BITLIFT: UNIFIED MIXED-PRECISION QUANTIZATION AND CACHING FOR EFFICIENT MOE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) models reduce computation through sparse activation, but moving expert weights can dominate decoding when the model exceeds device memory. Dynamic mixed precision and DRAM-caching can reduce this cost, yet their effectiveness depends on how weight representation, precision allocation, and cache residency interact. We introduce BITLIFT, a training-free framework that co-designs these components. A router-rank policy dynamically allocates a fixed nominal precision budget across selected experts. Each expert is encoded as a 2-bit base and two 1-bit refinements, jointly optimized for the target policy through weighted scale fitting and error compensation. A two-pool cache separates base and refinement planes and applies cache-resident precision refinement without fetching additional weights for an upgrade. Across three MoE model families, rank-assigned execution approaches independently optimized 3-bit quality at nominal budgets of 2.333–2.375 bits per routing slot, and outperforms the evaluated 2.5-bit baselines. On Qwen3-30B-A3B, cache-resident refinement matches uniform 4-bit WikiText-2 perplexity with a cache holding 50% of the 4-bit routed-expert footprint. At smaller matched capacities of 6.25–25%, it improves on uniform 3-bit perplexity. In this flash-limited regime, our analytical hardware model estimates a throughput gains of 1.17–$1.26\times$ over uniform 3-bit and 2.06–$2.49\times$ over uniform 4-bit inference.

1 INTRODUCTION

Mixture-of-Experts (MoE) models scale capacity while activating only a small subset of experts per token (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). *Conditional computation, however, is not conditional storage*: the full expert pool must remain available during inference. When it exceeds fast memory, missing expert weights are fetched from a slower tier, such as CPU memory or NVMe storage on servers, or flash on edge devices (Federici et al., 2025). At small decoding batch sizes, these transfers are difficult to amortize and can dominate inference latency (Tang et al., 2024; Skliar et al., 2025; Choi et al., 2025). Quantization (Nagel et al., 2021) reduces expert footprints and bytes transferred per miss, while caching retains weights for reuse. Static mixed precision fixes expert bit widths using calibration statistics (Chitty-Venkata et al., 2025; Duanmu et al., 2025; Deng et al., 2026), whereas dynamic methods adapt precision to token-dependent expert importance during inference (Wang et al., 2025; Tang et al., 2024; Choi et al., 2025).

Efficient dynamic-precision inference depends on the joint design of precision allocation, weight representation, and caching. HOBBIT (Tang et al., 2024) lacks sharing across precision levels; supporting 2-, 3-, and 4-bit execution would require nine payload bits per weight. SliceMoE (Choi et al., 2025) avoids duplication through shared bit slices and zero-point-aware truncation, but fits its representation independently of the execution policy. D^2MoE (Wang et al., 2025) combines a learned precision controller with sequential residual quantization, fixing the low-bit base before its refinements without jointly balancing accuracy across the policy's precision levels. At runtime, HOBBIT uses router-derived importance to choose the precision of experts loaded on cache misses, whereas SliceMoE uses gate-based requests for weight slices and predefined cache-retention priorities. These designs do not explicitly link representation fitting and cache allocation through a common precision prior, or specify resident-only upgrades beyond nominal requests that preserve fetch requests and replacement state.

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.

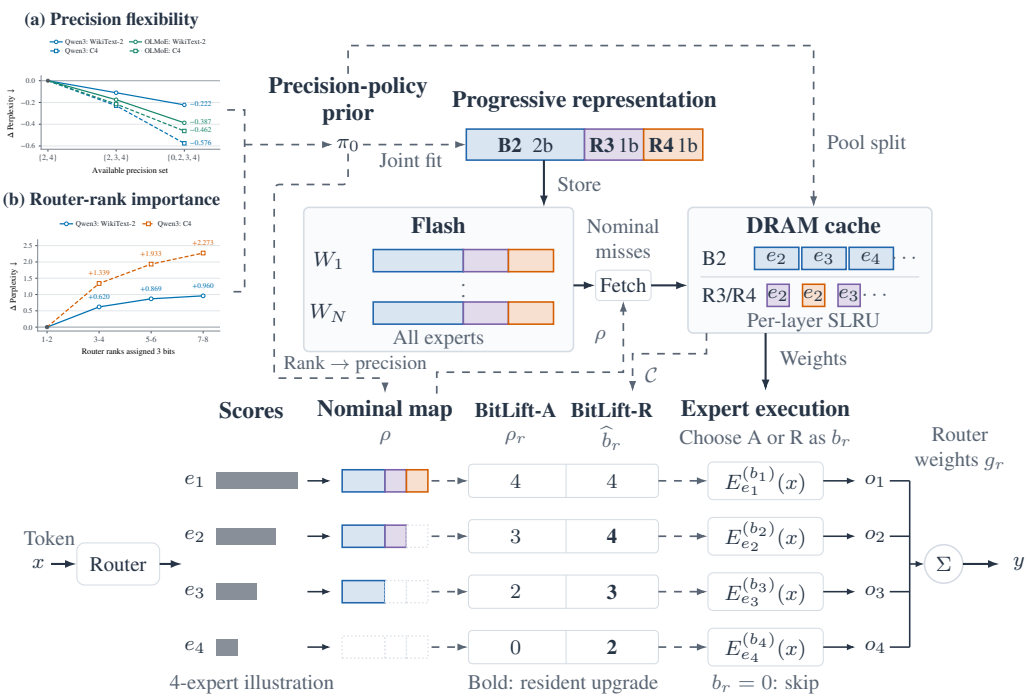

Figure 1: **BITLIFT overview.** Insets motivate the prior: (a) richer precision choices improve quality at a fixed bit budget; (b) higher-ranked experts benefit more from higher precision. The prior guides representation fitting, cache allocation, and nominal requests. BitLift-A executes nominal precisions; BitLift-R upgrades or revives experts using resident planes, without extra flash fetches or replacement updates. Solid arrows carry data; dashed arrows show policy dependencies.

We introduce BITLIFT, a training-free framework aligning quantization and caching through a fixed rank-to-precision prior that enforces a nominal average-bit budget (Figure 1). The prior guides policy-weighted scale fitting and error compensation to jointly optimize 2-, 3-, and 4-bit prefixes from a shared 2-bit base and two 1-bit refinements. It also guides allocation between dedicated base and shared refinement pools according to their object sizes and request patterns.

We introduce two variants of BITLIFT: BITLIFT-A executes the nominal assignments, while BITLIFT-R extends nominal prefixes with resident planes at runtime, raising execution precision without additional flash transfers for upgrades or changes to replacement state. We also combine BITLIFT with cache-aware routing (Skliar et al., 2025), which favors resident experts to reduce flash traffic and transfer latency. Quality recovered through refinement can support stronger routing boosts, reducing traffic at a given quality target.

Across Qwen3, OLMoE, and DeepSeek-V2-Lite, BITLIFT-A approaches uniform 3-bit GPTQ quality at 2.333–2.375-bit nominal average expert precision, with 16–29% less WikiText-2 flash traffic at matched, constrained cache budgets. BITLIFT-R further improves WikiText-2 perplexity and five-shot MMLU accuracy across tested capacities, with nearly unchanged traffic. On Qwen3, perplexity is comparable to uniform 4-bit with half the 4-bit routed-expert footprint cached. At 6.25–25% of that footprint, it outperforms uniform 3-bit GPTQ in perplexity, remaining flash-bound in our expert-weight throughput model, with analytical throughput ceilings 1.17–1.26× the baseline. Cache-aware routing lowers traffic at target quality, improving BITLIFT-R's observed perplexity–traffic frontier.

Our primary contributions are: **(i)** a unified, training-free framework connecting precision allocation, representation fitting, and caching; **(ii)** policy-aware joint quantization that outperforms all evaluated shared-representation baselines, under matched precision policies; **(iii)** resident precision upgrades without additional flash fetches for upgrades or replacement updates; and **(iv)** lower perplexity and higher average accuracy than evaluated state-of-the-art baselines at matched 2.5-bit budgets across three MoE families, with an analysis of potential throughput gains under a reference hardware model.

2 Problem Setup and Background

MoE routing and dynamic precision. For token representation x, an MoE layer selects K experts $e_1(x), \ldots, e_K(x)$ in decreasing router-score order. A precision policy assigns each selected expert a bitwidth $b_r(x) \in \mathcal{B}$, where $\mathcal{B}$ is the available precisions set, e.g., $\{0, 2, 3, 4\}$. The resulting output is

$$\widehat{y}(x) = \sum_{r=1}^{K} a_{e_r}(x) F_{e_r}^{(b_r)}(x), \tag{1}$$

where e_r and b_r denote the selected expert and its execution precision at rank r, depending on x.

Quantization. Post-training quantization combines a quantizer defining admissible reconstructions and a rounding algorithm assigning codes (Tseng et al., 2026). For weights $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and calibration inputs $X \in \mathbb{R}^{d_{\text{in}} \times n}$, the reconstruction loss of quantized weights Q is

$$\mathcal{L}(Q) = \|(W - Q)X\|_F^2 = \operatorname{tr}(E\mathcal{H}E^\top) = \|E\|_{\mathcal{H}}^2, \tag{2}$$

where $E = W - Q$ and $\mathcal{H} = XX^\top$ is the per-row Hessian of $\frac{1}{2}\mathcal{L}$.

(i) Quantizer. An affine b-bit quantizer represents weights as $q = s(c - z)$, with $c \in \{0, \ldots, 2^b - 1\}$. Range fitting selects the scale s and zero point z, which determine grid spacing and position. *(ii) Adaptive rounding.* For fixed quantizer parameters, adaptive rounding (Nagel et al., 2020) accounts for how code assignments jointly affect reconstruction error. GPTQ (Frantar et al., 2022) and LDLQ (Chee et al., 2023) round columns sequentially, using input correlations in $\mathcal{H}$ to adjust the remaining unquantized weights and compensate for the resulting rounding error.

Expert caching. An expert cache retains a subset of quantized expert weights in fast memory (e.g., DRAM) for reuse during decoding (Skliar et al., 2025). Missing required weights are fetched from a slower backing tier (e.g., flash), with a replacement policy evicting cached weights as needed.

(i) Cache capacity. We express cache capacity in *logical pages*, each holding one bit for each of the P_e weights in a routed expert. This makes the budget hardware-independent: D pages per layer across L MoE layers correspond to a model-dependent capacity of $M_{\text{cache}}(D) = LDP_e/8$ bytes.

(ii) Flash traffic. We measure flash traffic in bits per weight(t_{flash}), averaged over K routing slots per token and layer, and in decimal MB/token. For F weight bits fetched over N_{tok} decoding tokens,

$$t_{\text{flash}} = \frac{F}{N_{\text{tok}} L K P_e}, \qquad T_{\text{flash}} = \frac{F}{8 \times 10^6 N_{\text{tok}}} \quad \text{MB/token.} \tag{3}$$

3 Methodology

BitLift co-designs precision allocation, weight representation, and caching through a precision policy prior that guides both quantization and cache design. Cache-aware refinement exploits resident residual planes to increase execution precision without additional flash transfers.

3.1 Precision Policy Prior

Observation 1: Precision flexibility improves quality at a fixed budget. A richer precision set expands the feasible allocations under a fixed average bit budget. Our ablations show that this flexibility consistently improves predictive quality across the evaluated models and datasets (Figure 1(a)). We therefore adopt $\mathcal{B} = \{0, 2, 3, 4\}$ as the precision set for our policy prior.

Observation 2: Router rank provides a strong policy prior. Router rank provides an input-dependent importance ordering. Prior cache-aware routing work shows that perturbing top-ranked experts is particularly harmful (Skliar et al., 2025). Our ablation likewise finds higher Qwen3 perplexity when identical precision assignments shift to lower ranks at a fixed budget (Figure 1(b)). Rank-based allocation avoids calibrating thresholds on router scores while preserving their ordering and enforcing an exact bit budget.

We define the nominal precision prior $\pi_0(r) = \rho_r \in \mathcal{B}$, assigning ρ_r bits to rank r, with $\rho_1 \geq \cdots \geq \rho_K$ and $\sum_{r=1}^{K} \rho_r = KB$. For each model and budget, the schedule is fixed across tokens and layers. As illustrated in Figure 1, this prior specifies weight requests and connects representation fitting (Section 3.2) with cache design (Section 3.3).

3.2 POLICY-AWARE BIT-SLICED REPRESENTATION

To support precision set $\mathcal{B} = \{0, 2, 3, 4\}$ motivated above, we encode each expert weight matrix using a 2-bit base and two 1-bit residual planes. Nonzero precisions correspond to nested reconstructions:

$$Q^{(2)} = s_2(C_2 - z_2), \qquad Q^{(3)} = Q^{(2)} + s_3 R_3, \qquad Q^{(4)} = Q^{(3)} + s_4 R_4. \tag{4}$$

Here $C_2 \in \{0, 1, 2, 3\}$ and $R_3, R_4 \in \{-1, +1\}$ elementwise; s_2, s_3, s_4 are reconstruction scales and z_2 is the base zero point. Each nonzero precision request uses the corresponding prefix, allowing the policy to select precision without storing independent expert copies. We jointly optimize the shared representation to balance reconstruction accuracy across its precision endpoints:

$$\mathcal{J}_\lambda = \sum_{b \in \{2,3,4\}} \lambda_b \|W - Q^{(b)}\|_{\mathcal{H}}^2, \qquad \lambda_b \geq 0, \quad \sum_b \lambda_b = 1. \tag{5}$$

The precision prior π_0 guides quantization through reconstruction weights λ_b, which balance accuracy across precision levels. We select them through a calibration sweep (see Figure 3), evaluating each quantized model under the fixed policy π_0. They govern both *joint scale fitting* of the quantizer and *joint error propagation* during rounding, aligning both stages with the intended precision policy.

Policy-aware adaptive rounding. The three precision prefixes share the remaining working weights, so we combine their distinct reconstruction errors into a common policy-weighted correction. Let $\widetilde{W}$ denote the working weights after compensation from preceding columns. For fixed codes at column j, minimizing the local joint quadratic objective over a common correction yields

$$\overline{R}_{:,j} = \sum_{b \in \{2,3,4\}} \lambda_b \big(\widetilde{W}_{:,j} - Q^{(b)}_{:,j}\big), \qquad \widetilde{W}_{:,j+1:} \leftarrow \widetilde{W}_{:,j+1:} - \frac{\overline{R}_{:,j}}{U_{jj}} U_{j,j+1:}. \tag{6}$$

where upper-triangular U satisfies $U^\top U = \mathcal{H}^{-1}$. The weights λ_b balance prefix errors, while $\mathcal{H}$ determines the correction's distribution across remaining columns, influencing subsequent rounding decisions. Appendix A derives the optimal common correction given this column's fixed codes.

Policy-aware quantizer fitting. To reduce scale metadata and simplify dequantization, we tie $(s_2, s_3, s_4) = (8\tau, 2\tau, \tau)$, yielding uniformly spaced reconstruction levels at all three precisions. We initialize the 4-bit endpoint by matching an independently fitted affine INT4 grid, which determines the initial scale τ_0 and base zero point z_2 (details are in Appendix B). We then retain z_2 and fit τ jointly across the three precision endpoints.

For one output-row/input-channel group, let $w \in \mathbb{R}^{1 \times g}$ denote working weights and H_g the corresponding principal submatrix of $\mathcal{H}$. With fixed codes, write $q^{(b)} = \tau v_b$, where $v_2 = 8(C_2 - z_2)$, $v_3 = v_2 + 2R_3$, and $v_4 = v_3 + R_4$. The shared scale minimizes the group-local joint objective

$$\mathcal{J}_g(\tau) = \sum_{b \in \{2,3,4\}} \lambda_b (w - \tau v_b) H_g (w - \tau v_b)^\top, \qquad \tau \geq \tau_{\min}, \tag{7}$$

where $\tau_{\min} > 0$ is a scale floor. The closed-form solution is

$$\tau^* = \max\left\{\tau_{\min}, \frac{\sum_b \lambda_b \, w H_g v_b^\top}{\sum_b \lambda_b \, v_b H_g v_b^\top}\right\}. \tag{8}$$

Appendix C derives the update and proves fixed-code group-local improvement, with an ablation in Table 4. Algorithm 1 summarizes the procedure.

3.3 POLICY-AWARE CACHE DESIGN

We allocate cache capacity according to the distinct roles of base and refinement weights under the precision prior π_0. Each layer maintains a base pool with C_{B} expert slots and a shared refinement pool with C_{R} plane slots. Their two-page and one-page object sizes impose

$$2C_{\mathrm{B}} + C_{\mathrm{R}} = D, \qquad C_{\mathrm{B}}, C_{\mathrm{R}} \in \mathbb{Z}_{\geq 0}. \tag{9}$$

The base supports every nonzero precision, whereas R3 and R4 serve only requests of at least 3 and 4 bits, respectively. A single pool allows refinements to evict base objects, while three pools restrict capacity sharing between refinements. The allocation is selected under the target prior:

$$(C_{\mathrm{B}}^*, C_{\mathrm{R}}^*) = \operatorname*{arg\,min}_{(C_{\mathrm{B}}, C_{\mathrm{R}}) \in \mathcal{A}_D} \widehat{t}_{\mathrm{flash}}(\pi_0; C_{\mathrm{B}}, C_{\mathrm{R}}), \tag{10}$$

where $\mathcal{A}_D$ contains candidate splits, and $\widehat{t}_{\text{flash}}$ measures nominal-request traffic. We evaluate a small sweep on validation traces, average traffic equally across corpora, and use the same split for BITLIFT-A and BITLIFT-R. We maintain independent per-layer pools to prevent cross-layer evictions; the global alternative shares the same LD pages across layers. Within each pool, we use segmented LRU (SLRU) to limit cache pollution from one-off accesses: new objects enter a probationary segment and are promoted to a protected segment upon reuse. Section 4.5 isolates the effects of topology, capacity allocation, layer scope, and replacement policy at matched weight-payload capacity.

3.4 CACHE-RESIDENT PRECISION REFINEMENT

We introduce two execution modes. BITLIFT-A uses rank-assigned execution, setting $b_r = \rho_r$. BITLIFT-R adds cache-resident precision refinement while retaining the current set of K router-selected experts. Cache-aware routing is an optional extension evaluated separately below.

Cache-state based refinement. For each token and MoE layer, we determine nominal misses and execution precisions from the same pre-token cache state $\mathcal{C}$, before any admissions or evictions for that token. Let e_r occupy slot r and $\mathcal{P}_b(e)$ denote the planes required for its b-bit reconstruction. The nominal request supplies $\mathcal{P}_{\rho_r}(e_r)$, fetching its missing planes as needed. BITLIFT-R executes the highest prefix whose additional planes are already resident in $\mathcal{C}$:

$$\widehat{b}_r = \max\left\{b \in \mathcal{B},\ b \geq \rho_r : \mathcal{P}_b(e_r) \setminus \mathcal{P}_{\rho_r}(e_r) \subseteq \mathcal{C}\right\}, \tag{11}$$

With $\mathcal{P}_0(e) = \varnothing$, a nominally skipped expert can revive when its base is resident, using any complete resident refinements. An isolated $\mathrm{R4}$ cannot extend a 2-bit request unless $\mathrm{R3}$ is also resident; for a 3-bit request, the nominal fetch supplies $\mathrm{R3}$ if needed.

Cache updates and cost. Only nominal requests update cache admission, recency, and replacement state. Optional refinement and revival reads use *no-touch* semantics. Thus, with the same initial cache and nominal request trace, the two modes have identical cache-state evolution and transferred payload. This is not an end-to-end traffic guarantee: refined outputs can alter later hidden states and routing, so we measure the resulting traffic. Upgrades add resident-weight reads and computation; Section 4.6 analyzes these requirements relative to the flash service budget.

Optional cache-aware routing. To test compatibility with an existing traffic-reduction mechanism, we combine cache-resident refinement with cache-aware logit boosting (Skliar et al., 2025):

$$z'_e = z_e + \beta p_e, \qquad p_e = 2\mathbb{I}[\mathrm{B2}_e] + \mathbb{I}[\mathrm{R3}_e] + \mathbb{I}[\mathrm{R4}_e], \tag{12}$$

where p_e counts resident one-bit pages in pre-token cache state and $\beta \geq 0$ controls boost strength. We rank experts by boosted logits and assign the highest-ranked experts to nonzero-precision slots. Nominally skipped slots then receive remaining experts ranked highest by original logits; these slots remain eligible for resident revival. For uniform policies without skipping, boosted logits determine all K selections. Boosting changes expert selection and slot assignment, but mixing weights still use original logits. Resident refinement then applies to selected experts without fetching additional planes for upgrades. Section 4.7 evaluates this composition separately from default execution modes.

4 EXPERIMENTS

Models and datasets. We evaluate three MoE models: Qwen3-30B-A3B (Yang et al., 2025), OLMoE-1B-7B-0125 (Muennighoff et al., 2025), and DeepSeek-V2-Lite (Liu et al., 2024). Quantizer calibration uses WikiText-2 (Merity et al., 2016) training text. We measure perplexity on WikiText-2 test and C4 (Raffel et al., 2020) validation, and report zero-shot accuracy on ARC-Challenge, ARC-Easy, HellaSwag (Zellers et al., 2019), PIQA, WinoGrande, MathQA (Amini et al., 2019), and MMLU (Hendrycks et al., 2020) using the EleutherAI evaluation harness (Gao et al., 2024). For caching experiments, we evaluate five-shot MMLU and GSM8K (Cobbe et al., 2021) accuracy.

Implementation details and baselines. BITLIFT builds on asymmetric GPTQ (Frantar et al., 2022) with 128-weight groups. Following prior work (Huang et al., 2025; Deng et al., 2026), attention projections and dense or shared experts use 4-bit weights. We implement BITLIFT in PyTorch (Paszke et al., 2019) and Transformers (Wolf et al., 2020), with bit-packed expert planes and custom Triton kernels (Tillet et al., 2019) that reconstruct weight tiles within matrix multiplication.

Table 1: Predictive quality at the listed average bit budgets. Avg. denotes mean zero-shot accuracy (%) over seven tasks. For BITLIFT-A, average bits denote nominal precision averaged over routing slots. D2MoE's representation is evaluated under our precision policy.

		Perplexity ↓		Downstream Accuracy ↑							
Method	Bits	WT	C4	ARC-C	ARC-E	HellaSwag	PIQA	WinoGrande	MathQA	MMLU	Avg.
Qwen3-30B-A3B											
BF16	16	8.70	14.06	56.66	78.96	77.74	80.63	71.11	59.06	77.93	71.73
Uniform GPTQ	3	9.05	14.85	54.86	78.41	**76.25**	79.22	69.06	**54.34**	75.83	69.71
Uniform GPTQ	2.5	10.14	16.88	52.65	75.72	71.86	78.40	**70.32**	46.73	75.36	67.29
GEMQ	2.5	9.39	15.19	49.49	74.16	75	78.35	68.59	43.18	71.04	65.69
KBVQ-MoE	3	9.26	-	53.24	77.27	75.53	78.89	70.01	-	-	-
D2MoE	2.5	9.33	15.56	53.50	76.56	74.74	78.62	68.43	53.10	74.43	68.48
BITLIFT-A	2.5	**9.01**	**14.82**	**54.86**	**78.79**	75.97	79.60	68.98	54.27	**76**	**69.78**
BITLIFT-A	**2.375**	9.09	14.99	53.6	78.79	76	**79.71**	68.27	54.20	75.58	69.46
OLMoE-1B-7B-0125											
BF16	16	6.65	12.14	49.32	76.39	78.18	79.71	69.53	28.44	53.57	62.16
Uniform GPTQ	3	**7.35**	**12.95**	46.08	73.36	**76.28**	**79.38**	66.61	27.24	**51.23**	60.03
Uniform GPTQ	2.5	8.82	14.94	42.83	71.42	72.17	76.17	65.11	24.86	43.32	56.55
D2MoE	2.5	7.78	13.57	44.71	72.60	74.92	78.35	**66.77**	26.33	48.30	58.85
BITLIFT-A	2.5	7.49	13.08	**47.61**	73.78	75.75	78.84	66.61	28.24	50.07	**60.13**
BITLIFT-A	**2.375**	7.65	13.23	46.25	**73.82**	75.71	78.78	66.69	**29.01**	50.39	60.09
DeepSeek-V2-Lite											
BF16	16	6.31	9.32	48.89	76.22	77.96	80.30	71.51	39.66	55.01	64.22
Uniform GPTQ	3	6.56	9.93	47.01	73.11	76.13	79.27	70.17	37.32	51.75	62.11
Uniform GPTQ	2.5	7.04	11.02	44.45	72.22	73.21	78.24	69.77	34.61	50.49	60.43
GEMQ	2.5	6.65	10.51	46.93	**74.41**	73.39	77.91	69.46	35.61	47.37	60.73
D2MoE	2.5	6.65	10.12	45.65	73.23	74.79	78.56	**70.64**	36.92	51.33	61.59
BITLIFT-A	2.5	**6.55**	**9.80**	**47.53**	74.20	**76.49**	**79.43**	70.09	38.09	**51.82**	**62.52**
BITLIFT-A	**2.33**	6.59	9.86	46.76	73.27	76.38	79.38	69.53	**38.19**	51.13	62.09

We compare against uniform 3-bit and 2.5-bit GPTQ, GEMQ (Deng et al., 2026), KBVQ-MoE (Xu et al., 2026), and the D^{2}MoE representation (Wang et al., 2025). Following literature (Deng et al., 2026; Huang et al., 2025), the uniform 2.5-bit baseline uses 3-bit routed-expert weights in the first half of MoE layers and 2-bit weights in the remainder, giving 2.5 bits on average. We evaluate D^{2}MoE's representation under BITLIFT's 2.5-bit policies to compare representation quality at matched precision allocation. Appendix F provides further experimental details.

4.1 PREDICTIVE QUALITY UNDER DYNAMIC PRECISION

Table 1 reports results at 2.5 average bits and lower budgets of 2.375 bits for Qwen3 and OLMoE and 2.333 bits for DeepSeek. These evaluations use the native router ordering, with cache-resident refinement disabled. Exact precision schedules are given in Appendix F.

Quality at a matched budget. At 2.5 bits, BITLIFT-A achieves the lowest perplexity on both datasets and the highest mean zero-shot accuracy among the evaluated 2.5-bit methods across all three models. Compared with the D^{2}MoE representation under identical precision policies, BITLIFT-A improves mean accuracy by 0.93–1.30 percentage points. This comparison supports the benefit of optimizing the shared representation for its intended precision policy.

Quality at lower budgets. At 2.333–2.375 bits, BITLIFT-A retains mean zero-shot accuracy within 0.3 percentage points of uniform 3-bit GPTQ while reducing the average precision budget by approximately 21–22%. Maximum perplexity gaps are 0.30 on WikiText-2 and 0.28 on C4, both on OLMoE. Despite lower budgets, BITLIFT-A achieves lower perplexity and higher mean accuracy than 2.5-bit D^{2}MoE representation on every model. We next evaluate flash traffic at matched capacity.

4.2 FLASH TRAFFIC WITH POLICY-AWARE CACHING

BITLIFT-A's perplexity remains unchanged as cache capacity varies because execution uses the same rank-assigned precisions. At matched expert-weight cache capacity, Figure 2 (top row) shows that BITLIFT-A reduces WikiText-2 flash traffic by 16–29% relative to uniform 3-bit GPTQ across

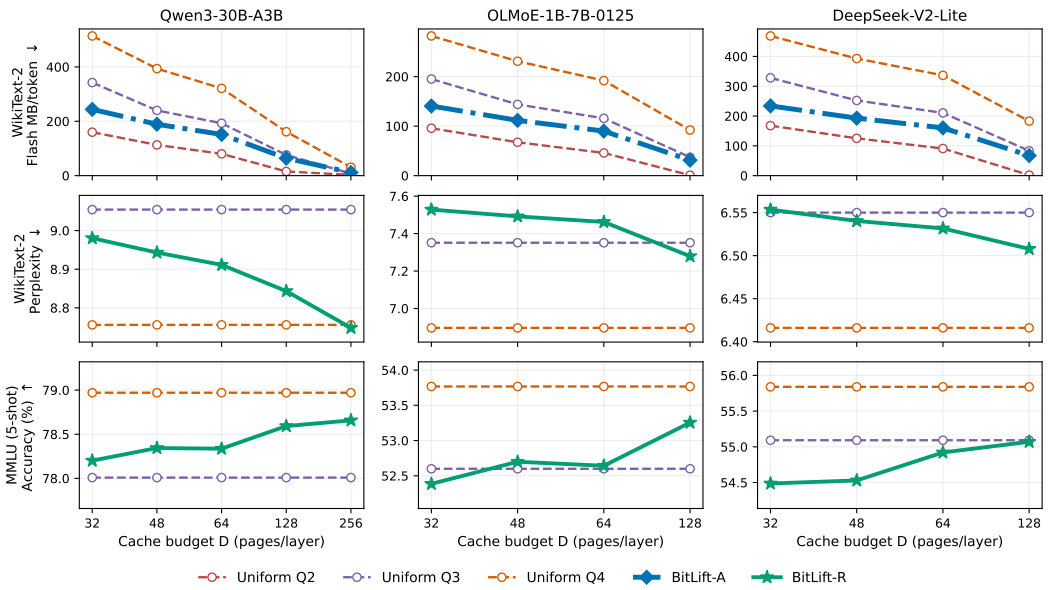

Figure 2: Traffic and quality at matched per-layer capacity D. Columns show three models. Top: WikiText-2 expert-weight flash traffic for BitLift-A and uniform Q2/Q3/Q4. Middle and bottom: WikiText-2 perplexity and five-shot MMLU accuracy for BitLift-R and uniform Q3/Q4. Uniform baselines use whole-expert SLRU; BitLift uses two-pool SLRU. Q2 accuracy is in Table 9.

three models at $D = 32$–128. Together with Section 4.1, these results show BitLift-A approaches uniform 3-bit perplexity with fewer expert-weight transfers under constrained cache capacity.

4.3 Effect of Cache-Resident Precision Refinement

At matched cache capacity, BitLift-R improves WikiText-2 perplexity and five-shot MMLU accuracy over BitLift-A at every evaluated budget across all three models. Figure 2 (bottom two rows) compares BitLift-R with uniform 3- and 4-bit GPTQ. At $D = 128$, it achieves lower perplexity than uniform 3-bit GPTQ while transferring 15–19% fewer expert-weight bytes. MMLU accuracy improves by 0.57–1.37 percentage points over BitLift-A at this budget. Perplexity improves further as cache capacity increases; on Qwen3 at $D = 256$, BitLift-R reaches perplexity comparable to uniform 4-bit GPTQ with 63% less flash traffic. These quality gains retain nearly the same flash traffic as BitLift-A, with a maximum increase of 1.93 MB/token on WikiText-2. The small traffic differences arise because refined outputs can alter subsequent routing. Appendix G reports the corresponding C4 and GSM8K results.

4.4 Policy-Aware Representation Quality

Policy-dependent reconstruction weighting. Figure 3 compares reconstruction weights under two Qwen3 precision policies. Among the tested settings, the policy dominated by 2-bit assignments favors $(\lambda_2, \lambda_3, \lambda_4) = (0.70, 0.15, 0.15)$, whereas the policy emphasizing 3- and 4-bit execution favors $(0.10, 0.45, 0.45)$. This supports selecting weights under the intended policy.

Shared representation quality. Table 2 compares representations under matched 2.5-bit policies on Qwen3 and DeepSeek. BitLift-A achieves lower perplexity than sequential residual quantization (Wang et al., 2025), MatGPTQ (Kleinegger et al., 2026), and the SliceMoE-style Choi et al. (2025) representation on WikiText-2 and C4. Its perplexity remains within 0.04 of the independent-copy reference while supporting 2-, 3-, and 4-bit execution from one four-bit payload.

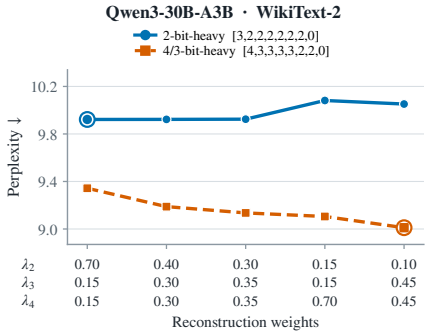

Figure 3: Preferred reconstruction weights $(\lambda_2, \lambda_3, \lambda_4)$ depend on the precision policy.

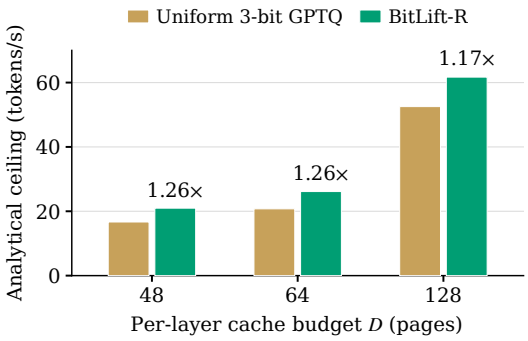

Figure 4: Analytical throughput ceilings in the flash-bound regime (Qwen3, WikiText-2). Labels show ratios to uniform 3-bit GPTQ.

Table 2: PPL under matched 2.5-bit routing policies.

	Qwen3-30B-A3B		DeepSeek-V2-Lite	
Method	**WT2**	**C4**	**WT2**	**C4**
Independent copies	**9.01**	14.85	**6.51**	**9.77**
Sequential residual	9.33	15.56	6.65	10.12
MATGPTQ	9.23	14.95	6.71	10.04
SliceMoE-style AMAT	9.66	15.60	6.77	10.08
Ours	**9.01**	**14.82**	6.55	9.80

Table 3: Qwen3 traffic reduction (%) at $D = 48$, relative to the configuration before each arrow. Topology uses LRU.

Design change	**WT2**	**C4**
Single → two pools	1.5	1.6
LRU → SLRU	2.6	3.2
Global → per-layer	13.6	20.4
Split $(16, 16) \rightarrow (18, 12)$	1.4	1.6

4.5 Cache Design Ablations

We vary one cache component at a time on fixed router traces, matching expert-weight payload capacity across designs (Table 3). On Qwen3 at $D = 48$, two pools reduce traffic relative to a single pool and perform similarly to the best tested three-pool allocation. Per-layer isolation provides the largest reduction at this budget: 13.6% on WikiText-2 and 20.4% on C4 relative to global pooling. SLRU also reduces traffic relative to LRU on Qwen3 and OLMoE over $D = 48$–128. On Qwen3 at $D = 48$, the selected split $(C_{\mathrm{B}}, C_{\mathrm{R}}) = (18, 12)$ minimizes traffic among the tested allocations for all evaluated 2.25–2.50-bit policies. Substantially lighter or heavier refinement demand favors different allocations, supporting split selection under the target policy. Appendix I reports the full cache ablations, and Appendix H.1 examines policy-dependent allocation.

4.6 Throughput Analysis

We translate measured expert-weight traffic into batch-one throughput ceilings using 4 GB/s for flash transfers and 68 GB/s for DRAM weight reads (Appendix E). On Qwen3 at matched budgets $D \in \{48, 64, 128\}$, the model yields WikiText-2 throughput ceilings of 21.0, 26.1, and 61.7 tokens/s for BitLift-R, respectively, or 1.17–1.26× those of uniform 3-bit GPTQ (Figure 4). Flash service exceeds expert-weight read and arithmetic service times at these budgets, even with refinement. BitLift-R's ceiling is within 1.1% of BitLift-A while providing quality gains in Section 4.3. With three services serialized, modeled rates remain 1.09–1.23× those of uniform 3-bit GPTQ (Table 6).

4.7 Cache-Aware Routing Tradeoff

Figure 5 shows that combining BitLift-R with cache-aware routing extends the observed perplexity–traffic frontier relative to uniform 3- and 4-bit GPTQ with cache-aware routing. Across Qwen3 budgets $D = 48, 64, 128$, BitLift-R offers operating points with lower flash traffic at comparable or better perplexity. Routing and refinement contribute complementary benefits. Stronger routing boosts favor cached experts, reducing flash traffic while increasing perplexity. Resident refinement improves

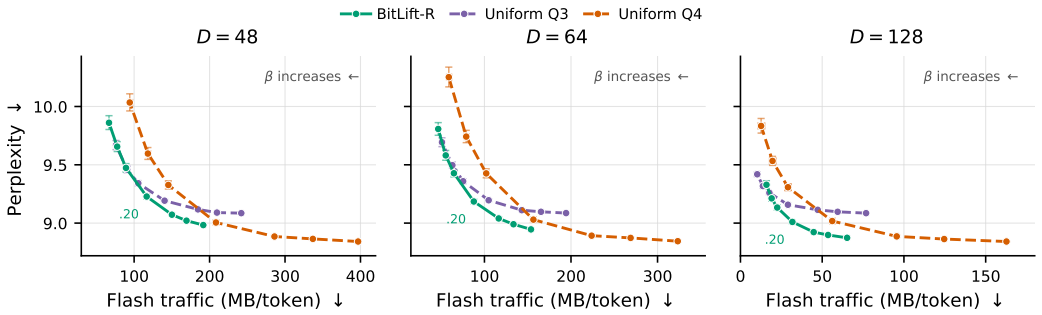

Figure 5: Cache-aware routing on Qwen3 (WikiText-2) at matched SLRU capacity. Green labels mark $\beta = 0.20$; whiskers show 95% paired-bootstrap intervals for changes from each method's $\beta = 0$ baseline (Appendix F). Lower-left is better.

perplexity at nearly unchanged traffic (Section 4.3), allowing stronger routing boosts before reaching a target perplexity. Their combination enables further flash-traffic reductions within a quality budget.

5 Related Work

MoE quantization and precision allocation. Expert-aware post-training quantization accounts for heterogeneous expert sensitivity and usage (Li et al., 2024; Fu et al., 2025; Huang et al., 2025; Chen et al.). Static mixed-precision methods fix expert bit widths using calibrated importance or hardware objectives (Chitty-Venkata et al., 2025; Duanmu et al., 2025; Deng et al., 2026). Dynamic methods adapt runtime precision: D^{2}MoE (Wang et al., 2025) learns a token-wise bit-width router, HOBBIT (Tang et al., 2024) uses gate-ranked high/low/skip decisions on cache misses, DyMoE (Huang et al., 2026) considers gate scores and depth, and DynaExq (Chu et al., 2025) promotes frequently used experts. BitLift's fixed rank-to-precision prior enforces an exact nominal budget over router-selected experts without controller training.

Shared multi-precision representations. A shared checkpoint supports multiple precisions. Bayesian Bits (Van Baalen et al., 2020) progressively quantizes residuals; MatQuant (Nair et al., 2025) jointly optimizes bit prefixes; MatGPTQ (Kleinegger et al., 2026) adds weighted cross-bit code selection to GPTQ. For MoE inference, D^{2}MoE (Wang et al., 2025) builds a 2+1+1-bit stream via sequential residual fitting, while SliceMoE (Choi et al., 2025) exposes compatible precisions via zero-point-aware truncation. BitLift optimizes shared 2, 3, and 4-bit prefixes with policy-calibrated reconstruction weights for joint scale fitting and second-order error compensation.

Expert caching and memory-aware serving. HOBBIT (Tang et al., 2024) and DyMoE (Huang et al., 2026) combine mixed-precision offloading with prefetching; SpecMD pairs speculative prefetching with Least-Stale eviction (Hoang et al., 2026). Cache-Conditional Experts reranks toward resident experts (Skliar et al., 2025), while SliceMoE combines bit-slice caching, precision selection, and prefill-guided warmup (Choi et al., 2025). BitLift's progressive representation and precision prior guide two-pool base–refinement caching with standard SLRU (Karedla et al., 1994).

6 Conclusion

BitLift unifies precision allocation, progressive quantization, and expert caching through a common precision policy prior. Across three MoE families, BitLift-A outperforms evaluated baselines at 2.5 average bits and approaches uniform 3-bit predictive quality at 2.333–2.375 bits, with lower flash traffic under constrained cache capacity. BitLift-R exploits resident planes to improve perplexity and downstream accuracy at nearly unchanged traffic. These quality gains complement cache-aware routing, enabling further traffic reductions at a given quality target. Our throughput analysis characterizes the resource conditions under which refinement costs can be hidden by flash transfers. Aligning representation fitting, cache allocation, and execution precision thus improves predictive quality and memory efficiency.

AI USE STATEMENT

We used generative AI tools to assist with literature review, language editing, document organization, and inspection of experiment outputs while preparing tables. All mathematical claims, citations, code changes, and reported numbers were reviewed and verified by the authors, who take responsibility for the final content.

ETHICS STATEMENT

This work studies compression and memory-efficient inference for publicly available pretrained language models. It does not involve human subjects or release personal data. Lowering inference memory and traffic may broaden access to large models; it does not remove the safety, bias, privacy, or misuse risks of the underlying models.

REFERENCES

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. 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)*, pp. 2357–2367, 2019.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in neural information processing systems*, 36: 4396–4429, 2023.

Zhixuan Chen, Xing Hu, Dawei Yang, Zukang Xu, Zhihang Yuan, Sifan Zhou, et al. Moequant: Enhancing quantization for mixture-of-experts large language models via expert-balanced sampling and affinity guidance. In *Forty-second International Conference on Machine Learning*.

Krishna Teja Chitty-Venkata, Jie Ye, and Murali Emani. Mopeq: Mixture of mixed precision quantized experts. In *2025 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 4082–4091. IEEE, 2025.

Yuseon Choi, Sangjin Kim, Jungjun Oh, Gwangtae Park, Byeongcheol Kim, and Hoi-Jun Yoo. Slicemoe: Bit-sliced expert caching under miss-rate constraints for efficient moe inference. *arXiv preprint arXiv:2512.12990*, 2025.

Kexin Chu, Dawei Xiang, Zixu Shen, Yiwei Yang, Zecheng Liu, and Wei Zhang. Dynamic expert quantization for scalable mixture-of-experts inference. *arXiv preprint arXiv:2511.15015*, 2025.

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

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. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=5Q4ts0pr9T.

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. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 14793–14806. PMLR, 13–19 Jul 2025.

Marco Federici, Davide Belli, Mart Van Baalen, Amir Jalalirad, Andrii Skliar, Bence Major, Markus Nagel, and Paul Whatmough. Efficient llm inference using dynamic input pruning and cache-aware masking. *Proceedings of Machine Learning and Systems*, 7, 2025.

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

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

Zhongqian Fu, Tianyi Zhao, Ning Ding, Xianzhi Yu, Xiaosong Li, Yehui Tang, and Yunhe Wang. Eaquant: Enhancing post-training quantization for moe models via expert-aware optimization. *arXiv preprint arXiv:2506.13329*, 2025.

Tomer Gafni, Asaf Karnieli, and Yair Hanani. Dual precision quantization for efficient and accurate deep neural networks inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 3284–3294, June 2025.

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.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Duc N.M Hoang, Mohammad Samragh, AJAY KUMAR JAISWAL, and Minsik Cho. SpecMD: A comprehensive study on speculative expert prefetching. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=hjmbstLYc9.

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. In *International Conference on Learning Representations*, volume 2025, pp. 68628–68648, 2025.

Yuegui Huang, Zhiyuan Fang, Weiqi Luo, Ruoyu Wu, Wuhui Chen, and Zibin Zheng. Dymoe: Dynamic expert orchestration with mixed-precision quantization for efficient moe inference on edge. *arXiv preprint arXiv:2603.19172*, 2026.

R. Karedla, J.S. Love, and B.G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994. doi: 10.1109/2.268884.

Maximilian Kleinegger, Elvir Crnčević, and Dan Alistarh. Matgptq: Accurate and efficient post-training matryoshka quantization. *arXiv preprint arXiv:2602.03537*, 2026.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Pingzhi Li, Xiaolong Jin, Zhen Tan, Yu Cheng, and Tianlong Chen. Quantmoe-bench: Examining post-training quantization for mixture-of-experts. *arXiv preprint arXiv:2406.08155*, 2024.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Stephen Merity et al. The wikitext long term dependency language modeling dataset. *Salesforce Metamind*, 9, 2016.

Micron Technology. LPDDR5X memory, n.d. URL https://www.micron.com/products/memory/lpddr-components/lpddr5x. Product specifications list data rates up to 10.7 Gbps. Accessed September 23, 2026.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. Olmoe: Open mixture-of-experts language models, 2025. URL https://arxiv.org/abs/2409.02060.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? Adaptive rounding for post-training quantization. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7197–7206. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/nagel20a.html.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

Pranav Ajit Nair, Puranjay Datta, Jeff Dean, Prateek Jain, and Aditya Kusupati. Matryoshka quantization. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=phVWcUSGYP.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Samsung Electronics. LPDDR5X: Product specifications, n.d.a. URL https://semiconductor.samsung.com/dram/lpddr/lpddr5x/. Product specifications list data rates up to 10.7 Gbps. Accessed September 23, 2026.

Samsung Electronics. UFS 4.0: Universal flash storage, n.d.b. URL https://semiconductor.samsung.com/estorage/ufs/ufs-4-0/. Product specifications list read speeds up to 4,200 MB/s. Accessed September 23, 2026.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Andrii Skliar, Ties van Rozendaal, Romain Lepert, Todor Boinovski, Mart Van Baalen, Markus Nagel, Paul N. Whatmough, and Babak Ehteshami Bejnordi. Mixture of cache-conditional experts for efficient mobile device inference. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=ul4W26KEKz.

Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. Hobbit: A mixed precision expert offloading system for fast moe inference, 2024. URL https://arxiv.org/abs/2411.01433.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Albert Tseng, Zhaofeng Sun, and Christopher De Sa. Model-preserving adaptive rounding, 2026. URL https://arxiv.org/abs/2505.22988.

Mart Van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. *Advances in neural information processing systems*, 33:5741–5752, 2020.

Haodong Wang, Qihua Zhou, Zicong Hong, and Song Guo. D2moe: Dual routing and dynamic scheduling for efficient on-device moe-based llm serving. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking*, pp. 574–588, 2025.

Wikipedia contributors. Apple A19, 2026a. URL https://en.wikipedia.org/wiki/Apple_A19. Secondary specification compilation; A19 Pro entry reports LPDDR5X-9600 and 76.8 GB/s. Accessed September 23, 2026.

Wikipedia contributors. List of Qualcomm Snapdragon systems on chips, 2026b. URL https://en.wikipedia.org/wiki/List_of_Qualcomm_Snapdragon_systems_on_chips. Secondary specification compilation; Snapdragon 8 Elite entry reports a 64-bit LPDDR5X interface at 5300 MHz and 84.8 GB/s. Accessed September 23, 2026.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020. URL https://arxiv.org/abs/1910.03771.

Zukang Xu, Zhixiong Zhao, Xing Hu, Zhixuan Chen, and Dawei Yang. Kbvq-moe: Klt-guided svd with bias-corrected vector quantization for moe large language models. *arXiv preprint arXiv:2602.11184*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Algorithm 1 Policy-aware residual quantization in BITLIFT.

Input: Weights $W \in \mathbb{R}^{m\times n}$, reconstruction metric $\mathcal{H} \succ 0$, block size h, group size g, scale floor $\tau_{\min} > 0$
Policy weights $\boldsymbol{\lambda} = (\lambda_2, \lambda_3, \lambda_4)$, $\lambda_b \geq 0$, $\sum_b \lambda_b = 1$

$Q^{(2)}, Q^{(3)}, Q^{(4)} \leftarrow 0$
$U \leftarrow \mathrm{Cholesky}(\mathcal{H}^{-1})^\top$
for $i = 0, h, 2h, \ldots, n-h$ **do**
  $J \leftarrow i : i+h$; $T \leftarrow i+h : n$
  $A \leftarrow W_{:,J}$; $E \leftarrow 0$
  for $k = 0, \ldots, h-1$ **do**
    $j \leftarrow i + k$
    if $j \bmod g = 0$ **then**
      $G \leftarrow j : j+g$
      $\Theta \leftarrow \mathrm{JointFit}(A_{:,k:k+g}, \mathcal{H}_{G,G}, \boldsymbol{\lambda}, \tau_{\min})$ ▷ Algorithm 2
    $\{Q^{(b)}_{:,j}\}_{b\in\{2,3,4\}} \leftarrow \mathrm{NestedQuant}(A_{:,k}; \Theta)$ ▷ Eq. equation 4
    $e \leftarrow U_{jj}^{-1} \sum_{b\in\{2,3,4\}} \lambda_b (A_{:,k} - Q^{(b)}_{:,j})$
    $A_{:,k:} \leftarrow A_{:,k:} - e\, U_{j,j:i+h}$
    $E_{:,k} \leftarrow e$
  $W_{:,T} \leftarrow W_{:,T} - E\, U_{J,T}$
return $Q^{(2)}, Q^{(3)}, Q^{(4)}$

■ BitLift operation ■ Standard GPTQ operation.

Algorithm 2 JOINTFIT: INT4-anchored scale fitting with fixed codes.

Input: Working weights $A_g \in \mathbb{R}^{m\times g}$, group metric $H_g \succeq 0$, scale floor $\tau_{\min} > 0$
Policy weights $\boldsymbol{\lambda} = (\lambda_2, \lambda_3, \lambda_4)$, $\lambda_b \geq 0$, $\sum_b \lambda_b = 1$
Output: Row-wise group parameters $\Theta = (\theta_1, \ldots, \theta_m)$

for each output row u **do**
  $w \leftarrow (A_g)_{u,:}$
  $(\Delta_4, z_4) \leftarrow \mathrm{FitINT4}(w)$ ▷ Same affine range fitting as the baseline
  $\tau_0 \leftarrow \max\{\tau_{\min}, \Delta_4/2\}$; $z_2 \leftarrow (z_4 - 3/2)/4$ ▷ Fix the floating zero point
  $\theta_0 \leftarrow (8\tau_0, 2\tau_0, \tau_0, z_2)$
  $(C_2, R_3, R_4) \leftarrow \mathrm{NestedCodes}(w; \theta_0)$ ▷ Hold codes fixed for the scale solve
  $v_2 \leftarrow 8(C_2 - z_2)$; $v_3 \leftarrow v_2 + 2R_3$; $v_4 \leftarrow v_3 + R_4$
  $a \leftarrow \sum_{b\in\{2,3,4\}} \lambda_b\, v_b H_g v_b^\top$; $d \leftarrow \sum_{b\in\{2,3,4\}} \lambda_b\, w H_g v_b^\top$
  $\tau \leftarrow \tau_0$
  if $a > 0$ **then**
    $\tau \leftarrow \max\{\tau_{\min}, d/a\}$ ▷ Eq. equation 8
  $\theta_u \leftarrow (8\tau, 2\tau, \tau, z_2)$
return Θ

NESTEDQUANT applies sequential nearest rounding to the representation in Eq. equation 4. It first selects the nearest 2-bit base level, then rounds the remaining residual to $\{-s_3, +s_3\}$ and the subsequent residual to $\{-s_4, +s_4\}$. NESTEDCODES returns the corresponding codes (C_2, R_3, R_4). JOINTFIT holds these codes and z_2 fixed during scale fitting; the main algorithm then rounds compensated columns using the fitted parameters.

A DERIVATION OF POLICY-AWARE ERROR COMPENSATION

We derive the common correction in Eq. equation 6, conditional on the current column's codes and quantizer parameters. The reconstruction objective separates over output rows, so it suffices to consider one row. We assume a positive-definite metric $\mathcal{H}$; in practice, damping replaces $XX^\top$ by $XX^\top + \gamma I$, $\gamma > 0$, for both the quadratic objective and its factorization.

Local joint objective. At column j, let $\mathcal{F} = \{j+1, \ldots, d_{\text{in}}\}$ index the remaining columns and $\mathcal{A} = \{j\} \cup \mathcal{F}$. The working weights $\widetilde{w}$ incorporate compensation from previously processed columns. For each precision $b \in \{2, 3, 4\}$, define the current residual $r_b = \widetilde{w}_j - q_j^{(b)}$. The metric restricted to

the active columns is

$$\mathcal{H}^{(j)} := \mathcal{H}_{\mathcal{A},\mathcal{A}} = \begin{bmatrix} h_{jj} & h_{j\mathcal{F}} \\ h_{\mathcal{F}j} & \mathcal{H}_{\mathcal{F}\mathcal{F}} \end{bmatrix}. \tag{13}$$

A common correction $\delta \in \mathbb{R}^{|\mathcal{F}|}$ updates $\widetilde{w}_{\mathcal{F}} \leftarrow \widetilde{w}_{\mathcal{F}} + \delta^{\top}$. Its contribution to the reconstruction error is $-\delta$, giving the local objective

$$\mathcal{J}_j(\delta) = \sum_{b\in\{2,3,4\}} \lambda_b \begin{bmatrix} r_b \\ -\delta \end{bmatrix}^{\top} \mathcal{H}^{(j)} \begin{bmatrix} r_b \\ -\delta \end{bmatrix}. \tag{14}$$

Using $\sum_b \lambda_b = 1$ and $\overline{r} = \sum_b \lambda_b r_b$, expansion yields

$$\mathcal{J}_j(\delta) = h_{jj} \sum_b \lambda_b r_b^2 - 2\overline{r}\, h_{j\mathcal{F}}\delta + \delta^{\top}\mathcal{H}_{\mathcal{F}\mathcal{F}}\delta. \tag{15}$$

Thus, the precision-dependent residuals enter the correction through their weighted mean $\overline{r}$. Setting the gradient to zero gives

$$\nabla_\delta \mathcal{J}_j = 2\mathcal{H}_{\mathcal{F}\mathcal{F}}\delta - 2\overline{r}\, h_{\mathcal{F}j} = 0, \qquad \delta^* = \overline{r}\, \mathcal{H}_{\mathcal{F}\mathcal{F}}^{-1} h_{\mathcal{F}j}. \tag{16}$$

Because $\mathcal{H}_{\mathcal{F}\mathcal{F}} \succ 0$, this is the unique minimizer over common corrections.

Connection to the factorized GPTQ update. Let U be upper triangular with $U^{\top}U = \mathcal{H}^{-1}$, and define $K^{(j)} = (\mathcal{H}^{(j)})^{-1}$. The block-inverse identity gives

$$\mathcal{H}_{\mathcal{F}\mathcal{F}}^{-1} h_{\mathcal{F}j} = -\frac{K^{(j)}_{\mathcal{F},j}}{K^{(j)}_{jj}}. \tag{17}$$

The trailing factor satisfies $K^{(j)} = U^{\top}_{\mathcal{A},\mathcal{A}} U_{\mathcal{A},\mathcal{A}}$. Since j is its first column, $K^{(j)}_{jj} = U^2_{jj}$ and $K^{(j)}_{\mathcal{F},j} = U_{jj} U^{\top}_{j,\mathcal{F}}$. Substitution therefore yields

$$\delta^* = -\frac{\overline{r}}{U_{jj}} U^{\top}_{j,\mathcal{F}}. \tag{18}$$

Applying this correction independently to each output row recovers Eq. equation 6, with $\overline{R}_{:,j} = \sum_b \lambda_b (\widetilde{W}_{:,j} - Q^{(b)}_{:,j})$. The policy weights therefore determine the residual supplied to GPTQ's existing compensation mechanism. The optimality result concerns the common continuous correction for fixed current-column reconstructions; it does not establish global optimality of the discrete code assignments.

B Tied-Scale Geometry and INT4 Initialization

The tied quantizer organizes its reconstruction levels hierarchically: the 2-bit base provides four levels, and each binary residual refines every level into two. We choose the relative scales so that each precision exposes a uniformly spaced grid, leaving one shared scale τ to be fitted to the precision policy. Figure 6 illustrates this construction.

Deriving the scale ratio. Set $s_4 = \tau > 0$. Around a fixed base reconstruction $q^{(2)}$, the two residual bits produce the four offsets

$$s_3 b_3 + \tau b_4 \in \{-s_3 - \tau,\, -s_3 + \tau,\, s_3 - \tau,\, s_3 + \tau\}, \qquad b_3, b_4 \in \{-1, +1\}. \tag{19}$$

For the ordered hierarchy $s_3 \geq \tau$, the consecutive gaps are 2τ, $2(s_3 - \tau)$, and 2τ. Equal spacing within this cluster therefore requires $s_3 = 2\tau$, giving offsets $\{-3\tau, -\tau, \tau, 3\tau\}$. Adjacent base reconstructions are separated by s_2; the gap between the last refined level of one cluster and the first of the next is $s_2 - 6\tau$. Matching this gap to 2τ gives

$$(s_2, s_3, s_4) = (8\tau, 2\tau, \tau). \tag{20}$$

The resulting 2-, 3-, and 4-bit grids contain 4, 8, and 16 levels, with spacings 8τ, 4τ, and 2τ, respectively. The hierarchy concerns prefixes of the stored representation; the numerical level sets need not contain one another.

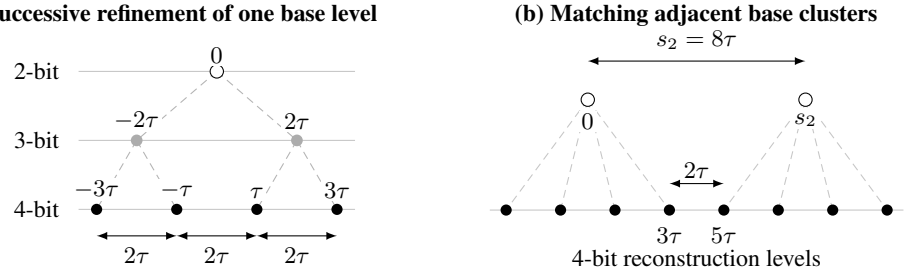

Figure 6: Geometry of the tied residual quantizer. Coordinates are offsets from a selected base reconstruction, shown at zero. (a) Successive binary refinements create four final levels per base level. (b) A base spacing of 8τ makes the gap between neighboring clusters equal to the within-cluster spacing 2τ. Open circles denote base levels; black circles denote 4-bit levels.

Initializing the affine grid. For each output-row/input-channel group, we initialize the shared representation by matching its 4-bit endpoint to an independently fitted affine INT4 grid, $\Delta_4(k - z_4)$, $k \in \{0, \dots, 15\}$, with step $\Delta_4 > 0$ and zero point z_4.

The tied construction yields sixteen uniformly spaced reconstruction levels with spacing 2τ. The smallest level is obtained with $c_2 = 0$ and $r_3 = r_4 = -1$, giving $q_{\min} = -\tau(8z_2 + 3)$. Each subsequent level adds 2τ, so the grid can be written as

$$\begin{aligned} q^{(4)}(k) &= q_{\min} + 2\tau k \\ &= 2\tau \left[k - \left(4z_2 + \tfrac{3}{2}\right)\right], \qquad k = 0, \dots, 15. \end{aligned} \tag{21}$$

Thus, the tied representation is an affine INT4 grid with step 2τ and effective zero point $4z_2 + 3/2$. Matching these parameters to the fitted grid gives

$$\tau_0 = \frac{\Delta_4}{2}, \qquad z_2 = \frac{z_4 - \frac{3}{2}}{4}. \tag{22}$$

We retain a floating base zero point z_2 to preserve this exact match. At $\tau = \tau_0$, the sixteen representable 4-bit levels coincide with those of the independently fitted INT4 quantizer; code assignments are determined separately. We subsequently fix z_2 and optimize τ under the joint reconstruction objective, preserving the effective zero point $4z_2 + 3/2 = z_4$ while allowing the grid spacing 2τ to change.

C Policy-Aware Scale Fitting

We derive the shared-scale update and its conditional improvement guarantee. For one output row and a group of g input channels, let $w \in \mathbb{R}^{1\times g}$ contain the working weights and $H_g \succeq 0$ be the corresponding principal submatrix of the reconstruction metric $\mathcal{H}$. The policy weights satisfy $\lambda_b \geq 0$ and $\sum_b \lambda_b = 1$. Throughout this solve, we hold w, H_g, the policy weights, the group codes, and the anchored zero point z_2 fixed.

Each reconstruction has the form $q^{(b)} = \tau v_b$, where $v_2 = 8(C_2 - z_2)$, $v_3 = v_2 + 2R_3$, and $v_4 = v_3 + R_4$, with codes restricted to the current group. These vectors are independent of τ. For $\tau \geq \tau_{\min} > 0$, expanding the group-local objective using the symmetry of H_g and $\sum_b \lambda_b = 1$ gives

$$\begin{aligned} \mathcal{J}_g(\tau) &= \sum_{b\in\{2,3,4\}} \lambda_b (w - \tau v_b) H_g (w - \tau v_b)^\top \\ &= \sum_b \lambda_b \left(w H_g w^\top - 2\tau w H_g v_b^\top + \tau^2 v_b H_g v_b^\top \right) \\ &= \tau^2 \sum_b \lambda_b v_b H_g v_b^\top - 2\tau \sum_b \lambda_b w H_g v_b^\top + w H_g w^\top. \end{aligned} \tag{23}$$

Collecting the quadratic, linear, and constant coefficients, define

$$a = \sum_b \lambda_b v_b H_g v_b^\top, \qquad d = \sum_b \lambda_b w H_g v_b^\top, \qquad c = w H_g w^\top. \tag{24}$$

Table 4: Joint scale refinement on Qwen3 under the same 2.5-bit policy. Reductions are relative to INT4 initialization.

	Reconstruction weights $(\lambda_2, \lambda_3, \lambda_4)$			
	$(0.10, 0.45, 0.45)$		$(0.30, 0.35, 0.35)$	
Scale configuration	WikiText-2 ↓	C4 ↓	WikiText-2 ↓	C4 ↓
INT4 initialization (τ_0)	9.16	15.03	9.18	15.02
Joint refinement ($\tau^\star$)	**9.01**	**14.82**	**9.13**	**14.91**
Perplexity reduction	0.15	0.21	0.05	0.11

The objective and its derivatives are therefore

$$\mathcal{J}_g(\tau) = a\tau^2 - 2d\tau + c, \qquad \mathcal{J}_g'(\tau) = 2a\tau - 2d, \qquad \mathcal{J}_g''(\tau) = 2a. \tag{25}$$

For $a > 0$, the objective is strictly convex. Setting its first derivative to zero gives the unconstrained minimizer $\tau = d/a$. Imposing $\tau \geq \tau_{\min}$ yields

$$\begin{aligned} \tau^* &= \max\left\{\tau_{\min}, \frac{\sum_b \lambda_b\, w H_g v_b^\top}{\sum_b \lambda_b\, v_b H_g v_b^\top}\right\}, \\ (s_2, s_3, s_4) &= (8\tau^*, 2\tau^*, \tau^*). \end{aligned} \tag{26}$$

If $a = 0$, positive semidefiniteness of H_g and nonnegative policy weights imply $d = 0$, so the objective is independent of τ; any feasible scale is optimal. The optimization concerns the group-local objective, which includes input correlations within the group while omitting cross-group terms from the full-layer objective.

Fixed-code improvement guarantee. For any feasible reference scale $\tau_{\text{ref}} \geq \tau_{\min}$, the change in objective is

$$\begin{aligned} \mathcal{J}_g(\tau_{\text{ref}}) - \mathcal{J}_g(\tau^*) = a(\tau_{\text{ref}} - \tau^*)^2 \\ + 2(a\tau^* - d)(\tau_{\text{ref}} - \tau^*) \;\geq\; 0. \end{aligned} \tag{27}$$

If the unconstrained optimum is feasible, $a\tau^* - d = 0$, so the improvement is exactly $a(\tau_{\text{ref}} - \tau^*)^2$. If the floor is active, $\tau^* = \tau_{\min}$ and both factors in the second term are nonnegative. Consequently, the fitted scale cannot increase the group-local joint objective relative to the INT4 initialization whenever $\tau_0 \geq \tau_{\min}$. If the raw initialization lies below the floor, the feasible reference is $\max\{\tau_{\min}, \tau_0\}$.

The guarantee compares scales under the same fixed codes, zero point, working weights, metric, and policy weights. It applies to the weighted group objective and permits tradeoffs between individual precisions. Changes to the codes or working weights define a new scale-fitting problem and are outside this fixed-code comparison.

Effect of joint scale refinement. We isolate the joint update of τ on Qwen3 under the 2.5-bit policy $(4, 3, 3, 3, 3, 2, 2, 0)$, keeping all other quantization and evaluation settings fixed. Table 4 shows that refinement improves both datasets under both reconstruction weightings. For the main configuration, $\boldsymbol{\lambda} = (0.10, 0.45, 0.45)$, perplexity decreases by 0.15 on WikiText-2 and 0.21 on C4.

D Reconstruction Across Precision Levels

A shared progressive representation couples reconstruction quality across precision levels. Sequential residual fitting commits to a 2-bit base before optimizing its refinements, whereas joint fitting adjusts the shared parameters to balance all prefixes under the policy-weighted objective. Figure 7 illustrates this distinction for the same 128-weight group and Hessian, with independently optimized GPTQ grids shown as dashed references.

With $(\lambda_2, \lambda_3, \lambda_4) = (0.10, 0.45, 0.45)$, joint fitting prioritizes the higher-precision prefixes. The normalized 2-bit error increases from 0.150 to 0.166, while the 3- and 4-bit errors decrease from

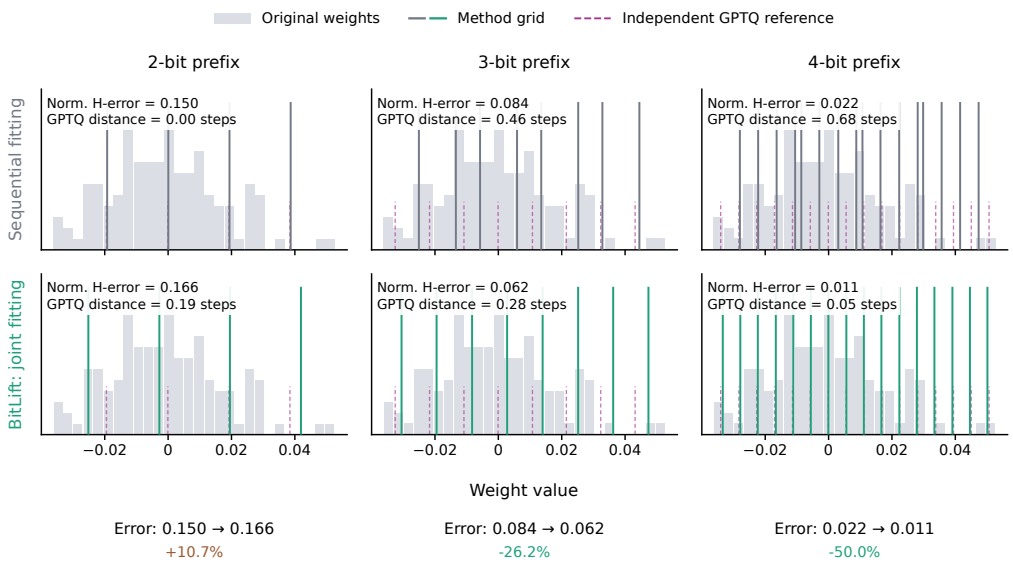

Figure 7: Reconstruction grids for the same 128-weight group. Rows compare sequential residual fitting and BITLIFT's joint fitting; columns show 2-, 3-, and 4-bit prefixes. Gray histograms show original weights, solid lines show reconstruction levels, and dashed magenta lines overlay independently optimized GPTQ grids.

0.084 to 0.062 and from 0.022 to 0.011, respectively. The resulting 4-bit error nearly matches the independent reference of 0.010. This example shows how allowing a modest change in the base can improve higher-precision reconstructions within the same shared representation.

E THROUGHPUT ANALYSIS

We use observed expert-weight traffic to derive resource ceilings for batch-one decoding, distinguishing flash transfers, resident expert-weight reads, and routed matrix operations. We then examine sensitivity to serialization and additional execution overhead. These analyses quantify throughput potential rather than measured end-to-end device performance.

E.1 TRAFFIC AND EXECUTION MODEL

Consider L MoE layers, K routing slots per layer, and P_e weights per routed expert. Let $\bar{b}_{\text{exec}}$ be the average executed precision over tokens, layers, and routing slots, counting unexecuted slots as zero. Writing $f = t_{\text{flash}}$ for the normalized flash traffic from Eq. (3), define

$$\Gamma = \frac{LKP_e}{8}, \qquad V_{\text{F}} = \Gamma f, \qquad V_{\text{R}} = \Gamma \bar{b}_{\text{exec}}, \tag{28}$$

where V_{F} and V_{R} are flash payload and packed expert-weight reads in bytes per token. The read model assumes each executed weight plane is read once from DRAM. Cache hits reduce V_{F} but still contribute to V_{R}.

For BitLift-A, $\bar{b}_{\text{exec}} = B$, with $B = 19/8$ for Qwen and OLMoE and $B = 14/6$ for DeepSeek. For BitLift-R, let u_3 and u_4 be the average numbers of optional one-bit refinement planes used per routing slot, and let r_0 be the rate at which nominally skipped slots are revived. Then

$$\bar{b}_{\text{exec}} = B + u_3 + u_4 + 2r_0. \tag{29}$$

The $2r_0$ term accounts for revived base planes; u_3 and u_4 also include refinements used by revived experts.

Each nominal policy skips one slot per layer. Counting a multiply and an add as two operations, $O_{\text{E}} = 2LKP_e((K-1)/K + r_0)$ for BITLIFT-A/R; for uniform precision, $O_{\text{E}} = 2LKP_e$. Refinement

adds packed reads and reconstruction work; reviving skipped experts also adds matrix operations. O_{E} counts only matrix arithmetic, with reconstruction overhead considered separately below.

E.2 Hardware Assumptions and Resource Ceilings

We use an illustrative memory configuration for smartphone-scale edge inference. Samsung and Micron specify LPDDR5X data rates up to 10.7 Gb/s per pin (Samsung Electronics, n.d.a; Micron Technology, n.d.). A reference 64-bit interface at this rate gives $10.7 \times 64/8 = 85.6$ GB/s peak bandwidth. We round this to 85 GB/s and allocate 80%, or 68 GB/s, to execution reads. Samsung specifies UFS 4.0 read speeds up to 4,200 MB/s (Samsung Electronics, n.d.b); we assume 4,000 MB/s sustained flash service for the analysis. The interface width, 80% allocation, and sustained rates are modeling assumptions, not measurements of a particular handset or expert-fetch workload. We also assume an ideal arithmetic budget of 45 TOPS; this is not a claim of verified sustained mixed-precision NPU performance on a particular device.

Device scope. Reported handset specifications place Snapdragon 8 Elite's 64-bit LPDDR5X interface at 5,300 MHz, corresponding to 84.8 GB/s with double data rate (Wikipedia contributors, 2026b), and the iPhone 17 Pro's A19 Pro at 76.8 GB/s with LPDDR5X-9600 (Wikipedia contributors, 2026a). These handset figures come from secondary specification compilations and provide context for the reference bandwidth; they are not sustained-bandwidth measurements. Our reported ceilings use the fixed 68 GB/s smartphone-scale reference. The 4 GB/s UFS-based flash assumption is likewise a reference configuration, not an iPhone storage specification; a device-specific estimate requires its own sustained storage rate and available DRAM capacity.

Let $\mathrm{BW}_{\mathrm{F}} = 4 \times 10^9$ bytes/s, $\mathrm{BW}_{\mathrm{D}} = 68 \times 10^9$ bytes/s of available execution-read bandwidth, and $C = 45 \times 10^{12}$ operations/s. The resource service times and expert-payload throughput ceiling are

$$\tau_{\mathrm{F}} = \frac{V_{\mathrm{F}}}{\mathrm{BW}_{\mathrm{F}}}, \qquad \tau_{\mathrm{D}} = \frac{V_{\mathrm{R}}}{\mathrm{BW}_{\mathrm{D}}}, \qquad \tau_{\mathrm{C}} = \frac{O_{\mathrm{E}}}{C}, \tag{30}$$

$$R_{\mathrm{E}} = \frac{1}{\max(\tau_{\mathrm{F}}, \tau_{\mathrm{D}}, \tau_{\mathrm{C}})} = \min\left(\frac{\mathrm{BW}_{\mathrm{F}}}{V_{\mathrm{F}}}, \frac{\mathrm{BW}_{\mathrm{D}}}{V_{\mathrm{R}}}, \frac{C}{O_{\mathrm{E}}}\right). \tag{31}$$

Equation (31) is a resource bound regardless of overlap. Attaining it requires sufficient overlap between transfers and execution; sequential layer dependencies and runtime overhead can reduce achievable rates. Because the paper uses decimal MB, the flash-only ceiling is simply $4000/T_{\mathrm{flash}}$ tokens/s. The ideal compute ceiling exceeds 12,000 tokens/s for every BitLift-R point, so a bandwidth term determines each tabulated estimate.

DRAM accounting. The 68 GB/s rate is the assumed bandwidth available to execution reads, with the remaining allocation providing allowance for ingestion and interference. We therefore do not add ingestion writes separately to Eq. (30). If instead bandwidth denotes total shared DRAM service, one must count both reads and writes:

$$\tau_{\mathrm{D}}^{\mathrm{total}} = \frac{V_{\mathrm{R}} + V_{\mathrm{F}} + V_{\mathrm{other}}}{\mathrm{BW}_{\mathrm{D}}^{\mathrm{total}}}. \tag{32}$$

This expression assumes one DRAM write per fetched byte. V_{other} accounts for additional DRAM traffic, including metadata, activations, and intermediate buffers.

E.3 Bandwidth Headroom and Cache Regimes

Under the available-read-bandwidth convention, let $\gamma = \mathrm{BW}_{\mathrm{D}}/\mathrm{BW}_{\mathrm{F}} = 17$. Expert-weight reads fit within the flash service time when

$$\bar{b}_{\mathrm{exec}} \le \underbrace{\gamma f}_{b_{\mathrm{limit}}}. \tag{33}$$

For a flash-limited nominal policy, the available bandwidth headroom is $b_{\mathrm{limit}} - B$ additional bits per routing slot, subject to the supported precision levels and cache residency. Under total DRAM accounting, the corresponding limit is $(\gamma_{\mathrm{total}} - 1)f - V_{\mathrm{other}}/\Gamma$, where $\gamma_{\mathrm{total}} = \mathrm{BW}_{\mathrm{D}}^{\mathrm{total}}/\mathrm{BW}_{\mathrm{F}}$.

Table 5 applies the model to the WikiText-2 measurements. Qwen has $L = 48$, $K = 8$, and $P_e = 4{,}718{,}592$, giving $\Gamma = 226{,}492{,}416$ bytes. OLMoE and DeepSeek have $\Gamma = 100{,}663{,}296$ and $168{,}689{,}664$ bytes, respectively. All calculations use unrounded source measurements.

Tight to moderate cache budgets. On Qwen, $D = 32$–128 spans 6.25–25% of the full 4-bit routed-expert footprint. BitLift-R remains flash-limited within the expert-payload model, despite increasing average execution precision from 2.665 to 3.134 bits on WikiText-2. At $D = 128$, it reads 709.92 MB/token from DRAM and fetches 64.83 MB/token from flash, giving service times of 10.44 and 16.21 ms/token. Its ceiling is therefore 61.7 tokens/s, close to BitLift-A's 62.4 tokens/s. The same resource regime holds for all evaluated OLMoE and DeepSeek budgets on WikiText-2. Here, reductions in flash traffic translate directly into higher resource ceilings, while cache-resident refinement mainly improves quality.

Moderate residency and DRAM saturation. At Qwen $D = 256$, the cache holds 50% of the 4-bit routed-expert footprint. BitLift-R's WikiText-2 flash payload falls to 11.55 MB/token, but its 3.606-bit average execution requires 816.62 MB/token of expert reads. Flash service takes 2.89 ms/token, whereas the reads require 12.01 ms/token. Consequently, the 346.2-token/s flash-only ceiling is capped at 83.3 tokens/s by DRAM. BitLift-A, with its fixed 537.92 MB/token of expert reads, reaches a higher ceiling of 126.4 tokens/s.

The same saturation dilutes the throughput benefit over uniform precision. On WikiText-2, uniform Q4's ceiling rises from 7.8 to 24.8 tokens/s over $D = 32$–128, making BitLift-R's ceiling 2.06–$2.49\times$ higher. At $D = 256$, Q4 is also DRAM-limited at 75.1 tokens/s: BitLift-R's 63% flash-traffic reduction then corresponds to only a $1.11\times$ higher expert-payload ceiling. Uniform Q3 reaches 100.1 tokens/s at this budget, above BitLift-R, whose resident refinements execute at higher precision. Thus the conversion from traffic savings to throughput benefits is most direct at tight to moderate budgets; as flash I/O shrinks, DRAM reads and execution precision become decisive. Cache fractions here exclude metadata, non-routed weights, and KV storage, as in Table 8.

Table 5: Expert-payload throughput analysis on WikiText-2. Execution precision and traffic columns describe BitLift-R; A and R denote BitLift-A and BitLift-R throughput ceilings. V_{F} and V_{R} are decimal MB/token. Rates assume 4 GB/s flash, 68 GB/s available execution-read bandwidth, and 45 TOPS for routed matrix operations. † marks a DRAM limit; all other listed points are flash-limited under this model. These are analytical ceilings, not measured device throughputs.

			MB/token		Ceiling (tok/s)	
Model	D	$\bar{b}_{\mathrm{exec}}$	V_{F}	V_{R}	BitLift-A	BitLift-R
Qwen3	32	2.665	245.70	603.49	16.4	16.3
	48	2.786	190.79	631.11	21.2	21.0
	64	2.878	153.09	651.92	26.4	26.1
	128	3.134	64.83	709.92	62.4	61.7
	256	3.606	11.55	816.62	126.4†	83.3†
OLMoE	32	2.569	140.76	258.62	28.4	28.4
	48	2.635	111.78	265.27	35.8	35.8
	64	2.692	90.13	270.97	44.4	44.4
	128	3.057	31.34	307.77	128.0	127.6
DeepSeek	32	2.548	233.85	429.74	17.1	17.1
	48	2.671	192.88	450.57	20.7	20.7
	64	2.758	160.09	465.24	25.0	25.0
	128	3.043	67.21	513.30	59.5	59.5

E.4 Serialization and Overhead Sensitivity

To examine dependence on overlap, we serialize all three modeled services for both methods, keeping the same traffic and reference rates:

$$T_{\mathrm{serial},m} = \tau_{\mathrm{F},m} + \tau_{\mathrm{D},m} + \tau_{\mathrm{C},m}, \qquad R_{\mathrm{serial},m} = 1/T_{\mathrm{serial},m}. \tag{34}$$

Table 6 shows that BitLift-R retains 1.09–$1.23\times$ the modeled rates of uniform 3-bit GPTQ without overlap at $D = 48, 64, 128$. This sensitivity calculation excludes unmodeled work and is not a lower bound on end-to-end speedup.

Table 6: Overlap sensitivity at the reference rates (Qwen3, WikiText-2). Ratios compare BITLIFT-R with uniform 3-bit GPTQ; ΔT is the overhead difference at break-even in the serialized model.

	Modeled R/Q3 rate ratio		
D	Resource ceiling	Serialized	ΔT (ms/token)
48	1.26×	1.23×	13.0
64	1.26×	1.22×	10.3
128	1.17×	1.09×	2.4

Additional execution cost. Our Triton kernels reconstruct weight tiles within matrix multiplication, avoiding a full dequantized expert buffer. The resource model does not time unpacking, metadata access, dispatch, or other full-model work. Let H_m denote additional per-token time beyond the serialized service terms, so $T_m = T_{\text{serial},m} + H_m$. BITLIFT-R retains the advantage when

$$H_{\text{R}} - H_{\text{Q3}} < \underbrace{T_{\text{serial,Q3}} - T_{\text{serial,R}}}_{\Delta T}. \tag{35}$$

The break-even differences are 13.0, 10.3, and 2.4 ms/token at the three budgets, respectively. These are allowable overhead differences, not measured kernel costs; tighter caches provide greater tolerance for additional execution work.

Bandwidth sensitivity. When both methods are flash-limited at a common effective flash bandwidth, their ceiling ratio is $R_{\text{E,R}}/R_{\text{E,Q3}} = V_{\text{F,Q3}}/V_{\text{F,R}}$; the absolute flash rate cancels. At 4 GB/s flash and the reference arithmetic rate, both methods remain flash-limited for available DRAM read bandwidths above approximately 13.2, 17.0, and 43.8 GB/s at $D = 48, 64, 128$, respectively. For fixed traffic and other service rates, reducing the common flash bandwidth increases ΔT because BITLIFT-R fetches fewer bytes. These comparisons assume the same effective service rates for both methods; differences in fetch efficiency or execution overhead must be accounted for separately.

F EXPERIMENTAL DETAILS

Models. We evaluate the base checkpoints of Qwen3-30B-A3B, OLMoE-1B-7B-0125, and DeepSeek-V2-Lite. Table 7 summarizes their parameter composition and routing structure. Qwen3 and OLMoE select eight experts per MoE layer, while DeepSeek selects six and additionally uses shared experts.

Table 7: Model architecture and parameter composition. Expert parameters include routed and shared experts; non-expert parameters include attention, embeddings, routers, normalization, and the language-model head. Router share is a subset of non-expert share. The expert count is per layer times the number of MoE layers; DeepSeek has 64 routed and two shared experts per MoE layer, excluding its initial dense layer. Qwen3 and OLMoE have no shared experts. Top-K selection applies only to routed experts.

		Parameter share (%)				
Model	Parameters	Expert	Non-expert	Router	Experts × layers	K
Qwen3-30B-A3B	30.5B	94.95	5.05	0.041	128×48	8
OLMoE-1B-7B-0125	6.9B	93.11	6.89	0.030	64×16	8
DeepSeek-V2-Lite	15.7B	94.51	5.49	0.022	$(64 + 2) \times 26$	6

Quantization and calibration. BITLIFT builds on asymmetric group-wise GPTQ (Frantar et al., 2022), with group size 128, block size 128, and Hessian damping factor 0.05. Calibration uses 128 samples of 2,048 tokens each from the WikiText-2 training split. Perplexity evaluation also uses a sequence length of 2,048 tokens. We apply Group-Aware Reordering (GAR) (Gafni et al., 2025), which reorders columns within groups and/or moves complete groups while preserving group membership. Affine range fitting uses `maxshrink=0.8` and `grid=100`. The fitted INT4

grid initializes the shared representation; joint scale fitting then adjusts its scale under the policy-weighted objective. Routed experts support 2-, 3-, and 4-bit execution, with zero-bit assignments denoting skipping. Attention projections and dense or shared experts remain at 4 bits (Huang et al., 2025; Deng et al., 2026). We implement quantization and evaluation in PyTorch and Transformers, using bit-packed expert planes and custom Triton kernels that reconstruct weight tiles within matrix multiplication.

Precision schedules and budgets. For Table 1, Qwen3 and OLMoE use $(4, 3, 3, 3, 3, 2, 2, 0)$ and $(4, 3, 3, 3, 2, 2, 2, 0)$ at nominal average precisions of 2.5 and 2.375 bits, respectively. DeepSeek uses $(4, 3, 3, 3, 2, 0)$ and $(4, 3, 3, 2, 2, 0)$ at 2.5 and $14/6 \approx 2.333$ bits. Entries follow decreasing router score. The budget $B = K^{-1} \sum_{r=1}^{K} \rho_r$ averages assigned precision over all routing slots, including skipped slots. It describes nominal execution requests; the shared representation stores four payload bits per weight, excluding quantizer metadata. Each schedule is fixed across tokens and layers, while the expert identities occupying its ranks vary with routing.

Predictive-quality evaluation and baselines. Table 1 reports WikiText-2 test and C4 validation perplexity, together with zero-shot accuracy on ARC-Challenge, ARC-Easy, HellaSwag, PIQA, WinoGrande, MathQA, and MMLU. Average accuracy is the unweighted mean over these seven tasks. BF16 provides the full-precision reference. Uniform 3-bit GPTQ uses independently optimized 3-bit weights. Following GEMQ (Deng et al., 2026), uniform 2.5-bit GPTQ assigns 3 bits to routed experts in the first half of MoE layers and 2 bits in the remainder, yielding 2.5 bits on average. We also compare with GEMQ, KBVQ-MoE, and the D^{2}MoE representation (Wang et al., 2025); the latter uses BITLIFT-A's 2.5-bit schedules rather than its learned precision controller. GEMQ reports average assigned bitwidth across experts, while KBVQ-MoE reports nominal weight bitwidth excluding additional representation overhead. Table 1 compares predictive quality at these reported budgets. These precision-policy comparisons disable resident refinement and cache-aware routing boosts.

Table 2 isolates representation quality under the same 2.5-bit rank policies. Independently optimized 2-, 3-, and 4-bit copies provide a reference, requiring nine payload bits per weight to retain all three precisions. Shared alternatives include sequential residual fitting, MatGPTQ, and SliceMoE-style AMAT. The MatGPTQ configuration uses an additional rounding bit; AMAT obtains lower-precision reconstructions through zero-point-aware truncation of a shared INT4 representation. Appendix D illustrates the resulting reconstruction tradeoffs.

Cache configuration and capacity. Caching experiments use the 2.375-bit schedule for Qwen3 and OLMoE and the $14/6$-bit schedule for DeepSeek. All three progressive checkpoints use $(\lambda_2, \lambda_3, \lambda_4) = (0.10, 0.45, 0.45)$. Both BitLift modes use per-layer two-pool SLRU with identical base and refinement allocations; BITLIFT-A executes assigned precisions, while BITLIFT-R enables cache-resident refinement and revival. Uniform 2-, 3-, and 4-bit GPTQ use independently optimized checkpoints and whole-expert SLRU, with $\lfloor D/b \rfloor$ expert slots per layer at bit width b. Every SLRU cache targets a 75% protected fraction. All designs receive the same weight-payload budget within a model; whole-expert caches may leave fewer than b pages unused because of integer slot allocation.

Table 8 converts the logical budget into physical capacity using $M_{\text{cache}} = LDP_e/8$ bytes and $1\,\text{GiB} = 2^{30}$ bytes. The fraction of the complete 4-bit routed-expert footprint is $D/(4E)$, where E is the number of routed experts per layer. Capacity accounting excludes metadata, shared experts, and non-routed weights.

Table 8: Routed-expert cache capacity and fraction of the full 4-bit routed-expert footprint. Capacity excludes metadata and non-routed weights, including shared experts. A dash denotes a budget not evaluated.

	Qwen3		**OLMoE**		**DeepSeek**	
D (pages/layer)	GiB	4-bit (%)	GiB	4-bit (%)	GiB	4-bit (%)
32	0.844	6.25	0.375	12.50	0.838	12.50
48	1.266	9.38	0.562	18.75	1.257	18.75
64	1.688	12.50	0.750	25.00	1.676	25.00
128	3.375	25.00	1.500	50.00	3.352	50.00
256	6.750	50.00	–	–	–	–

Caching perplexity and flash traffic. The caching analyses in Appendix G use disjoint 2,048-token windows: the complete tokenized WikiText-2 test split and 256 C4 validation windows selected with shuffle seed 0. The C4 evaluation therefore uses 524,288 tokens, more than the predictive-quality evaluation in Table 1. The different C4 samples can yield different absolute perplexities for the same checkpoint; methods within each comparison use identical windows.

Cache replay starts empty and includes compulsory loads. Cache-only ablations replay identical router traces, so replacement choices change traffic without changing model outputs. Refinement experiments execute the modified model and measure both perplexity and traffic, since refined outputs can affect subsequent routing. Optional refinement and revival reads leave cache replacement state unchanged. We count transferred weight payload only, excluding metadata, and convert normalized traffic using $T_{\text{flash}} = t_{\text{flash}} LKP_e/(8 \times 10^6)$ MB/token.

Downstream caching evaluation. We use `lm-eval` 0.4.12 with five-shot prompting on all 14,042 MMLU and 1,319 GSM8K examples. MMLU reports accuracy; GSM8K uses `exact_match,flexible-extract`. Table 9 includes the uniform 2-bit accuracy references omitted from the main figures to keep the 3- and 4-bit operating range readable. In BITLIFT-R, the nominally skipped expert can execute only when its base is resident, with any additional refinements required to form a complete prefix also resident.

Routing–refinement evaluation. On Qwen3 at $D \in \{48, 64, 128\}$, we sweep $\beta \in \{0, 0.05, 0.10, 0.20, 0.30, 0.35, 0.40\}$ for BITLIFT-R and uniform 2-, 3-, and 4-bit GPTQ. Evaluation uses three 64-window WikiText-2 samples of 131,072 tokens each, selected with window-shuffle seeds 0, 1, and 2. Within each seed, all methods use the same windows and matched per-layer cache budgets, with the SLRU configurations above.

Each positive-β point in the routing-boost figure reports a 95% paired-bootstrap interval for its change from the corresponding $\beta = 0$ control. We use 48 matched batches across the three seeds and 50,000 bootstrap replicates, stratified by seed. The reference points have zero-length intervals because the plotted uncertainty concerns changes relative to those references. To compare traffic at a target perplexity, we linearly interpolate between measured boost settings.

G ADDITIONAL CACHING QUALITY AND TRAFFIC

We extend the caching results in Sections 4.2 and 4.3 with C4 perplexity and flash traffic, and five-shot GSM8K accuracy. BITLIFT-A uses rank-assigned execution; BITLIFT-R adds cache-resident precision refinement. As in the main experiments, uniform GPTQ baselines use whole-expert SLRU and both BITLIFT modes use per-layer two-pool SLRU, with a 75% protected fraction at matched expert-weight cache capacity. WT2 denotes WikiText-2.

Predictive quality under dynamic precision. Table 9 reports five-shot MMLU and GSM8K accuracy with cache-resident refinement disabled. BITLIFT-A retains MMLU accuracy within 0.93 percentage points of uniform 3-bit GPTQ and achieves higher GSM8K accuracy on all three models. BF16 provides a full-precision reference; independently optimized uniform 2-, 3-, and 4-bit GPTQ provide fixed-width comparisons, including the 2-bit results omitted from the main quality panels.

Table 9: Five-shot MMLU and GSM8K accuracy (%, higher is better). BF16 is the full-precision reference; uniform GPTQ models are optimized independently at each bit width.

	Qwen3		OLMoE		DeepSeek	
Method	MMLU	GSM8K	MMLU	GSM8K	MMLU	GSM8K
BF16	79.58	84.91	55.18	51.93	58.03	37.98
Uniform 2-bit GPTQ	71.52	80.52	45.55	16.00	48.85	17.21
Uniform 3-bit GPTQ	78.01	81.96	52.60	38.97	55.09	30.10
Uniform 4-bit GPTQ	78.97	85.14	53.77	45.94	55.84	34.65
BITLIFT-A	78.02	86.81	51.89	40.64	54.16	30.25

Flash traffic with policy-aware caching. BITLIFT-A's perplexity remains unchanged as cache capacity varies because execution uses the same rank-assigned precisions. Table 10 reports the selected cache splits and expert-weight flash traffic on WikiText-2 and C4. Traffic decreases with capacity on both corpora across all three models. At matched cache capacity over $D = 32$–128, BITLIFT-A reduces flash traffic by 12–29% relative to uniform 3-bit GPTQ and 50–66% relative to uniform 4-bit GPTQ across both corpora.

Table 10: Flash traffic for BITLIFT-A with per-layer two-pool SLRU. C_{B} and C_{R} count base objects and refinement planes, with $2C_{\mathrm{B}} + C_{\mathrm{R}} = D$; base share is $2C_{\mathrm{B}}/D$. Traffic is reported as t_{flash} and decimal MB/token, following Eq. equation 3.

				WT2		C4	
Model	D	$C_{\mathrm{B}}/C_{\mathrm{R}}$	Base (%)	t_{flash}	MB/token	t_{flash}	MB/token
Qwen3	32	12/8	75.0	1.076	243.8	1.074	243.2
	48	18/12	75.0	0.835	189.1	0.822	186.2
	64	24/16	75.0	0.669	151.6	0.648	146.9
	128	48/32	75.0	0.283	64.1	0.263	59.6
	256	85/86	66.4	0.051	11.5	0.051	11.7
OLMoE	32	13/6	81.2	1.397	140.6	1.378	138.8
	48	20/8	83.3	1.109	111.7	1.088	109.5
	64	27/10	84.4	0.894	90.0	0.874	88.0
	128	48/32	75.0	0.310	31.2	0.304	30.6
DeepSeek	32	13/6	81.2	1.387	234.0	1.402	236.5
	48	18/12	75.0	1.144	193.0	1.150	194.0
	64	24/16	75.0	0.950	160.2	0.948	159.9
	128	48/32	75.0	0.399	67.3	0.395	66.6

Effect of cache-resident precision refinement. At matched cache capacity, BITLIFT-R improves C4 perplexity and five-shot GSM8K accuracy over BITLIFT-A at every evaluated budget across all three models. Table 11 reports perplexity changes on both corpora; C4 reductions span 0.24–0.76 for Qwen3, 0.13–0.39 for OLMoE, and 0.053–0.138 for DeepSeek. At $D = 128$, BITLIFT-R achieves lower perplexity than uniform 3-bit GPTQ on both corpora while transferring 12–19% fewer expert-weight bytes. GSM8K accuracy improves by 1.97–3.79 percentage points over BITLIFT-A at this budget. On Qwen3 at $D = 256$, BITLIFT-R reaches perplexity comparable to uniform 4-bit GPTQ: 0.008 lower on WikiText-2 and 0.090 higher on C4, with 58–63% less flash traffic.

Table 11: Perplexity change from BITLIFT-A to BITLIFT-R, $\Delta\mathrm{PPL} = \mathrm{PPL}_{\mathrm{R}} - \mathrm{PPL}_{\mathrm{A}}$. Negative values indicate improvement; – denotes an unevaluated budget.

	Qwen3		OLMoE		DeepSeek	
D	WT2	C4	WT2	C4	WT2	C4
32	−0.123	−0.236	−0.126	−0.131	−0.026	−0.053
48	−0.160	−0.313	−0.162	−0.176	−0.039	−0.078
64	−0.192	−0.391	−0.192	−0.212	−0.048	−0.094
128	−0.260	−0.562	−0.374	−0.393	−0.072	−0.138
256	−0.355	−0.762	–	–	–	–

H ADDITIONAL POLICY ABLATIONS

The main experiments use one selected rank policy per model in both BitLift modes. We add two Qwen3 ablations on WikiText-2 to examine how the cache allocation and the final quality–traffic operating point change under alternative rank-precision policies. These experiments are targeted policy comparisons rather than a uniform sweep over average bitwidth, and they do not establish a universally optimal policy across models or datasets.

H.1 POLICY-DEPENDENT CACHE SPLIT

We first replay seven precision policies over the same fixed router trace at $D = 48$, varying (C_{B}, C_R) over seven feasible allocations. Holding the trace fixed isolates the traffic effect of base and residual demand; it does not measure policy-induced changes in hidden states or routing. Table 12 shows that the traffic-minimizing allocation adapts at the extremes. A residual-light 1.875-bit policy favors $22/4$, whereas residual-heavy policies favor $14/20$ or $12/24$. Nevertheless, $18/12$ remains optimal for every tested graded-like policy between 2.25 and 2.50 bits.

Table 12: Policy-dependent base/residual allocation for Qwen3 at $D = 48$ on a fixed WikiText-2 router trace. Demand reports the policy-implied Q2/residual page mix. The final column is the traffic penalty from using the selected $18/12$ split rather than the per-policy optimum. Lower traffic is better.

Rank-precision policy	Avg. bits	Q2/residual demand	Best C_{B}/C_R	Best MB/token	$18/12$ penalty
$[3, 2, 2, 2, 2, 2, 2, 0]$	1.875	93.3/6.7%	22/4	119.28	11.44%
$[3, 3, 3, 3, 2, 2, 2, 0]$	2.250	77.8/22.2%	**18/12**	167.23	0
$[3, 3, 3, 3, 3, 2, 2, 0]$	2.375	73.7/26.3%	**18/12**	179.94	0
$[4, 3, 3, 3, 2, 2, 2, 0]$	2.375	73.7/26.3%	**18/12**	189.07	0
$[4, 3, 3, 3, 3, 2, 2, 0]$	2.500	70.0/30.0%	**18/12**	202.87	0
$[4, 4, 4, 3, 3, 3, 2, 0]$	2.875	60.9/39.1%	14/20	253.38	3.43%
$[4, 4, 4, 4, 4, 4, 4, 0]$	3.500	50.0/50.0%	12/24	332.76	14.19%

The optimum need not exactly equal the nominal demand ratio because temporal reuse, eviction, and the discrete split grid also affect traffic. The result instead shows that the chosen allocation is stable for nearby graded policies while meaningfully different residual demand can justify a different partition.

H.2 COMPARISON OF RANK-PRECISION POLICIES

We next compare five targeted policies using the same progressive checkpoint and the fixed 75/25 cache split at $D \in \{48, 64, 128\}$. The 2.125- and 2.25-bit variants test low-cost policies that still retain 4-, 3-, 2-, and zero-bit assignments; the 2.375-bit graded policy is the selected BitLift design; a matched-bit no-R4 policy removes the highest refinement level; and a 2.625-bit no-zero policy raises the assigned precision floor. For each policy, we evaluate BitLift-A (assigned execution) and BitLift-R (cache-resident refinement).

Across the four policies that retain an assigned 4-bit rank, the BitLift-R points exhibit a shallow U-shaped quality–traffic trend at every cache budget. Moving from 2.125 or 2.25 bits toward the

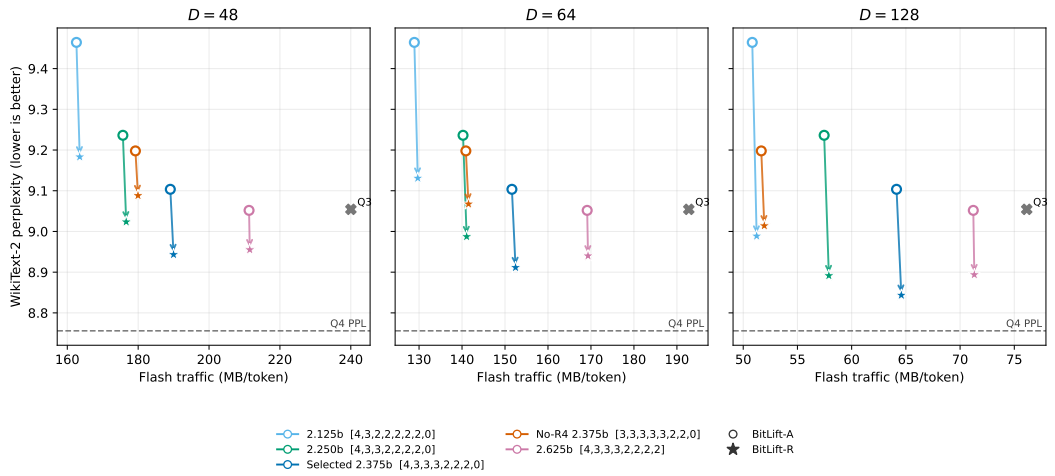

Figure 8: Comparison of Qwen3 rank-precision policies on WikiText-2. Circles show BitLift-A and stars show BitLift-R, with each arrow connecting the same policy. Uniform references use whole-expert SLRU at matched capacity. The uniform Q3 point is shown explicitly; the dashed line gives Q4 perplexity because its traffic lies outside the cropped range. All policies use the fixed 75/25 base/residual split. Lower-left is better.

selected graded policy improves quality, but increasing the assigned floor further with the 2.625-bit no-zero policy slightly worsens final perplexity while increasing traffic by roughly 10–11%. The selected graded policy therefore attains the lowest BitLift-R perplexity at each budget (8.943, 8.911, and 8.843 for $D = 48, 64, 128$) and lies at the favorable knee of the observed comparison.

The policies immediately to its left remain competitive lower-traffic alternatives. The 2.25-bit policy is only 0.080, 0.076, and 0.048 PPL worse while using 7–10% less traffic. At $D = 128$, even the 2.125-bit policy reaches 8.989 PPL at 51.2 MB/token, slightly better than the uniform Q3 reference at 9.054 PPL and 76.1 MB/token. The matched-bit no-R4 control never places an R4 plane in the cache and consequently cannot use resident 4-bit upgrades. These comparisons explain why the graded policy is favorable in the tested setting without suggesting that small changes to the policy cause a sharp degradation.

I Cache Design Ablations

We isolate topology, replacement, layer scope, and capacity allocation on Qwen3-30B-A3B and OLMoE-1B-7B-0125 under the 2.375-bit policy $(4, 3, 3, 3, 2, 2, 2, 0)$. Each comparison replays identical nominal requests at matched weight-payload capacity, with precision upgrades and routing boosts disabled; model outputs therefore remain unchanged. All traffic values report t_{flash} (Eq. equation 3): transferred bits per weight averaged over routing slots, including skipped slots. Lower is better. D denotes pages per layer, with $2C_{\text{B}} + C_{\text{R}} = D$. WT2 abbreviates WikiText-2.

Table 13: Qwen3 topology with per-layer LRU at $D = 48$. Parentheses give separate R3/R4 page allocations. Reductions are relative to uncached BitLift-A ($t_{\text{flash}} = 2.375$). Bold marks the best results.

	WT2		C4	
Topology	$t_{\text{flash}} \downarrow$	Reduction (%)	$t_{\text{flash}} \downarrow$	Reduction (%)
Single pool	0.8705	63.35	0.8629	63.67
Two pools	**0.8574**	**63.90**	**0.8488**	**64.26**
Three pools (10/2)	0.8579	63.88	0.8493	64.24
Three pools (9/3)	0.8612	63.74	0.8529	64.09

Cache topology. We fix per-layer LRU at $D = 48$. Two- and three-pool designs reserve 36 pages for 18 base objects; refinements share the remaining 12 pages or receive separate allocations. Two pools reduce traffic relative to one pool by 1.5% on WT2 and 1.6% on C4 (Table 13). The best three-pool split is only 0.0005 higher in t_{flash} on either dataset. Thus, the tested designs support reserving base capacity while sharing refinement capacity according to reuse.

Replacement policy. With per-layer two-pool allocations fixed, we compare LRU with SLRU, which admits new objects to a probationary segment and promotes reused objects to a protected segment targeting 75% of capacity. SLRU reduces traffic on both models and datasets at $D = 48$–128 (Table 14). At $D = 32$, it nearly ties LRU on Qwen3 but increases OLMoE traffic by 1.39–2.15%; at Qwen3 $D = 256$, differences fall below 1%. These results support SLRU as the default at moderate capacities.

Table 14: Replacement at matched per-layer two-pool allocations. $\Delta = 100(t_{\text{SLRU}}/t_{\text{LRU}} - 1)$; negative values favor SLRU. Lower traffic is bold.

		WT2			C4		
Model	D	LRU	SLRU	Δ (%)	LRU	SLRU	Δ (%)
Qwen3	32	1.0765	**1.0763**	−0.03	1.0755	**1.0737**	−0.16
	48	0.8574	**0.8348**	−2.64	0.8488	**0.8220**	−3.17
	64	0.6943	**0.6694**	−3.59	0.6791	**0.6484**	−4.52
	128	0.2960	**0.2832**	−4.32	0.2751	**0.2630**	−4.40
	256	0.0509	**0.0506**	−0.62	0.0517	**0.0515**	−0.50
OLMoE	32	**1.3780**	1.3972	+1.39	**1.3494**	1.3784	+2.15
	48	1.1216	**1.1093**	−1.09	1.0950	**1.0882**	−0.63
	64	0.9142	**0.8940**	−2.21	0.8893	**0.8745**	−1.66
	128	0.3266	**0.3104**	−4.96	0.3196	**0.3045**	−4.73

Per-layer versus global scope. We hold two-pool SLRU fixed and compare D pages per layer with LD pages shared globally. Per-layer caching prevents cross-layer evictions; global pooling redistributes capacity between layers. Per-layer caching yields lower traffic on both models and datasets at $D = 32$–64, whereas global pooling becomes preferable at $D = 128$, with larger gains at Qwen3 $D = 256$ (Table 15). This reversal supports choosing layer scope according to the available capacity.

Table 15: Layer scope at matched total capacity under two-pool SLRU. $\Delta = 100(t_{\text{global}}/t_{\text{per-layer}} - 1)$; positive values favor per-layer caching. Lower traffic is bold.

		WT2			C4		
Model	D	Per-layer	Global	Δ (%)	Per-layer	Global	Δ (%)
Qwen3	32	**1.076**	1.764	+63.9	**1.074**	1.921	+78.9
	48	**0.835**	0.966	+15.7	**0.822**	1.033	+25.6
	64	**0.669**	0.694	+3.6	**0.648**	0.668	+3.0
	128	0.283	**0.266**	−5.9	0.263	**0.248**	−5.7
	256	0.051	**0.040**	−21.3	0.051	**0.043**	−16.0
OLMoE	32	**1.397**	1.804	+29.1	**1.378**	1.869	+35.6
	48	**1.109**	1.288	+16.1	**1.088**	1.281	+17.7
	64	**0.894**	1.019	+13.9	**0.874**	1.026	+17.3
	128	0.310	**0.307**	−1.0	0.304	**0.298**	−2.0

Base–refinement allocation. We vary only the split under per-layer SLRU (Table 16). Here, C_{B} counts two-page base objects, C_{R} counts one-page refinements, and the base share is $2C_{\text{B}}/D$. At $D = 48$, the selected splits—$(18, 12)$ for Qwen3 and $(20, 8)$ for OLMoE—minimize traffic among the listed candidates on both datasets; devoting all capacity to bases worsens OLMoE traffic. At $D = 128$, $(48, 32)$ is best except on Qwen3 C4, which slightly favors $(43, 42)$ (0.2618 versus 0.2630). This sensitivity motivates selecting allocations using observed reuse under the target policy.

Table 16: Local split sensitivity under per-layer SLRU. Base share is $2C_{\mathrm{B}}/D$. Bold entries are the evaluation configurations, which need not minimize each dataset's traffic.

	Qwen3				OLMoE			
D	$C_{\mathrm{B}}/C_{\mathrm{R}}$	Base (%)	WT2	C4	$C_{\mathrm{B}}/C_{\mathrm{R}}$	Base (%)	WT2	C4
48	16/16	66.7	0.8470	0.8355	18/12	75.0	1.1152	1.0946
	18/12	**75.0**	**0.8348**	**0.8220**	**20/8**	**83.3**	**1.1093**	**1.0882**
	20/8	83.3	0.8466	0.8332	24/0	100.0	1.1887	1.1728
128	43/42	67.2	0.2853	0.2618	43/42	67.2	0.3255	0.3162
	48/32	**75.0**	**0.2832**	**0.2630**	**48/32**	**75.0**	**0.3104**	**0.3045**
	53/22	82.8	0.3052	0.2903	53/22	82.8	0.3308	0.3264