WINDQuant: Weight-Informed Neural Decision-Making for Global Mixed-Precision LLM Quantization

Anonymous ACL submission

Abstract

Quantization is an effective approach to reduce the memory footprint and inference cost of large language models (LLMs), yet maintaining performance in the ultra-low-bit regime remains challenging. Existing post-training methods often suffer from severe accuracy degradation, while quantization-aware training requires costly retraining and additional resources. Moreover, most mixed-precision strategies rely on coarse-grained or heuristic sensitivity analysis that overlooks fine-grained variations within weight matrices. We propose WINDQuant, a reinforcement-learning-based allocation controller for ultra-low-bit LLM quantization. Rather than introducing another low-level quantization operator, WINDQuant learns how to assign bit-widths and quantization treatments to fine-grained column chunks under a global storage budget. By operating at the column-chunk level, WINDQuant enables flexible and fine-grained precision assignment within layers under a global target bit-width. The implementation combines PPO with activation-aware calibration, lightweight per-unit quantizer fitting, and explicit effective-bit accounting of the learned mixed-precision plan. Experiments on LLaMA models demonstrate that WINDQuant achieves competitive performance in ultra-low-bit settings while reducing optimization overhead relative to retraining-based approaches, highlighting reinforcement learning as a practical controller for adaptive mixed-precision quantization.

1 Introduction

Large Language Models (LLMs) have achieved remarkable performance across reasoning, language understanding, and generation tasks (Brown et al., 2020; Chowdhery et al., 2023). However, this progress comes at the cost of massive computational and memory requirements. Modern frontier models contain hundreds of billions to trillions of parameters, requiring terabytes of VRAM for full-precision deployment (Fedus et al., 2022). Such resource demands make direct deployment on edge devices or resource-constrained environments infeasible. Quantization has emerged as one of the most promising solutions to reduce inference cost by representing model weights and activations with lower-precision data types (Dettmers et al., 2022).

Despite its success, low-bit quantization remains highly challenging, especially in the sub-3-bit regime (Banner et al., 2019). Post-Training Quantization (PTQ) methods are efficient and fast but tend to suffer severe degradation at ultra-low precision (Nagel et al., 2019). Quantization-Aware Training (QAT) methods improve robustness but require expensive retraining and often rely on knowledge distillation, significantly increasing computational overhead (Jacob et al., 2018). Moreover, most existing approaches apply uniform or coarse-grained quantization strategies, overlooking the fact that not all weights contribute equally to model performance (Dong et al., 2019).

Recent empirical findings indicate that a small fraction of salient weights disproportionately affects model accuracy (Lin et al., 2024). Furthermore, sensitivity exhibits structured locality across layers and channels, suggesting that different substructures within a layer respond differently to quantization (Dong et al., 2019). This observation motivates a more fine-grained and adaptive allocation strategy. Instead of treating each tensor, group, or unit as an independent local quantization problem, we view ultra-low-bit mixed-precision quantization as a global budgeted allocation problem, where assigning more precision to one unit necessarily reduces the budget available to others.

In this work, we introduce WINDQuant (Weight-Informed Decision-making for Reinforcement Learning in Quantization), a reinforcement learning framework for mixed-precision quantization. We define a unit as a column-level chunk within a

layer and formulate quantization as a finite-horizon sequential decision problem with a global target bit budget. At each step, the agent observes statistical, structural, and activation-aware features of a unit and selects one of seven actions: skip or quantize at 8, 4, 3, 2, 1.58, or 1 bit. The training objective combines PPO (Schulman et al., 2017) with budget-aware action masking, dense step-level shaping, and an episode-end quality penalty computed from a fast perplexity proxy. This distinction is central to our formulation. Conventional PTQ and QAT methods primarily specify how a given tensor, group, or block should be quantized once the target precision is fixed. WINDQuant instead addresses the complementary allocation problem: given a pretrained model decomposed into thousands of sub-layer units, the policy decides which units should be kept at higher precision, aggressively quantized, or assigned ternary/binary operators under a global bit budget.

Unlike prior RL-based approaches that operate at layer granularity (Wang et al., 2019), WINDQuant performs fine-grained unit-level allocation, enabling more expressive and adaptive mixed-precision policies. Our framework integrates activation-aware protection mechanisms and supports multiple quantization operators spanning 1-bit to 8-bit regimes. Experiments on LLaMA models demonstrate that WINDQuant achieves competitive performance in ultra-low-bit settings while significantly reducing computational and memory overhead compared to training-based approaches.

Overall, the contributions of our research can be articulated as follows:

- We propose **WINDQuant**, a reinforcement learning–based framework that reformulates fine-grained mixed-precision quantization for large language models as a sequential decision-making problem, enabling adaptive bit-width allocation beyond static heuristic sensitivity ranking.

- We demonstrate the effectiveness of the proposed approach across a range of modern large language models, including LLaMA3 (1B, 3B, 8B, 70B), showing its generality and applicability to contemporary architectures.

- We show that WINDQuant reaches competitive ultra-low-bit performance on LLaMA models up to 70B parameters while avoiding full model retraining, requiring only lightweight policy optimization and local per-unit quantizer fitting.

2 Related Works

2.1 Quantization of Large Language Models

Quantization methods for LLMs are commonly divided into post-training quantization (PTQ) and quantization-aware training (QAT) (Lang et al., 2024; Li et al., 2024). PTQ approaches compress pretrained models without additional training by applying calibration and reconstruction techniques (Lin et al., 2024; Xiao et al., 2023). Their primary strength lies in efficiency, but performance degrades significantly in ultra-low-bit settings, where fixed or heuristic precision assignments cannot adequately compensate for accumulated quantization error. QAT methods address this by incorporating quantization effects during training (Liu et al., 2024a), yielding stronger accuracy retention at the cost of substantial retraining overhead. Despite their complementary strengths, both paradigms typically treat precision allocation as a static optimization problem, motivating adaptive approaches that learn allocation policies automatically.

2.2 Reinforcement Learning for Neural Network Compression

Reinforcement learning has emerged as a promising framework for automated model compression, framing design decisions as sequential optimization problems. HAQ (Wang et al., 2019) pioneered the use of deep deterministic policy gradient (DDPG) to determine layer-wise bit-widths for hardware-aware mixed-precision quantization without manual heuristics. ReLeQ (Elthakeb et al., 2020) applied Q-learning to learn heterogeneous bit-width assignments, demonstrating that RL-based policies can approach near-lossless accuracy across different architectures. Beyond quantization, AMC (He et al., 2018) uses DDPG to automate channel pruning ratios, and N2N learning (Ashok et al., 2017) applies policy gradient methods to jointly optimize architecture and compression decisions. These works share a common insight: compression decisions involve complex, interdependent trade-offs that are naturally suited to sequential decision-making formulations.

However, existing RL-based quantization methods face several limitations when applied to large

language models. First, operating at layer granularity leads to coarse allocation that ignores within-layer sensitivity variations. Second, reward evaluation typically requires repeated full-model inference, creating computational bottlenecks at the billion-parameter scale. Third, the action spaces and state representations used in prior work were designed for smaller vision models and do not account for the structural properties of transformer-based LLMs, such as the distinct sensitivity profiles of attention and MLP components.

These limitations motivate our approach: a fine-grained, sub-layer RL formulation with efficient reward evaluation, specifically designed for large-scale transformer quantization. Unlike HAQ and ReLeQ, which assign one bit-width per layer, WINDQuant operates at the column-chunk level, enabling more precise adaptation to within-layer sensitivity.

2.3 Mixed-Precision and Sensitivity-Aware Quantization

Empirical findings show that only a small fraction of salient weights dominate model accuracy, with sensitivity structured along channels rather than uniformly distributed (Lin et al., 2024; Dong et al., 2019). Recent elastic quantization operators–such as Learned Step-Size Quantization (LSQ) and Stretched Elastic Quantization (SEQ)–adaptively learn per-group quantization boundaries, outperforming fixed-scale methods at ultra-low precision (Liu et al., 2025). Most existing mixed-precision approaches apply coarse layer-level or mask-based heuristics that ignore fine-grained within-layer variation (Li et al., 2024). These operators serve as quantization primitives within our RL framework, enabling the agent to leverage optimized quantizers at each precision level.

3 Methodology

3.1 Problem Definition

Let $\mathcal{M}$ denote a pretrained model with N quantization units. Each weight matrix is decomposed into column-level chunks, referred to as *units*, motivated by empirical observations that weight importance exhibits channel-localized structure (Lin et al., 2024). Each unit i has n_i weights and is assigned an action a_i. Because the implementation allows skip decisions and INT8-protected weights, the realized storage cost is an effective bit count $B_i(a_i)$ rather than just a nominal action bit-width. We seek a policy π that satisfies a global storage budget:

$$\sum_{i=1}^{N} B_i(a_i) \leq B_{\text{target}} \tag{1}$$

while preserving model quality under a soft constraint:

$$\mathcal{L}(\mathcal{M}_Q; \mathcal{D}) - \mathcal{L}(\mathcal{M}; \mathcal{D}) \leq \varepsilon \tag{2}$$

where $\mathcal{M}_Q$ is the quantized model and $\mathcal{D}$ is the calibration corpus used during RL reward evaluation. Downstream reasoning benchmarks are used only after training. Rather than assigning bit-widths independently per layer using local heuristics, we formulate this as a sequential decision process where the policy considers global compression state and previously made decisions, enabling adaptive allocation under a global budget.

3.2 MDP Formulation

We cast the bit-width assignment problem as a finite-horizon MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$.

3.2.1 State Space

The state at step t encodes both local weight statistics and global compression context. Table 1 details the 15-dimensional state representation. Following prior RL-based compression methods (Elthakeb et al., 2020; Wang et al., 2019), we include distributional features of the current weight unit (mean, variance, sparsity, outlier fraction) to estimate quantization sensitivity. The state also includes episode-level signals such as progress, remaining bit budget, and urgency, allowing the agent to adapt its strategy as the compression budget becomes constrained. Calibration statistics derived from activation-aware analysis provide information about activation scales, which correlate with weight importance. Structural indicators help the policy distinguish between attention and MLP components, which exhibit different sensitivity to quantization.

In the implementation, budget-related state features are normalized by an FP32 reference total ($32 \sum_i n_i$) for numerical stability, even though realized storage may include FP16 skip actions and INT8-protected weights.

3.2.2 Action Space

The action space defines the set of discrete precision levels that the agent can assign to each quantization unit:

Dim.	Feature	Formula
0	mean	$\bar{W}$
1	std	σ_W
2	abs_mean	$\frac{1}{n}\sum_i \lvert W_i \rvert$
3	sparsity	$\frac{\lvert\{i:\lvert W_i\rvert<10^{-6}\}\rvert}{n}$
4	outlier_frac	$\frac{\lvert\{i:\lvert W_i\rvert>0.5\cdot P_{99}\}\rvert}{n}$
5	progress	$\frac{t}{N}$
6	bits_used_ratio	$\frac{B_{used}}{B_{FP32}}$
7	budget_remaining	$\frac{B_{target}-B_{used}}{B_{target}}$
8	urgency	$(1-\frac{B_r}{b_t n_r})_+$
9-11	calib_stats	$(\bar{s}, \sigma_s, \max(s))$
12	is_attention	$1[\text{“}attn\text{”} \in name]$
13	is_mlp	$1[\text{“}mlp\text{”} \in name]$
14	target_norm	$b_{target}/8$

Table 1: State feature definitions used in WINDQuant.

$$\mathcal{A} = \{16; 8; 4; 3; 2; 1.58; 1\} \tag{3}$$

The actions correspond to supported bit-widths of the quantization engine, ranging from full-precision (16-bit), conventional low-precision formats (8- and 4-bit) to ultra-low-bit representations (3-, 2-, and 1-bit). We additionally include the 1.58-bit format, which corresponds to ternary quantization schemes. By allowing the agent to choose among these discrete precision levels, the policy can adaptively allocate higher precision to sensitive units while aggressively compressing less critical components.

In practice, the categorical policy is combined with budget-aware action masking, and the selected action is further clamped by layer-specific minimum precisions for early layers, attention projections, and MLP/gate blocks.

Episode complexity. For LLaMA-3.1-8B with column chunk size G=256, each episode processes N≈3,584 units (32 layers × 7 linear projections per layer × 16 chunks per projection). Reward evaluation uses 64 Open-Platypus samples truncated to sequence length 256 and serves as a fast training proxy rather than a full downstream benchmark pass. In our saved 8B runs, a 100-episode training job completes in tens of hours on a single H200 GPU.

3.2.3 Transition and Discount

The transition is deterministic: given the current state and action, the environment applies the selected quantization to the current unit and advances to the next. The discount factor is set to $\gamma = 0.99$.

3.3 Reward Design

The mixed-precision allocation problem balances compression against proxy quality under a target average bit-width. The total episode return combines local per-step feedback with an episode-end proxy evaluation:

$$G = \sum_{t=1}^{T} \gamma^{T-t} r_{\text{step}}(t) + R_{\text{final}} \tag{4}$$

where $r_{\text{step}}(t)$ nudges the policy toward budget-feasible local decisions and R_{final} scores the realized mixed-precision plan. This hybrid structure mitigates the credit assignment problem without replacing the terminal objective.

Episode-end reward. The final reward balances compression efficiency against model quality through a weighted combination of objectives:

$$\mathcal{R} = \alpha G_c - P_{\text{qual}} - P_{\text{budget}} - P_{\text{ppl}} - P_{\text{skip}} + B_{\text{quality}} \tag{5}$$

where G_c measures normalized bit reduction relative to an FP32 reference and is capped to account for INT8 protection overhead; P_{qual} penalizes relative loss increase beyond a tolerance threshold ε=0.3, modulated by a Lagrange multiplier λ; P_{budget} penalizes violation of the target bit-width with a super-linear penalty; P_{ppl} imposes a piecewise penalty on the perplexity ratio $\rho = \text{PPL}_Q/\text{PPL}_0$ using soft and hard thresholds of 3 and 20; P_{skip} discourages excessive skipping beyond 10% of units; and B_{quality} provides an additional bonus when the perplexity ratio remains below 2.

Per-step reward. To provide dense temporal feedback, step-level rewards use lightweight shaping signals: skip actions incur a small penalty, allocations at or below the current target receive a positive signal proportional to saved bits, and over-target actions receive a mild negative reward. The dominant learning signal still comes from the episode-end reward.

Constraint enforcement. We handle the quality constraint (Eq. 2) via Lagrangian relaxation: a dual variable λ is updated by projected gradient ascent after each episode, increasing the quality penalty weight when degradation is excessive and relaxing it otherwise. The bit budget is not

enforced by the dual variable; instead, the implementation combines budget-aware action masks with an episode-end budget penalty. Additionally, we adopt a curriculum learning strategy where the target bit-width is progressively reduced (e.g., $3.0 \to 2.5 \to 2.0 \to 2.0$). This staged optimization stabilizes exploration and allows the policy to learn meaningful quantization patterns before aggressive compression is applied.

3.4 Policy Optimization

We optimize the precision-allocation policy using Proximal Policy Optimization (PPO) (Schulman et al., 2017), a stable first-order policy gradient method. The policy $\pi_\theta(a \mid s)$ is parameterized by a neural network that maps state representations to a categorical distribution over bit-width actions. In the implementation, the actor-critic network is a two-layer MLP with LayerNorm and ReLU activations, followed by separate actor and critic heads over the seven-way action space.

PPO's clipped surrogate objective prevents excessively large policy updates by constraining the probability ratio between the updated and previous policies, approximating a trust-region constraint without the computational cost of second-order methods. Advantages are estimated using Generalized Advantage Estimation (GAE) (Schulman et al., 2015) with $\gamma = 0.99$ and $\lambda_{\text{GAE}} = 0.95$, which provides smooth credit assignment across sequential bit-width decisions–critical since early allocation decisions affect later budget feasibility.

The full optimization objective combines the clipped surrogate loss, value function regression ($c_1 = 0.5$), and entropy regularization ($c_2 = 0.05$). In the saved v4 runs, PPO performs 3 optimization epochs per roll-out. The entropy term discourages premature convergence to deterministic bit-width assignments, which is important because early decisions influence remaining budget flexibility and final model quality.

3.5 Quantization Engine and Bit Accounting

After the policy selects an action for a column chunk, the quantization engine applies the corresponding low-bit operator: elastic binarization for 1-bit weights, SEQ-style operators for 1.58/2-bit weights, and LSQ-style sparse quantization for 3/4/8-bit weights. Each operator performs lightweight local scale fitting on the current unit without backpropagating through the full model. If the reconstruction error of a selected operator is excessive, the implementation falls back to a higher-precision operator. Additional implementation details are provided in Appendix C.

We also use activation-aware salient weight protection (Lin et al., 2024). For each unit, a small fraction of salient weights is identified using the product of weight magnitude and activation scale and stored in INT8; the remaining weights use the bit-width selected by the policy. Therefore, the realized storage cost differs from the nominal action bit-width. For a unit i with $n_i^{(p)}$ protected weights and $n_i^{(q)} = n_i - n_i^{(p)}$ quantized weights, the effective bit cost is

$$B_i = \begin{cases} 16 \cdot n_i, & \text{if action = 16,} \\ b_i \cdot n_i^{(q)} + 8 \cdot n_i^{(p)}, & \text{otherwise.} \end{cases} \tag{6}$$

All reported average bit-widths use $\bar{b} = \sum_i B_i / \sum_i n_i$, so the reported compression level includes the overhead of INT8-protected weights.

In summary, WINDQuant should be understood as an end-to-end allocation-and-quantization pipeline, as illustrated in Figure 1.

4 Experiments

In this section, we evaluate WINDQuant across a range of LLaMA-3 family models to assess its ability to balance reasoning performance, language modeling quality, and computational efficiency under ultra-low-bit mixed-precision settings. Our method is compared against PTQ approaches–AWQ (Lin et al., 2024), GPTQ (Frantar et al., 2022), OmniQ (Shao et al., 2023), SpinQuant (Liu et al., 2024b), TesseraQ (Li and Panda, 2024), GPTVQ (van Baalen et al., 2024), SignRound (Cheng et al., 2024) –QAT methods–LLM-QAT (Liu et al., 2024a), EfficientQAT (Chen et al., 2025), –and vector quantization (AQLM (Egiazarian et al., 2024)).

4.1 Experiment Settings

We evaluate WINDQuant on LLaMA-3.2-1B, LLaMA-3.2-3B, LLaMA-3.1-8B, and LLaMA-3-70B. All models are quantized to ultra-low bit-width, approximately 2-bit mixed precision.

For the main tables, we evaluate language modeling quality and reasoning performance using the WikiText-2 (Wiki2) test set (Merity et al., 2016) and zero-shot commonsense reasoning benchmarks, including ARC-E, ARC-c (Clark et al., 2018), BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019),

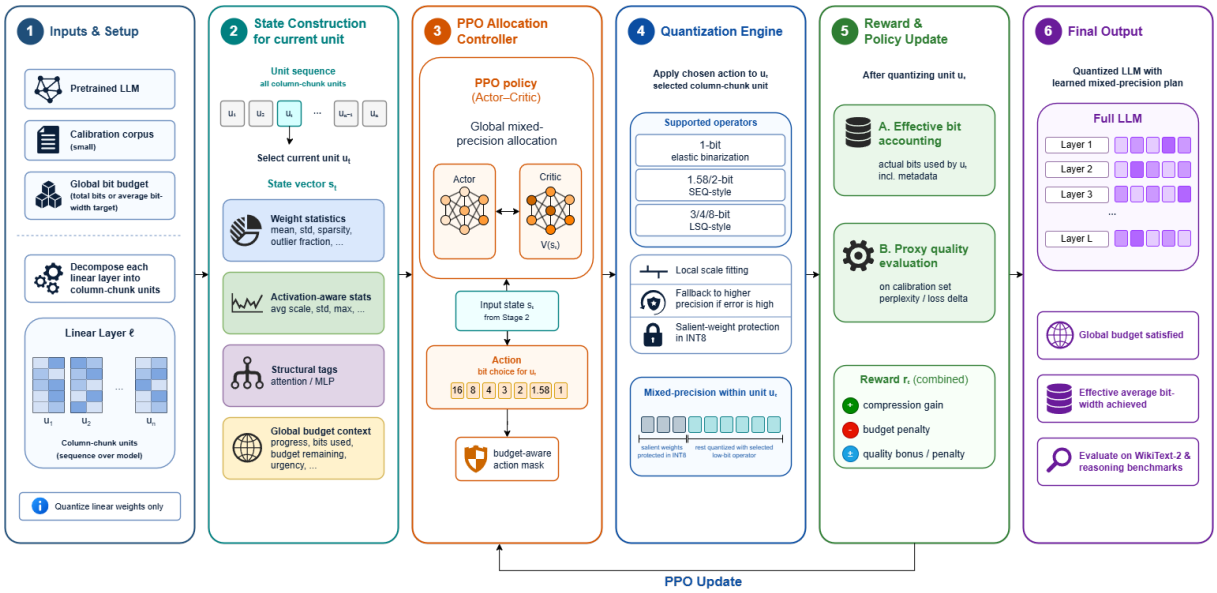

Figure 1: Visualization of the WINDQuant pipeline within a single quantization step.

and Winogrande (Sakaguchi et al., 2021). All experiments are conducted on a single NVIDIA H200 GPU (140GB VRAM). We report macro-average accuracy across tasks, giving each benchmark equal weight. For baseline evaluation, we reproduce all feasible methods using official implementations or widely adopted codebases, following hyperparameters recommended by the corresponding papers and repositories. All resulting models are evaluated with the same downstream benchmark pipeline. We do not apply additional method-specific tuning beyond official recommendations, since our goal is to compare methods under their intended usage rather than engineer custom rescue configurations for individual baselines. Full baseline configurations are provided in Appendix A.2.

During quantization, all models are initialized from pretrained checkpoints. Following standard practice, we quantize linear weights while excluding embedding and output layers to preserve stability. Unless otherwise stated, WINDQuant uses column chunks of size 256, 64 Open-Platypus calibration samples truncated to sequence length 256, 10 local quantizer-optimization steps per unit, and PPO with 3 epochs per roll-out, learning rate 5×10^{-4}, clipping parameter 0.2, and entropy coefficient 0.05.

Training budgets are model-dependent in the saved runs used for this paper. The main 1B, 3B and 8B runs use 100 episodes with curriculum $\{3.0, 2.5, 2.0, 2.0\}$, while 70B runs use 24 episodes for runtime reason. Activation-aware protection stores salient weights in INT8; depending on the run, fixed salient rates are set at 3%. During training, the reward is computed on the Open-Platypus proxy corpus; downstream benchmarks are used only for post-training evaluation.

4.2 Results & Discussion

Table 2 reports the main evaluation results under a practical single-GPU optimization budget. This setting reflects the intended use case of WINDQuant: obtaining competitive ultra-low-bit models without relying on extremely long retraining or search procedures. Compared with the previous version, we include additional recent low-bit baselines, including GPTVQ, SignRound, and TesseraQ, to cover vector-quantization, rounding-based, and tensor/block-reconstruction approaches. Methods whose recommended configurations require substantially larger optimization budgets are not omitted from the paper; instead, they are treated as long-budget reference methods and compared separately in Table 3, where runtime and memory cost are shown alongside quality.

Main comparison. Table 2 shows that WINDQuant achieves the strongest average downstream performance across all three evaluated scales while keeping WikiText-2 perplexity in a stable regime. On 1B, WINDQuant reaches 49.30 average accuracy, outperforming EfficientQAT (46.58), SignRound (45.09), GPTVQ (41.44), and TesseraQ (39.38). On 3B, WINDQuant obtains 61.98 average accuracy at 1.99 effective

Model	Method	#Bits	ARC-c ↑	BoolQ ↑	PIQA ↑	HS ↑	Wino ↑	Avg ↑	Wiki2 ↓
1B	FP	16	36.7	63.8	74.8	64.2	60.7	60.04	8.62
	WINDQuant	2.01	**28.0**	52.7	**64.3**	**47.0**	**54.5**	**49.30**	**19.64**
	AWQ	2	26.0	50.7	52.2	26.0	50.7	41.12	1.7×10^5
	GPTQ	2	25.0	41.2	50.6	26.9	51.0	38.94	1.2×10^5
	LLM-QAT	2	25.2	37.8	52.3	25.6	49.1	38.00	6260
	OmniQ	2	26.2	37.8	53.6	25.9	48.9	38.48	7945
	SpinQuant	2	25.7	44.0	50.7	26.5	51.5	39.68	56.34
	EfficientQAT	2.12	24.3	**60.3**	59.8	35.7	52.8	46.58	29.32
	GPTVQ	2	23.8	45.9	55.7	31.2	50.6	41.44	50.93
	SignRound	2	26.0	53.2	59.8	33.3	53.0	45.09	4.96×10^4
	TesseraQ	2	23.3	40.4	52.8	30.0	50.4	39.38	30.44
3B	FP	16	46.3	74.1	78.2	74.1	69.5	67.61	6.93
	WINDQuant	1.99	**37.8**	66.8	**73.9**	**65.7**	**65.7**	**61.98**	**10.40**
	AWQ	2	25.5	37.8	52.9	26.3	50.2	38.54	6.6×10^4
	GPTQ	2	24.0	43.7	51.3	27.7	47.7	38.88	4.7×10^4
	LLM-QAT	2	25.7	37.8	50.7	41.5	54.4	42.02	6471
	OmniQ	2	24.2	37.8	50.7	26.1	48.9	37.54	9657
	SpinQuant	2	27.0	46.2	52.4	26.4	47.9	39.98	57.40
	EfficientQAT	2.12	24.8	43.0	64.1	37.1	55.0	44.80	21.49
	AQLM	2.27	37.6	**70.5**	73.8	64.4	63.0	61.86	11.29
	GPTVQ	2	20.7	37.9	52.4	27.9	49.8	37.74	17.57
	SignRound	2	31.3	61.5	67.4	48.9	56.5	53.13	1.01×10^5
	TesseraQ	2	24.6	54.6	60.1	37.3	54.1	46.13	30.22
8B	FP	16	55.03	83.05	81.12	79.3	74.58	74.62	5.56
	WINDQuant	1.96	**49.9**	**77.1**	**78.4**	**75.4**	70.3	**70.22**	7.90
	AWQ	2	25.7	42.9	50.3	25.9	48.5	38.66	2.8×10^5
	GPTQ	2	24.2	50.5	52.7	30.2	50.4	41.60	10092
	LLM-QAT	2	21.6	37.8	50.3	26.1	49.9	37.14	29.50
	OmniQ	2	25.7	37.8	53.1	26.4	50.6	38.72	4.2×10^4
	SpinQuant	2	25.7	47.7	50.8	26.2	49.5	39.98	30.20
	EfficientQAT	2.12	31.1	60.6	70.3	56.3	59.3	55.52	14.70
	AQLM	2.27	45.7	75.6	**78.4**	73.9	**71.6**	69.04	**6.90**
	GPTVQ	2	21.9	37.8	53.9	27.7	50.5	38.37	12.15
	SignRound	2	33.1	64.6	69.3	56.7	60.1	56.75	4.99×10^4
	TesseraQ	2	22.1	57.1	53.3	54.0	58.4	48.97	100.86

Table 2: Main results on LLaMA-3 1B, 3B, and 8B under approximately 2-bit quantization within a practical single-GPU optimization budget. Higher is better (↑) except WikiText-2 perplexity (↓). All reproduced baselines are run with official or recommended configurations and evaluated using the same benchmark pipeline. We include standard standalone PTQ methods (AWQ, GPTQ, OmniQuant, SpinQuant), QAT baselines (LLM-QAT, EfficientQAT), vector-quantization baselines (AQLM, GPTVQ), and recent ultra-low-bit baselines (SignRound, TesseraQ). Degenerate perplexity values are retained for transparency, but are interpreted as evidence of instability in the direct 2-bit setting rather than as the primary basis for WINDQuant's advantage. Long-budget methods whose recommended optimization cost substantially exceeds the practical-budget setting are discussed separately in Table 3.

bits, slightly above AQLM (61.86 at 2.27 bits) and substantially above SignRound (53.13), TesseraQ (46.13), and EfficientQAT (44.80). On 8B, WINDQuant reaches 70.22 average accuracy at 1.96 bits, outperforming AQLM (69.04 at 2.27 bits), EfficientQAT (55.52), SignRound (56.75), TesseraQ (48.97), and GPTVQ (38.37).

Interpretation. The degenerate perplexities of several standalone PTQ baselines should be interpreted as evidence that direct 2-bit PTQ is unstable, not as the sole basis for WINDQuant's advantage. The more informative comparison is against recent low-bit and vector-quantization baselines. AQLM is the closest competitor, achieving slightly better WikiText-2 perplexity on 8B, but WINDQuant obtains higher average accuracy at a lower effective bit-width. SignRound reaches competitive downstream scores on some tasks, but its WikiText-2 perplexity is highly unstable under our reproduced 2-bit setting. Overall, these results suggest that WINDQuant's learned global allocation policy provides a favorable compression–accuracy trade-off by deciding how a limited precision budget should be distributed across the model, rather than only improving local quantization of individual groups.

Efficiency comparison. Table 3 compares quality, optimization time, and peak memory consumption on LLaMA-3.1-8B. This comparison is im-

Method	Time	GPU Mem.	Avg↑	Wiki2 ↓
WINDQuant	49	17.1	70.2	**7.9**
LLM-QAT	0.5	126.9	42.0	7137
EfficientQAT	14	8.1	44.8	14.7
ParetoQ†	1777	82.4	**70.9**	8.0

Table 3: Efficiency comparison on LLaMA-3.1-8B under approximately 2-bit quantization. WINDQuant and reproduced baselines are measured in our environment. † ParetoQ is included as a long-budget reference rather than a practical-budget baseline: its quality numbers are taken from the original paper (Liu et al., 2025), while its optimization cost is estimated from running the recommended configuration on our single-H200 setup. Runtime is reported in hours and GPU memory in GB.

Method	#Bits	Avg ↑	Wiki2 ↓
FP	16	**77.28**	**2.53**
WINDQuant	2.02	**71.48**	5.10
AQLM	2.27	70.04	**4.57**
TesseraQ	2.00	66.78	7.47
EfficientQAT	2.12	38.90	1.2×10^7

Table 4: Results on LLaMA-3-70B under approximately 2-bit quantization. We attempted WINDQuant together with several baselines that were feasible at smaller scales. In our single-H200 setup, WINDQuant completed with stable quality, while other attempted baselines either failed to maintain a usable perplexity regime or did not complete within the available memory and runtime budget.

portant because ultra-low-bit quantization is not only an accuracy problem: methods that require extremely long optimization may be impractical for iterative compression or deployment-oriented workflows.

WINDQuant requires 49 hours and 17.1 GB of peak GPU memory while achieving 70.2 average accuracy and a WikiText-2 perplexity of 7.9, giving the best quality–efficiency trade-off among the reproduced practical-budget baselines. LLM-QAT is faster in wall-clock time, but requires 126.9 GB of memory and produces a degenerate perplexity of 7137. EfficientQAT has lower memory usage and shorter runtime, but its average accuracy remains substantially lower and its perplexity is nearly twice that of WINDQuant.

ParetoQ is included as a long-budget reference rather than in the practical-budget main table. Although it reports comparable quality, with 70.9 average accuracy and 8.0 WikiText-2 perplexity, it requires 1777 hours and 82.4 GB of memory under its recommended configuration. This supports the central claim of WINDQuant: learned fine-grained allocation can approach the quality of much heavier ultra-low-bit optimization methods while remaining substantially cheaper to optimize.

The computational profile of WINDQuant is dominated by policy rollout, lightweight local quantizer fitting, and calibration-based proxy evaluation rather than full model retraining, avoiding global backpropagation through the full model and reducing optimization cost relative to retraining-heavy alternatives.

Quantization on LLaMA-3-70B. Quantization at the 70B scale presents a significantly more challenging regime, where conventional PTQ methods degrade severely under ultra-low-bit constraints and several stronger baselines become difficult to run within a single-GPU budget. We therefore restrict the 70B comparison to methods that completed with reportable outputs in our environment or had directly comparable available results. As shown in Table 4, WINDQuant achieves 71.48 average accuracy at 2.02 bits, preserving more than 92% of the full-precision score while keeping WikiText-2 perplexity at 5.10. AQLM obtains slightly lower average accuracy (70.04) at a higher effective bit-width of 2.27 bits, while EfficientQAT fails to maintain a usable language-modeling regime under our reproduced setting, with WikiText-2 perplexity increasing to 1.2×10^7. These results suggest that the learned allocation policy remains effective at larger model scales, where fine-grained precision assignment can exploit the redundancy of high-capacity models while avoiding the cost of full retraining.

5 Conclusion

In this work, we introduced WINDQuant, a reinforcement learning framework for ultra-low-bit mixed-precision quantization of large language models. By formulating precision allocation as a sequential decision-making problem, WINDQuant assigns bit-widths to fine-grained column-chunk units under a global storage budget. The framework combines PPO-based allocation, activation-aware calibration, lightweight local quantizer fitting, and explicit effective-bit accounting to produce learned mixed-precision plans. Across LLaMA models from 1B to 70B parameters, WINDQuant achieves a competitive compression–quality trade-off while avoiding full model retraining and substantially reducing optimization cost compared with heavier training-based approaches. These results suggest that reinforcement learning is a promising direction for adaptive, budget-aware model compression.

Limitations

This work focuses on weight-only ultra-low-bit quantization. We do not quantize activations or KV cache states, so the reported bit-widths reflect weight-storage cost rather than full serving memory or latency.

WINDQuant uses a lightweight calibration proxy to train the RL policy. This makes policy optimization practical at billion-parameter scale, but the proxy may not fully capture downstream task accuracy, long-context behavior, or domain shift across calibration corpora. Our proxy analysis suggests that it tracks final WikiText-2 perplexity along the training trajectory, but broader validation across datasets and models remains necessary.

The current pipeline also includes practical stabilizers such as activation-aware salient weight protection, local per-unit scale fitting, fallback rules, and budget-aware action masking. These components are important for robustness in the ultra-low-bit regime, but they make WINDQuant a complete allocation-and-quantization pipeline rather than a pure RL policy alone. More extensive ablations over state features, reward components, and protection mechanisms would further clarify the contribution of each component.

Finally, experiments are primarily conducted on LLaMA-family models in a single-GPU setting. Although we evaluate scales from 1B to 70B parameters, additional architectures, calibration corpora, and deployment environments are needed to establish broader generality.

References

Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. 2017. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*.

Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in neural information processing systems*, 32.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. 2025. Efficientqat: Efficient quantization-aware training for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10081–10100.

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

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of machine learning research*, 24(240):1–113.

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. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pages 2924–2936.

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. *arXiv preprint arXiv:1803.05457*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35:30318–30332.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 293–302.

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

Ahmed T Elthakeb, Prannoy Pilligundla, Fatemehsadat Mireshghallah, Amir Yazdanbakhsh, and Hadi Esmaeilzadeh. 2020. Releq: A reinforcement learning approach for automatic deep quantization of neural networks. *IEEE micro*, 40(5):37–45.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

Jiedong Lang, Zhehao Guo, and Shuyu Huang. 2024. A comprehensive study on quantization techniques for large language models. In *2024 4th International conference on artificial intelligence, robotics, and communication (ICAIRC)*, pages 224–231. IEEE.

Min Li, Zihao Huang, Lin Chen, Junxing Ren, Miao Jiang, Fengfa Li, Jitao Fu, and Chenghua Gao. 2024. Contemporary advances in neural network quantization: A survey. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE.

Yuhang Li and Priyadarshini Panda. 2024. Tesseraq: Ultra low-bit llm post-training quantization with block reconstruction. *arXiv.org*.

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 on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2024a. Llm-qat: Data-free quantization aware training for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 467–484.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024b. Spinquant: Llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*.

Zechun Liu, Changsheng Zhao, Hanxian Huang, Sijia Chen, Jing Zhang, Jiawei Zhao, Scott Roy, Lisa Jin, Yunyang Xiong, Yangyang Shi, and 1 others. 2025. Paretoq: Improving scaling laws in extremely low-bit llm quantization. *arXiv preprint arXiv:2502.02631*.

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

Markus Nagel, Mart Van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-free quantization through weight equalization and bias correction. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1325–1334.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*.

Mart van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. 2024. Gptvq: The blessing of dimensionality for llm quantization. *arXiv.org*.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8612–8620.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800.

A Experiment details

A.1 Complete results for LLaMa-3-70B

Table 5 provides the complete per-task results for LLaMA-3-70B under approximately 2-bit quantization.

WINDQuant achieves strong and consistent performance across all tasks at the 70B scale. In particular, it remains closest to full precision on PIQA (81.7 vs. 83.8) and HellaSwag (78.3 vs. 85.6), indicating that commonsense reasoning is largely preserved under extreme quantization. This suggests that a significant portion of high-level semantic reasoning can be maintained even when the majority of weights operate at ultra-low precision.

On more challenging benchmarks such as ARC-c, WINDQuant reaches 51.2 compared to 58.6 in full precision. The larger gap on ARC-c indicates that tasks requiring multi-step or structured reasoning remain more sensitive to aggressive compression, even at the 70B scale.

Overall, WINDQuant achieves an average accuracy of 73.46, maintaining most of the full-precision performance (79.14) at only 2.02 bits. This result highlights that the proposed policy scales effectively to high-capacity models, where redundancy in parameter space can be better exploited through fine-grained precision allocation.

In terms of language modeling quality, WINDQuant attains a WikiText-2 perplexity of 5.10, which remains close to the full-precision baseline (2.53). In contrast, EfficientQAT exhibits extremely high perplexity (1.2×10^7), indicating a failure to preserve model quality under ultra-low-bit constraints. This further reinforces that maintaining a stable perplexity regime is critical for scaling quantization methods to large models.

A.2 Baseline Configuration Details

To ensure a fair evaluation, all baselines were configured following the recommendations in their respective publications and official repositories whenever possible. Table 6 summarizes the exact hyperparameters used in our experiments. Unless otherwise stated, all methods quantize linear weight matrices only while excluding the embedding and output head layers for stability. For weight-only PTQ and vector-quantization baselines, we evaluate approximately 2-bit weight quantization under each method's intended configuration.

PTQ baselines. AWQ and GPTQ were run through the `llmcompressor` one-shot pipeline with asymmetric 2-bit quantization and group size 128, using 512 Open-Platypus calibration samples at sequence length 2048, following the recommended settings of Lin et al. (2024) and Frantar et al. (2022). OmniQuant uses learnable weight clipping (LWC) only, optimized on 128 WikiText-2 samples at sequence length 2048 for 10 epochs, with the 8B run reduced to 5 epochs due to runtime constraints, following Shao et al. (2023). SpinQuant follows its two-stage procedure: it first learns R3 orthogonal rotations on 128 C4 samples and then applies W2A16 post-training quantization with group size 128, consistent with Liu et al. (2024b).

We additionally include recent ultra-low-bit baselines that are closer to the target regime of this paper. SignRound is evaluated as a 2-bit weight-only baseline with group size 128, 128 calibration samples from `NeelNanda/pile-10k`, sequence length 2048, 200 optimization iterations, asymmetric quantization, batch size 8, and AMP enabled. TesseraQ is evaluated using its 2-bit weight-only w2g128 configuration with 128 WikiText-2 calibration samples at sequence length 2048, `reduce_memory=True`, no weight clipping, and no loaded transform. These baselines are included to strengthen the comparison beyond conventional PTQ recipes and to cover recent rounding-based and tensor/block-reconstruction approaches in the ultra-low-bit setting.

QAT baselines. EfficientQAT follows its two-stage training protocol: Block-AP with 1024 Open-Platypus samples at sequence length 1024, followed by E2E-QP on 2048 Deita-10K samples at sequence length 2048 with learning rate 2e-5 and gradient accumulation 16 (32 for 8B), matching Chen et al. (2025). LLM-QAT uses W2A8KV4 quantization with 1 epoch of training on synthesized data, learning rate 2e-5, cosine scheduling, and knowledge distillation, consistent with Liu et al. (2024a). We retain LLM-QAT in the main comparison because it represents an established QAT-style low-bit LLM compression baseline, while EfficientQAT provides a stronger recent QAT reference.

Vector quantization baselines. AQLM uses a 1×16 codebook with in-group size 8, trained for 10 epochs with early stopping patience 3 on 1024 WikiText-2 samples at sequence length 4096, following Egiazarian et al. (2024). GPTVQ is evaluated as a 2-bit weight-only vector-quantization

Method	#Bits	ARC-e ↑	ARC-c ↑	PIQA ↑	HS ↑	Wino ↑	Avg ↑	Wiki2 ↓
FP	16	86.6	58.6	83.8	85.6	81.1	79.14	2.53
WINDQuant	2.02	**80.3**	**51.2**	**81.7**	**78.3**	75.8	**73.46**	5.10
AQLM	2.27	79.0	50.1	78.1	63.5	**79.5**	70.04	**4.57**
EfficientQAT	2.12	26.5	38.0	53.7	26.4	49.0	38.72	1.2×10^7
TesseraQ	2.00	78.7	47.4	78.2	57.9	71.7	66.78	7.47

Table 5: Full per-task results on LLaMA-3-70B under ∼2-bit quantization. Higher is better (↑) except WikiText-2 perplexity (↓).

Method	Group	Calib. Dataset	Calib. n	Seq. Len	Key Hyperparameters
AWQ	128	Open-Platypus	512	2048	Asym. quantization at 2–3 bit; sym. at ≥4 bit
GPTQ	128	Open-Platypus	512	2048	Asym. quantization at 2–3 bit; Hessian-weighted rounding
OmniQuant	128	WikiText-2	128	2048	LWC only; 10 epochs (5 for 8B); LR 0.01
SpinQuant	128	C4	128	2048	R3 learned rotations; W2A16KV16
SignRound	128	NeelNanda/pile-10k	128	2048	2-bit weight-only; 200 iterations; asym=True; batch size 8; AMP=True
TesseraQ	128	WikiText-2	128	2048	2-bit weight-only; w2g128 config; reduce_memory=True; no weight_clip; no load_transform
LLM-QAT	—	Synthesized	—	2048	W2A8KV4; 1 epoch; LR 2e-5; cosine schedule; KD
EfficientQAT	64	Open-Platypus	1024	1024/2048	Block-AP 1 ep + E2E-QP 5 ep on Deita-10K
AQLM	8	WikiText-2	1024	4096	1×16 codebook; 10 epochs; early-stop 3
GPTVQ	256	WikiText-2	128	2048	2-bit weight-only; vq-dim=1; codebook-bits=8; kmeans-iters=100; Hessian-weighted lookups; M-step; Mahalanobis initialization

Table 6: Exact configurations used for the baseline methods in our experiments. "Group" denotes the quantization group size when applicable. "Calib. n" denotes the number of calibration samples. The upper block contains standard and recent PTQ baselines, the middle block contains QAT baselines, and the lower block contains vector-quantization baselines. RTN and QuIP are omitted from the main configuration table because they are reported only in the extended appendix comparison.

baseline with group size 256, 128 WikiText-2 calibration samples at sequence length 2048, vq-dim=1, codebook-bits=8, 100 k-means iterations, Hessian-weighted lookups, M-step refinement, and Mahalanobis initialization. Including both AQLM and GPTVQ allows us to compare WINDQuant against two vector-quantization approaches with different optimization and codebook designs.

Regarding unusually high perplexity of PTQ baselines. We emphasize that the poor performance of standard PTQ methods (AWQ, GPTQ, RTN, SpinQuant, QuIP) at 2-bit is consistent with findings in prior work (Egiazarian et al., 2024): these methods were primarily designed and validated at 3–4 bit precision, and their quality collapses rapidly below 3 bits. Our SpinQuant results (e.g., Wiki2 PPL of 56.34 on 1B, 57.4 on 3B) are broadly consistent with the ranges reported in the SpinQuant paper for 2-bit settings. RTN, AWQ, and GPTQ produce degenerate outputs (10^4–10^6 perplexity) because their quantization noise overwhelms the signal at 2-bit group-128 granularity. No additional tuning was performed beyond the official recommended configurations, as our goal was to evaluate each method under its standard settings.

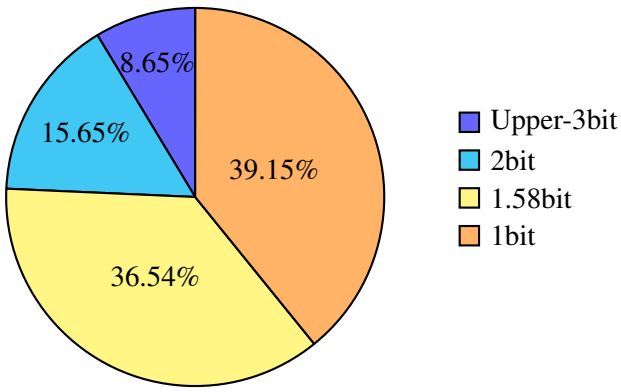

Figure 2: Action distribution of the RL quantization policy when quantizing LLaMA-3.1-8B. Higher-bit actions (≥3 bits and skip) are grouped into "Upper-3bit", showing that the policy strongly favors ultra-low-bit configurations (1–2 bits).

A.3 Learned policy behavior

As shown in Figure 2, the RL agent strongly favors ultra-low-bit configurations, with approximately

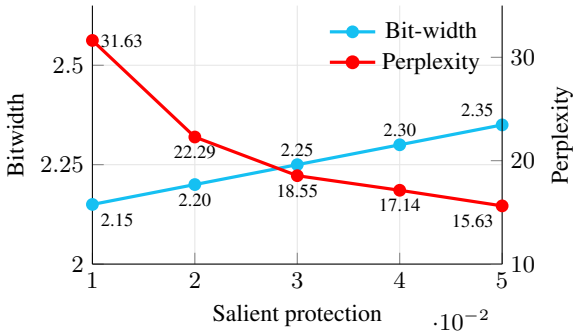

Figure 3: Effect of salient protection on average bit-width and WikiText-2 perplexity for 5-episode 2-bit quantization of LLaMA-3.2-1B. Increasing salient protection raises the effective bit-width while generally improving perplexity.

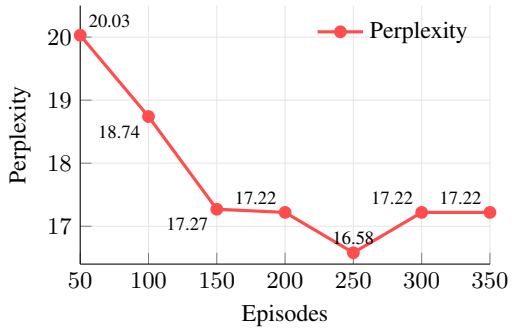

Figure 4: Evolution of WikiText-2 perplexity over RL training episodes. Perplexity decreases rapidly during the early stage of training and shows limited additional benefit after 200 episodes, indicating convergence of the learned allocation policy.

Eps	50	100	150	200	250	300	350
Wik2	20.03	18.74	17.27	17.22	16.58	17.22	17.22
Prxy	10.19	9.86	9.29	9.29	9.06	9.28	9.27

Table 7: Final WikiText-2 perplexity and training-time proxy perplexity across RL training episodes.

75% of weights assigned to 1-bit and 1.58-bit, and only a small fraction (8.65%) allocated to higher precision (≥3 bits). This distribution indicates that WINDQuant does not simply preserve many units at high precision. Instead, it learns an aggressive allocation policy that assigns most units to binary or ternary-like regimes while reserving higher precision for a small subset of more sensitive units. This behavior aligns with the core design goal of WINDQuant: exploiting fine-grained redundancy through learned allocation rather than relying on uniform or layer-wise precision choices.

B Design Choices and Empirical Justification

B.1 Quality vs. Bit-width Trade-offs

As shown in Figure 3, increasing the salient protection ratio leads to a clear trade-off between effective bit-width and language modeling quality. When the protection ratio is very low (e.g., 0.01), the model operates under stricter compression (around 2.15 bits) but suffers from significantly degraded perplexity (31.63). As the protection ratio increases, perplexity improves rapidly, dropping to 22.29 at 0.02 and further to 18.55 at 0.03. Beyond this point, the marginal improvement in perplexity diminishes while the effective bit-width continues to rise, reaching 2.35 at 0.05 in the figure. Based on this observation, we choose a protection ratio of 0.03 as a balanced operating point: it significantly improves perplexity compared to more aggressive compression, while keeping the bit-width close to the target ultra-low regime. This analysis is intended to select a stable protection rate rather than to isolate the contribution of the RL policy.

Salient protection alone only determines a small set of INT8-preserved weights; the precision of the remaining majority of parameters is still decided by the learned allocation policy. We therefore combine this protection-rate study with the heuristic allocator comparison in Section B.3, where the same protection mechanism and quantization operators are kept fixed while the PPO policy is replaced by a deterministic sensitivity rule.

B.2 Quality vs. Run-time Trade-offs

Table 7 and Figure 4 show that the optimization process converges rapidly within the first 200 episodes. WikiText-2 perplexity decreases from 20.03 at 50 episodes to 18.74 at 100 episodes, and further to 17.27 at 150 episodes. At 200 episodes, perplexity reaches 17.22, after which no consistent improvement is observed. Although a lower value of 16.58 appears at 250 episodes, this gain is not sustained, as perplexity returns to 17.22 at 300 and 350 episodes. This indicates that the policy has effectively converged by 200 episodes, and later improvements are not stable. Across checkpoints from 50 to 350 episodes, the training-time proxy is strongly correlated with final WikiText-2 perplexity (Pearson $r = 0.997$, Spearman $\rho = 0.935$). Although this analysis is limited to one training trajectory, it suggests that the proxy provides a useful signal for guiding policy optimization.

From an efficiency standpoint, extending train-

ing beyond 200 episodes is not justified. Under the default setting, every additional 50 episodes increases runtime by approximately 7 hours. Increasing the budget from 200 to 350 episodes therefore adds 21 hours of computation without yielding a reliable improvement in perplexity. Even the extension from 200 to 250 episodes incurs an additional 7 hours while failing to produce a consistent gain. These results support selecting 200 episodes as the default training budget, as it captures the majority of the performance improvement while avoiding unnecessary computational overhead.

B.3 Comparison with Heuristic Sensitivity Allocator

A natural question is whether the RL-learned policy provides value beyond a well-designed deterministic heuristic. To investigate this, we implement a heuristic sensitivity allocator that uses the identical infrastructure as WINDQuant—the same activation-aware saliency protection (3% salient rate), the same per-unit quantizers (elastic binarization, SEQ, LSQ), and the same effective-bit accounting—but replaces the PPO policy with a deterministic rule.

Heuristic rule. For each unit, we compute a saliency score as the mean product of activation scale and weight magnitude. Units are ranked by saliency (descending), and bits are assigned greedily: starting from a 2-bit default for all units, the least sensitive units are pushed to 1.58-bit or 1-bit to free budget, and the most sensitive units are promoted to 3-bit or 4-bit if budget permits. The target average bit-width is 2.0.

Experimental setup. Both WINDQuant (50 episodes) and the heuristic allocator are run on LLaMA-3.2-1B with 5 random seeds $\{42, 123, 456, 789, 2026\}$ to estimate variance. We report mean $\pm$ standard deviation of proxy perplexity and average bit-width.

Discussion. The results in Table 8 show that WINDQuant consistently outperforms the deterministic heuristic under an identical quantization pipeline, indicating that the improvement is not attributable to the underlying quantizers, saliency definition, or protection mechanism, but rather to the learned allocation policy. While the heuristic captures a reasonable first-order approximation of sensitivity by ranking units based on saliency, it remains fundamentally myopic: decisions are made greedily and independently, without accounting for global budget interactions or downstream effects of earlier allocations.

Method	Bitwidth	Perplexity
Heuristic	2.182	8.875
WINDQuant	2.066 ± 0.001	8.756 ± 0.055

Table 8: Comparison of WINDQuant (RL) vs. deterministic heuristic allocator on LLaMA-3.2-1B at $\sim$2-bit, averaged over 5 seeds. Both methods use identical saliency protection, quantizers, and bit accounting. WINDQuant involve 50 episodes in the quantization process.

In contrast, the RL policy explicitly optimizes a sequential objective under a global bit constraint, enabling it to discover non-trivial allocation patterns that better balance compression and quality. In particular, the policy can implicitly trade off local sensitivity against global budget pressure, selectively allocating higher precision only when it yields sufficient marginal benefit, while compensating with more aggressive compression elsewhere. The heuristic allocator can approach WINDQuant in proxy perplexity only by using a noticeably larger effective bit-width. WINDQuant achieves a comparable perplexity at roughly 0.12 fewer bits, indicating that the RL policy learns a more efficient allocation under the global budget. This behavior cannot be replicated by a static ranking-based heuristic without additional tuning or hand-crafted rules.

Moreover, the near-zero variance observed across seeds suggests that the learned policy is stable and not sensitive to stochastic training noise in this setting. Overall, these results support the claim that the advantage of WINDQuant arises from its ability to perform coordinated, budget-aware allocation, rather than from auxiliary components shared with the heuristic baseline.

B.4 Sensitivity to Chunk Size

The column-chunk size G controls the granularity of the quantization units. Smaller G creates more units per layer, enabling finer-grained precision allocation at the cost of more RL decisions per episode. Larger G reduces overhead but limits the agent's ability to adapt to within-layer sensitivity variations.

We evaluate three chunk sizes—$G \in \{128, 256, 512\}$—on LLaMA-3.2-1B with 100 episodes and identical hyperparameter (target 2.0-bit, salient rate 0.03, curriculum $\{3.0, 2.5, 2.0, 2.0\}$).

G	Units/ep	PPL ↓	BW	Time (h)
128	88	8.78	2.03	20.4
256	74	8.87	2.02	14.2
512	52	10.03	2.03	7.2

Table 9: Effect of chunk size G on proxy perplexity, average bit-width, number of RL decisions per episode, and total runtime for LLaMA-3.2-1B at ∼2-bit quantization over 100 episodes.

As shown in Table 9, the chunk size G introduces a clear trade-off between quantization granularity and computational efficiency. A smaller chunk size ($G = 128$) provides finer-grained control, achieving the best perplexity (8.78), but incurs the highest runtime (20.4 hours). In contrast, a larger chunk size ($G = 512$) reduces the number of RL decisions and total runtime (7.2 hours), but results in a clear degradation in perplexity (10.03), indicating limited capacity to capture within-layer sensitivity variations.

The intermediate setting $G = 256$ provides a balanced trade-off. It achieves a perplexity of 8.87, which is very close to the result at $G = 128$ (a difference of 0.09 PPL), while reducing total runtime to 14.2 hours, corresponding to a 30.2% reduction. This demonstrates that $G = 256$ preserves most of the performance benefits of finer granularity while offering a substantial improvement in efficiency, making it a suitable default choice for subsequent experiments.

C Implementation Details

C.1 Pipeline

Figure 1 illustrates the overall workflow. Given a pretrained model $\mathcal{M}$, we first decompose each weight matrix into column-level quantization units. At each time step t, the environment constructs a state vector s_t summarizing statistical properties of the current unit, structural layer information, and global compression indicators. The RL agent selects an action a_t, which is executed by the quantization engine using elastic binarization at 1-bit, SEQ-style operators at 1.58/2-bit, and LSQ-style sparse quantization at 3/4/8-bit, with fallback to a higher precision if reconstruction error is excessive. After each action, a small step reward is recorded. Once all units are processed, a fast perplexity pass on calibration text produces the episode-end reward used by PPO.

C.2 Per-unit Scale Optimization

At each step, the selected quantization operator fits its scale parameter by running a small local optimization (typically 10 Adam steps) on the reconstruction loss for the current column chunk. SEQ variants use soft assignments and per-group scales, whereas LSQ-style operators learn step sizes with straight-through estimation for discrete rounding. If the reconstruction error remains too large, the implementation can fall back to the next higher precision. This optimization is entirely local to the unit: it requires no model-wide backward pass and does not interact with the PPO gradient. Group size is set to the chunk size G=256, matching the unit granularity.

C.3 Salient Weight Protection

We adopt activation-aware weight quantization (Lin et al., 2024) to identify and protect salient weights. The saliency of each weight is computed as the product of its magnitude and the corresponding per-channel activation scale, estimated during a calibration forward pass. This measure approximates the contribution of a weight to output variance and prioritizes parameters associated with frequently activated channels.

Given a protection rate r (configured per run; the saved experiments in this repository mainly use fixed salient rates in the 2%–3% range), the top-r fraction of weights by saliency are preserved at INT8 precision, while remaining weights use the agent-selected bit-width. The implementation can also suppress protection for tensors that do not satisfy heavy-tail criteria when a fixed salient rate is not supplied.

Effective bit-width accounting. The presence of FP16 and INT8-protected parameters means the effective per-unit bit cost differs from the nominal action bit-width. For each unit i with agent-selected action b_i, let $n_i^{(p)}$ denote the number of protected parameters and $n_i^{(q)} = n_i - n_i^{(p)}$ the remainder:

$$B_i = \begin{cases} 16 \cdot n_i, & \text{if action = 16,} \\ b_i \cdot n_i^{(q)} + 8 \cdot n_i^{(p)}, & \text{otherwise.} \end{cases} \quad (7)$$

The average bit-width reported in all experiments uses $\bar{b} = \sum_i B_i / \sum_i n_i$, which faithfully reflects the true storage cost including protection overhead.