ULMoE: Algorithm-System Co-design for Efficient Ultra-Low-Bit MoE Compression and Serving

Anonymous authors
Paper under double-blind review

Abstract

Mixture-of-Experts (MoE) models now dominate frontier LLMs. However, the massive parameter count of experts makes them expensive to fit into GPU memory, and the traffic of reading expert weights on every decode step makes them expensive to serve. We show that ultra-low-bit weight quantization addresses both challenges, and that it pays off more in MoE than in dense models, because sparse routing makes the memory-bound region of grouped GEMM wider than dense GEMM, covering the practical serving batch sizes. Achieving practical serving speedup requires the quantization format and the kernel to be designed together. Methods such as vector quantization and expert-level mixed precision are designed for accuracy alone, and although they reach two bits, no grouped GEMM kernel can execute them efficiently. Methods designed around kernels achieve efficient serving performance but stop at four bits. We present *ULMoE*, an algorithm-system co-design for accurate and efficient ultra-low-bit MoE serving. The quantization format spends 2.5 bits per weight and places the quantization group scales on two integer levels. The metadata for 64 weights fits in one aligned 32-bit word for efficient memory access. On top of the format we build a fast W2A16 grouped GEMM kernel and integrate it into SGLang. Across seven MoE models from three families, ULMoE compresses expert weights $\sim 6\times$ at 2.6 to 2.8 effective bits and loses 1.8 average accuracy points. At matched interactivity it delivers $5.7\times$ the serving throughput of BF16 on Qwen3.5-35B and $1.7\times$ that of Int4 on Qwen3.5-122B.

1 Introduction

Mixture-of-Experts (MoE) architectures have become the dominant recipe for scaling large language models in recent years. Routing each token to a small subset of experts keeps per-token computation constant while growing total model capacity by an order of magnitude. However, the cost of that benefit is memory. Every expert must stay resident in memory even though few are active per token, so routed experts hold $\sim 95\%$ of the weight memory across common open MoE models (Figure 1a), and during decoding their weight traffic dominates step latency (Figure 1b). Workstations and edge devices are important platforms for the local deployment of open-weight models. Even when serving a single user, local workloads can expose parallelism: agent workflows may issue concurrent model calls for independent subtasks, while speculative decoding verifies multiple candidate tokens in each target-model forward pass. These forms of parallelism motivate efficient serving for local deployment, with support for moderate concurrency. For MoE models, the memory constraint and large weight traffic make them hard to serve efficiently.

Weight-only quantization addresses these constraints, and the benefit of weight compression in MoE serving is structural. Sparse routing keeps the experts computation memory-bound, so compression keeps converting into speedup at batch sizes where dense models have long gone compute-bound (Section 3.1). However, no existing quantization method delivers that benefit fully for MoE models, because the algorithm and the system are designed separately. Algorithm-first methods reach two bits with codebooks (Tseng et al., 2024; Egiazarian et al., 2024), rotations or expert-level mixed precision (Huang et al., 2025b). However, those mechanisms cannot be executed efficiently with grouped GEMM, which is the fastest path for experts computation with one unified launch covering

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

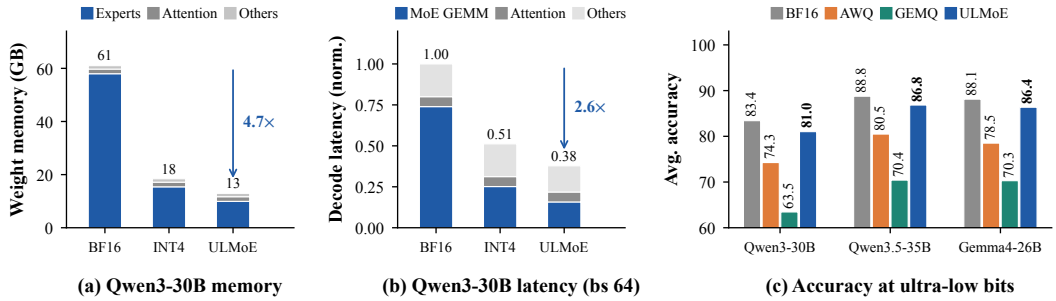

Figure 1: ULMoE overview. (a) Routed experts hold 95% of weight memory for Qwen3-30B-A3B, and ULMoE compresses them by ∼**6**×, compressing the whole model **4.7**× (GPTQ Int4: 3.4×). (b) Decode latency at batch 64: ULMoE cuts the step latency **2.6**× versus BF16. (c) Across the three models shown under comparable budget, AWQ loses 8–10 average accuracy points and GEMQ 18–20, while ULMoE loses under 3.

all E experts. So the compression these methods claim does not turn into throughput gain. System-first formats serve efficiently, but the bit-widths deployed in production stacks stop at four bits (Frantar et al., 2024), so everything below that width remains unexplored in practice.

In this paper we present ULMoE, an algorithm-system co-design for ultra-low-bit MoE compression and serving. **On the algorithm side**, we design a quantization format built on a two-level hierarchy of integer scales. It shrinks the scaling group to 8 weights for better accuracy, while reducing the overheads of extra metadata. The format is

$$\hat{w} = (q - z) \cdot s_{\text{int}} \cdot m \cdot s_o,$$

built from a 2-bit code q, 1-bit zero z and integer multiplier s_{int} per micro-group of 8 weights, a 7-bit scale code m per macro-group of 64, and one FP16 scale s_o per output channel shared across experts. The zero and two inner scales are integers, the metadata describing 64 weights sums to exactly one aligned 32-bit word and the format costs 2.5 bits per weight. Salient input channels for each expert are protected by additional 2-bit columns in the same quantization format. **On the system side**, the ULMoE format preserves the unified structure of experts, which can be executed efficiently with grouped GEMM. We build a W2A16 grouped GEMM kernel with efficient dequantization, running one decode path and epilogue for all E experts, and fuse the salience gather and the residual columns into the activation kernels the MoE block already runs. Our contributions are as follows:

- We give a roofline analysis of the routed grouped GEMM. Its weight-traffic term stays memory-bound over a batch range E/k_r times wider than the dense one, long after a dense layer has gone compute-bound, extending the speedup to larger batch sizes for serving.
- We propose a novel 2.5-bit quantization format, built from a two-level integer scale hierarchy and top-k salient channel protection. It lowers weight quantization MSE while leaving every expert in a single unified representation.
- We implement an efficient W2A16 grouped GEMM kernel that keeps one decode path and one epilogue across all experts, and fuses the salience gather and the residual replication into the activation kernels the MoE block already runs. We further integrate it into SGLang.
- We show that ULMoE advances the state of the art in both accuracy and serving. At 2.6-2.8 bits it leads to a 1.8 accuracy point drop on average compared to BF16 in 7 MoE models, where AWQ and GEMQ at the same budget lose 5.4–18.3 and 15.3–29.9 points, and it exceeds four-bit quantization in both serving throughput and resident-batch capacity across different models.

2 Background

MoE model inference. An MoE layer replaces the dense feed-forward block with E expert MLPs and a router. For a token with hidden state $x \in \mathbb{R}^H$, the router scores every expert with a small linear layer, keeps the k_r highest scores, and renormalizes them into gating weights g_e. Each expert holds

three matrices, $\mathrm{gate}_e, \mathrm{up}_e \in \mathbb{R}^{I \times H}$ and $\mathrm{down}_e \in \mathbb{R}^{H \times I}$, and computes $\mathrm{down}_e\big(\phi(\mathrm{gate}_e(x)) \odot \mathrm{up}_e(x)\big)$ with ϕ a SiLU or GELU gate. The layer output is the gated sum $\sum_e g_e \cdot \mathrm{expert}_e(x)$ over the k_r selected experts.

Serving engines such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) store the experts of one layer as two 3-D tensors, $[E, 2I, H]$ for the fused gate and up projections and $[E, H, I]$ for down, and execute the layer as a grouped GEMM. The engine sorts the (token, expert) pairs by expert, pads each expert's block to a tile boundary, and launches one kernel in which every tile multiplies its own tokens by its expert's weights. One launch covers all E weight matrices under a single instruction stream, layout and set of launch parameters, and the uniformity makes it efficient on GPUs. Because the expert tensors are by far the largest in the model, the MoE block dominates decode latency (Section 3.1).

LLM quantization. Post-training quantization (PTQ) is the standard way for LLM compression. The most commonly used weight-only quantization format splits each weight matrix into groups of g consecutive input channels and stores every group with one floating-point scale s and one integer zero z,

$$q_i = \mathrm{clamp}\big(\,\mathrm{round}(w_i/s + z),\, 0,\, 2^b - 1\big), \qquad \hat{w}_i = (q_i - z) \cdot s, \tag{1}$$

so a weight costs b bits and a group costs the $(16 + b)$ bits of a 16-bit floating-point s and a b-bit integer z, giving $b + (16 + b)/g$ bits per weight. A smaller g tracks the local weight range more closely and lowers the error, but the $(16 + b)/g$ term grows in proportion. Standard quantization recipes choose a group size of 128 (Lin et al., 2026; Frantar et al., 2024; Liang et al., 2026). Recent works (Tseng et al., 2024; Egiazarian et al., 2024) push quantization bit-width below three bits with weight rotation and vector quantization with learned codebooks. Both are difficult to deploy, because codebook dequantization needs table lookups that are inefficient on GPUs, and a rotated basis must be materialized at module boundaries. These costs keep such methods at batch size 1 GEMV implementation and out of batched serving. Appendix D provides a detailed view of existing quantization methods.

High-performance weight-only quantization kernels such as AWQ (Lin et al., 2026) and Marlin (Frantar et al., 2024) hide dequantization overhead inside the GEMM pipeline, but they target four-bit dense layouts. We extend that line of work to MoE grouped GEMMs and to lower bit widths.

MoE quantization. Most MoE-specific quantization methods converge on a recipe we group as expert-level mixed precision (Duanmu et al., 2025; Deng et al., 2026; Huang et al., 2025b), which scores every expert on a calibration set and solves a small budgeted allocation that hands three or four bits to the top-scoring experts and one or two bits to the tail. The average bits-per-weight numbers are attractive, but per-expert widths break the homogeneity a grouped GEMM depends on, partitioning the (token, expert) pairs into one sub-GEMM per precision class, per projection and per layer, and stopping the expert dimension from packing as one 3-D tensor with uniform strides. MxMoE (Duanmu et al., 2025) generates a fused group-GEMM that runs several precisions concurrently by holding every precision's tile configuration to a common warp count. It shows the recipe can be made to run, but running it costs a hand-tuned micro-kernel per precision class and a tile shape that is not the best for any of them. MiLo (Huang et al., 2025a) instead keeps one width for every expert and goes to three bits, recovering the lost accuracy with a mixture of low-rank compensators. However, its complex memory alignment of 3-bit representations leads to limited speedup, and the low-rank branches incur extra launch overheads.

ULMoE takes the opposite route from existing methods. Everything the grouped GEMM sees stays uniform, with the same bits, layout and decode path. The adaptivity moves inside the format, where a two-level hierarchy of integer scales gives more fine-grained grouping, and top-k residual columns in the unified format better preserve salient input channels.

3 ULMoE: Algorithm-System Co-design

Section 3.1 sets out the motivation of ultra-low-bit quantization for MoE serving, Section 3.2 develops the ULMoE quantization format, and Section 3.3 develops the serving path.

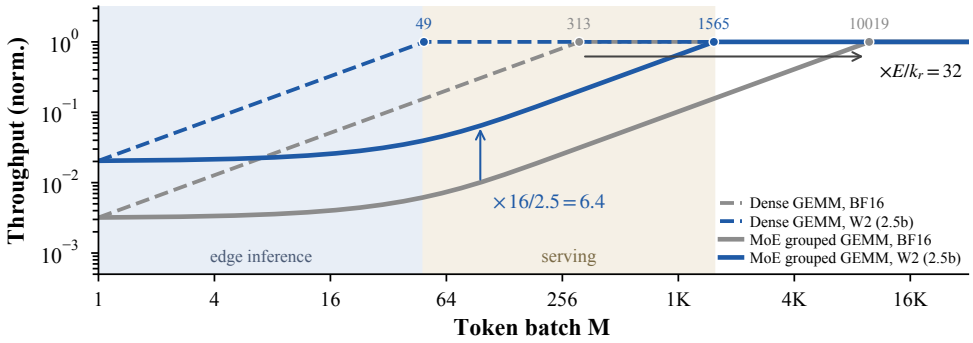

Figure 2: The roofline of equation 4 on RTX PRO 6000 GPU ($R \approx 313$) with E=256, top-8. W2 is at 2.5 bits per weight, with 2-bit codes plus a 0.5-bit metadata per weight. In the *edge-inference* region everything is memory-bound. In the *serving* region the MoE grouped GEMM is still memory-bound, and there W2 attains $6.4\times$ the BF16 throughput theoretically.

3.1 Analysis for Ultra-Low-Bit MoE Quantization

Expert grouped GEMMs dominate weight traffic in MoE serving (Figure 1b), and they stay memory-bound at much larger batches than dense GEMMs do. A roofline argument gives the reason. A dense projection with weights $[N, K]$, b bytes per weight, and M tokens performs $2MNK$ FLOPs against bNK weight bytes, so its arithmetic intensity is $2M/b$ and grows with the batch alone. It stays memory-bound until that intensity reaches the machine balance R, the hardware's peak arithmetic throughput divided by its peak memory bandwidth, which puts the knee at $M^{\star}_{\text{dense}} = bR/2$. For grouped GEMM, we analyze the roofline under the ideal case of independent token routing across different batches, each selecting k_r distinct experts with probability k_r/E per expert. The expected number of experts activated is

$$\bar{A}(M) = E\left[1 - \left(1 - \frac{k_r}{E}\right)^{M}\right]. \tag{2}$$

With each activated expert's weights read once, the Mk_r routed pairs yield a weight-only arithmetic intensity of

$$I_{\text{MoE}}(M) \approx \frac{2Mk_r}{b\,\bar{A}(M)}. \tag{3}$$

At small batches ($Mk_r \ll E$), $\bar{A}(M) \approx Mk_r$, so intensity stays near $2/b$. Expected coverage of all experts reaches 99% at $Mk_r \approx 5E$. As $\bar{A}(M) \to E$, tokens per activated expert approach $\bar{m} = Mk_r/E$, giving the memory-bound condition

$$\bar{m} = \frac{Mk_r}{E} < M^{\star}_{\text{dense}} \quad \Longleftrightarrow \quad M < \frac{E}{k_r} M^{\star}_{\text{dense}}. \tag{4}$$

This indicates the memory-bound region of an MoE grouped GEMM is E/k_r times wider than a dense GEMM's. On our evaluation GPU (RTX PRO 6000, ~500 dense BF16 TFLOPS over 1.6 TB/s, $R \approx 313$) with E=256 and top-8 routing, a dense layer at the format's 2.5 bits goes compute-bound at $M \approx 49$ while the MoE grouped GEMM holds to $M \approx 1565$, with the factor of $E/k_r = 32$ (Figure 2). Between those two knees lies the batch range serving actually occupies, where the compression benefit diminishes for dense GEMM, and the MoE grouped GEMM can still benefit from the reduced memory access.

Using fine-grained quantization groups is a reliable way to cut quantization error, and Figure 4 shows weight MSE falling steadily as a GPTQ (Frantar et al., 2023) Int2 group shrinks from 128 to 8. However, the scales and zeros lead to $18/g$ bits of extra memory overhead. At g=8, a 16-bit scale and a zero cost 2.3 bits per weight, enough to push a two-bit format past the byte budget of Int4. Reaching accurate and effective two-bit quantization therefore means *compressing the scales, not just the weights*. For MoE, the grouped GEMM needs one unified data type, one layout and one decode path, so that a single launch still covers all E experts.

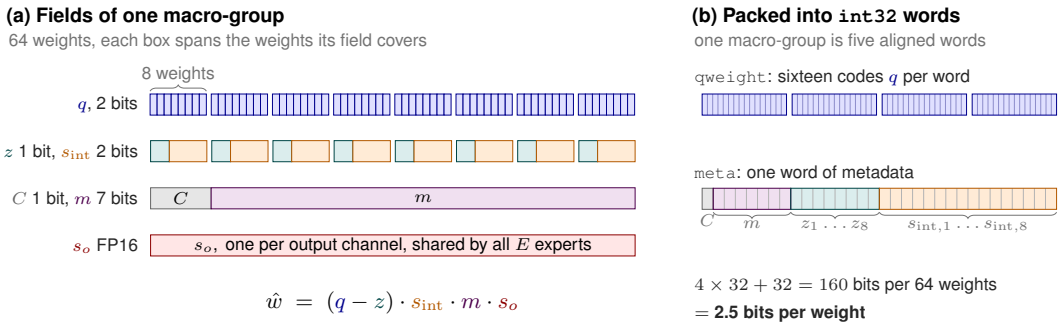

Figure 3: The ULMoE format. (a) The fields of one macro-group. (b) The same macro-group packed into five 32-bit words.

3.2 ULMoE Quantization Algorithm

A routed MoE layer contains gate/up weights $W^{gu} \in \mathbb{R}^{E\times 2I\times H}$ and down weights $W^{d} \in \mathbb{R}^{E\times H\times I}$. ULMoE quantizes only these routed expert tensors and leaves everything else in BF16. We use K for a projection's input dimension, which is H for gate/up and I for down, and N for its output dimension.

The ULMoE format includes three levels of scales in total as illustrated in Figure 3a. The **micro-group** sets the granularity to reduce error. Along the input dimension a projection's weights are cut into micro-groups of 8 channels, and each micro-group carries a zero $z \in \{1, 2\}$ and a two-bit multiplier s_{int}, which is three bits per group. This makes the extra metadata of a small group affordable. The **macro-group** supplies the value those integers multiply. Eight micro-groups form a macro-group of 64 channels holding a 7-bit scale code m and a one-bit ladder $C \in \{5, 7\}$ that fixes the range of s_{int} to $\{C-3, \ldots, C\}$, so each macro-group picks the ladder its weights need. $C = 7$ gives finer relative-scale spacing for macro-groups whose micro-groups have similar dynamic ranges, and $C = 5$ covers a wider range of relative scales. The **anchor** carries one floating-point value s_o per output channel, shared across all E experts of the layer. It turns floating-point scale storage from $O(ENK)$ entries into $O(N)$ and lets the epilogue scale the accumulator by one expert-independent vector instead of gathering per-expert tables. *The word layout* fixes the field widths so that they sum to exactly one machine word, $1 + 7 + 8 \times 1 + 8 \times 2 = 32$ (Figure 3b). Every stored weight then reconstructs as

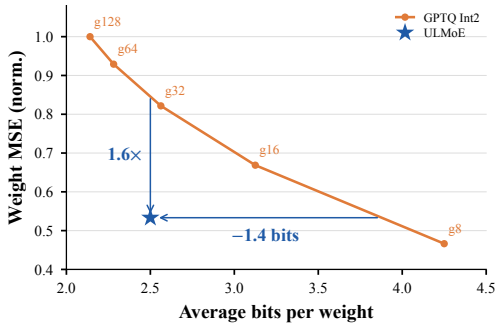

Figure 4: Bits versus weight MSE on expert weights of Qwen3.5-35B, relative to GPTQ Int2 at g=128, whose 16-bit scale and 2-bit zero cost $18/g$ bits. ULMoE format is **1.6**× lower at equal bits or **1.4** bits cheaper at equal error.

$$\hat{w} = (q - z) \cdot s_{\text{int}} \cdot m \cdot s_o, \tag{5}$$

with a 2-bit code $q \in \{0, 1, 2, 3\}$ per weight. A macro-group of 64 weights therefore costs 128 bits of codes and one 32-bit metadata word, which is 2.5 bits per weight. The final format (without additional residual columns) is `qweight` of shape $[E, K/16, N]$ and `meta` of shape $[E, K/64, N]$. A macro-group's whole decode state arrives in a single aligned load and every field comes out with one shift and one mask against immediate operands.

The order in which those fields spend the budget is not arbitrary. Micro-groups of 8 take the granularity, and the integer ladder and the shared anchor make the scales cheap enough to afford it. Figure 4 measures MSE on routed-expert weights. The 2.5-bit ULMoE quantization format reaches the weight MSE of a GPTQ format with $g \approx 10$, which is $1.6\times$ lower than GPTQ Int2 in the same 2.5-bit budget.

With the format fixed, quantization is a search over a finite set. Each macro-group needs a ladder and a 7-bit scale, each micro-group inside it needs a multiplier and a zero. Algorithm 1 runs

Algorithm 1 QUANTIZE(W, k): two-phase quantization of routed-expert projection.

Input: weights $W \in \mathbb{R}^{E\times N\times K}$, residual budget k, step floor $\epsilon = 10^{-8}$
Output: `qweight` and `meta` for $K + k$ columns, anchor $s_o \in \mathbb{R}^N$ in FP16

▷ *Phase 1 (prepare): fix each macro-group's ladder, then compute the anchor.*

1: **for all** macro-groups $v = (v_1, \ldots, v_8)$ of W **do**
2: $\quad \sigma_C(v) \leftarrow \max_j \mathrm{range}(v_j)/(3C)$ for each $C \in \mathcal{C}$
3: $\quad C_v^\star \leftarrow \arg\min_{C\in\mathcal{C}} \sum_{j\le 8} \min_{s\in\mathcal{S}(C),\, z\in\mathcal{Z}} \left\| v_j - D\big(v_j,\, \sigma_C(v)\, s,\, z\big)\right\|^2$
4: $\quad \sigma_v \leftarrow \sigma_{C_v^\star}(v)$
5: **end for**
6: $\mu(V) \leftarrow$ maximum of σ_v over the macro-groups and experts sharing each output channel of V
7: $\tilde{R} \leftarrow W_{:k} - \hat{W}_{:k}$, with $\hat{W}$ from lines 11–14 on $W_{:k}$ at anchor $\mathrm{FP16}\big(\mu(W)/127\big)$
8: $s_o \leftarrow \mathrm{FP16}\big(\max\big(\mu(W),\, \mu(\tilde{R})\big)/127\big)$

▷ *Phase 2 (quantize): quantize every macro-group with the anchor.*

9: **for all** macro-groups v of W, then of $R \leftarrow W_{:k} - \hat{W}_{:k}$ **do**
10: $\quad$ for a macro-group of R, run lines 2–4 on it first ▷ *obtain $C_v^\star$ for R*
11: $\quad m_v \leftarrow \mathrm{clamp}\big(\mathrm{round}(\sigma_v/s_o),\, 0,\, 127\big)$
12: $\quad$ **for** $j = 1$ to 8 **do**
13: $\quad\quad (s_j, z_j) \leftarrow \arg\min_{s\in\mathcal{S}(C_v^\star),\, z\in\mathcal{Z}} \left\| v_j - D\big(v_j,\, \max(m_v s_o s,\, \epsilon),\, z\big)\right\|^2$
14: $\quad\quad q_j \leftarrow \mathrm{clamp}\big(\mathrm{round}\big(v_j/\max(m_v s_o s_j,\, \epsilon) + z_j\big),\, 0,\, 3\big)$ ▷ *ϵ keeps the step positive*
15: $\quad$ **end for**
16: **end for**

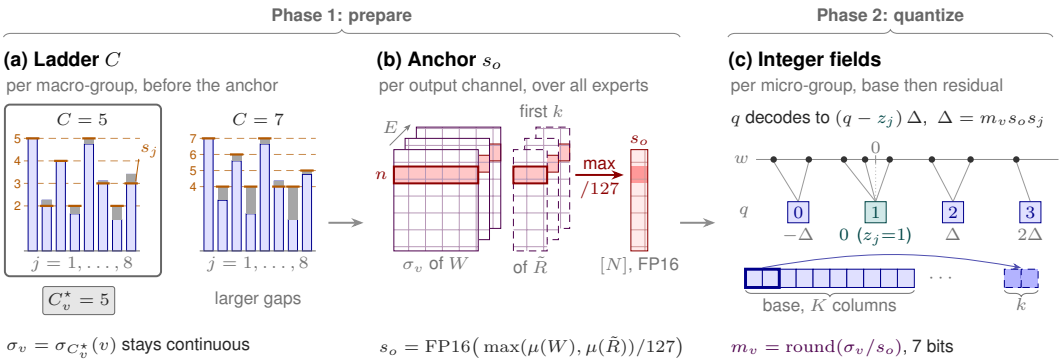

Figure 5: Algorithm 1 on one projection. (a) Each macro-group keeps the ladder that achieves the lower weight MSE. (b) Each output channel's anchor is its largest continuous macro scale, taken over all experts. (c) The macro scale rounds to m_v, and each micro-group's parameters are re-selected.

in two phases, and Figure 5 illustrates both phases. It writes $\mathcal{C} = \{5, 7\}$ and $\mathcal{Z} = \{1, 2\}$ for the ladders and zeros, $\mathcal{S}(C) = \{C-3, \ldots, C\}$ for the multipliers of ladder C, and $D(w, s, z) = \big(\mathrm{clamp}(\mathrm{round}(w/s + z), 0, 3) - z\big)\, s$ for the reconstruction at step s and zero z. **Phase 1** is the prepare stage, fixing every macro-group's ladder and then the anchor. The ladder (lines 2–4) is chosen against a continuous macro scale, which lets it be fixed before the anchor computation. In line 2, $\mathrm{range}(v_j)/3$ is the min-max step of micro-group v_j on a four-level grid, so $\sigma_C(v)\, C$ is the step of the widest micro-group and the ladder gives each micro-group a fraction s/C of it, from $(C-3)/C$ to 1. The anchor (lines 6–8) is then the largest scale any output channel needs, over all experts and over both the weights and their residual columns, divided by the largest 7-bit code and stored in FP16, with the residual term coming from one discarded pass over the salient columns at a provisional anchor. **Phase 2** quantizes the values at the computed anchor (lines 9–14), rounding each macro scale to its 7-bit code m_v and re-enumerating every micro-group's multiplier and zero at the rounded step, since the rounding perturbs every effective micro scale. It passes over the base columns and then over the residual they leave, and a macro-group of that residual takes its ladder from its own values (line 10). The search is a constant number of vectorized candidate evaluations per weight, with no iterative optimization, unlike Hessian-compensation pipelines, and activation information enters only through channel ordering and residual allocation.

In order to better preserve the salient input channels, we append same-anchor 2-bit residual columns for the k=128 most salient channels, yielding $K_{\text{ext}} = K + k$ uniformly encoded columns processed by one kernel. Algorithm 3 uses one shared gate/up permutation across experts and expert-specific, routing-weighted salience for down, measured with temporarily quantized gate/up weights. Down-column permutations are absorbed offline into corresponding gate/up rows, requiring no runtime indexing. Calibration proceeds layerwise using activations from previously quantized layers, caching and replaying one layer at a time to enable single-GPU calibration.

3.3 ULMoE System Design

The ULMoE serving path is a grouped GEMM kernel over routed (token, expert) pairs with two small activation kernels in front of it. Offline quantization writes out four things per MoE module: the packed 2-bit codes, the 32-bit metadata words, the shared FP16 output scales, and one activation permutation for gate/up. Each grouped GEMM is preceded by one small kernel that prepares its activations, and in both cases the additional residual column mechanism is fused into work the block already performs. Before gate/up projections, the salient-first gather and the replication of the top-k channels are a single indexed load, and before down projection the replication is folded into the gated activation, so neither projection pays a launch for the residual columns and both GEMMs see an ordinary matrix of K_{ext} columns.

The grouped GEMM itself runs with BF16 activations, FP32 accumulation and the routing weight folded into the down projection's output stage, and it decodes weights inside the main loop. A macro-group's whole state is one aligned 32-bit load, every field falls out with a shift and a mask against immediates, and each pair of weights is reconstructed by three packed instructions and a single conversion, with no table lookup and no division (Algorithm 2). The only floating-point scale, s_o, reaches the FP32 accumulator once per output channel. One launch therefore covers all E experts, since every tile shares a data type, a layout and a decode path, and the output stage is expert-independent by construction of s_o. Appendix C ablates the 32-bit aligned word design for `meta` using an unpacked variant with the same bits and weights.

4 Evaluation

4.1 Evaluation Setup

We evaluate seven MoE models spanning three families, including Qwen3-30B-2507 (Yang et al., 2025), Qwen3.5-35B (Qwen Team, 2026), Qwen3.6-35B (Qwen Team, 2026), Qwen3.5-122B (Qwen Team, 2026), Gemma4-26B (Team et al., 2026b), GLM4.5-Air (Team et al., 2025), and GLM4.7-Flash (Team et al., 2025). For calibration, every model uses the same recipe of 512 calibration sequences of 2048 tokens from Nemotron post-training dataset v2 (Nathawani et al., 2025).

We evaluate nine challenging benchmarks: the non-reasoning tasks (MMLU (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), HumanEval (Chen et al., 2021), IFEval (Zhou et al., 2023)) with LM-EVALUATION-HARNESS (Gao et al., 2024), and the reasoning tasks (AIME24 (Zhang & Math-AI, 2024), AIME25 (Zhang & Math-AI, 2025), MATH500 (Lightman et al., 2023), GPQA-Diamond (Rein et al., 2023), LiveCodeBench-v6 (Jain et al., 2024)) with LightEval (Habib et al., 2023). Every reasoning task result is the mean of four runs with sampling seeds 0 through 3. We compare against AWQ (Lin et al., 2026) and GEMQ (Deng et al., 2026) at an aligned average experts bit-width. For all methods and models, only the routed experts are quantized. All of the accuracy evaluations are done with real quantized checkpoints.

We evaluate serving performance on top of SGLang. *Offline* evaluation fixes an explicit batch size and sweeps batch 1 to 256 at input sequence length 128 and output sequence length 1024, isolating a format's per-step cost from scheduling. *Online* evaluation sweeps requested concurrency together with the server's cap on running requests, and reports the Pareto envelope of output throughput against interactivity, the per-user decode rate $1000/\text{TPOT}_{\text{p50}}$. Cross-format ratios are read at interactivity anchors fixed in advance. Every experiment runs on NVIDIA RTX PRO 6000 GPUs by default. Appendix B provides more system evaluation results, including kernel benchmarks, evaluation of tensor parallel setting, evaluation of long context workloads, combination with speculative decoding, and evaluation on different GPU. Appendix A details the implementation of ULMoE, and Appendix E shows the baseline configurations, harness and measurement protocol in full.

Table 1: Downstream accuracy of ULMoE against BF16 and same two-bit-budget baselines. MMLU and GSM8K are 5-shot, and LCB is LiveCodeBench-v6.

Method	MMLU	GSM8K	HumanEval	IFEval	AIME24	AIME25	MATH500	GPQA	LCB	Avg
Qwen3-30B-2507										
BF16	81.04	86.13	93.29	91.13	85.00	81.67	97.80	71.97	62.71	83.42
AWQ	77.67	84.38	86.59	87.77	63.33	56.67	95.90	62.88	53.14	74.26
GEMQ	73.71	78.32	85.37	90.05	40.00	36.67	92.00	42.93	32.00	63.45
ULMoE	**80.15**	**85.29**	**89.63**	**90.41**	**80.00**	**78.33**	**96.90**	**68.06**	**60.43**	**81.02**
Qwen3.5-35B										
BF16	83.40	90.60	95.12	92.45	91.67	90.00	98.60	84.09	73.00	88.77
AWQ	80.39	86.96	90.24	89.45	80.00	71.67	97.40	78.03	50.29	80.49
GEMQ	80.37	88.78	85.98	88.85	46.67	43.33	91.70	76.67	31.43	70.42
ULMoE	**82.34**	**89.84**	**93.29**	**91.61**	**89.17**	**85.84**	**98.80**	**82.07**	**68.57**	**86.84**
Qwen3.6-35B										
BF16	84.52	88.55	96.95	91.49	90.00	90.83	98.90	84.22	75.72	89.02
AWQ	81.23	88.93	90.24	89.57	81.67	73.33	97.30	77.02	64.00	82.59
GEMQ	81.06	88.78	82.93	88.85	60.00	40.00	85.60	71.21	33.71	70.24
ULMoE	**84.00**	**89.76**	**95.12**	**90.65**	**90.83**	**85.84**	**99.00**	**82.32**	**73.71**	**87.91**
GLM4.7-Flash										
BF16	73.50	86.28	85.98	87.17	76.67	65.00	97.10	71.21	46.29	76.58
AWQ	64.39	67.25	70.73	73.26	41.67	36.67	87.40	50.76	32.57	58.30
GEMQ	56.07	53.22	61.59	70.24	16.67	26.67	78.20	42.93	14.86	46.72
ULMoE	**69.88**	**81.96**	**75.00**	**84.05**	**77.50**	**63.33**	**95.60**	**64.90**	**45.72**	**73.10**
Gemma4-26B										
BF16	83.86	87.72	95.12	93.76	90.00	87.50	99.00	80.31	75.86	88.13
AWQ	77.76	82.64	93.29	92.57	73.33	61.67	97.40	70.20	57.71	78.51
GEMQ	81.17	86.81	90.85	92.93	43.33	56.67	90.40	68.18	22.29	70.29
ULMoE	**81.74**	**87.64**	**95.12**	**93.88**	**87.50**	**81.67**	**98.50**	**76.64**	**74.57**	**86.36**
Qwen3.5-122B										
BF16	87.12	88.86	92.68	93.88	91.67	90.00	99.20	86.36	75.29	89.45
AWQ	85.52	85.06	89.63	92.69	76.67	75.00	98.10	84.34	69.14	84.02
GEMQ	84.20	82.34	89.02	91.21	60.00	71.67	94.50	61.11	33.71	74.20
ULMoE	**86.22**	**89.46**	**93.90**	**93.29**	**89.17**	**87.50**	**99.40**	**86.11**	**72.14**	**88.58**
GLM4.5-Air										
BF16	81.65	84.69	87.20	89.69	87.50	80.83	98.00	68.18	60.71	82.05
AWQ	79.21	87.11	56.10	81.18	80.00	66.67	92.00	59.09	22.29	69.29
GEMQ	77.43	86.13	50.61	80.87	70.00	63.33	89.20	56.57	6.86	64.56
ULMoE	**80.36**	**87.79**	**87.20**	**89.33**	**82.50**	**79.17**	**97.15**	**67.43**	**60.14**	**81.23**

4.2 Accuracy Performance

Table 1 reports downstream task accuracy. ULMoE maintains competitive average accuracy relative to BF16: average drops span 0.8 to 3.5 points with an average of 1.8, at $\sim 6\times$ reduction of BF16 expert size. The remaining loss is 1.3 points on the four non-reasoning tasks against 2.2 on the five reasoning tasks. Under a comparable bit budget, AWQ loses 5.4 to 18.3 average points and GEMQ loses 15.3 to 29.9.

Extended accuracy evaluation. Table 2 shows the results for FP4-native DeepSeek-V4-Flash-0731 (DeepSeek-AI et al., 2026) and Table 3 shows the results for diffusion MoE models. DeepSeek-V4-Flash contains native FP4 expert weights, and ULMoE re-quantizes them to W2. With *max* reasoning effort, it leads to 0.4 point average drop over five reasoning tasks. For diffusion decoders over four generation tasks including GSM8K (Cobbe et al., 2021), MBPP (Austin et al., 2021), HumanEval (Chen et al., 2021), and MATH500 (Lightman et al., 2023), the average drop is 1.3 points for LLaDA2.0-mini (Bie et al., 2025) and 0.7 for DiffusionGemma (Team et al., 2026a).

Table 2: Downstream accuracy of FP4-native MoE DeepSeek-V4-Flash-0731 with *max* reasoning effort.

Method	AIME24	AIME25	MATH500	GPQA	LCB	Avg
DeepSeek-V4-Flash-0731						
FP4	95.00	98.34	98.80	91.67	93.14	95.39
ULMoE	95.00	98.34	98.40	91.41	92.00	**95.03**

Table 3: Downstream accuracy of diffusion-decoder MoEs.

Method	GSM8K	MBPP	HumanEval	MATH500	Avg
LLaDA2.0-mini					
BF16	92.04	67.20	84.76	76.20	80.05
ULMoE	91.66	65.20	81.10	77.20	**78.79**
DiffusionGemma					
BF16	96.44	67.20	89.02	77.60	82.57
ULMoE	96.21	66.40	90.24	74.60	**81.86**

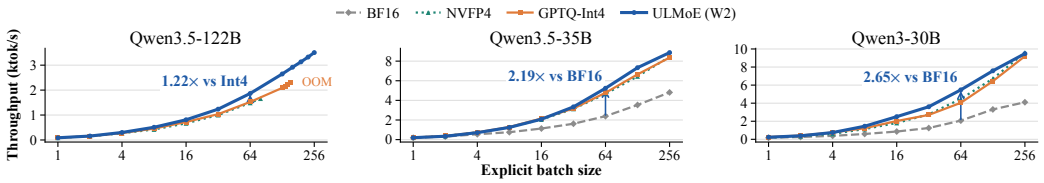

Figure 6: Offline decode throughput at ISL 128 / OSL 1024, single GPU. Arrows mark the speedup at batch 64 against BF16 where BF16 fits and against Int4 otherwise. The 122B Int4 curve ends where it runs out of memory.

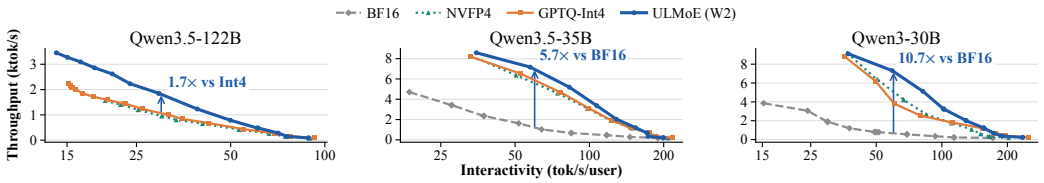

Figure 7: Online throughput-interactivity Pareto frontiers, same workload and hardware. Arrows are read at the interactivity anchors 30 tok/s/user for the 122B, 60 for the 35B and 30B.

4.3 Serving Performance

Offline. Figure 6 sweeps explicit batch size with forced generation length. ULMoE reaches 1.10-1.35× over Int4 and 2.2-2.6× over BF16 at batch 64, with NVFP4 tracking Int4. Memory headroom appears as curve extent: on Qwen3.5-122B, Int4 runs out of memory beyond batch ∼152 while ULMoE continues to 256 at 3504 tokens/s. At the kernel level (Figure 8), our W2A16 grouped GEMM has the lowest median latency at every routed batch size against Marlin W4A16 and FlashInfer NVFP4 W4A4 (Ye et al., 2025), ∼1.4× compared with the strongest baseline.

Online. Offline equal-batch comparison understates what compression buys in deployment. Comparing at matched interactivity (Figure 7) fixes the per-step budget, so a cheaper step serves more concurrent requests and per-step advantage converts into throughput. ULMoE sustains 1.70× the Int4 throughput on the capacity-bound 122B model at 30 tok/s/user, and 1.85× (30B) and 1.17× (35B) at 60. Against BF16 the gains reach 5.7× (35B) and 10.7× (30B). The spread tracks capacity pressure rather than kernel speed.

5 Conclusion

In this paper we present ULMoE, an algorithm-system co-design for ultra-low-bit MoE serving. Exploiting the observation that routed grouped GEMM remains memory-bound over a batch range E/k_r times wider than dense GEMM, ULMoE uses a unified-decode 2.5-bit format with two-level integer scales. Across seven MoE models ULMoE compresses routed experts ∼6× for an averaged drop of 1.8 average accuracy points, and at matched interactivity it delivers 5.7× the serving throughput of BF16 on Qwen3.5-35B and 1.7× that of Int4 on Qwen3.5-122B.

AI USE STATEMENT

Large language models were used to assist with language editing and data visualization, including improving grammar, clarity, and readability, as well as generating and refining plotting code and figure presentation. They were not used to formulate research ideas, design experiments, perform scientific analyses, or draw conclusions. The underlying data, visualization choices, and all AI-assisted outputs were carefully reviewed and verified by the authors. All research contributions, technical claims, interpretations, and conclusions remain entirely the work and responsibility of the authors.

REPRODUCIBILITY STATEMENT

We have made substantial efforts to facilitate the reproduction of the results reported in this paper. To this end, we provide detailed descriptions of the experimental setup, including the model configurations and hardware specifications.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Ling Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. Llada2.0: Scaling up diffusion language models to 100b, 2025. URL https://arxiv.org/abs/2512.15745.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding, 2026. URL https://arxiv.org/abs/2602.06036.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang

Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Jianing Deng, Song Wang, Dongwei Wang, Zijie Liu, Tianlong Chen, Huanrui Yang, and Jingtong Hu. Gemq: Global expert-level mixed-precision quantization for moe llms, 2026. URL https://arxiv.org/abs/2605.23078.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression, 2023. URL https://arxiv.org/abs/2306.03078.

Haojie Duanmu, Xiuhong Li, Zhihang Yuan, Size Zheng, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. Mxmoe: Mixed-precision quantization for moe with accuracy and performance co-design, 2025. URL https://arxiv.org/abs/2505.05799.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization, 2024. URL https://arxiv.org/abs/2401.06118.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

Elias Frantar, Roberto L. Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language models, 2024. URL https://arxiv.org/abs/2408.11743.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. Lighteval: A lightweight framework for llm evaluation, 2023. URL https://github.com/huggingface/lighteval.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Beichen Huang, Yueming Yuan, Zelei Shao, and Minjia Zhang. Milo: Efficient quantized moe inference with mixture of low-rank compensators, 2025a. URL https://arxiv.org/abs/2504.02658.

Wei Huang, Yue Liao, Jianhui Liu, Ruifei He, Haoru Tan, Shiming Zhang, Hongsheng Li, Si Liu, and Xiaojuan Qi. Mixture compressor for mixture-of-experts llms gains more, 2025b. URL https://arxiv.org/abs/2410.06270.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yesheng Liang, Haisheng Chen, Zihan Zhang, Song Han, and Zhijian Liu. Paroquant: Pairwise rotation quantization for efficient reasoning llm inference, 2026. URL https://arxiv.org/abs/2511.10645.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2026. URL https://arxiv.org/abs/2306.00978.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving, 2025. URL https://arxiv.org/abs/2405.04532.

Dhruv Nathawani, Shuoyang Ding, Vitaly Lavrukhin, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Boris Ginsburg, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v2, August 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v2.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

DiffusionGemma Team, Adrien Ali Taïga, James Assiene, Daniele Calandriello, Rahma Chaabouni, João Gante, Tamara von Glehn, Nate Keating, Chris Knutsen, Martin Kukla, Tianlin Liu, Ivan Lobov, Ofir Nabati, João Gabriel Oliveira, Nicolas Perez-Nieves, Nastasia Prutianova, Bobak Shahriari, Jean Tarbouriech, Pavel Tyletski, Çağlar Ünlü, Cindy Wu, Glenn Cameron, Jerome Connor, Sertan Girgin, Maarten Grootendorst, Alon Levkovitch, Eliya Nachmani, Omar Sanseviero, Piotr Stanczyk, Quentin Berthet, Andrew Campbell, Clément Crepy, Valentin De Bortoli, Arnaud Doucet, Romuald Elie, Alexandre Galashov, Klaus Greff, Alexis Jacq, David Ruhe, Yu-Han Wu, Sebastian Flennerhag, Brendan O'Donoghue, George Scrivener, and Shantanu Thakoor. Diffusiongemma technical report, 2026a. URL https://arxiv.org/abs/2608.00146.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, Johan Ferret, Filippo Galgani, Sertan Girgin, Jean-Bastien Grill, Maarten Grootendorst, Jiaxian Guo, Cassidy Hardin, Yanzhang He, Steven M. Hernandez, Omri Homburger, Léonard Hussenot, Juyeong Ji, Armand Joulin, Aishwarya Kamath, Parnian Kassraie, Olivier Lacombe, Preethi Lahoti, Gaël Liu, Gus Martins, Luciano Martins, Tatiana Matejovicova, Ramona Merhej, Nikola Momchev, Sneha Mondal, Ryan Mullins, Sindhu Raghuram Panyam, Shreya Pathak, Sarah Perrin, André Susano Pinto, Etienne Pot, Angéline Pouget, Alexandre Ramé, Sabela Ramos, Douglas Reid, David Rim, Morgane Rivière, Karsten Roth, Louis Rouillard, Omar Sanseviero, Pier Giuseppe Sessa, Shane Settle, Danila Sinopalnikov, Sara Smoot, Piotr Stanczyk, Andreas Steiner, Lawrence Stewart, Ilya Tolstikhin, Michael Tschannen, Anton Tsitsulin, Nino Vieillard, Renjie Wu, Pingmei Xu, Haichuan Yang, Edouard Yvinec, Biao Zhang, Li Zhang, Joe Zou, Nicolas Aagnes, Abdelrahman Abdelhamed, Jakub Adamek, Shivani Agrawal, Shubham Agrawal, Ibrahim Alabdulmohsin, Jean Baptiste Alayrac, Uri Alon, Chandramouli Amarnath, Ankesh Anand, Chrysovalantis Anastasiou, Setareh Ariafar, François-Xavier Aubet, Kyriakos Axiotis, Federico Barbero, Joelle Barral, Alexei Bendebury, Urs Bergmann, Stanley Bileschi, Kat Black, Mathieu Blondel, Sebastian Borgeaud, Arthur Bražinskas, Ryan Burnell, Robert Busa-Fekete, Mu Cai, Daniele Calandriello, Glenn Cameron, Charlotte Caucheteux, Rahma Chaabouni, Garima Chadha, Jetha Chan, Blake Jianhang Chen, Jesse Chen, Lin Chen, Xu Chen, Derek Cheng, Tzu hsiang Chien, Nikolai Chinaev, Yi Chou, Zhaohui Chu, Benjamin Coleman, Pooja Consul, Sam Conway-Rahman, Scott Crowell, Dylan Cutler, Vivek Dani, Samira Daruki, Anil Das, Daniel Deutsch, Nishanth Dikkala, Li Ding, Qiuhan Ding, Shenil Dodhia, Konstantin Donhauser, Tulsee Doshi, Anca Dragan, Alex Druinsky, Sahil Dua, Zoltan Egyed, Danielle Eisenbud, Daniel Eppens, Cindy Fan, Bahare Fatemi, Yassir Fathullah, Vlad Feinberg, Milen Ferev, Sebastian Flennerhag, Takumi Fujimoto, João Gabriel Oliveira, Isaac Galatzer-Levy, João Gante, Simon Geisler, Soham Ghosal, Antonious M. Girgis, Tamara von Glehn, Alec Go, Alhaad Gokhale, Alex Grills, Yiming Gu, Mayank Gupta, Pramod Gupta, Guru Guruganesh, Raia Hadsell, Hamza Harkous, Jitendra Harlalka, Demis Hassabis, Anja Hauth, Joe Heyward, Arian Hosseini, Chih-Yang Hsia, I-Hung Hsu, Xiaopeng Huang, Yangsibo Huang, Kevin Hui, Adrian Hutter, Te I, Fotis Iliopoulos, Advait Jain, Ganesh Jawahar, Ziwei Ji, Qilin Jin, Melvin Johnson, Kandarp Joshi, Arun Kandoor, Wang-Cheng Kang, Koray Kavukcuoglu, Mehran Kazemi, Kathleen Kenealy, Amr Khalifa, Phoebe Kirk, Ivan Korotkov, Suraj Kothawade, Vitaly Kovalev, Neel Kovelamudi, Adam Kraft, Ravin Kumar, Vivek Kumar, Harish Kuppam, Justin Lannin, Chen-Yu Lee, Seungji Lee, Dmitry Lepikhin, Alon Levkovitch, Dongdong Li, Qiujia Li, Valentin Liévin, Ethan Lin, Ziqian Lin, Casper Liu, Tianlin Liu, Tianqi Liu, Xin Liu, Ivan Lobov, Mayank Lunayach, Min Ma, Gagan Madan, Andrii Maksai, Eric Malmi, Michal Matuszak, Daniel McDuff, Gaurav Menghani, Maciej Mikuła, Daniil Mirylenka, Karolis Misiunas, Vedant Misra, Andreea Mitran, Kareem Mohamed, Maksim Mukha, Eric Noland, James O'Donnell, Brendan O'Donoghue, Kate Olszewska, Bernett Orlando, Wanqiong Pan, Rina Panigrahy, Unnati Parekh, Nicolas Perez-Nieves, Chunjong Park, Eric Paskie, Liqian Peng, Bryce Petrini, Slav Petrov, Jonas Pfeiffer, Bilal Piot, Martyna Plomecka, Siim Poder, Octavio Ponce, Arijit Pramanik, David Racz, Anish Rajan, Michelle Ramanovich, Anand Rao, Marvin Ritter, Vitor Rodrigues, Evan Rosen, Mikołaj Rybiński, Noveen Sachdeva, Michaël E. Sander, Rohit Sathyanarayana, Sagar Savla, Samuel Schmidgall, Tal Schuster, George Scrivener, Benoit Seguin, Andrew Sellergren, Aliaksei Severyn, Izhak Shafran, Dhruv Shah, Bobak Shahriari, Yuan Shangguan, Ashish Shenoy, Pradeep Shenoy, Rakesh Shivanna, Pauline Sho, Lucas Spangher, Wojciech Stokowiec, Tim Strother, Yao Su, Yinghao Sun, Mukund Sundararajan, Andrea Tacchetti, Mor Hazan Taege, Pouya Tafti, Jean Tarbouriech, Chetan Tekur, Shantanu Thakoor, Rahul Thapa,

Madeleine Traverse, Lenart Treven, Tao Tu, Chien Te Tung, Çağlar Ünlü, Petar Veličković, Malini Pooni Venkat, Sagar Gubbi Venkatesh, Vidya Venkiteswaran, Francesco Visin, Alex Vitvitskyi, Kiran Vodrahalli, Weiyi Wang, Xin Wang, Tris Warkentin, Jan Wassenberg, John Wieting, Cindy Wu, Lechao Xiao, Hao Xu, Yuhui Xu, Fuzhao Xue, Arun Yadav, Jun Yan, Antoine Yang, Lin Yang, Ming-Hsuan Yang, Ziyu Ying, Jae Hyeon Yoo, Morteza Zadimoghaddam, Sajjad Zafar, Fred Zhang, Jiageng Zhang, Jianyi Zhang, Xiaofan Zhang, Chao Zhao, David Zhou, and Chen Zou. Gemma 4 technical report, 2026b. URL https://arxiv.org/abs/2607.02770.

GLM-4.5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks, 2024. URL https://arxiv.org/abs/2402.04396.

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, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving, 2025. URL https://arxiv.org/abs/2501.01005.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang:

Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

A ULMoE Implementation Details

Packed layout. The metadata word β of Algorithm 2, one per macro-group of each output channel of each expert in `meta` $[E, K/64, N]$, holds the ladder selector in bit 0, the macro scale code m in bits 1–7, the eight micro-group zeros in bits 8–15 with z equal to the bit plus one, and the eight 2-bit multiplier offsets in bits 16–31, with $s_{\text{int}} = 2 + 2C_{\text{idx}} + \text{offset}$ recovering $\mathcal{S}(5) = \{2, \ldots, 5\}$ and $\mathcal{S}(7) = \{4, \ldots, 7\}$. The `qweight` $[E, K/16, N]$ stores sixteen 2-bit codes per word, code i at bit $2i$. Before export, a one-time permutation of these bit positions moves the two codes of decode pair p to bits $2p$ and $16{+}2p$, which places them in the two halves of the word and in the same micro-group, so the kernel reaches both with one shift and one packed mask.

Algorithm 2 Dequantizing one macro-group inside the W2A16 grouped GEMM.

Input: code words $P_0, \ldots, P_3$ and metadata word β of one macro-group
Output: weights $\hat{w}_1, \ldots, \hat{w}_{64}$ up to the anchor s_o

1: $C_{\text{idx}} \leftarrow \beta \,\&\, 1, \quad m \leftarrow (\beta \gg 1) \,\&\, \texttt{0x7F}$ ▷ *the macro-group fields*
2: **for all** code words $w \in \{0, \ldots, 3\}$ and pairs $p \in \{0, \ldots, 7\}$ **do**
3: $\quad j \leftarrow 2w + \lfloor p/4 \rfloor + 1$ ▷ *the micro-group holding both codes*
4: $\quad z_j \leftarrow \big((\beta \gg (7 + j)) \,\&\, 1\big) + 1$ ▷ $z_j \in \mathcal{Z}$
5: $\quad s_j \leftarrow 2 + 2C_{\text{idx}} + \big((\beta \gg (14 + 2j)) \,\&\, 3\big)$ ▷ $s_j \in \mathcal{S}(C^\star)$
6: $\quad \textit{lanes} \leftarrow \texttt{lop3.b32}\big(P_w \gg 2p,\ \texttt{0x00030003},\ \texttt{0x43004300},\ \texttt{0xEA}\big)$
7: $\quad \textit{lanes} \leftarrow \texttt{sub.bf16x2}\big(\textit{lanes},\ \text{dup}(\texttt{0x4300} + z_j)\big)$ ▷ *each lane* $q - z_j$
8: $\quad h \leftarrow \texttt{cvt.rn.bf16.s32}(s_j\, m)$ ▷ *one conversion per pair*
9: $\quad \textit{lanes} \leftarrow \texttt{mul.rn.bf16x2}\big(\textit{lanes},\ \text{dup}(h)\big)$ ▷ *each lane one* $\hat{w}$
10: $\quad \hat{w}_{16w+2p+1},\ \hat{w}_{16w+2p+2} \leftarrow$ the two lanes
11: **end for**

Efficient dequantization operations. Algorithm 2 illustrates the dequantization the grouped GEMM runs, and the permutation above lets it skip converting the codes. Expanding a code the ordinary way means shifting it down, masking it, converting the integer to BF16, subtracting the zero and multiplying by the scale: five operations per weight and one trip through the conversion pipe. Instead the kernel writes codes where a BF16 value already keeps its integers. On $[128, 256)$ the exponent is fixed and the unit in the last place is exactly 1, so the low mantissa bits of $\texttt{0x4300} = \text{BF16}(128)$ hold a small integer directly. One `lop3.b32` masks two 2-bit codes and injects them into two BF16 lanes at once (line 6), one `sub.bf16x2` yields $(q - z)$ for both (line 7), and after the macro-group's integer product is converted once, one `mul.bf16x2` finishes both weights (line 9). Per pair that is six operations and one conversion against the ordinary path's ten and two. The offline bit shuffle above allows it, since it places a decode pair at bits $2p$ and $16{+}2p$ so that one shift positions both.

The conversion in line 8 is not exact integer arithmetic. BF16 carries an eight-bit significand, so it represents consecutive integers exactly only up to 256, then every second integer up to 512 and every fourth one up to 1024, while the maximum of product $s_j m$ reaches $7 \times 127 = 889$. The largest relative error of the BF16 rounding is 0.386%, at $s_j m = 259$. All of our accuracy and serving evaluations are done with real quantized checkpoints, and the small relative error leads to no change on downstream accuracy performance since it's much smaller than the weight rounding error.

Cross-expert shared output scales. Algorithm 1 computes one anchor per logical output-channel index, as the largest continuous macro scale over all macro-groups and experts sharing that index, taken over the weights and the provisional residual together, divided by 127 and stored in FP16 (lines 6–8). Dividing by 127 maps that largest scale to the top 7-bit code, and because the residual

enters the same maximum, one s_o serves both the base and the residual columns. A macro-group whose scale rounds to $m{=}0$ decodes to zero, since decode multiplies by m, and the quantizer floors its step at a small ϵ so the search stays defined.

Effective bit-width per model. The format itself costs 2.5 bits per weight, and the only thing that varies across models is how the fixed residual budget amortizes. A projection with input dimension K is extended to $K + k$ columns of the same format, so it costs $2.5\,(K{+}k)/K$ bits per weight, with $k{=}128$ for all models. Gate/up has $K = H$ and $2IH$ weights while down has $K = I$ and IH weights, so the model-level figure is the parameter-weighted average of the two,

$$\text{average bits} \;=\; 2.5 \cdot \frac{2I\,(H+k) + H\,(I+k)}{3IH},$$

which depends only on the architecture and not on the data. Across the evaluated models this gives 2.6 (GLM4.5-Air and DeepSeek-V4-Flash), 2.7 (GLM4.7-Flash, Qwen3.5-122B, Gemma4-26B, DiffusionGemma, and Qwen3-30B-2507) and 2.8 (LLaDA2.0-mini, Qwen3.5-35B, and Qwen3.6-35B) bits per weight. The shared s_o vectors and the gate/up permutation add $O(N{+}H)$ further entries per layer, independent of E and negligible at scale.

Offline calibration. Algorithm 3 is the whole offline quantization pass. It walks the decoder in depth order on activations produced by already-quantized previous layers, so each layer is calibrated against the input distribution it will actually see, and it holds one layer's replay rather than a full forward, which keeps 512-sequence calibration of a 122B MoE within a single GPU. Down salience is measured under a temporary quantization of gate/up so that the statistic reflects propagated error rather than the BF16 forward.

Calibration uses 512 sequences of 2048 tokens drawn with a fixed seed from a general post-training instruction corpus. Quantizer settings are identical across all models and experiments: macro/micro groups $64/8$, ladder candidates $\mathcal{C} = \{5, 7\}$, zeros $\mathcal{Z} = \{1, 2\}$, residual budget $k{=}128$, with no per-model hyperparameter tuning. On a single RTX PRO 6000 GPU, quantizing the routed experts takes 37 minutes for Qwen3.5-35B and 77 minutes for Qwen3.5-122B.

Algorithm 3 ULMoE quantization pipeline.

Input: decoder layers $f_1, \ldots, f_L$, calibration batches $\mathcal{D}$, residual budget k
Output: `qweight`, `meta`, `oscale` per expert projection, `perm` per MoE layer

$\mathcal{H} \leftarrow$ inputs of f_1 over $\mathcal{D}$, cached with each layer's call arguments ▷ *one recording pass*
for $\ell = 1$ to L **do**
  if f_ℓ has no routed experts **then**
    $\mathcal{H} \leftarrow f_\ell(\mathcal{H})$ **and continue**
  end if
  $\mathcal{X} \leftarrow$ inputs of the MoE block of f_ℓ over $\mathcal{H}$
  $\pi \leftarrow \text{argsort}^{\downarrow} \sum_{x \in \mathcal{X}} |x|$ ▷ *gate/up salience, one order for all experts*
  $\tilde{W}^{gu} \leftarrow \textsc{Quantize}\big(W^{gu}[\cdot, \cdot, \pi],\, k\big)$ with π undone ▷ *Algorithm 1, temporary copy*
  for $e = 1$ to E **do**
    $\mathcal{X}_e \leftarrow$ tokens of $\mathcal{X}$ routed to e, with routing weights $p_e(x)$
    $\rho_e \leftarrow \text{argsort}^{\downarrow} \sum_{x \in \mathcal{X}_e} |p_e(x)| \cdot \big|\phi(g) \odot u\big|$, where $[g; u] = \tilde{W}^{gu}_e x$
    reorder the rows of W^{gu}_e and the columns of W^{d}_e by ρ_e
  end for
  $\big(\texttt{qweight}, \texttt{meta}, \texttt{oscale}\big)^{gu} \leftarrow \textsc{Quantize}\big(W^{gu}[\cdot, \cdot, \pi],\, k\big), \quad \texttt{perm} \leftarrow \pi$
  $\big(\texttt{qweight}, \texttt{meta}, \texttt{oscale}\big)^{d} \leftarrow \textsc{Quantize}\big(W^{d},\, k\big)$ ▷ *no runtime order for down*
  $\mathcal{H} \leftarrow f_\ell(\mathcal{H})$ with its experts quantized ▷ *error propagation*
end for
pair-align every `qweight` for the kernel and export the checkpoint

B Additional Serving and Kernel Results

Kernel microbenchmarks. Figure 8 isolates the routed grouped GEMM on Qwen3.5-122B expert shapes ($E{=}256$, top-8 routing) on an RTX PRO 6000 GPU, comparing our W2A16 kernel against

Marlin MoE W4A16 and FlashInfer SM120 NVFP4 (the fastest W4A4 backend on SM120) on two projection shapes from Qwen3.5-122B. Latency is measured with a cold L2 cache, achieved by flushing it before each timed iteration. Each point is the median of 50 timed iterations after 15 warmup iterations, repeated over 3 data seeds, timed with CUDA events. W2A16 has the lowest median latency at every routed batch size $M \in [1, 512]$. Converting the same measurements to effective throughput, it reaches 100 TFLOPS at M=512 where the strongest baseline reaches ∼72, a ∼1.4× speedup over the Marlin Int4 kernel.

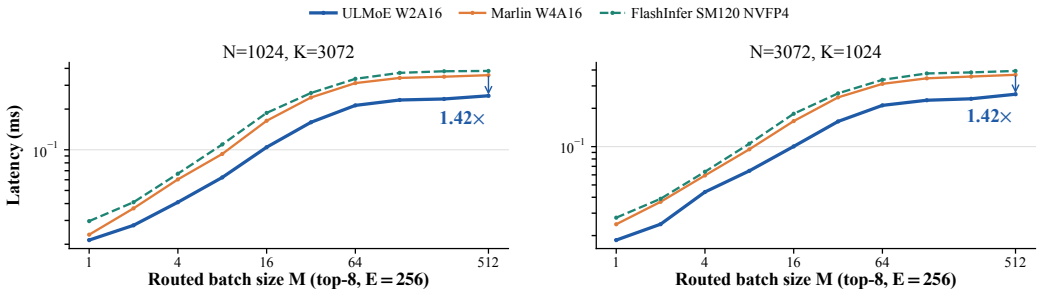

Figure 8: Grouped GEMM microbenchmarks on RTX PRO 6000.

Qwen3-235B under tensor parallelism. Figure 9 reports serving performance of Qwen3-235B on a system with two RTX PRO 6000 GPUs (TP2, ISL 128 / OSL 1024). In the offline benchmark, ULMoE reaches 3390 tokens/s at batch 256 versus 2560 for Int4 (1.3×). In the online benchmark, the envelope gap reaches 2.6× at 30 tok/s/user of interactivity.

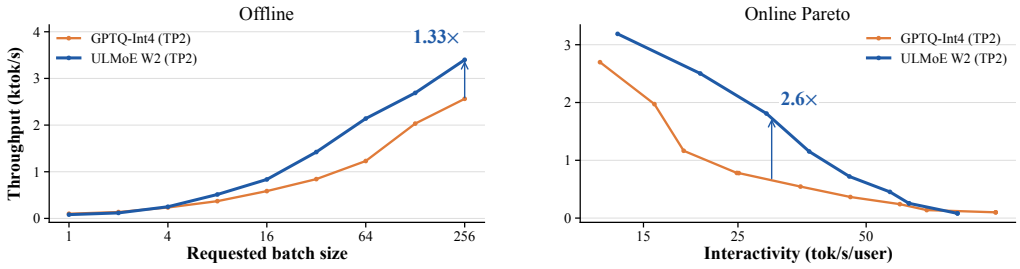

Figure 9: Qwen3-235B under TP2 (ISL 128 / OSL 1024). Offline explicit-batch throughput (left) and online Pareto envelope (right).

Long-context workload. Figure 10 repeats offline and online serving evaluation protocols at ISL 1024 / OSL 10240, where the KV cache competes with weights for memory. We set the context length to 16384 for this workload. On Qwen3.5-122B, Int4 cannot allocate beyond batch ∼48 (1120 tokens/s) while ULMoE continues to batch ∼112 at 2000 tokens/s, and its online envelope reaches ∼1.6× the Int4 throughput at matched interactivity.

Composition with speculative decoding. Speculative decoding is an optimization orthogonal to quantization, and we evaluate the efficiency of composing the two. We pair ULMoE and GPTQ-Int4 with DFlash (Chen et al., 2026), a block diffusion draft that proposes 8 tokens per step and verifies them in one target pass, and repeat the protocols of Section 4.1 on Qwen3.5-35B and Qwen3.5-122B with MATH500 prompts and natural stopping. Accepted length is 5.66 to 6.00 out of 8 for both quantization methods at both scales and across the whole batch range, so the draft is indifferent to the quantization format, and speculation leaves the output unchanged. Figure 11 reports offline and online serving results for the two models. On Qwen3.5-35B, ULMoE leads by 1.17× at concurrency

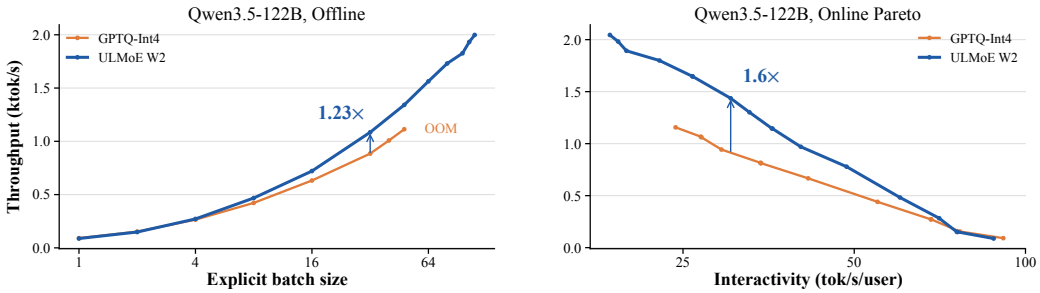

Figure 10: Long-context serving (ISL 1024 / OSL 10240) on Qwen3.5-122B. Offline explicit-batch generation (left) and online Pareto frontier (right). On the 122B model Int4 cannot allocate beyond batch $\sim$48.

32 offline, and the lead widens to 1.24× at 200 tok/s/user online. On Qwen3.5-122B it leads by 1.31× at concurrency 8 offline, and widens to 1.95× at 88 tok/s/user.

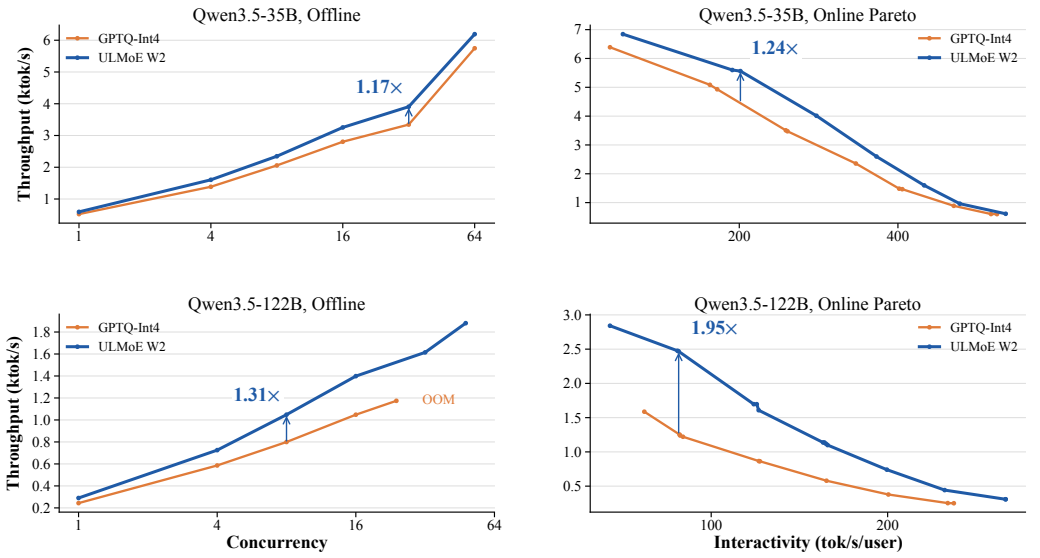

Figure 11: DFlash block speculative decoding (block 8, MATH500 prompts): Qwen3.5-35B (top) and Qwen3.5-122B (bottom), offline explicit-concurrency generation (left) and online Pareto frontier (right).

Extended serving evaluation on NVIDIA L40S GPU. Every measurement above uses an RTX PRO 6000 GPU. To further validate the efficiency of ULMoE, we extend the evaluation to an L40S GPU. Packed BF16 arithmetic requires SM90 or later, so the L40S with SM89 build of the kernel runs the same sequence on FP16 lanes. There $\mathrm{FP16}(1024) = \texttt{0x6400}$ takes the role of `0x4300`, since FP16 also has unit spacing over $[1024, 2048)$, and the mask immediate becomes `0x64006400` with `sub.f16x2` and `mul.rn.f16x2` in place of their BF16 forms. The instruction count per pair is unchanged.

Figure 12 shows the grouped-GEMM microbenchmark on an NVIDIA L40S, with 48 GB of GDDR6 at 864 GB/s and 362 dense BF16 TFLOPS. Machine balance rises to $R \approx 419$, which widens the memory-bound region, moving the dense knee at 2.5 bits to $M \approx 65$ and the MoE knee to $M \approx 1047$ at $E/k_r = 16$ and $M \approx 2095$ at $E/k_r = 32$, so the regime the format targets still covers

the whole batch range these models are served at. W2A16 has the lowest median latency at every batch size, running 1.33 to 1.54× faster than Marlin W4A16. The average speedup over Marlin is 1.42×.

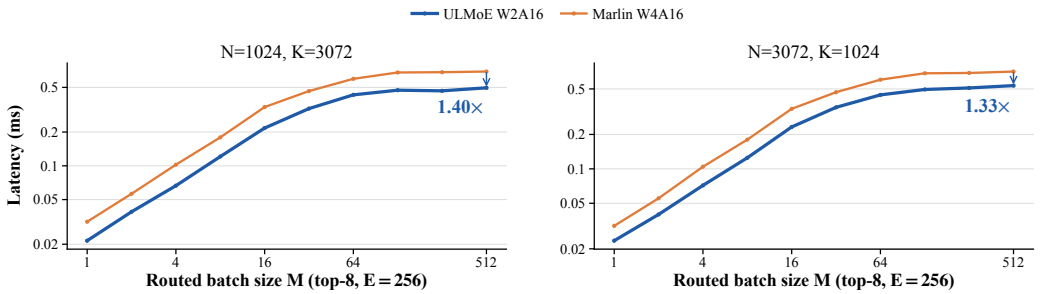

Figure 12: Grouped GEMM microbenchmarks on an NVIDIA L40S, same protocol as Figure 8.

Figure 13 runs both protocols of Section 4.1 on the same board against GPTQ-Int4 expert quantization of each model. On Qwen3-30B-A3B, ULMoE leads offline at every batch size, by 1.07 to 1.37× with an average of 1.20×. Online, the lead widens to 1.81× at the 60 tok/s/user anchor (2332 against 1288 tokens/s).

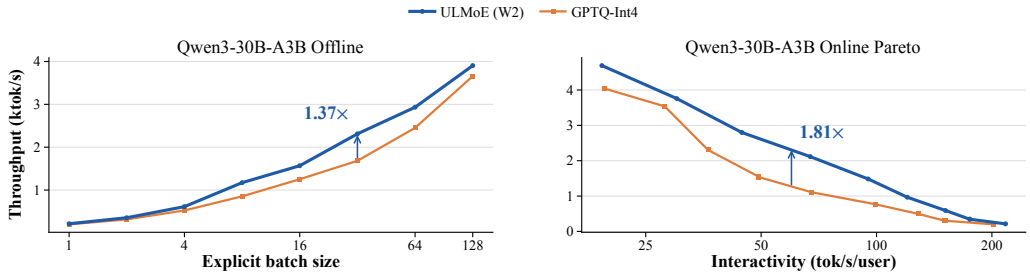

Figure 13: Qwen3-30B-A3B serving on one L40S. Offline explicit-batch throughput (left) and online Pareto envelope (right), same workload and protocol as Section 4.1.

C Ablation Study

Quantization format. We ablate the design of residual columns and ladder choice C. Table 4 removes one component of the format at a time and re-runs the whole calibration pipeline: (a) drops the residual columns entirely (k=0), leaving every input channel at two bits. Variants (b) and (c) replace the per-macro-group ladder selector with a single fixed ladder, C=7 and C=5 respectively, which keeps the bit budget but removes the format's ability to match each macro-group's dynamic range. The results show that the residual columns better preserve the precision of salient channels, while the choice of ladder C enhances the format's ability to match each macro-group's dynamic range.

Calibration data size. Calibration data enters ULMoE quantization pipeline only through the two orderings of Algorithm 3, because the search of Algorithm 1 reads the weights alone. Table 5 varies the corpus from 128 to 512 sequences at the fixed length of 2048 tokens. Accuracy declines slightly as the corpus shrinks and stays close to the default throughout, so the recipe remains robust against different calibration data sizes.

Kernel design. The kernel ablation isolates the one decision that is not forced by the format: whether the 32 metadata bits of a macro-group are addressed as a single aligned word or as separate

Table 4: Ablation study. Each row modifies one component of the full recipe and re-runs the complete calibration pipeline. Cells average the non-reasoning and the reasoning tasks of Table 1.

	Qwen3.5-35B		Gemma4-26B	
Variant	Non-reasoning	Reasoning	Non-reasoning	Reasoning
Full recipe (k=128, $C \in \{5, 7\}$)	**89.27**	**84.89**	**89.59**	**83.78**
(a) k=0, no residual columns	86.20	80.75	87.13	80.82
(b) single ladder C=7	87.98	83.44	86.31	82.29
(c) single ladder C=5	87.28	83.19	86.55	81.88

Table 5: Ablation of calibration data size on Qwen3.5-35B, at the sequence length of 2048 tokens used throughout. Cells average the non-reasoning and the reasoning tasks of Table 1.

Calibration set	Non-reasoning	Reasoning
512 sequences (default)	89.27	**84.89**
256 sequences	89.11	84.30
128 sequences	**89.34**	83.80

fields. Both variants carry the same 32 bits per macro-group, which is 0.5 bits per weight against the 2.3 that a conventional scale and zero would cost at g=8, and both move the same metadata bytes. The unpacked variant splits the bits across three tensors, each contiguous along the output dimension. Its inner loop issues three loads, three index computations and three live destination registers where the packed kernel issues one load and recovers every field from one resident register by a shift and a mask against immediate operands. The cost is therefore paid in issue slots, address arithmetic and register pressure rather than in bandwidth. Table 6 measures both variants on the shapes of Figure 8. The packed one runs 1.06 to 1.27× faster, 1.13× on average. The two variants decode the same weights, since feeding the unpacked kernel three tensors reproduces the packed output bit for bit.

Table 6: Speedup of the packed metadata layout compared with the unpacked metadata layout, measured on an RTX PRO 6000.

Batch M	N=1024, K=3072	N=3072, K=1024
16	1.27×	1.15×
64	1.11×	1.06×
256	1.11×	1.07×

D The Landscape of MoE Applicable Quantization

Table 7 summarizes the approaches discussed in Section 2 by bit allocation, serving path, and the obstacle that keeps each from converting compression into MoE serving throughput. The two nearest to ULMoE are the two that already use a second scale level. SpQR (Dettmers et al., 2023) quantizes weight groups to 3 bits under a 16-bit second level, and protects the residue with an unstructured sparse set served by a CSR kernel. QServe (Lin et al., 2025) makes the second level an integer against a per-channel float so that dequantization survives the main loop of a W4A8 GEMM. ULMoE differs from both in three ways that the two-bit MoE setting forces. Its hierarchy has two integer scaling levels under a shared floating-point anchor, and its top level is shared across the expert dimension, which has no analogue in a dense matrix and is what keeps scale storage at $O(N)$ as E grows. And its protected channels are ordinary columns of the same format rather than a sparse side-channel, so one launch and one decode path still cover every expert.

Table 7: Comparison of quantization approaches applicable to MoE experts.

Method	Bit allocation	Serving path	Key obstacle
Uniform group-wise Int4 (Lin et al., 2026)	uniform per group	real grouped kernel	accuracy floor at four bits, 16-bit scales per group
Group-wise Int3 with LoRA (Huang et al., 2025a)	uniform per group with low rank branches	bespoke kernel, padded or word-straddling packing	zero-waste packing costs more complex packed-lane decode, low-rank branch leads to extra launch overheads
Expert-level mixed precision (Deng et al., 2026; Huang et al., 2025b)	1–4 bits *per expert*, allocated by a model-wide linear program	one Triton launch over the selected experts, bit-width read per expert at run time	a run-time bit-width forces per-expert stride tables in place of one uniform 3-D tensor and a per-element division, shift and mask in the main loop, and the fused path is written for batch one
Sparse-quantized, two-level scales (Dettmers et al., 2023)	3–4 bits, 3-bit statistics per 8–32 weights, 16-bit statistics above them	dense quantized matmul *plus* a CSR sparse matmul	outliers are unstructured, so the grouped GEMM gains a second heterogeneous path
Integer second-level scale (Lin et al., 2025)	4 bits, per-channel float scale $\times$ per-group integer scale	real W4A8 kernel, dequant kept in the main loop	two levels suffice at four bits but not at two, first level is per matrix so tables still grow with E
Codebook / vector quantization (Tseng et al., 2024)	$\sim$2 bits + codebook	table lookups in GEMM inner loop	lookup-bound decode on small expert tiles
Rotation-based (Liang et al., 2026)	2–4 bits, rotated basis	extra transforms at layer boundaries	rotated basis must be materialized or absorbed at layer boundaries
ULMoE (ours)	2-bit uniform with compressed and hardware-friendly metadata + in-format top-k residual	single grouped kernel, uniform across experts	–

E Implementation and Evaluation Details

Baseline configurations. AWQ uses activation statistics to scale salient input channels so that their precision is better preserved. We implement AWQ using 2-bit weight codes with group-wise scales and zero points and a group size of 32, yielding an average bit-width of 2.6, matching the bitwidth of ULMoE for fair comparison. GEMQ (Deng et al., 2026) is the strongest expert-level mixed-precision baseline. It runs a GPTQ backbone under a global linear program that allocates a bit-width to every expert in the model at once, scoring experts by their contribution to end-to-end quantization error rather than layer by layer, and it iterates importance estimation and allocation to a fixed average budget, which we set to the same average bit-width as ULMoE for each model. It then finetunes the routers against the quantized experts. Both baseline methods consume the same calibration set as ULMoE. Serving baselines (GPTQ-Marlin Int4, NVFP4, BF16) run in the same engine version with identical scheduler, attention backend and workload settings.

Accuracy evaluation. Accuracy benchmarks use each task's standard protocol (MMLU and GSM8K 5-shot, the remainder zero-shot), scored with LM-EVALUATION-HARNESS V0.4.10 for the non-reasoning tasks and LIGHTEVAL V0.13.0 for the reasoning tasks. Each reasoning task result is the mean of four runs with sampling seeds 0-3 under fixed generation settings shared across all methods for a given model. The reasoning models are sampled with temperature $= 1.0$, `top_p` $= 0.95$, `top_k` $= 20$, `min_p` $= 0.0$, and presence penalty $= 1.5$, with a maximum output length of 64k tokens per prompt. Harness versions, prompt templates and seeds are pinned across methods.

For the diffusion-decoder models, the evaluation uses each model's released inference procedure. Both decode greedily at the harness level, with temperature 0.0, `do_sample` false. LLaDA2.0-mini uses the `multi_bd` strategy with naive sampling, a block and page size of 32, buffer size 1, up to 1024 denoising steps, and a model length of 4096. DiffusionGemma uses block and page size 256, buffer size 1, 48 denoising steps, and a model length of 262144. Its entropy-bound denoising is internally stochastic: $t \in [0.4, 0.8]$, entropy bound 0.1, confidence threshold 0.005, stability threshold two consecutive stable steps, at most 48 denoising steps and a repetition run of 32. Output budgets are 8192 generated tokens for GSM8K and 2048 for HumanEval, MBPP and MATH500 on LLaDA2.0-mini, where the engine's 4096 model length and `max_new_tokens` cap prompt plus output at 4096 regardless. DiffusionGemma allows 8192 for GSM8K and 4096 for the other three.

Serving evaluation. Kernel microbenchmarks and single-GPU serving experiments run on an NVIDIA RTX PRO 6000 GPU. The Qwen3-235B experiment uses two such GPUs under tensor parallelism. The serving engine is a mainstream open-source stack SGLang v0.5.15. Offline experiments force full generation length ($\mathrm{ignore_eos}$). Online experiments are closed-loop. For each value of the server's cap on concurrently running requests, $\mathrm{MRR} \in \{32, 64, 128, 256\}$, a fresh server is launched, so allocator state cannot leak across caps, and the SGLang benchmark client then holds a fixed number of requests in flight, $N \in \{1, 2, 4, \ldots, 256\}$, sending a new request as soon as one completes. Each request uses a random prompt of 128 tokens, with generation configured for 1024 output tokens. Every point is the median over three repeats. The envelope is the set of points that no other point dominates in both throughput and interactivity, taken over all caps and concurrencies. On the capacity-bound Qwen3.5-122B the cap ladder stops where memory runs out, at 152 for Int4, and the concurrency ladder adds intermediate points near that limit.

Interactivity is $1000/\mathrm{TPOT}_{\mathrm{p50}}$: within a repeat the per-user decode rate is taken from that repeat's median time per output token, and the value reported for a point is the median over repeats. The interactivity anchors used for cross-format ratios (60 tok/s/user for the 30B and 35B models, 30 for the 122B) were fixed from each model's operating range before any ratio was computed, rather than taken as a maximum over the envelope. Tensor parallelism is one for every single-GPU point, context length 2048, static memory fraction 0.9, and the radix prefix cache is disabled so that a repeated prompt cannot be served from cache. Activations are BF16 for both formats, with the mamba state also BF16 on the hybrid models. Decode CUDA graphs are captured normally on both sides, over the batch ladder up to the server's running-request cap. The page size is 64. The attention backend is FlashInfer for the Qwen families, including the linear-attention path on the hybrid Qwen3.5 models. Gemma4 uses the Triton attention backend. Offline points drive an in-process engine at an explicit batch size, so all requests in a batch start together and generation length is forced. Online points drive a launched server through its HTTP endpoint with `request-rate=inf` and a fixed in-flight cap, which is continuous batching under a closed loop.