.38	51.46	17.00	45.50	34.60
	Slim-LLM	2	**16.22**	**36.93**	52.57	28.78	22.61	25.38	**55.09**	**20.00**	55.87	37.19
	PB-LLM	1.7	5.71e5	7.48e5	52.12	25.00	21.33	25.80	48.46	12.80	40.15	32.24
	PT^2-LLM	1.6	37.54	153.78	54.62	36.78	19.71	27.65	53.67	13.40	55.72	37.36
	Ours	**1.7**	20.04	53.98	**62.08**	**55.35**	**26.62**	**33.99**	53.99	17.00	**62.26**	**44.47**
Q-8B	FP16	16	6.99	10.40	79.30	82.10	52.60	58.90	72.10	32.80	82.90	65.81
	AWQ	2	1.66e7	1.31e7	52.60	26.60	22.60	25.50	50.00	12.00	44.60	33.41
	GPTQ	2	106.41	347.02	57.40	30.60	20.50	33.40	52.50	16.40	48.70	37.07
	Slim-LLM	2	37.60	143.57	50.59	49.49	17.63	32.96	54.40	16.20	53.30	39.22
	PB-LLM	1.7	1.91e4	2.26e4	53.05	26.56	20.73	25.73	49.17	14.60	40.31	32.88
	PT^2-LLM	1.6	31.82	138.84	56.64	40.53	18.86	30.65	55.09	14.20	58.04	39.14
	Ours	**1.7**	**15.66**	**38.72**	**64.25**	**59.30**	**28.07**	**36.08**	**58.48**	**21.40**	**62.23**	**47.12**
Q-14B	FP16	16	6.38	9.68	80.58	83.46	55.80	61.85	74.19	35.00	86.76	68.23
	AWQ	2	2.68e7	2.18e7	53.00	24.60	23.00	25.30	50.70	20.00	46.20	34.69
	GPTQ	2	37.90	74.50	56.31	34.64	20.65	33.30	52.72	17.20	46.33	37.31
	Slim-LLM	2	22.85	68.38	61.83	52.54	29.35	31.52	52.04	20.40	61.20	44.13
	PB-LLM	1.7	2.89e4	2.44e4	54.08	25.93	20.73	25.76	47.99	15.00	38.04	32.50
	PT^2-LLM	1.6	16.48	68.13	62.95	53.03	23.63	33.65	59.75	20.60	62.17	45.11
	Ours	**1.7**	**11.78**	**26.14**	**68.93**	**67.42**	**34.39**	**41.17**	**64.72**	**26.00**	**65.93**	**52.65**
L3-8B	FP16	16	6.14	9.45	79.54	80.13	50.34	60.13	73.40	34.60	81.01	65.59
	AWQ	2	1.70e5	2.10e5	52.72	24.16	21.50	25.58	49.33	14.60	62.17	35.72
	GPTQ	2	1480.43	394.74	52.12	25.72	**21.59**	26.72	49.17	13.60	44.16	33.30
	Slim-LLM	2	38.21	390.02	55.77	32.15	19.11	27.83	48.78	13.20	44.83	34.52
	PB-LLM	1.7	73.08	104.15	56.64	33.08	17.15	27.98	51.07	12.40	55.44	36.25
	PT^2-LLM	1.6	32.19	129.83	56.86	34.22	18.43	30.36	53.28	13.80	57.58	37.79
	Ours	**1.7**	**24.09**	**66.45**	**59.63**	**39.52**	18.60	**30.40**	**53.83**	**17.00**	**63.03**	**40.29**

split and C4(Allen Institute for AI). We evaluate zero-shot accuracy on seven downstream benchmarks: ARC-Easy&Challenge(Clark et al., 2018), HellaSwag(Zellers et al., 2019), PIQA(Bisk et al., 2019), WinoGrande(Sakaguchi et al., 2019), OpenbookQA(Mihaylov et al., 2018), and BoolQ(Clark et al., 2019).

4.2 Perplexity and Zero-shot Accuracy Results

Table 1 reports perplexity and zero-shot accuracy across Qwen3-Base(Yang et al., 2025) and Llama3 backbones(Meta, 2024). Overall, QTEA achieves the best average zero-shot accuracy among sub-2-bit methods on all evaluated models while using only 1.7-bit weights. On Qwen3, the advantage becomes more pronounced as model size increases.

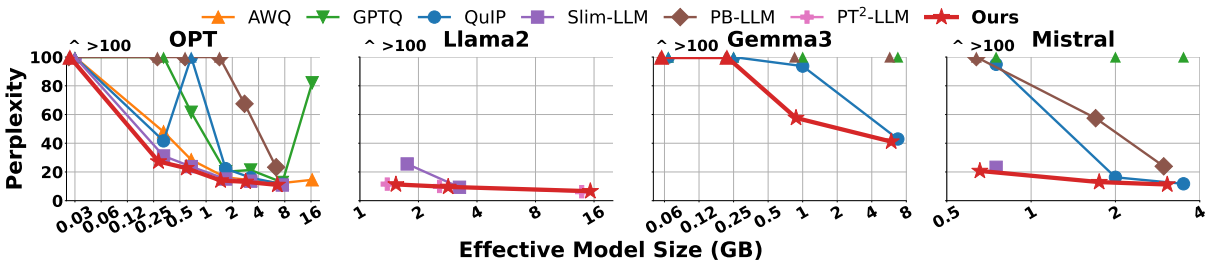

Figure 6: WikiText2 perplexity comparison across different model families and model sizes. Results with perplexity greater than 100 are omitted.

For Qwen3-14B, QTEA improves average downstream accuracy from 45.11% of the strongest baseline to 52.65%, a 16.7% relative gain. It also reduces WikiText2 perplexity from 16.48 to 11.78 and C4 perplexity from 68.13 to 26.14, corresponding to 1.40× and 2.61× reductions. Across smaller Qwen3 models, QTEA consistently remains the strongest sub-2-bit method, indicating favorable scaling behavior.

QTEA also generalizes to the Llama3 backbone. On Llama3-8B, it improves average zero-shot accuracy from 37.79% to 40.29%, giving a 6.6% relative gain over the best competing sub-2-bit method. Meanwhile, it lowers WikiText2 and C4 perplexity by 1.34× and 1.95×, respectively.

4.3 Perplexity vs. Model Size

We further study how low-bit PTQ methods scale with model size. Figures 1 and 6 report WikiText2 perplexity across Qwen3, OPT (Zhang et al., 2022), Llama2 (Touvron et al., 2023), Gemma3 (Team et al., 2025), and Mistral (Liu et al., 2026). Overall, QTEA shows stable scaling behavior and consistently achieves strong perplexity under a sub-2-bit weight budget.

On Qwen3, QTEA obtains the best perplexity among sub-2-bit methods from 0.6B to 14B parameters. Its perplexity decreases smoothly from 63.58 on Qwen3-0.6B to 11.78 on Qwen3-14B, reducing perplexity over PT^2-LLM by up to 2.29×. Across OPT, Gemma3, Llama2, and Mistral, QTEA remains the most stable method and stays competitive with the strongest ternary PTQ baseline (see Fig. 6). QTEA slightly outperforms PT^2-LLM on Llama2-7B and remains competitive on larger models. These results indicate that the proposed ternary base and sparse residual compensation generalize across model scales and architectures.

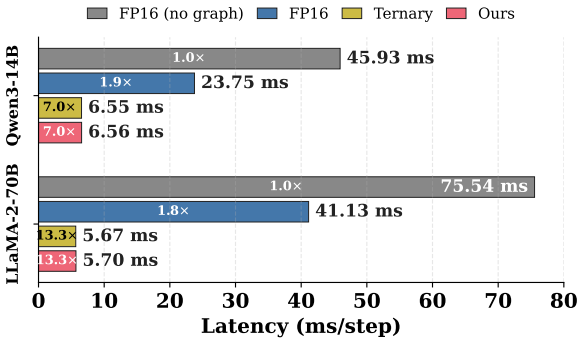

Figure 7: End-to-end per-token latency on GPU with CUDA Graph.

Shape	Latency		Energy	
	FP16	Ours	FP16	Ours
1×5120 · 5120×5120	1008.25	269.96	72.39	22.28
1×17408 · 17408×5120	3428.04	917.86	246.11	75.73
1×4096 · 4096×4096	645.28	183.20	46.33	14.47
1×4096 · 4096×12288	1935.83	445.34	139.00	41.35

Table 2: Latency (μs) and energy (μJ) comparison between FP16 dense matrix multiplication and our method on customized hardware. See Appendix A for details.

4.4 Efficiency

Figure 7 reports end-to-end generation latency per token on GPU. With CUDA Graph, QTEA reduces FP16 latency from 23.75 to 6.56 ms/token on Qwen3-14B, and from 41.13 to 5.70 ms/token on Llama2-70B, achieving 3.62× and 7.22× speedups, respectively. Compared with FP16 inference without CUDA Graph, the speedups reach 7.0× and 13.3×. QTEA also matches the latency of the ternary-only kernel, showing that the semi-sparse residual path adds negligible overhead.

We further evaluate QTEA on customized hardware using a TSMC 22nm standard-cell library and memory power reference tables. As shown in Table 2, QTEA achieves an average 3.83× latency speedup over FP16 dense matrix multiplication and reduces energy consumption by 69.4% on average. It demonstrates QTEA's potential for extreme effi-

ciency optimization. More hardware-level details are provided in Appendix A.

4.5 Ablation Study

4.5.1 Salient Metrics

Table 3: Ablation of salient-column scoring metrics.

Scoring Metric	Qwen3-14B		Llama3-8B	
	Avg ZS	Δ	Avg ZS	Δ
$\max(W^2 \cdot H_{diag}^2)$	52.65	–	40.29	–
$\mathrm{mean}(W^2 \cdot H_{diag}^2)$	50.18	−2.47	35.63	−4.66
$\max\lvert W\rvert$	49.53	−3.12	33.77	−6.52

As shown in Table 3, $\max(W^2 \cdot H_{diag}^2)$ achieves the best average zero-shot accuracy on both Qwen3-14B and Llama3-8B. This suggests that preserving LLM capability depends more on protecting extreme high-impact values than on selecting columns with large average importance.

4.5.2 Salient Sparse Pattern

Table 4: Ablation of salient-residual structures.

Salient Structure	Qwen3-14B	Llama3-8B
Pure ternary	34.73	27.80
Column 1.5%	50.90	33.61
Column 5%	52.96	40.58
Global 1.5%	**53.56**	**40.86**
Column 2.5% + 1:2 Sparse	51.00	33.49
Column 10% + 1:8 Sparse	51.72	36.29
Ours: Column 5% + 1:4 Sparse	52.65	40.29

Table 4 compares different salient-residual structures. Global 1.5% serves as a hardware-unfriendly unstructured upper bound. QTEA remains close to this, trailing by only 0.91 and 0.57 points on Qwen3-14B and Llama3-8B, while using a hardware-friendly column 5% + 1:4 semi-sparse structure. This trades some accuracy, 0.31pp and 0.29pp on Qwen3-14B and Llama3-8B, for 4× lower residual storage, showing that column-semi sparse residuals offer the benefits of unstructured salient weights while preserving GPU-friendly access.

4.5.3 Rescale Refinement

Table 5 shows that column-wise rescale refinement is most effective when applied inside GPTQ column-by-column quantization. Applying rescale refinement after GPTQ severely degrades accuracy, while refining rescale during quantization preserves the GPTQ error-compensation trajectory. The drop from removing ternary reassignment indicates that rescale factor and the ternary assignments should be optimized jointly.

Table 5: Ablation of column-wise rescale refinement.

Variant	Qwen3-14B	Llama3-8B
Full QTEA	**52.65**	**40.29**
Rescale before Column-wise Quant.	52.44	39.84
Rescale after Column-wise Quant.	40.28	37.32
No Rescale	50.63	37.49
No ternary update iteration	51.75	38.99

4.5.4 Error Decay

Table 6: Ablation of error decay.

Variant	Qwen3-14B		Llama3-8B	
	Avg ZS	Δ	Avg ZS	Δ
Full QTEA	52.65	–	40.29	–
w/o Error Decay	52.19	−0.46	37.55	−2.74

Table 6 shows that disabling error decay consistently reduces accuracy, especially on Llama3-8B, confirming that position-dependent error propagation helps stabilize ultra-low-bit quantization.

5 Conclusion

We present QTEA, a sub-2-bit weight-only PTQ framework for large language models. QTEA quantizes all weights into a compact ternary base and uses column-semi sparse FP8 residuals to compensate the most harmful quantization errors, rather than bypassing quantization with high-precision salient weights. It further improves the ternary representation with column-wise rescale refinement inside GPTQ-style quantization, and stabilizes order-dependent error propagation through error decay.

Experiments across multiple model families show that QTEA achieves a strong accuracy–compression trade-off. On Qwen3-14B and Llama3-8B, it substantially improves zero-shot accuracy and reduces perplexity over prior sub-2-bit PTQ methods. Ablations validate the effectiveness of the salient scoring metric, column-semi sparse residuals, rescale refinement, and error decay. Finally, our LUT-based kernel translates the compact representation into practical acceleration, achieving significant GPU speedups and promising latency and energy reductions on customized hardware. These results demonstrate that QTEA is a practical path toward accurate and efficient sub-2-bit LLM deployment.

Limitations

This work focuses on weight-only post-training quantization. We do not quantize activations or KV cache, and therefore the reported compression and speedups mainly target the weight-memory bottleneck in low-batch autoregressive decoding. The benefit may be smaller in settings where activation computation, KV-cache traffic, or other system overheads dominate runtime.

Although QTEA is evaluated across several model families, including Qwen3, Llama, OPT, Gemma3, and Mistral, our experiments are still limited to decoder-only language models. Further evaluation is needed on instruction-tuned models, mixture-of-experts models, multimodal models, and longer-context workloads. In addition, the current hyperparameters, such as the salient-column ratio, 1:4 residual sparsity, group size, and calibration setting, are kept fixed in most experiments; adapting these choices to different architectures may further improve the accuracy–efficiency tradeoff.

QTEA also relies on specialized kernels to fully realize its inference benefits. While the column-semi sparse residual structure is more hardware-friendly than unstructured sparsity, its practical speedup still depends on the target hardware, memory hierarchy, kernel implementation, and integration with inference runtimes such as CUDA Graph. Finally, our customized-hardware results are architecture-level estimates rather than complete post-layout measurements. They do not fully model clock-tree power, routing parasitics, post-layout wire effects, or workload-specific switching activity, and should therefore be interpreted as evidence of architectural potential rather than final silicon measurements.

QTEA is a model compression and inference-efficiency technique, and does not introduce new training data or new model capabilities by itself. However, by reducing the memory footprint and serving cost of LLMs, it may make large-scale deployment more accessible, which can amplify both beneficial applications and existing risks of LLM misuse, such as spam generation, misinformation, or automated harmful content generation. In addition, ultra-low-bit quantization may change model behavior in ways that are not fully captured by perplexity and standard zero-shot benchmarks, including calibration, robustness, bias, or safety-alignment properties. Therefore, quantized models should be re-evaluated on task-specific and safety-critical benchmarks before deployment, especially in high-stakes or user-facing applications.

Acknowledgments

We use publicly available models and datasets only for research evaluation and follow their original licenses, terms of use, and intended-use restrictions. The evaluated model families, including Qwen3, Llama, OPT, Gemma3, and Mistral, are used for quantization and inference-efficiency evaluation. The benchmark datasets, including WikiText-2, C4, ARC, HellaSwag, PIQA, WinoGrande, OpenBookQA, and BoolQ, are used only for standard research benchmarking. We do not redistribute the original model weights or datasets. Any released code or quantized artifacts are intended for research on model compression and efficient inference, and will be distributed with a license that respects the access conditions and redistribution restrictions of the corresponding upstream artifacts.

We used AI assistants to support code implementation, and proofreading for grammar, clarity, and typographical errors. All technical ideas, experimental design, results, analyses, and final claims were developed, checked, and approved by the authors. The authors take full responsibility for the content of the paper.

References

Allen Institute for AI. C4. https://huggingface.co/datasets/allenai/c4. Hugging Face dataset, accessed May 15, 2026.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language. *Preprint*, arXiv:1911.11641.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip: 2-bit quantization of large language models with guarantees. *Preprint*, arXiv:2307.13304.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *Preprint*, arXiv:1905.10044.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers. *Preprint*, arXiv:2210.17323.

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. 2020. Sparse gpu kernels for deep learning. *Preprint*, arXiv:2006.10901.

Hao Gu, Lujun Li, Hao Wang, Lei Wang, Zheyu Wang, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Sirui Han, and Yike Guo. 2026. Btc-llm: Efficient sub-1-bit llm quantization via learnable transformation and binary codebook. *Preprint*, arXiv:2506.12040.

Yipin Guo, Yilin Lang, and Qinyuan Ren. 2024. Gptqt: Quantize large language models twice to push the efficiency. *Preprint*, arXiv:2407.02891.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Preprint*, arXiv:2102.00554.

Hong Huang, Decheng Wu, Qiangqiang Hu, Guanghua Yu, Jinhai Yang, Jianchen Zhu, Xue Liu, and Dapeng Wu. 2026. Sherry: Hardware-efficient 1.25-bit ternary quantization via fine-grained sparsification. *Preprint*, arXiv:2601.07892.

Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. 2024. Billm: pushing the limit of post-training quantization for llms. ICML'24. JMLR.org.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. 2025. Slim-llm: Salience-driven mixed-precision quantization for large language models. *Preprint*, arXiv:2405.14917.

Zhiteng Li, Xianglong Yan, Tianao Zhang, Haotong Qin, Dong Xie, Jiang Tian, zhongchao shi, Linghe Kong, Yulun Zhang, and Xiaokang Yang. 2026. Arb-llm: Alternating refined binarizations for large language models. *Preprint*, arXiv:2410.03129.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for llm compression and acceleration. In *MLSys*.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, Alexandre Sablayrolles, Amélie Héliou, Amos You, Andy Ehrenberg, Andy Lo, Anton Eliseev, Antonia Calvi, Avinash Sooriyarachchi, Baptiste Bout, and 101 others. 2026. Ministral 3. *Preprint*, arXiv:2601.08584.

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

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

Meta. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *Preprint*, arXiv:1809.02789.

Gunho Park, Baeseong park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2024. LUT-GEMM: Quantized matrix multiplication based on LUTs for efficient inference in large-scale generative language models. In *The Twelfth International Conference on Learning Representations*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *Preprint*, arXiv:1907.10641.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2024. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *Preprint*, arXiv:2402.04396.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. 2023. Bitnet: Scaling 1-bit transformers for large language models. *Preprint*, arXiv:2310.11453.

Jinheng Wang, Hansong Zhou, Ting Song, Shijie Cao, Yan Xia, Ting Cao, Jianyu Wei, Shuming Ma, Hongyu Wang, and Furu Wei. 2025. Bitnet.cpp: Efficient edge inference for ternary llms. *Preprint*, arXiv:2502.11880.

Xianglong Yan, ChengZhu Bao, Zhiteng Li, Tianao Zhang, Kaicheng Yang, Haotong Qin, Ruobing Xie, Xingwu Sun, and Yulun Zhang. 2026. PT^2-LLM: Post-training ternarization for large language models. In *The Fourteenth International Conference on Learning Representations*.

Xianglong Yan, Tianao Zhang, Zhiteng Li, Haotong Qin, and Yulun Zhang. 2025. Progressive binarization with semi-structured pruning for llms. *Preprint*, arXiv:2502.01705.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Haoran You, Yipin Guo, Yichao Fu, Wei Zhou, Huihong Shi, Xiaofan Zhang, Souvik Kundu, Amir Yazdanbakhsh, and Yingyan Celine Lin. 2024. Shiftaddllm: Accelerating pretrained llms via post-training multiplication-less reparameterization. *Preprint*, arXiv:2406.05981.

Mengxia Yu, De Wang, Qi Shan, Colorado J Reed, and Alvin Wan. 2025. The super weight in large language models. *Preprint*, arXiv:2411.07191.

Zhihang Yuan, Yuzhang Shang, and Zhen Dong. 2024. PB-LLM: Partially binarized large language models. In *The Twelfth International Conference on Learning Representations*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models. *Preprint*, arXiv:2205.01068.

Jiaqi Zhao, Miao Zhang, Ming Wang, Yuzhang Shang, Kaihao Zhang, Weili Guan, Yaowei Wang, and Min Zhang. 2025. Ptq1.61: Push the real limit of extremely low-bit post-training quantization methods for large language models. *Preprint*, arXiv:2502.13179.

A Hardware Evaluation Details

This appendix provides the hardware-level evaluation setup for the proposed ternary-LUT computation engine. The evaluated workload is a matrix-vector multiplication,

$$y = xW, \tag{12}$$

where the input vector has shape $[1, K]$, the weight matrix has shape $[K, N]$, and the output has shape $[1, N]$. We compare our ternary-LUT engine against a dense FP16 matrix computation baseline.

Hardware assumptions. All arithmetic units are evaluated under a 1GHz target frequency. The dense baseline uses 26 parallel FP16 MAC lanes, where each lane computes one FP16 multiply-accumulate operation. For our ternary engine, each 128-column group is partitioned into 26 five-column packs with padding size 2. For each five-column pack, we build a half-LUT for all ternary-weighted combinations and use packed ternary weights as LUT addresses. The pre-compute stage, which computes the scaled activation $x_i v_i$, is conservatively modeled using one FP16 MAC lane. The LUT construction stage uses 26 FP16 ternary MAC, one for each pack.

Compute logic cost. The dense 26-lane baseline has a compute-logic area of 35,573.98μ^2. In comparison, the ternary compute logic consists of one pre-compute MAC and 26 ternary MAC, resulting in an area of 28,454.25μ^2. Thus, before considering SRAM accesses, the ternary compute logic uses about 80% of the dense baseline area.

SRAM modeling. SRAM capacity and SRAM read/write energy are modeled separately from compute-logic area. For example, the ternary engine uses 26 128×16 SRAM macros for the half-LUTs therefore power consumption is estimated with 128×16 SRAM array reference table at certain operating condition.

Memory traffic. For each row-group, the dense FP16 baseline reads 128 FP16 weights. In contrast, the ternary engine reads 26 packed 8-bit weight addresses, which are equivalent to 13 16-bit SRAM reads. Therefore, the ternary engine replaces repeated dense FP16 weight reads with compact packed-weight reads and LUT accesses.

Latency model. The ternary engine processes the matrix in 128-column groups. For each group, the

latency includes precomputation, LUT construction, and lookup/accumulation. The latency is estimated as

$$T_{\text{ternary}} = \left\lceil \frac{K}{128} \right\rceil (652 + 977 + N), \qquad (13)$$

where 652 cycles correspond to precomputation, 977 cycles correspond to LUT construction, and the lookup/accumulation stage processes one output row per cycle with 26 parallel LUT channels. The dense 26-lane FP16 baseline latency is estimated as

$$T_{\text{dense}} = \frac{KN}{26}. \qquad (14)$$

End-to-end hardware results. Table 2 summarizes the hardware-level latency and energy estimates. Across the four evaluated matrix shapes, our ternary-LUT engine achieves an average latency speedup of 3.83× over the dense FP16 baseline. The average energy reduction is 69.4%. The main benefit comes from replacing dense FP16 MAC operations and repeated FP16 weight reads with LUT-based accumulation over compact ternary weights. Although the ternary engine introduces LUT scratch SRAM traffic, the total energy remains substantially lower because the latency and persistent weight-read traffic are both greatly reduced.

Storage comparison. The ternary representation also reduces persistent weight storage. For each 128-column group, five ternary weights are packed into one 8-bit address, resulting in approximately 1.6 bits per weight before scale and offset metadata. In the evaluated shapes, the packed ternary weight storage is about 9.85× smaller than dense FP16 weight storage.

Limitations of the estimate. The reported numbers should be interpreted as architecture-level estimates. Compute power is based on synthesized arithmetic units, while SRAM capacity and read/write energy are estimated separately from SRAM macro data. Clock-tree power, routing parasitics, post-layout wire effects, and workload-specific switching activity are not fully modeled. Therefore, the results are mainly intended to show the architectural trend: the ternary-LUT design trades a small amount of scratch SRAM and LUT construction traffic for much lower FP16 compute complexity, reduced persistent weight traffic, shorter latency, and lower total energy.

B Model size.

Table 7: Model size comparison on Llama2-7B.

Method	Size (GB)	Reduction	Perplexity
FP16	13.48	–	–
GPTQ	2.19	6.16×	52.22
Slim-LLM	2.30	5.86×	15.84
PB-LLM	2.91	4.63×	66.41
BiLLM	2.93	4.60×	32.48
ARB-LLM	3.23	4.17×	16.44
PT^2-LLM	**1.88**	**7.17**×	11.56
Ours	2.12	6.35×	**11.02**

Table 7 compares the total model storage of QTEA with prior low-bit quantization methods on Llama2-7B, with baseline numbers taken from PT^2-LLM (Yan et al., 2026). QTEA uses approximately 1.7 bpw payload plus 0.3 bpw metadata, embeddings, LayerNorm and the LM head. Under this setting, QTEA reduces the Llama2-7B checkpoint size from 13.48 GB in FP16 to 2.12 GB, corresponding to a 6.35× storage reduction. Compared with 2-bit methods such as GPTQ and Slim-LLM, QTEA achieves a smaller model size; compared with PT^2-LLM, it uses a modestly larger payload but provides substantially better accuracy, e.g., around 7–8 percentage points higher zero-shot accuracy on Qwen3 models. Following prior comparisons, embeddings, the LM head, and LayerNorm parameters are kept in FP16.