AlignQuant: TILE-ALIGNED MIXED-PRECISION QUANTIZATION FOR EFFICIENT LLM GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Fine-grained mixed-precision quantization promises efficient large language model inference, but local precision choices can conflict with regular GPU storage and computation units. This precision-boundary mismatch limits the translation of compression into practical acceleration. We introduce *AlignQuant*, a post-training quantization method that uses GPU-compatible two-dimensional weight tiles as the common unit of precision allocation, compact storage, and execution. This shared partition lets precision follow sensitivity within output channels. Joint prefill/decode calibration scores precision reductions using projection-output perturbations weighted by language-model loss gradients under quantized activations. Phase-normalized scores prioritize higher precision for tiles important to either phase under a model-wide weight-storage budget. Each tile stores one selected representation, while phase-specialized kernels reuse the packed model and expand lower-bit weights for INT8 computation with 8-bit activations. Across four LLMs spanning 3B to 14B parameters, *AlignQuant* achieves up to $2.50\times$ generation speedup over BF16 while preserving model quality. Evaluations further cover three GPUs and contexts up to 64K tokens. These results show that local precision flexibility and regular GPU execution can coexist through a shared tile unit. The implementation is available at `https://anonymous.4open.science/r/AlignQuant-0B64`.

1 INTRODUCTION

Large language model (LLM) inference is increasingly constrained by parameter storage, memory traffic, and computation. Post-training quantization (PTQ) reduces these costs through lower-precision representations, while the resulting quantization error makes accuracy preservation a central challenge (Dettmers et al., 2022; Xiao et al., 2023; Frantar et al., 2022; Lin et al., 2024b). Uniform-bit quantization addresses this challenge by improving the quantized representation at a fixed weight precision (Frantar et al., 2022; Lin et al., 2024b). Mixed-precision quantization additionally exploits differences in sensitivity by assigning higher precision to sensitive weight regions under a weight-storage budget (Chen et al., 2024; Huang et al., 2024; Zheng et al., 2026; Li et al., 2026; Nie et al., 2026; Hooper et al., 2025). However, when fine-grained precision boundaries cross regular storage or GPU computation units, the resulting **precision-boundary mismatch** makes efficient deployment depend on how mixed-bit weights are packed, loaded, and converted for computation (Zhao et al., 2024; Lin et al., 2025).

Figure 1 illustrates how precision allocation, weight representation, and GPU execution jointly shape model quality and inference efficiency. Existing methods manage this tension through different precision-sharing and execution structures. Channel- and output-feature-level methods maintain regular partitions by sharing one bit-width across each channel or output feature, coupling weights with different sensitivities to the same precision choice (Chen et al., 2024; Zheng et al., 2026). Finer-grained approaches expose more local choices through quantization groups and hardware-aligned blocks, together with reordering, mixed formats, or specialized execution support (Huang et al., 2024; Li et al., 2026; Nie et al., 2026; Hooper et al., 2025). System-oriented quantizers co-design low-bit formats with weight layouts, dequantization, and GPU data paths to control these deployment costs (Zhao et al., 2024; Lin et al., 2025). These approaches expose a structural trade-off: coarser sharing restricts where higher precision can be placed, while finer allocation requires coor-

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.

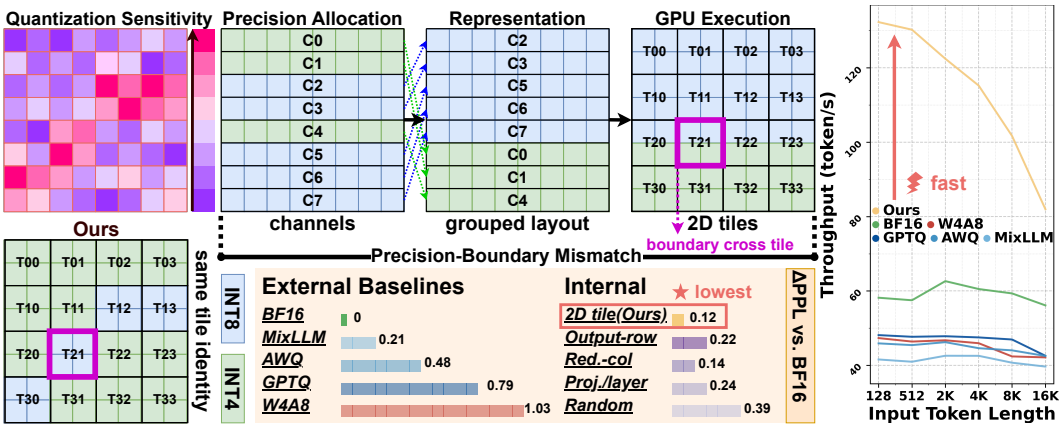

Figure 1: Comparison of precision allocation, representation, and GPU execution. Colors denote 8-bit/4-bit weight precision, and purple outlines track logical tiles. The schematic shows how channel-wise precision grouping can create a precision-boundary mismatch with regular two-dimensional GPU tiling, while *AlignQuant* uses a common weight-tile partition across allocation, storage, and execution. The quality panel reports WikiText-2 PPL increase over BF16, and the right panel reports generation throughput across prompt lengths on Llama-3.2-3B.

dinating the compact storage of heterogeneous weights with the formats used for GPU computation. This leads to our central question: *How can sensitivity-driven local precision choices be preserved in compact weight storage and direct GPU execution?*

Our key insight is to use GPU-compatible two-dimensional **weight tiles as a shared unit** for local precision selection, storage, and execution. Each tile receives its own bit-width, allowing precision to vary within output channels while providing a regular unit for storing and loading the selected weights. Our analysis supports this choice by showing that quantization sensitivity is localized within output channels. At a matched 4-bit tile fraction, two-dimensional tile allocation improves WikiText-2 perplexity over the output-row control. However, turning these local precision choices into efficient inference presents two coupled challenges. First, prefill processes prompt tokens and decode generates new tokens using the same weights. A shared precision assignment must therefore combine sensitivity evidence from both phases under one model-wide weight-storage budget. Second, the selected mixed-bit weights must retain their compact format in GPU memory and during loading while supporting integer computation tailored to each phase's workload (Lin et al., 2025).

We introduce *AlignQuant*, a tile-aligned mixed-precision method that constructs one compact model for efficient LLM generation. Joint loss-aware calibration assigns 4-bit or 8-bit precision under a model-wide budget, giving higher-precision priority to tiles important to either phase. Packing stores one selected representation per tile, preserving 4-bit weights in compact form through storage and loading. Phase-specialized kernels expand 4-bit weights on-chip to share INT8 computation with 8-bit weights and activations, directly consuming the same packed model.

Across Llama-3.2-3B, Qwen3-4B, Ministral-3-8B, and Qwen3-14B, *AlignQuant* achieves maximum 2.50× inference speedups over BF16 on an NVIDIA A100, NVIDIA H100 NVL, and RTX 4090 GPUs while preserving model quality. These results show how preserving precision granularity through weight storage and computation translates local quantization choices into practical inference efficiency. Our contributions are as follows:

1. **Tile-Aligned Precision Design:** We develop a two-dimensional weight-tile organization that permits local bit-width choices within output channels. The same partition defines the regions evaluated during calibration and the units used for compact storage and GPU loading.
2. **Joint Prefill/Decode Allocation:** We develop a loss-aware rule that scores local projection-output changes from 8-to-4-bit substitutions under 8-bit activations. Joint prefill/decode calibration produces one precision map under a model-wide storage budget, prioritizing 8-bit retention for tiles important to either phase.
3. **Compact Storage and Direct Execution:** We develop prefill- and decode-specialized integer kernels that directly consume a compact model with one selected representation per tile. Compact 4-bit weight loads and on-chip expansion let both weight formats share INT8 arithmetic for efficient inference.

2 Related Work

Mixed-precision LLM quantization couples three design dimensions: the granularity at which precision varies, the representation used to realize heterogeneous precision, and the numerical transformations used to reduce quantization error.

Sensitivity-aware precision allocation has progressively moved from coarse model structure toward localized decisions. Layer-wise methods reduce the number of precision decisions and preserve regular operators, while coupling regions of different sensitivity to one bit-width across each layer (Lee et al., 2025; Zhao et al., 2025). Channel-, output-feature-, and group-wise methods capture finer heterogeneity through structured units, coupling the weights within each unit to one precision and, under layer-local budgets, constraining high-precision placement (Chen et al., 2024; Huang et al., 2024; Zheng et al., 2026; Harma et al., 2026). Block-level schemes expose more local choices through hardware-aligned partitions, using global budgets, Fisher-weighted perturbation, or outlier-aware clustering for bit-width selection (Li et al., 2026; Nie et al., 2026; Hooper et al., 2025; Xie et al., 2025). Per-weight salience aggregation captures additive contributions, while whole-block precision changes interact through the projection output (Nie et al., 2026; Hooper et al., 2025). Phase-adaptive methods vary precision across inference phases and generation steps, adding runtime selection among quantized weight representations (Chen et al., 2025). Allocation therefore depends on spatial granularity, sensitivity evidence, and precision sharing across inference phases.

Low-bit representation and execution approach the same trade-off from the systems side. Low-bit systems co-design numerical formats, packing, data movement, and kernels for efficient GPU computation (Dettmers et al., 2022; Zhao et al., 2024; Lin et al., 2025; Hu et al., 2025a). Within this category, uniform weight formats couple sensitive and less sensitive regions to the same bit-width (Lin et al., 2025; Hu et al., 2025a). Executable mixed-precision systems organize heterogeneous weights through output-feature partitions, reordered blocks, expert groups, or hardware-supported formats, tying their precision choices to corresponding layouts, indexing, or arithmetic paths (Zheng et al., 2026; Nie et al., 2026; Duanmu et al., 2025; Liu et al., 2026; Jang & Tambe, 2025). Representation-centric methods use sparse outliers, residual subspaces, reusable binary codes, or fractional-bit quantizers to improve fidelity or precision flexibility, with corresponding indexed accesses, auxiliary products, or specialized reconstruction (Dettmers et al., 2023; Saxena et al., 2024; Park et al., 2026; Lee & Song, 2026). Inference efficiency therefore depends on how compact representations map to actual GPU arithmetic.

Transformation-based quantization reshapes weight and activation distributions to reduce quantization error. Scaling methods protect salient weights or redistribute quantization difficulty between weights and activations, with flexibility limited to feature-wise rescaling (Xiao et al., 2023; Lin et al., 2024b). Rotation- and permutation-based methods redistribute outliers across feature dimensions, while transforms spanning precision partitions change the weights assigned to each quantized region (Ashkboos et al., 2024; Liu et al., 2025; Lin et al., 2024a). Affine and jointly optimized scaling and orthogonal transformations offer richer distribution shaping, while spatial bit-width placement remains a separate decision (Ma et al., 2024; Hu et al., 2025b; Sun et al., 2024). Runtime smoothing adapts channel scales during inference to control dynamic ranges, while loss-sensitive weight placement requires a separate allocation criterion (Yi et al., 2025). These approaches improve numerical fidelity, while executable mixed precision also requires compatible transformation, allocation, and execution partitions.

3 *AlignQuant*

AlignQuant uses a common two-dimensional weight-tile partition to connect **precision allocation, compact storage, and GPU execution**. Each tile uses 4-bit or 8-bit weights, while uniform 8-bit activations provide a common input format for hardware-accelerated INT8 computation. Offline calibration produces a precision map under a model-wide weight-storage budget. This map records the selected bit-width at each tile coordinate and determines the weights stored in one packed model shared by prefill and decode.

Figure 2 follows a tile from candidate construction and joint calibration to compact storage and integer computation. We evaluate each replacement of 8-bit weights with 4-bit weights through its local projection-output change, weighted by gradients of the full model loss. Prefill and decode provide separate calibration evidence, which is combined to select one representation per tile. The runtime uses the selected weights and scales with execution schedules adapted to the two phases.

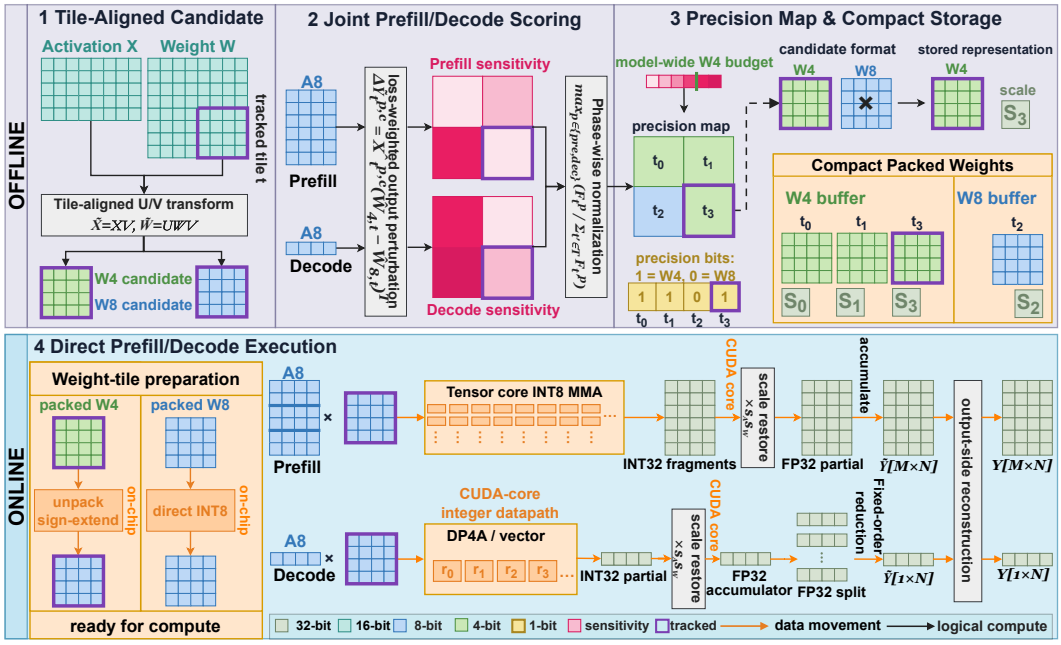

Figure 2: **Overview of *AlignQuant*.** Purple outlines track one logical weight tile. *(1) Candidates.* Tile-aligned U/V transforms prepare 4-bit and 8-bit weight candidates. *(2) Joint scoring.* Loss-weighted output changes score each 8-bit to 4-bit replacement in prefill and decode; the larger normalized score determines 8-bit priority. *(3) Compact storage.* The budget assigns 4-bit precision to the lowest-scoring tiles, storing one selected representation and scale per tile. *(4) Direct execution.* With 8-bit activations, prefill uses Tensor Core multiplication and decode uses DP4A dot products. The illustrated path expands 4-bit weights to INT8 on chip; long prefill reuses a temporary INT8 buffer. Scaled accumulation and output reconstruction produce Y.

3.1 Tile-Level Precision Assignment

We introduce offline precision allocation to select 8-to-4-bit replacements with the **smallest estimated impact under a storage budget**. The allocation leverages per-tile 4-bit and 8-bit candidates, multi-phase activation and gradient evidence, and a target 4-bit fraction.

For a linear projection $Y = XW^\top$, let $X \in \mathbb{R}^{M\times K}$ be the input activations and $W \in \mathbb{R}^{N\times K}$ the weights, where M is the number of activation rows and K, N are the input and output dimensions. Each row of W corresponds to an output feature and each column to an input feature. We partition all quantized weight matrices into equal-sized two-dimensional tiles, with model-wide set $\mathcal{T}$. Each $t \in \mathcal{T}$ is identified by its layer, projection, and matrix position, allowing **different regions within an output channel to use different bit-widths**.

Let $\rho \in [0, 1]$ denote the target fraction of 4-bit tiles. Since all tiles contain the same number of weights, each 8-to-4-bit replacement saves the same storage. Thus, the budget **determines how many tiles use 4-bit weights, while the scores determine which ones**.

Quantized candidates. As shown in Figure 2(1), candidate construction applies orthogonal input and output transformations (V along K, U along N) to redistribute outliers and minimize quantization error (Ashkboos et al., 2024; Liu et al., 2025), yielding symmetric 4-bit and 8-bit candidates per tile. Aligning groups with tile boundaries keeps each transformed weight tile ($\widetilde{W} = UWV$, $\widetilde{X} = XV$) within its assigned precision region. The paired output reconstruction preserves the floating-point projection (Appendix B.1).

Let $\widehat{W}_{4,t}$ and $\widehat{W}_{8,t}$ denote the reconstructed 4-bit and 8-bit candidates of the same transformed tile. Multiplying the values by their own scale puts both candidates in **the same numerical coordinates for comparison**. The precision assignment selects the integer values and scale retained for inference. Calibration and inference use the same 8-bit activation quantizer, so candidate comparisons reflect deployed inputs (Appendices B.2 and F.1).

Output perturbation from precision reduction. Joint scoring in panel (2) of Figure 2 begins by measuring the **local projection-output change** caused by replacing tile t's 8-bit candidate with its 4-bit candidate. This change depends on both the candidate weight difference and the activations multiplied by it. Let $p \in \{\text{pre}, \text{dec}\}$ denote the inference phase, $\mathcal{C}_p$ its calibration set, and $c \in \mathcal{C}_p$ one example. We write $\widehat{X}_t^{p,c}$ for the transformed 8-bit activations reconstructed with their scales. This slice contains the scored activation rows and the input columns associated with tile t. Holding these activations and the other weight-tile candidates fixed, the substitution produces

$$\Delta \widetilde{Y}_t^{p,c} = \widehat{X}_t^{p,c} \left(\widehat{W}_{4,t} - \widehat{W}_{8,t} \right)^\top . \tag{1}$$

The matrix product combines candidate errors across the tile's input dimensions, capturing how they **reinforce or cancel in the projection output**. We then weight this local change by its importance to the full language-model loss.

Fisher-weighted tile sensitivity. Projection-output coordinates differ in their influence on the language-model loss. We weight squared output perturbations by squared loss gradients using the diagonal empirical Fisher (Hooper et al., 2025). The gradients carry information from downstream model computation, giving higher scores to substitutions that affect loss-sensitive outputs.

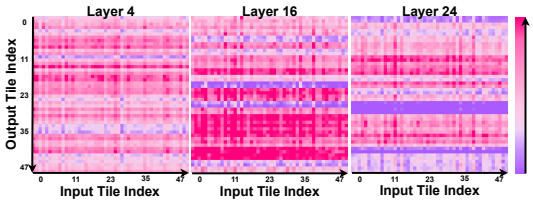

Figure 3: Tile-level sensitivity of query-projection weights in Llama-3.2-3B at three decoder depths. Purple and pink indicate lower and higher sensitivity under a shared scale, revealing pronounced within-matrix and across-depth variation.

Let $\mathcal{L}_c$ denote the full language-model loss averaged over n_c valid prediction tokens. Let $\widetilde{Y}_t$ denote the transformed projection-output slice with the same row and feature coordinates as $\Delta \widetilde{Y}_t^{p,c}$. Tiles in the same output-feature group share these output coordinates, while their input columns determine their respective perturbations. The phase-specific score is

$$F_t^p = \frac{1}{2} \sum_{c \in \mathcal{C}_p} \left\| n_c \nabla_{\widetilde{Y}_t} \mathcal{L}_c \odot \Delta \widetilde{Y}_t^{p,c} \right\|_F^2 , \tag{2}$$

where $\odot$ denotes elementwise multiplication and $\| \cdot \|_F$ is the Frobenius norm. The factor n_c restores the summed-loss gradient from the token-averaged loss. The squared norm sums the squared, gradient-weighted perturbations over the scored rows and output features, and the outer sum aggregates calibration examples within the phase.

The score combines **the local projection change** with its **downstream loss sensitivity**. Forming the perturbation before weighting captures the combined effect of changing all weights in the tile together. A larger F_t^p gives tile t higher priority for retaining 8-bit weights in phase p. Figure 3 shows local sensitivity variation within query-projection weights at three decoder depths, illustrating the value of assigning precision to individual tiles. Calibration forward and gradient-capture details are given in Appendix C.1, and phase-specific observation construction in Appendix C.2.

Joint prefill/decode allocation. Prefill and decode supply distinct activation and gradient observations for the same weights. To combine their evidence in one precision map, we first express each tile's score relative to its phase's model-wide total. We then retain the larger relative score:

$$F_t = \max_{p \in \{\text{pre},\text{dec}\}} \frac{F_t^p}{\sum_{t' \in \mathcal{T}} F_{t'}^p} . \tag{3}$$

Each ratio expresses tile t's share of one phase's total sensitivity score. Normalization makes the phase scores comparable, while the maximum retains **each tile's greater relative sensitivity across the two phases**. A tile with high relative sensitivity in either phase therefore receives higher priority for retaining 8-bit weights.

AlignQuant assigns 4-bit weights to the $\text{round}(\rho|\mathcal{T}|)$ lowest-scoring tiles and 8-bit weights to the rest. Since each 4-bit selection saves the same number of weight bytes, the **model-wide ranking determines where the fixed high-precision budget is spent**. Figure 4 shows that high-sensitivity tiles are concentrated unevenly across layers and projections. Model-wide ranking lets the allocation of 8-bit weights follow this distribution, with a common budget coordinating all quantized projections. Tie handling and the fixed-budget selection argument are given in Appendix D.4.

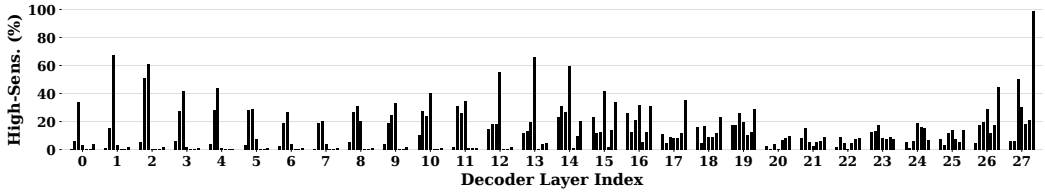

Figure 4: Distribution of high-sensitivity tiles across layers. Each group contains 7 linear projections, ordered as the query, key, value, attention output projections, feed-forward gate, up, and down projections. Bar height reports the percentage of tiles in each projection whose sensitivity scores fall within the global top 10%, using a single threshold over all tiles in the model. The strong concentration across layers and projections motivates model-wide precision allocation.

3.2 Compact Mixed-Precision Representation

In panel (3) of Figure 2, packing stores **one selected integer representation per tile** together with its FP32 reconstruction scale. Each byte stores two signed 4-bit values or one signed 8-bit value, so a 4-bit tile uses **half the integer-weight storage** of an equally sized 8-bit tile. The selected scale accompanies those values through storage and computation.

Selected 4-bit and 8-bit tiles occupy separate contiguous buffers. A one-bit precision label selects the buffer for each logical tile. Within that buffer, the row offset marks the start of its output-tile row, and the count of preceding same-precision tiles determines its position. This mapping lets the runtime **locate the selected weights and scale from the original tile coordinate**. Indexing rules, bit ordering, and kernel layouts are detailed in Appendix E.1.

With a target fraction ρ of 4-bit tiles, nominal integer-weight storage is $4\rho + 8(1-\rho) = 8 - 4\rho$ bits per quantized weight; the realized value uses the rounded tile count. Complete packed storage also includes reconstruction scales and indexing metadata. Appendices E.2 and G.7 give storage accounting and memory-measurement details.

3.3 Direct Prefill/Decode Execution

The execution stage prepares the selected weights for GPU integer computation (Figure 2, panel (4)). Both weight formats use a common INT8 path, with schedules that exploit weight reuse in prefill and parallel dot products in decode.

Integer operand preparation. The runtime first applies the input-side transform V and quantizes the activations to 8 bits using the same numerical procedure as calibration. In the direct mixed-tile kernels, 4-bit values **stay packed during GPU-memory loading and expand on chip** into signed INT8 operands. Sign extension preserves each selected integer value, and the stored scale preserves the reconstructed value of the 4-bit candidate. 8-bit weights provide signed INT8 operands directly.

Prefill weight reuse. Prefill processes multiple activation rows, so its Tensor Core INT8 schedule reuses each weight tile across their output computations. Short-prefill kernels prepare mixed-tile operands on chip. For long-prefill projections containing both formats, the runtime expands the selected weights into a temporary INT8 buffer in GPU memory once per projection call. The Tensor Core schedule **reuses the prepared weights across activation-row groups**, reducing the conversion cost per group while preserving each tile's values and scale.

Decode parallelism. Single-sequence decode has one activation row ($M = 1$), giving a matrix-vector workload. The kernel directly loads the packed weight tiles and forms INT8 operands on chip. DP4A accumulates four products of signed 8-bit operands per instruction into an INT32 result. Work is distributed **across output features and input-column groups**, allowing multiple GPU threads to contribute to each output in parallel. Their partial dot products are combined, drawing parallelism from both weight-matrix dimensions.

Output reconstruction. For both schedules, each tile product first accumulates in INT32. The corresponding activation and weight scales restore its contribution before FP32 accumulation across input tiles. This per-tile rescaling combines contributions from both weight precisions in **the same numerical coordinates**. Split-reduction schedules combine their FP32 partials in a fixed order, after which the output-side transform U restores the projection output. The phase transition reuses the same precision map and selected weights.

Table 1: End-to-end inference efficiency for **Llama-3.2-3B-Instruct**. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold and underlined values indicate the best and worst results, respectively, among the compared methods excluding the reference columns.

Metric	Unit	Upper Ref.	Lower Ref.	Ours	Others			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP
TTFT ↓	ms	17.47	22.51	**19.67**	21.99	23.07	27.48	86.99
TPOT ↓	ms/token	17.19	21.12	**7.54**	20.77	21.78	24.05	82.33
Throughput ↑	tok/s	58.18	47.33	**132.28**	48.13	45.91	41.58	12.15
Resident GPU Mem. ↓	GiB	5.99	2.08	2.72	**2.11**	2.12	2.87	2.15
Peak VRAM ↓	GiB	6.09	2.16	2.72	**2.20**	2.21	2.92	2.37
Energy ↓	J/1K tokens	2280.68	2322.34	**1267.92**	2047.43	1732.24	2172.71	22535.65

Shared activation preparation. Projections with matching inputs and transforms (e.g., Q/K/V or gate/up) share input preparation. Each group **transforms and quantizes its input once**, reusing 8-bit activations across projections while retaining independent weights, scales, and output transforms (Appendices E.3 and E.4).

Together, these designs preserve local precision flexibility while making each 4-bit choice reduce stored weight bytes. Joint calibration and phase-specific weight preparation support efficient deployment and generation with one compact model.

4 Experiments

4.1 Experimental Setup

We evaluate whether *AlignQuant* improves inference efficiency while preserving model quality, with allocation controls examining the value of local precision placement. The main text focuses on Llama-3.2-3B-Instruct; Appendix H extends the evaluation to Qwen3-4B-Instruct-2507, Ministral-3-8B-Instruct-2512-BF16, and Qwen3-14B.

Datasets and Metrics. We evaluate language modeling via WikiText-2 perplexity and reasoning/understanding via HellaSwag, MMLU, ARC-Challenge, LAMBADA, PIQA, and WinoGrande. Long-context capabilities are assessed using LV-Eval QA (16K–64K length levels) and LongEval line-retrieval exact-match accuracy.

Configuration and Comparisons. The default model uses 64×64 tiles, 60% 4-bit weights, and 8-bit activations. BF16 is the original-model reference; uniform 4-bit weights provide the low-bit reference. Structural and random controls test placement (Appendix G.1), alongside GPTQ, AWQ, MixLLM, and SFMP (Frantar et al., 2022; Lin et al., 2024b; Zheng et al., 2026; Nie et al., 2026).

Hardware and Measurement. Primary measurements use an NVIDIA A100-SXM4 40 GB GPU, with additional Llama-3.2-3B tests on H100 NVL and RTX 4090. We report time to first token (TTFT), time per output token (TPOT), throughput, memory, and energy over the full inference path (Appendix G).

4.2 Inference Efficiency

BF16 retains the original optimized model; Hugging Face FP16, shown as Torch FP16, provides the FP16 reference used in prior quantization evaluations (Lin et al., 2024b). On Llama-3.2-3B, *AlignQuant* reaches 132.28 tokens/s, $2.27\times$ **BF16 and** $2.75\times$ **GPTQ throughput**, while reducing resident memory from 5.99 to 2.72 GiB and achieving the lowest energy in Table 1. The compact selected representation therefore converts low-bit projection efficiency into complete-model generation gains.

Varying request shape shows how the benefit changes with the balance between prompt processing and generation. *AlignQuant* leads the displayed throughput comparisons in Figures 1 and 5. For long prompts, **longer continuations amplify the gain over BF16**: repeated token generation gives the packed-load decode path a larger share of the request. The same selected model therefore supports different request shapes while its execution schedules exploit their available reuse and parallelism. Additional system results and runtime ablations appear in Appendices H.4 and H.5.

Table 2: General and ablation results for **Llama-3.2-3B-Instruct**. Accuracy values are reported as percentages, while WikiText-2 is reported with word perplexity. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold values indicate the best result among the non-reference methods.

Benchmark	Metric	Upper Ref.	Lower Ref.	Ours	Others				Ablation			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP	Output-row	Red.-col	Proj./layer	Random
WikiText-2 ↓	PPL	12.49	13.51	**12.61**	13.27	12.97	12.69	12.88	12.71	12.63	12.73	12.88
HellaSwag ↑	Acc.	71.52	68.88	**70.74**	52.62	50.41	51.44	70.38	70.53	**70.74**	70.45	70.24
MMLU ↑	Acc.	60.66	58.30	**60.53**	59.91	59.24	60.50	58.53	60.05	59.81	59.79	59.89
ARC-C ↑	Acc.	52.56	51.11	**52.73**	45.73	47.44	47.35	50.77	50.68	51.88	50.68	50.34
LAMBADA ↑	Acc.	64.23	61.65	**65.66**	63.79	64.42	65.28	64.55	63.67	63.30	63.57	63.81
PIQA ↑	Acc.	77.31	76.61	**76.68**	75.08	75.73	76.01	76.22	76.61	75.79	76.44	76.66
WinoGrande ↑	Acc.	71.43	69.22	**71.22**	70.24	68.75	70.43	68.82	69.53	69.61	69.61	68.75

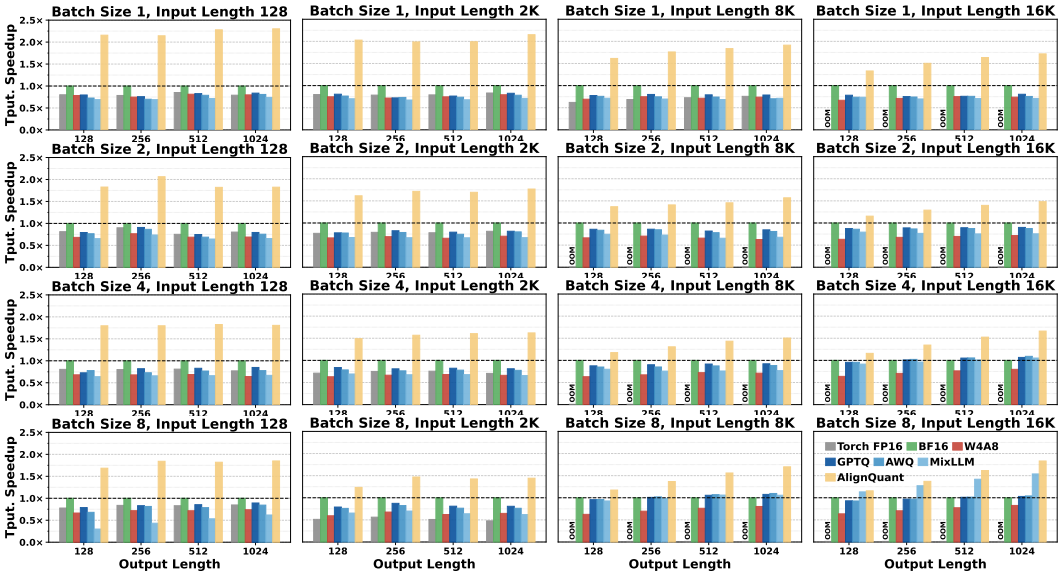

Figure 5: Generation-throughput speedup relative to BF16 on Llama-3.2-3B. Rows vary batch size over 1, 2, 4, and 8; columns vary prompt length over 128, 2K, 8K, and 16K tokens. Each panel varies output length; OOM denotes an out-of-memory run.

4.3 General Quality

Beyond generation speed, we evaluate general language tasks (Table 2). *AlignQuant* achieves **the lowest perplexity and highest or tied-highest accuracy** among quantized configurations, showing that one loss-aware allocation supports diverse tasks while retaining efficient inference.

Precision-allocation ablation. *AlignQuant* reaches 65.66% on LAMBADA and 71.22% on WinoGrande, versus 63.30%–63.81% and 68.75%–69.61% for structured and random placements. These gains support **assigning precision to local regions within output channels**; additional models appear in Appendices H.1 and H.3.

4.4 Long-Context Evaluation

Beyond general benchmarks, we evaluate long-context information retrieval and reasoning at LV-Eval's 16K, 32K, and 64K length levels. Using the dataset indices in Table 3: ① *cmrc_mixup* (Chinese reading comprehension); ② *dureader_mixup* (Chinese QA); ③ *hotpotwikiqa_mixup* (multi-hop Wikipedia evidence retrieval); ④ *lic_mixup* (long-input comprehension and localization); ⑤–⑦ *loogle_CR/MIR/SD_mixup* (comprehension/reasoning, multiple-information retrieval, and short-dependency QA); and ⑧–⑨ *multifieldqa_en/zh_mixup* (English and Chinese multi-field retrieval).

LooGLE-SD (⑦) shows how selective precision behaves as context grows (Table 3). At the 64K level, *AlignQuant* scores 48.06 alongside BF16's 48.81, versus 47.24 for MixLLM and 41.69 for AWQ. Its margin over AWQ grows from 2.35 to 6.37 points, showing that **precision retained in sensitive regions remains useful at longer contexts**.

Allocation at long contexts. Comprehension and reasoning further distinguish precision placement: at the 64K level, *AlignQuant* scores 24.81 on LooGLE-CR (⑤), versus 22.42 for reduction-column

Table 3: Long-context QA results for **Llama-3.2-3B-Instruct** on LV-Eval. Scores are reported as percentages. Cell colors are relative to BF16: green for higher scores, blue for lower scores, and gray for BF16. Bold values indicate the best result among the non-reference methods.

Dataset	Len	Upper Ref.	Lower Ref.	Ours	Others				Ablation			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP	Output-row	Red.-col	Proj./layer	Random
①	16k	21.41	17.78	**22.28**	16.63	18.29	20.05	18.21	21.61	21.52	19.05	18.43
	32k	18.61	13.55	**17.97**	13.85	16.39	16.13	15.57	15.43	15.67	14.98	13.58
	64k	14.19	11.75	**13.14**	12.02	11.99	12.40	12.88	12.66	12.87	11.93	11.94
②	16k	16.46	15.05	**17.42**	16.23	15.41	17.20	17.15	17.38	17.34	16.63	16.57
	32k	17.16	15.18	**15.99**	15.23	15.97	15.29	15.73	15.20	15.08	15.14	14.69
	64k	17.17	14.69	**17.40**	14.94	16.84	16.52	15.86	15.42	14.98	14.75	14.12
③	16k	23.76	20.90	**23.39**	20.46	22.44	23.01	22.73	18.99	22.02	21.58	21.18
	32k	29.75	19.55	**29.16**	27.97	27.59	28.35	28.19	22.88	26.87	26.44	22.76
	64k	24.79	20.01	**24.38**	23.20	23.54	23.80	23.70	22.70	23.81	23.87	22.24
④	16k	11.23	9.57	12.86	9.85	12.65	10.38	12.38	**13.21**	12.60	11.92	11.48
	32k	11.74	8.93	**11.83**	9.24	11.09	10.70	11.27	11.01	11.30	10.78	9.97
	64k	9.77	7.13	**10.00**	7.29	9.90	9.27	9.23	9.89	9.97	8.78	8.24
⑤	16k	26.11	23.05	**25.96**	25.17	25.36	24.57	25.50	23.51	25.30	23.36	23.73
	32k	25.69	22.13	**25.94**	24.19	25.09	25.19	**25.94**	23.02	24.68	22.96	23.45
	64k	24.34	20.42	**24.81**	23.24	24.77	23.00	24.68	20.57	22.42	20.31	20.97
⑥	16k	18.55	17.34	19.87	19.28	19.48	19.39	**20.29**	17.49	19.30	19.07	17.39
	32k	19.06	16.53	**18.79**	18.26	18.60	17.93	18.59	16.93	17.44	17.60	16.78
	64k	19.19	16.85	**19.40**	18.31	17.19	19.14	17.37	17.53	18.04	18.12	17.17
⑦	16k	53.80	44.96	**52.94**	45.93	50.59	52.42	49.71	51.22	52.91	51.33	49.81
	32k	49.48	32.97	**49.59**	36.73	46.50	49.45	41.38	46.42	48.23	46.76	45.31
	64k	48.81	30.48	**48.06**	40.28	41.69	47.24	43.21	45.87	47.62	46.05	40.33
⑧	16k	34.83	34.15	**35.89**	32.47	32.55	34.16	34.46	34.35	35.21	32.56	33.72
	32k	32.85	26.47	**32.96**	30.52	30.18	31.21	32.74	32.64	31.11	30.78	29.30
	64k	36.52	29.12	34.55	30.06	**34.90**	33.69	34.28	34.33	34.10	34.49	30.63
⑨	16k	19.05	13.67	**20.32**	20.16	13.08	19.91	14.35	17.77	17.47	19.65	15.66
	32k	13.80	7.63	**12.48**	12.41	11.14	12.03	11.41	11.08	11.86	11.99	10.90
	64k	11.64	8.53	**13.20**	12.08	9.98	10.41	10.44	10.08	11.67	10.24	9.86

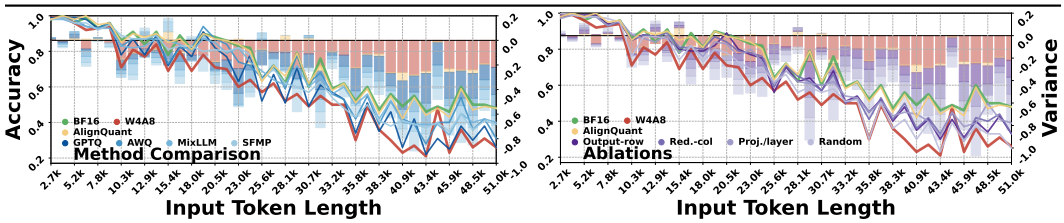

Figure 6: LongEval line-retrieval accuracy for **Llama-3.2-3B-Instruct** across input lengths from 2.7K to 51.0K tokens. Lines report exact-match accuracy and bars show differences relative to BF16. **Left:** *AlignQuant* compared with baselines. **Right:** *AlignQuant* compared with ablations.

and 20.57 for output-row allocation. This supports **local precision choices within output channels** (Appendix H.2).

Retrieval across input lengths. LongEval resolves length effects across 39 settings from 2.7K to 51.0K tokens (Figure 6). Short inputs are similar, while longer inputs separate the methods: at the longest input, *AlignQuant* matches BF16 at 48%, versus 40% for MixLLM and 31% for AWQ, preserving successful retrieval at the demanding end of the sweep.

Retrieval allocation controls. At the longest inputs, *AlignQuant* achieves 48% accuracy, outperforming structured (33%–39%) and random (25%) placements. This demonstrates the value of **preserving precision within targeted weight regions** to retain information as context grows.

Across these evaluations, *AlignQuant* combines **quality preservation and faster generation in one compact model**. Loss-aware tile scoring identifies the weight regions worth protecting, while tile-aligned storage and integer execution make those selected weights efficient to use. The common tile structure connects these numerical and computational choices throughout deployment.

5 Conclusion

AlignQuant demonstrates that fine-grained precision choices can coexist with regular GPU computation when the precision unit supports both sensitivity evaluation and efficient execution. Our results show this preserves quality while boosting inference efficiency, suggesting a broader principle for future quantizers: design precision units for compact storage and regular execution, then allocate bits based on local output sensitivity.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript polishing and code development. The authors reviewed all AI-assisted content and take responsibility for the final work.

REPRODUCIBILITY STATEMENT

Appendices B through E detail the numerical construction, calibration procedure, allocation derivations, and GPU implementation. Appendix G specifies the experimental configurations, evaluation protocols, and storage and runtime measurements.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

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

Hao Mark Chen, Fuwen Tan, Alexandros Kouris, Royson Lee, Hongxiang Fan, and Stylianos Venieris. Progressive mixed-precision decoding for efficient llm inference. In *International Conference on Learning Representations*, volume 2025, pp. 37767–37783, 2025.

Zihan Chen, Bike Xie, Jundong Li, and Cong Shen. Channel-wise mixed-precision quantization for large language models. *arXiv preprint arXiv:2410.13056*, 2024.

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

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

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

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. *arXiv preprint arXiv:2505.05799*, 2025.

Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 1074–1084, 2019.

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.

Simla Burcu Harma, Danila Mishin, Zhengyuan Su, Ayan Chakraborty, Elizaveta Kostenok, Dongho Ha, Babak Falsafi, Martin Jaggi, Yunho Oh, and Amir Yazdanbakhsh. Mxsens: Sensitivity-aware mixed-precision quantization for efficient llm inference. *arXiv preprint arXiv:2607.17733*, 2026.

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.

Coleman Hooper, Charbel Sakr, Ben Keller, Rangharajan Venkatesan, Kurt Keutzer, Sophia Shao, and Brucek Khailany. Fgmp: fine-grained mixed-precision weight and activation quantization for hardware-accelerated llm inference. *arXiv preprint arXiv:2504.14152*, 2025.

Huanqi Hu, Bowen Xiao, Shixuan Sun, Jianian Yin, Zhexi Zhang, Xiang Luo, Chengquan Jiang, Weiqi Xu, Xiaoying Jia, Xin Liu, et al. Liquidgemm: Hardware-efficient w4a8 gemm kernel for high-performance llm serving. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1619–1630, 2025a.

Xing Hu, Yuan Cheng, Dawei Yang, Zukang Xu, Zhihang Yuan, Jiangyong Yu, Chen Xu, Zhe Jiang, and Sifan Zhou. Ostquant: Refining large language model quantization with orthogonal and scaling transformations for better distribution fitting. *arXiv preprint arXiv:2501.13987*, 2025b.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. Slim-llm: Salience-driven mixed-precision quantization for large language models. *arXiv preprint arXiv:2405.14917*, 2024.

Wonsuk Jang and Thierry Tambe. Blockdialect: Block-wise fine-grained mixed format quantization for energy-efficient llm inference. *arXiv preprint arXiv:2501.01144*, 2025.

Deokjae Lee and Hyun Oh Song. Q-palette: Fractional-bit quantizers toward optimal bit allocation for efficient llm deployment. *Advances in Neural Information Processing Systems*, 38:11525–11558, 2026.

Sangjun Lee, Seung-taek Woo, Jun-gyu Jin, Changhun Lee, and Eunhyeok Park. Amq: Enabling automl for mixed-precision weight-only quantization of large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35520–35538, 2025.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can context length of open-source llms truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

Xinlin Li, Timothy Chou, Josh Fromm, Zichang Liu, Yunjie Pan, and Christina Fragouli. Scalebits: Scalable bitwidth search for hardware-aligned mixed-precision llms. *arXiv preprint arXiv:2602.17698*, 2026.

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. Duquant: Distributing outliers via dual transformation makes stronger quantized llms. *Advances in Neural Information Processing Systems*, 37:87766–87800, 2024a.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024b.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Wenyuan Liu, Haoqian Meng, Yilun Luo, Peng Zhang, and Xindian Ma. Micromix: Efficient mixed-precision quantization with microscaling formats for large language models. In *International Conference on Learning Representations*, volume 2026, pp. 112668–112687, 2026.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, volume 2025, pp. 92009–92032, 2025.

Yuexiao Ma, Huixia Li, Xiawu Zheng, Feng Ling, Xuefeng Xiao, Rui Wang, Shilei Wen, Fei Chao, and Rongrong Ji. Affinequant: Affine transformation quantization for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 50932–50951, 2024.

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

Xin Nie, Haicheng Zhang, Liang Dong, Beining Feng, Jinhong Weng, and Guiling Sun. Sfmp: Fine-grained, hardware-friendly and search-free mixed-precision quantization for large language models. *arXiv preprint arXiv:2602.01027*, 2026.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Gunho Park, Jeongin Bae, Beomseok Kwon, Byeongwook Kim, Se Jung Kwon, and Dongsoo Lee. Anybcq: Hardware efficient flexible binary-coded quantization for multi-precision llms. In *International Conference on Learning Representations*, volume 2026, pp. 125106–125122, 2026.

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

Utkarsh Saxena, Sayeh Sharify, Kaushik Roy, and Xin Wang. Resq: Mixed-precision quantization of large language models with low-rank residuals. *arXiv preprint arXiv:2412.14363*, 2024.

Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, et al. Flatquant: Flatness matters for llm quantization. *arXiv preprint arXiv:2410.09426*, 2024.

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

Xilong Xie, Liang Wang, Limin Xiao, Meng Han, Lin Sun, Shuai Zheng, and Xiangrong Xu. Fineq: Software-hardware co-design for low-bit fine-grained mixed-precision quantization of llms. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7. IEEE, 2025.

Ke Yi, Zengke Liu, jianwei zhang, Chengyuan Li, Tong Zhang, Junyang Lin, and Jingren Zhou. Rotated runtime smooth: Training-free activation smoother for accurate int4 inference. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 17168–17184, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/2bda52aca6d214904eceffbce50f2e8c-Paper-Conference.pdf.

Tao Yuan, Xuefei Ning, Dong Zhou, Zhijie Yang, Shiyao Li, Minghui Zhuang, Zheyue Tan, Zhuyu Yao, Dahua Lin, Boxun Li, et al. Lv-eval: A balanced long-context benchmark with 5 length levels up to 256k. *arXiv preprint arXiv:2402.05136*, 2024.

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.

Junchen Zhao, Ali Derakhshan, Dushyant Bharadwaj, Jayden Kana Hyman, Junhao Dong, Sangeetha Abdu Jyothi, and Ian Harris. Impq: Interaction-aware layerwise mixed precision quantization for llms. *arXiv e-prints*, pp. arXiv–2509, 2025.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209, 2024.

Zhen Zheng, Xiaonan Song, and Chuanjie Liu. Mixllm: Llm quantization with global mixed-precision between output-features and highly-efficient system design. *Proceedings of Machine Learning and Systems*, 8:1771–1785, 2026.

Table 4: Notation for the numerical construction, allocation, and execution of quantized weight tiles.

Category	Symbol	Description
Projection	$X \in \mathbb{R}^{M\times K}$	Input activations
	$W \in \mathbb{R}^{N\times K}$	Projection weights
	$Y = XW^\top$	Projection output
	M, K, N	Activation rows, input features, and output features
Coordinates	U, V	Orthogonal output- and input-feature transforms
	$\widetilde{\cdot}, \widehat{\cdot}$	Transformed coordinates and quantized reconstructions
Tiling	b_N, b_K	Weight-tile dimensions
	$R = N/b_N,\ C = K/b_K$	Tile-row and tile-column counts per projection
	$\mathcal{T},\ t$	Model-wide tile set and one tile identifier
	$\rho,\ z_t$	Target W4 fraction and selected precision $z_t \in \{4, 8\}$
Calibration	$p \in \{\mathrm{pre}, \mathrm{dec}\}$	Inference phase
	$\mathcal{C}_p,\ c$	Phase-specific observations and one observation
	$\mathcal{L}_c,\ n_c$	Mean prediction loss and its valid token count
Scoring	$\widehat{W}_{4,t}, \widehat{W}_{8,t}$	Reconstructed weight candidates
	$\widehat{X}_t^{p,c}$	Reconstructed A8 input slice associated with tile t
	$\widetilde{Y}_t, \Delta\widetilde{Y}_t^{p,c}$	Output slice and its candidate-induced change
	$F_t^p,\ F_t$	Phase-specific score and joint allocation score
Storage	$Q_{b,t}, s_{b,t};\ Q_t, s_t$	Candidate and selected integer values and scales
	$\beta_{r,j}$	Precision label: 1 for W4 and 0 for W8
	T_4, T_8	Selected W4 and W8 tile counts per projection

A Notation and Reading Guide

Table 4 collects the notation used throughout the method. Appendix B defines the quantized candidates, Appendix C describes calibration, and Appendix D derives the allocation rule. Storage and execution details appear in Appendix E, followed by the design discussion in Appendix F.

Tiles in one output-tile row share output features but span different input columns. Their scores pair the shared output-gradient slice with each tile's input-dependent perturbation.

B Numerical Construction of Quantized Tiles

B.1 Tile-Aligned Transformations

Block-diagonal transforms $U = \mathrm{diag}(U_0, \ldots, U_{R-1})$ and $V = \mathrm{diag}(V_0, \ldots, V_{C-1})$ redistribute values **within each weight tile** (Ashkboos et al., 2024; Liu et al., 2025). The transformed tile is $\widetilde{W}_{r,j} = U_r W_{r,j} V_j$. With $\widetilde{X} = XV$ and $\widetilde{W} = UWV$, output reconstruction preserves the floating-point projection:

$$\widetilde{Y}U = (XV)(UWV)^\top U = XVV^\top W^\top U^\top U = XW^\top. \tag{4}$$

For the default 64×64 configuration, stored seeds define $V_j = \Phi_{v_j}$ and $U_r = \Phi_{u_r}^\top$, with $\Phi_{-1} = I$, $\Phi_0 = H_{64}$, and $\Phi_s = D_{L,s} P_s H_{64} D_{R,s}$ for $s = 1, \ldots, 6$. Here, P_s is a permutation, $D_{L,s}$ and $D_{R,s}$ are diagonal sign matrices, and H_{64} is the normalized Hadamard transform with final butterfly scaling $1/8$.

B.2 Weight Candidates

For each precision $b \in \{4, 8\}$, let $q_{\min,b} = -2^{b-1}$ and $q_{\max,b} = 2^{b-1} - 1$. Each tile uses zero point zero and nearest-integer rounding with ties to even:

$$Q_b(A; s) = \mathrm{clip}_{[q_{\min,b}, q_{\max,b}]}(\mathrm{round}_{\mathrm{even}}(A/s)) \,. \tag{5}$$

The initial scale covers both tile endpoints:

$$s_{0,b}(A) = \max\left\{\frac{\max(0, \max A)}{q_{\max,b}}, \frac{\max(0, -\min A)}{|q_{\min,b}|}, 10^{-12}\right\}. \tag{6}$$

We fit **a separate scale at each precision** by testing $s_i = \max\{(1/4 + i/128)s_{0,b}(A), 10^{-12}\}$ for $i = 0, \ldots, 128$ and selecting

$$s_b^\star(A) = \arg\min_{s_i} \|A - s_i Q_b(A; s_i)\|_F^2. \tag{7}$$

Search and integer construction use FP64, with grid order resolving equal errors. The candidate contains $Q_{b,t} = Q_b(\widetilde{W}_t; s_b^\star)$ and $s_{b,t} = \mathrm{FP32}(s_b^\star)$. Scoring compares their reconstructions $\widehat{W}_{b,t} = s_{b,t}Q_{b,t}$, and packing retains the selected pair $(Q_t, s_t) = (Q_{z_t,t}, s_{z_t,t})$.

B.3 Activation Quantization and Padding

After applying V, activation quantization uses one scale per group of 64 features and b_M rows, where $b_M = 1$ for $M = 1$ and $b_M = 16$ otherwise. Each group A uses $s_A^\star = s_{0,8}(A)$ from equation 6 and $Q_A = Q_8(A; s_A^\star)$. CUDA preparation computes in FP64 and stores signed INT8 values with FP32 scales; calibration follows the same grouping and endpoint rule with FP64 reconstruction. Zero padding completes $M' = \lceil M/b_M \rceil b_M$ rows, and the output is sliced back to M rows. The prediction mask selects the tokens contributing to calibration loss.

B.4 Transform Selection

For each compatible input-sharing group $\mathcal{G}$, transform selection uses collected inputs $X_\mathcal{G}$, a shared V, and projection-specific U_ℓ. Let $\mathcal{R}_4$ denote 4-bit quantization and reconstruction with FP64 fitted scales, using 16×64 activation groups and 64×64 weight tiles. The search minimizes

$$E_\mathcal{G} = \sum_{\ell \in \mathcal{G}} \left\| \mathcal{R}_4(X_\mathcal{G} V)\mathcal{R}_4(U_\ell W_\ell V)^\top U_\ell - X_\mathcal{G} W_\ell^\top \right\|_F^2. \tag{8}$$

Search over the eight seeds initializes a uniform input transform and then a uniform output transform for each projection. Two coordinate sweeps refine input groups followed by output groups, with seed order resolving ties. The resulting transforms feed W4 and W8 candidate scoring under A8 inputs.

C Calibration Procedure

C.1 Forward Values and Gradient Observations

Calibration uses reconstructed 8-bit weights, the selected transforms, and 8-bit activations while keeping model parameters fixed. Outputs return to original coordinates and the model's activation type. For a captured output gradient G, the frozen-weight straight-through rule computes $G_X = GW_{\text{original}}$ using the pretrained weights. Since $Y = \widetilde{Y}U$, scoring pairs the reconstructed input slices with gradients in the same transformed output coordinates:

$$\widetilde{G} = GU^\top. \tag{9}$$

C.2 Prefill and Decode Observations

Each record contains a prompt and continuation, with loss $\mathcal{L}_c = n_c^{-1} \sum_{\tau \in \mathcal{I}_c} \ell_{c,\tau}$ over $n_c = |\mathcal{I}_c|$ prediction targets. Prefill scores **all continuation targets** in one causal pass, beginning with the target predicted from the final prompt position.

Decode builds the prompt's key/value cache and feeds continuation tokens sequentially to predict the next token. Up to four deterministically spaced steps are scored, each with a **one-token loss** ($n_c = 1$) and one-row projection observations. The accumulated cache supplies fixed context for each scored step.

C.3 Computing the Precision Map

For each input-tile column, the scorer evaluates equations 1 and 2 using FP64 inputs, reconstructed candidates, and accumulation. This produces one tile-score matrix per projection and phase. Equation 3 combines the scores for model-wide ranking. The allocator selects the target number of 4-bit tiles and packs each tile's selected integer values and scale, following the ordering rules in Appendix D.4.

D Derivations and Properties of the Allocation Score

D.1 Local Output Perturbation

A tile replaces several weights at once, and their errors combine through the input. For output group r, the reconstructed projection is $\widehat{\widetilde{Y}}_{:,r} = \sum_j \widehat{X}_{:,j}\widehat{W}_{r,j}^\top$. Holding the inputs and other tiles fixed, the W8-to-W4 substitution at $t = (r, j)$ gives

$$\Delta\widetilde{Y}_t^{p,c} = \widehat{X}_t^{p,c}\widehat{W}_{4,t}^\top - \widehat{X}_t^{p,c}\widehat{W}_{8,t}^\top = \widehat{X}_t^{p,c}(\widehat{W}_{4,t} - \widehat{W}_{8,t})^\top. \tag{10}$$

For one output coordinate, write the input as x and the candidate weight difference as δw. Then

$$(x^\top \delta w)^2 = \sum_i x_i^2 \delta w_i^2 + 2\sum_{i<i'} x_i x_{i'} \delta w_i \delta w_{i'}. \tag{11}$$

The cross terms capture reinforcement and cancellation between errors contributing to that output.

D.2 Loss-Weighted Quadratic Form

The factor n_c converts the gradient of the mean token loss to the gradient of its sum:

$$\nabla_{\widetilde{Y}_t} \sum_{\tau \in \mathcal{I}_c} \ell_{c,\tau} = n_c \nabla_{\widetilde{Y}_t} \mathcal{L}_c. \tag{12}$$

For each observation, let $d_t^{p,c} = \mathrm{vec}(\Delta\widetilde{Y}_t^{p,c})$ and $g_t^{p,c} = \mathrm{vec}(n_c \nabla_{\widetilde{Y}_t}\mathcal{L}_c)$. Equivalently, equation 2 becomes

$$F_t^p = \frac{1}{2}\sum_{c \in \mathcal{C}_p} (d_t^{p,c})^\top \,\mathrm{diag}(g_t^{p,c} \odot g_t^{p,c}) d_t^{p,c}. \tag{13}$$

The diagonal weights act on projection-output coordinates, after the weight errors within each tile have combined through the input. This gives a loss-weighted score for the whole tile substitution. Scores use the captured calibration state and the gradient rule in Appendix C.1; downstream quality is measured on the assembled model with all selected substitutions applied.

D.3 Phase Normalization

Assume $F_t^p \geq 0$ and a positive total score in each phase. The normalized scores $\overline{F}_t^p = F_t^p / \sum_{t'} F_{t'}^p$ sum to one. For any phase-wide multiplier $\alpha_p > 0$,

$$\frac{\alpha_p F_t^p}{\sum_{t'} \alpha_p F_{t'}^p} = \overline{F}_t^p. \tag{14}$$

Normalization therefore retains relative tile importance when the overall score scale changes.

The joint score $F_t = \max\{\overline{F}_t^{\mathrm{pre}}, \overline{F}_t^{\mathrm{dec}}\}$ retains each tile's stronger relative importance. For any set $\mathcal{S}$ of tiles assigned lower precision, $\max_p \sum_{t\in\mathcal{S}} \overline{F}_t^p \leq \sum_{t\in\mathcal{S}} F_t$. Fixed-budget ranking therefore minimizes **a common upper bound on both phases' total normalized scores**. The budget then determines which positions in this joint ranking receive W8.

D.4 Fixed-Budget Selection

Let $k = \mathrm{round}(\rho|\mathcal{T}|)$ and keep the candidate scores fixed. Equal-area tiles give every W8-to-W4 substitution the same integer-storage saving, so allocation selects

$$\arg\min_{\mathcal{S}\subseteq\mathcal{T},\ |\mathcal{S}|=k} \sum_{t\in\mathcal{S}} F_t. \tag{15}$$

Order the scores as $F_{t_1} \le \cdots \le F_{t_{|\mathcal{T}|}}$. For any feasible set with sorted ranks $j_1 < \cdots < j_k$, each $j_i \ge i$, giving

$$\sum_{i=1}^{k} F_{t_{j_i}} \ge \sum_{i=1}^{k} F_{t_i}.$$

The first k tiles attain this minimum. The endpoints $k = 0$ and $k = |\mathcal{T}|$ select the empty set and the full tile set, respectively.

The implementation resolves equal scores by lexicographic order of the module-and-coordinate identifier. The tile count uses nearest-integer rounding with ties to even.

E Packed Representation and GPU Execution

E.1 Locating and Unpacking a Tile

The kernel starts from a logical tile coordinate (r, j). Its precision label $\beta_{r,j} = \mathbf{1}[z_{r,j} = 4]$ selects the W4 or W8 buffer. Labels follow row-major tile order, with eight labels per byte and the first label in the least significant bit.

Each buffer stores its selected tiles in row order. For precision $b \in \{4, 8\}$, the physical tile index is

$$\mathrm{index}_b(r, j) = \underbrace{\sum_{r'<r} \sum_{j'=0}^{C-1} \mathbf{1}[z_{r',j'} = b]}_{\text{stored row offset}} + \underbrace{\sum_{j'<j} \mathbf{1}[z_{r,j'} = b]}_{\text{position within the row}}. \tag{16}$$

The weight scale uses the same index. For example, a row with precisions W4, W8, W4 stores its first and third tiles consecutively in the W4 buffer.

Within each tile, values are viewed in input-by-output order and permuted into the CUTLASS `ColumnMajorTensorOpMultiplicandCrosswise<8,32>` layout used by the integer kernels. Both precisions use this element order. W8 stores one signed value per byte, while W4 packs consecutive values q_0, q_1 as

$$\mathrm{byte}(q_0, q_1) = (q_0 \bmod 16) + 16(q_1 \bmod 16). \tag{17}$$

A nibble v expands to v for $v < 8$ and to $v - 16$ otherwise. This sign extension preserves the selected integer value and its reconstruction with the stored scale.

E.2 Stored Bytes

Let $T_4 + T_8 = RC$ and $B = b_N b_K$. The integer weights occupy $BT_4/2 + BT_8$ bytes. The packed projection additionally contains one FP32 scale per tile, the precision labels, and two INT32 row-offset arrays of length $R + 1$:

$$S_{\mathrm{packed}} = \frac{B}{2} T_4 + BT_8 + 4(T_4 + T_8) + \left\lceil \frac{RC}{8} \right\rceil + 8(R + 1). \tag{18}$$

For 64×64 tiles, the W4 and W8 buffers have shapes $[T_4, 64, 32]$ and $[T_8, 64, 64]$, respectively.

With realized W4 fraction $\rho_{\mathrm{real}} = T_4/(T_4 + T_8)$, the integer-weight cost is $8 - 4\rho_{\mathrm{real}}$ bits per weight. Complete packed bits per weight are $8S_{\mathrm{packed}}/[B(T_4 + T_8)]$. Transform seeds and matrix geometry accompany the serialized tensors.

Runtime allocations also include prepared activations, outputs, and execution workspace. For a mixed projection using long-prefill preparation, the temporary INT8 weights, scales, and row offsets occupy $BRC + 4RC + 4(R + 1)$ bytes for that call.

Table 5: Computation assigned to one thread block before optional splitting of the input dimension.

Schedule	Output region	Threads	Arithmetic
Prefill M16	16×64	128	Tensor Core INT8 MMA
Prefill M64	64×64	256	Tensor Core INT8 MMA
Single-sequence decode	1×64	256	DP4A dot products

E.3 Operand Preparation and Scheduling

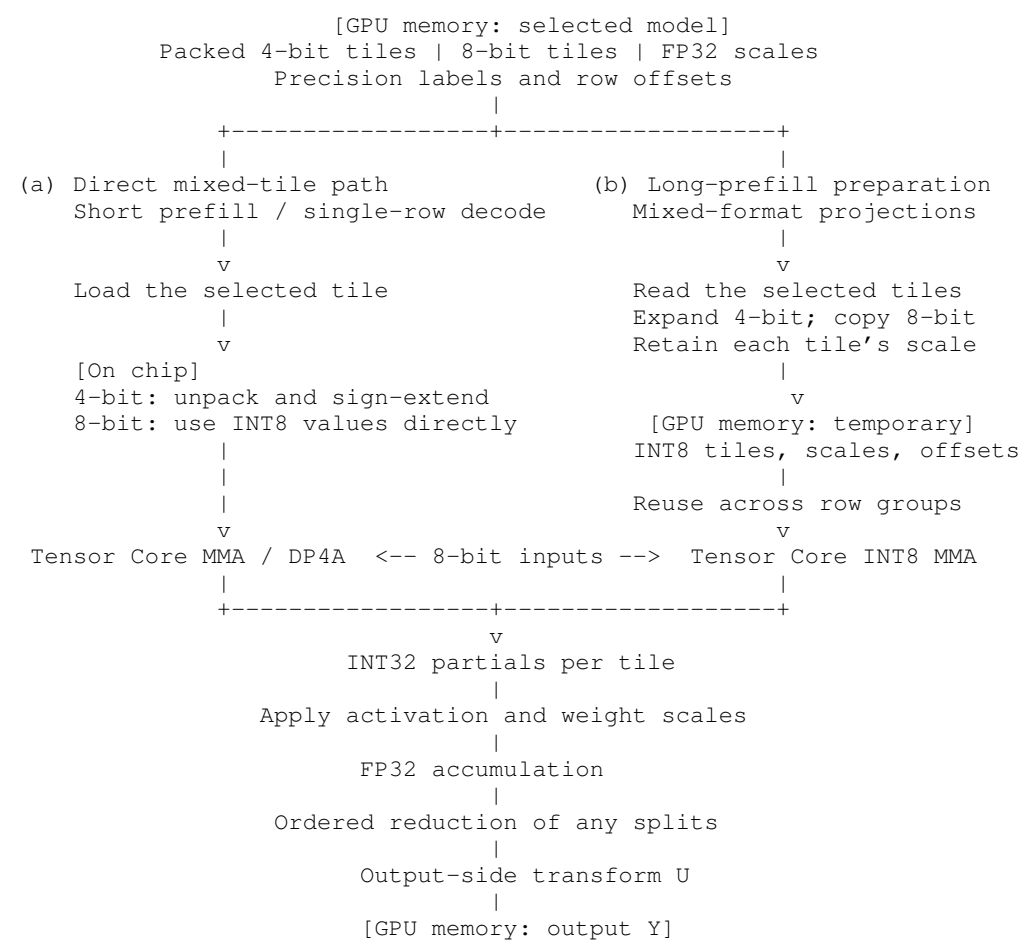

Figure 7: **Weight preparation and data movement.** Both paths start from the same selected mixed-precision model. Direct kernels expand packed 4-bit weights on chip, while long prefill prepares a temporary INT8 buffer once per projection call and reuses it across activation-row groups. Both paths retain the selected integer values and scales and receive separately prepared 8-bit activations. Integer partials are rescaled before FP32 accumulation and output reconstruction.

Figure 7 shows the two weight-preparation paths, both supplied with activations transformed by V and quantized to 8 bits. Table 5 lists the prefill and decode thread-block geometries.

Prefill. Prefill selects M16 below 128 padded rows and M64 otherwise, using $16 \times 8 \times 32$ INT8 MMA instructions. The M64 path receives activation tiles of shape $[M'/16, C, 16, 64]$, with one scale per 16-row group. Two shared-memory stages overlap tile loading and multiplication.

Direct kernels prepare selected weights on chip. For mixed-format projections with $M' \geq 1024$, a temporary INT8 buffer is prepared once per call, retaining the selected values and scales in logical tile order with its own row offsets. Reuse across activation-row groups amortizes this preparation over the prompt.

Decode. For $M = 1$, the kernel directly loads packed tiles, expands 4-bit values on chip, and distributes work across output features and input-column groups. DP4A accumulates four signed 8-bit products per instruction into INT32. In split execution, four adjacent lanes cooperate on one output feature, each handling 16 input coordinates per tile.

Accumulation and reconstruction. For activation-row group m and output-tile row r, integer tile products are rescaled before FP32 accumulation:

$$\widehat{\widehat{Y}}_{m,r} = \sum_{j=0}^{C-1} s^X_{m,j} s^W_{r,j} \left(Q^X_{m,j} (Q^W_{r,j})^\top \right). \tag{19}$$

Each parenthesized product accumulates in INT32. For S splits, split s covers input-tile columns $\lfloor Cs/S \rfloor$ through $\lfloor C(s+1)/S \rfloor - 1$. FP32 partials are summed in increasing split order before U_r restores output coordinates. The interface returns the original row count and model activation type, adding bias when present.

E.4 Shared Projection Inputs

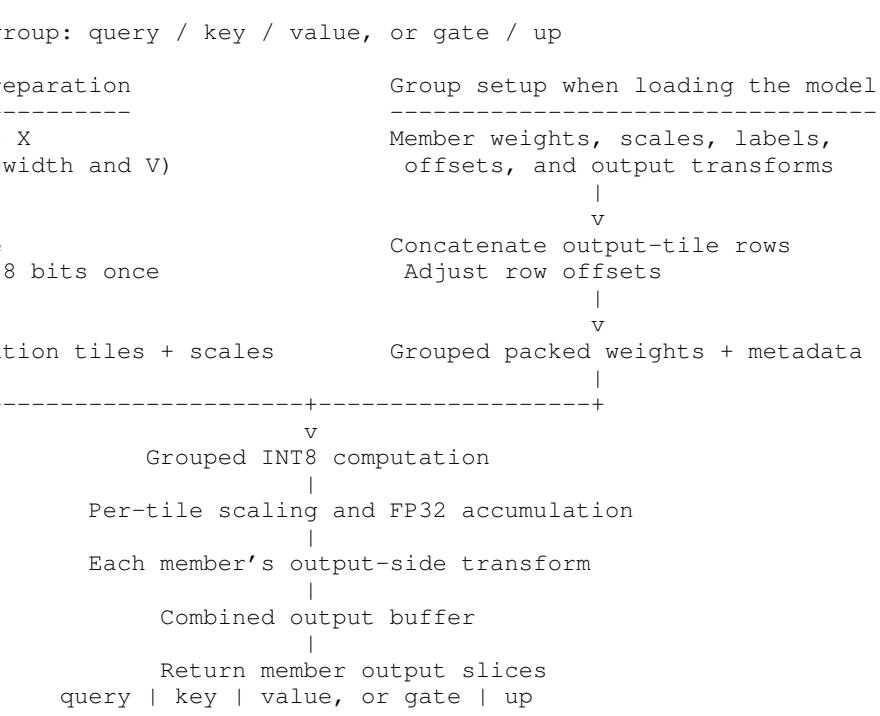

Figure 8: **Shared input preparation across compatible projections.** A group contains query, key, and value projections, or gate and up projections, sharing the same input, input width, and transform V. Their selected weights are assembled along the output dimension during model setup. Each input is transformed and quantized once for grouped execution, which preserves each member's weight scales and output transform. The resulting combined output supplies the individual projection slices.

Compatible query, key, and value projections, or gate and up projections, share the same input, input width, and input-side transform V. During model setup, their packed weight rows and output-transform metadata are concatenated, with row offsets adjusted to locate each member's tiles. As shown in Figure 8, each input is transformed and quantized once, and grouped execution computes a combined output. Each projection retains its selected weights, scales, and output transform and receives its corresponding output slice. The combined result is retained until all group members have consumed their slices.

F Design Rationale

F.1 Activation Precision

One A8 input group contributes to several output tiles, each of which can independently select W4 or W8. Using the same reconstructed input makes the candidate comparison specific to the weight substitution and lets compatible projections share activation preparation. A8 also supplies the operand width used by both Tensor Core INT8 multiplication and DP4A (Xiao et al., 2023; Lin et al., 2025).

For a fixed represented range, A8 provides 256 integer levels and A4 provides 16. An A4 variant would recalibrate the candidate scores using its reconstructed inputs and prepare those values for the chosen arithmetic. Mixed activation precision would add format choices for input groups and coordinate them across their consuming projections. Uniform A8 keeps the input representation shared while the weight allocation follows local sensitivity.

F.2 Precision Granularity and Execution

The weight tile determines which values share a precision choice, while activation groups and thread blocks determine how those values are used. A 64×64 weight tile can therefore support a single decode row or multiple prefill rows. Smaller weight tiles provide more allocation choices and increase scale and label storage, whose contribution is approximately $33/(b_N b_K)$ bits per weight before row offsets and byte alignment.

Joint calibration assigns the shared weights using evidence from both phases. At runtime, direct loading forms INT8 operands on chip, while long prefill reuses prepared operands across many rows. These schedules preserve the selected values and scales while adapting their preparation to the available reuse.

G Experimental Protocols

G.1 Models and Comparison Configurations

Evaluated models are listed in Section 4.1. General quality and LV-Eval compare GPTQ and AWQ on all four models, adding MixLLM and SFMP on Llama-3.2-3B and the two Qwen models. The reported LongEval results use Llama-3.2-3B with the comparison set in Figure 6. Cross-model A100 measurements include all four external methods on every model.

GPTQ and AWQ use 4-bit weights with group size 128. MixLLM uses `kernels_mixllm::gemm`; the evaluated SFMP implementation unpacks and dequantizes INT4 weights for floating-point multiplication. Reported SFMP system measurements characterize this evaluated dequantization-based backend. The four allocation controls share the 60% 4-bit tile target. Output-row shares precision across reduction-direction tiles within an output-feature group, while reduction-column shares precision across output groups within a reduction-tile column. At the default tile size, these groups span 64 output features for Output-row and 64 input features for Red.-col. Projection/layer-constrained allocation restricts the budget within projection/layer partitions, while random allocation selects the same number of 4-bit tiles using a sensitivity-independent random ranking. The endpoints use uniform 4-bit or 8-bit weights with 8-bit activations.

Exact checkpoint revisions are pinned in the released model presets. *AlignQuant* quantizes the decoder query, key, value, and attention-output projections together with the feed-forward gate, up, and down projections; other modules retain their original representation.

G.2 Calibration Settings

Calibration uses 64 ordered document-summary pairs from Multi-News training rows 544–607 (Fabbri et al., 2019). The first 24 records provide prefill observations. Prompt text is truncated to 4096 tokens, and summary text to 128 tokens before the end-of-sequence token is appended. Decode uses 24 deterministic matched-prefix windows constructed by the released calibration script, each with 32 continuation tokens, and scores up to four deterministic decode steps per window. The default

record order and observation positions are selected deterministically. Candidate construction and scoring follow Appendices B and C, and the default model-wide allocation assigns 4-bit weights to the lowest-scoring 60% of tiles.

G.3 General Quality Evaluation

WikiText-2 reports word-level perplexity (Merity et al., 2016). General Avg is the equal-weight mean accuracy on HellaSwag, MMLU, ARC-Challenge, LAMBADA, PIQA, and WinoGrande (Zellers et al., 2019; Hendrycks et al., 2020; Clark et al., 2018; Paperno et al., 2016; Bisk et al., 2020; Sakaguchi et al., 2021). For each model, all methods use the same evaluator, tokenizer, task configuration, prompt format, evaluation examples, and reported metric field.

G.4 Long-Context Evaluation

LongEval (Li et al., 2023) measures exact-match retrieval at line counts $200, 300, \ldots, 4000$, with token-length axes determined by each model's tokenizer. LV-Eval (Yuan et al., 2024) uses the nine datasets identified in Section 4.4 at its 16K, 32K, and 64K length levels, retaining the official task scores. Qwen3-14B uses the adopted YaRN configuration at 32K and 64K.

G.5 System Evaluation

Hardware settings follow Section 4.1, with A100 runs conducted at Polaris. The evaluated *AlignQuant* runtime uses adaptive decode scheduling, shared activation preparation, and CUDA Graph execution. All compared methods use matched CUDA Graph settings, attention backend, KV-cache configuration, and measurement protocol; method-specific quantization kernels follow their evaluated implementations. The cross-model workload uses batch size 1, a 128-token prompt, and 512 generated tokens. The reference Llama-3.2-3B A100 measurement uses two warm-up runs and five timed repetitions, reporting median latency, throughput, and energy. Measurements include input and weight preparation, projection computation, scale restoration, reduction, and output conversion. Headline generation speedups are BF16-to-*AlignQuant* TPOT ratios for matching models, GPUs, and workloads. Candidate construction, calibration, and packing precede request timing; online operand preparation, including the long-prefill INT8 buffer, is included.

The 80-setting sweep combines prompt lengths $\{128, 2048, 8192, 16384\}$ tokens, output lengths $\{128, 256, 512, 1024\}$ tokens, and batch sizes $\{1, 2, 4, 8, 16\}$. It compares *AlignQuant*, BF16, Hugging Face FP16, uniform 4-bit weights with 8-bit activations, GPTQ, AWQ, and MixLLM; Figure 5 shows batches 1, 2, 4, and 8. The separate peak-memory sweep uses prompt lengths of 128, 512, 2K, 4K, 8K, and 16K tokens (Table 21).

G.6 Ablation Protocols

Sensitivity-localization statistics cover all four models. Tile-size and calibration-data studies use Llama-3.2-3B, Qwen3-4B, and Ministral-3-8B. Tile sizes 32×32, 64×64, and 128×128 are evaluated with held-out prefill and decode NLL. Calibration uses 8, 16, 24, and 48 documents with seeds 0, 1, and 2; Table 17 reports the mean, sample standard deviation, minimum, and maximum of General Avg across seeds. Main-table results use the deterministic record construction in Appendix G.2. Means and standard deviations in this sweep summarize the sampled runs within each calibration-set size.

On Llama-3.2-3B, the budget sweep uses $\rho \in \{0, 0.4, 0.5, 0.6, 0.7, 0.8, 1\}$ with the final runtime (Table 16). Runtime controls use fixed decode splitting, separate activation preparation, or individual kernel launches under the same A100 generation workload (Table 22).

G.7 Storage and Memory Accounting

Packed BPW sums the bytes in equation 18 over quantized projections, multiplies by eight, and divides by their total weight count. Full-model storage adds the remaining parameters and serialization metadata. Resident and peak allocated GPU memory are reported in GiB, with activations, key/value states, and temporary weights contributing according to their runtime lifetimes.

Table 6: General benchmark results for **Qwen3-4B-Instruct-2507**. Accuracy values are reported as percentages, while WikiText-2 is reported with word perplexity. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold values indicate the best result among the compared quantized methods, excluding the reference columns.

Benchmark	Metric	Upper Ref.	Lower Ref.	Ours	Others			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP
WikiText-2 ↓	PPL	13.07	14.44	**13.15**	13.72	13.71	13.19	13.96
HellaSwag ↑	Acc.	69.14	66.87	**68.85**	51.53	51.71	68.81	67.91
MMLU ↑	Acc.	74.72	72.74	**74.22**	73.09	73.03	72.10	71.00
ARC-Challenge ↑	Acc.	66.38	64.08	**66.64**	61.35	61.09	66.54	64.76
LAMBADA ↑	Acc.	63.40	58.14	**63.38**	59.95	62.51	63.21	61.25
PIQA ↑	Acc.	76.06	76.66	**76.67**	75.19	75.08	76.33	76.06
WinoGrande ↑	Acc.	66.38	66.54	**66.77**	66.22	65.27	65.98	65.90

Table 7: Ablation results for **Qwen3-4B-Instruct-2507**. Accuracy values are reported as percentages, while WikiText-2 is reported with word perplexity. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold values indicate the best result among the non-reference methods.

Benchmark	Metric	Upper Ref.	Lower Ref.	Ours	Ablation			
		BF16	W4A8	*AlignQuant*	Output-row	Red.-col	Proj./layer	Random
WikiText-2 ↓	PPL	13.07	14.44	**13.15**	13.40	13.32	13.35	13.88
HellaSwag ↑	Acc.	69.14	66.87	**68.85**	68.15	68.54	67.91	67.34
MMLU ↑	Acc.	74.72	72.74	74.22	74.14	**74.30**	74.19	73.41
ARC-Challenge ↑	Acc.	66.38	64.08	**66.64**	65.70	66.30	65.96	65.44
LAMBADA ↑	Acc.	63.40	58.14	63.38	62.88	**63.42**	63.13	59.93
PIQA ↑	Acc.	76.06	76.66	**76.67**	76.17	76.33	76.17	76.01
WinoGrande ↑	Acc.	66.38	66.54	**66.77**	66.14	66.69	66.22	65.27

H Additional Results and Analysis

H.1 General Quality Results

To examine whether the quality preservation extends across model families and sizes, we repeat language modeling and downstream evaluation on Qwen3-4B, Ministral-3-8B, and Qwen3-14B. Tables 6, 10, and 8 show a recurring pattern: selective W8 retention improves the reported task scores over W4A8 while preserving the predictive behavior of the original model. For Qwen3-4B, LAMBADA accuracy reaches 63.38%, alongside 63.40% for BF16 and 58.14% for W4A8. For Qwen3-14B, HellaSwag accuracy is 78.77% alongside 78.83% for BF16, and WikiText-2 perplexity is 10.94 alongside 10.78. The preservation therefore appears in both correct-answer prediction and the probabilities assigned to text.

This behavior is consistent with the numerical object evaluated during allocation. As detailed in Appendices D.1 and D.2, each score measures the output change caused by replacing a reconstructed W8 candidate with its W4 candidate under the observed A8 inputs. The loss-gradient weighting connects that local change to the model's prediction objective. Each model supplies its own candidate differences, activations, and gradients, so the same procedure can assign W8 to different regions under a common storage budget. The downstream results support these selected regions as useful locations for retaining additional numerical resolution.

Allocation controls. The matched-budget controls examine how much of this quality depends on the spatial freedom of the allocation. On Qwen3-4B, *AlignQuant* achieves 13.15 perplexity, compared with 13.32 to 13.88 for the structured and random placements (Table 7). On Qwen3-14B, LAMBADA accuracy reaches 69.65%, compared with 67.55% to 67.79% for the structured controls and 66.45% for random placement (Table 9). These comparisons hold the W4 fraction fixed, so the improved predictions reflect a more effective placement of the available W8 tiles. The model-wide selection in Appendix D.4 permits this capacity to follow the tile scores across projections. Together, these results support retaining local choices in model-wide allocation.

Table 8: General benchmark results for **Qwen3-14B**. Accuracy values are reported as percentages, while WikiText-2 is reported with word perplexity. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold values indicate the best result among the compared quantized methods, excluding the reference columns.

Benchmark	Metric	Upper Ref.	Lower Ref.	Ours	Others			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP
WikiText-2 ↓	PPL	10.78	11.49	**10.94**	11.14	11.24	10.96	11.39
HellaSwag ↑	Acc.	78.83	77.92	**78.77**	59.97	60.18	78.70	78.25
MMLU ↑	Acc.	80.72	79.62	**80.15**	79.74	79.92	78.58	77.98
ARC-Challenge ↑	Acc.	69.97	68.94	**69.65**	67.32	65.70	69.62	69.11
LAMBADA ↑	Acc.	67.86	65.86	**69.65**	67.01	67.22	69.12	68.19
PIQA ↑	Acc.	79.65	78.56	**79.87**	79.33	79.82	79.33	79.82
WinoGrande ↑	Acc.	75.43	74.59	**75.66**	75.14	73.95	74.27	72.69

Table 9: Ablation results for **Qwen3-14B**. Accuracy values are reported as percentages, while WikiText-2 is reported with word perplexity. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold values indicate the best result among the non-reference methods.

Benchmark	Metric	Upper Ref.	Lower Ref.	Ours	Ablation			
		BF16	W4A8	*AlignQuant*	Output-row	Red.-col	Proj./layer	Random
WikiText-2 ↓	PPL	10.78	11.49	10.94	10.97	**10.88**	11.02	11.22
HellaSwag ↑	Acc.	78.83	77.92	**78.77**	78.65	78.56	78.61	78.45
MMLU ↑	Acc.	80.72	79.62	80.15	**80.30**	80.24	80.03	80.00
ARC-Challenge ↑	Acc.	69.97	68.94	**69.65**	69.03	69.20	69.54	68.34
LAMBADA ↑	Acc.	67.86	65.86	**69.65**	67.79	67.55	67.79	66.45
PIQA ↑	Acc.	79.65	78.56	79.87	79.60	79.87	**80.36**	79.16
WinoGrande ↑	Acc.	75.43	74.59	**75.66**	75.06	74.51	74.59	74.35

H.2 LV-Eval Task Results

The additional LV-Eval experiments examine whether the shared precision maps preserve information use as context length increases across different models. Qwen3-4B maintains a nearly level CMRC trajectory, scoring 24.95, 25.30, and 24.74 at 16K, 32K, and 64K, with the final value matching BF16 (Table 11). Ministral-3-8B preserves a different length response on LooGLE-SD: its score changes from 59.44 at 16K to 56.18 at 64K, alongside the BF16 change from 58.47 to 55.09 (Table 12). Preservation here follows each model's task-specific trajectory across lengths. The selected representation retains the first example's nearly level response and the second's gradual change.

Qwen3-14B shows a stronger dependence on selective precision at the largest context length (Table 13). On LooGLE-SD, *AlignQuant* scores 21.51, 18.96, and 17.86 from 16K to 64K, while W4A8 scores 18.54, 15.58, and 8.55. A related pattern appears in English multi-field question answering (⑧), where the 64K score is 14.28 with *AlignQuant*, 14.66 with BF16, and 5.89 with W4A8. The separation across these tasks shows that the value of higher precision extends to several forms of long-input information use. The retained W8 regions help preserve the model's ability to extract and use relevant evidence at lengths where the precision choice has a larger effect on the answer.

Allocation controls. The fixed-budget comparisons further distinguish the effect of precision placement. For Qwen3-14B on LooGLE-SD, the strongest structured control changes from 21.03 at 16K to 10.11 at 64K, compared with 21.51 and 17.86 for *AlignQuant*. For Ministral-3-8B on LooGLE-CR at 64K, *AlignQuant* scores 26.77 versus 21.45 to 23.37 for the structured and random allocations. The available W8 capacity is the same, yet the locally selected map preserves more contextual information in both cases.

The shared-map design connects these results to the calibration procedure in Appendices C.2 and D.3. Prefill and decode contribute their own activation and gradient observations, and their normalized scores determine one selected representation per tile. That representation is reused across the evaluated context lengths. The task trajectories show that the resulting choices continue to protect useful predictions as the amount of input context changes.

Table 10: General benchmark and ablation results for **Ministral-3-8B-Instruct-2512-BF16**. Accuracy values are reported as percentages, while WikiText-2 is reported with word perplexity. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold values indicate the best result among the non-reference methods.

Benchmark	Metric	Upper Ref.	Lower Ref.	Ours	Others		Ablation			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	Output-row	Red.-col	Proj./layer	Random
WikiText-2 ↓	PPL	9.73	10.24	9.87	9.90	9.93	9.90	**9.86**	9.90	10.01
HellaSwag ↑	Acc.	78.79	78.09	**78.74**	58.17	58.13	78.40	78.62	78.59	78.42
MMLU ↑	Acc.	77.85	76.46	77.22	77.05	76.88	77.18	**77.26**	76.98	76.77
ARC-Challenge ↑	Acc.	69.03	68.00	**68.77**	64.25	63.57	68.17	68.34	68.34	68.52
LAMBADA ↑	Acc.	70.99	69.67	71.76	70.83	71.03	70.75	71.73	**71.86**	71.45
PIQA ↑	Acc.	80.52	79.27	**79.92**	78.51	79.05	79.22	79.87	**79.92**	79.71
WinoGrande ↑	Acc.	77.98	75.06	76.87	76.72	76.09	**77.03**	76.87	76.48	76.40

Table 11: Long-context question answering results for **Qwen3-4B-Instruct-2507** on LV-Eval across nine datasets and three context lengths. Scores are reported as percentages. Cell colors are relative to BF16: green for higher scores, blue for lower scores, and gray for BF16. Bold values indicate the best result among the non-reference methods.

Dataset	Len	Upper Ref.	Lower Ref.	Ours	Others				Ablation			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP	Output-row	Red.-col	Proj./layer	Random
①	16k	24.57	23.04	24.95	23.86	24.94	24.20	23.89	**25.57**	24.89	23.49	23.37
	32k	25.68	21.27	25.30	23.90	**26.56**	25.01	26.04	23.79	22.89	21.48	21.38
	64k	24.74	20.39	**24.74**	21.12	23.99	23.75	23.18	22.57	21.66	20.03	20.96
②	16k	19.20	18.75	**19.90**	16.96	18.89	18.92	19.35	19.67	19.56	19.40	19.70
	32k	19.77	18.01	**19.26**	18.41	17.63	19.01	18.68	18.94	19.06	18.76	18.40
	64k	19.83	19.51	**20.65**	18.92	19.15	19.86	19.15	18.62	19.40	19.50	19.19
③	16k	4.78	2.72	**5.01**	4.76	4.80	4.85	4.43	4.95	4.94	4.80	3.89
	32k	4.46	2.65	4.77	4.66	**4.87**	4.80	4.53	4.75	4.63	4.45	3.48
	64k	4.67	2.62	**4.76**	3.76	4.64	4.51	4.12	4.37	4.49	4.37	4.06
④	16k	22.31	21.31	**22.57**	20.93	20.98	22.54	21.03	21.32	22.02	21.15	20.64
	32k	22.91	18.51	**22.20**	20.99	21.61	21.16	21.67	18.46	20.04	19.01	18.62
	64k	21.65	17.97	**22.63**	22.05	22.49	22.26	22.10	17.41	19.63	18.47	17.69
⑤	16k	14.77	12.19	**14.65**	13.92	13.90	14.36	14.06	14.54	14.55	14.30	13.38
	32k	14.42	11.16	**14.30**	13.12	**14.30**	14.04	14.11	13.88	13.95	14.01	13.34
	64k	14.24	10.01	**13.75**	12.60	13.71	13.04	12.84	12.95	13.07	13.23	12.36
⑥	16k	10.26	8.10	10.91	10.16	10.66	10.79	10.27	**11.00**	10.82	10.49	9.02
	32k	9.88	7.33	**9.89**	9.19	9.58	9.15	9.47	9.63	9.21	9.66	7.60
	64k	9.07	6.30	**9.60**	8.97	8.64	9.38	8.70	8.57	8.21	8.67	6.72
⑦	16k	24.43	21.08	23.96	22.75	23.54	23.74	22.65	23.57	23.60	**24.06**	22.46
	32k	22.95	18.90	22.15	21.54	21.90	**22.20**	22.14	21.31	21.37	22.04	20.47
	64k	21.96	17.88	**20.46**	19.99	20.05	20.30	19.46	19.04	19.11	19.75	18.26
⑧	16k	18.89	16.46	**19.78**	18.84	19.15	19.38	18.39	19.42	19.38	19.60	17.46
	32k	18.32	16.35	**19.51**	19.43	18.66	18.60	18.72	19.40	19.18	18.86	17.12
	64k	18.61	15.14	17.93	18.00	17.30	17.89	16.83	17.92	**18.02**	17.82	15.80
⑨	16k	17.70	13.11	**17.28**	15.66	15.69	**17.28**	16.54	16.32	17.02	16.25	13.94
	32k	16.25	12.33	**16.77**	15.01	15.59	16.76	14.83	15.01	16.25	15.19	13.21
	64k	15.20	12.58	**15.12**	13.75	14.84	14.29	14.74	14.85	14.88	14.33	12.70

H.3 Allocation and Numerical Ablations

H.3.1 Allocation Structure and Sensitivity Localization

We examine sensitivity variation and precision maps to explain the quality gains from the allocation controls defined in Appendix G.1. Table 14 reports mixed W4/W8 assignments in 59.81% of the output-tile rows of Llama-3.2-3B and 87.37% to 90.17% of those in the other models. Each such row contains input regions contributing to the same output-feature group but receiving different precisions. The selected maps therefore use the within-channel allocation freedom throughout substantial portions of the models. The sensitivity CV values also show that the extent of local variation differs across models, providing different distributions for the common allocation procedure.

The perturbation construction explains how these regions receive distinct scores. Tiles in one output-tile row share output-gradient coordinates, while their input slices and W4/W8 candidate differences produce different local output changes in equation 10. The allocator can therefore distinguish their effects even though they contribute to the same output features. Figure 3 shows this local variation, and Figure 4 shows how high-sensitivity regions are distributed across projections. The precision

Table 12: Long-context question answering results for **Ministral-3-8B-Instruct-2512-BF16** on LV-Eval across nine datasets and three context lengths. Scores are reported as percentages. Cell colors are relative to BF16: green for higher scores, blue for lower scores, and gray for BF16. Bold values indicate the best result among the non-reference methods.

Dataset	Len	Upper Ref.	Lower Ref.	Ours	Others		Ablation			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	Output-row	Red.-col	Proj./layer	Random
①	16k	39.93	39.98	**43.79**	42.75	41.71	40.24	42.30	40.41	41.51
	32k	40.88	37.41	**41.10**	40.45	39.48	39.46	40.44	39.78	38.81
	64k	39.94	35.49	**41.00**	40.10	40.31	39.54	40.21	39.70	38.76
②	16k	14.16	14.94	**15.82**	14.38	14.22	14.62	14.84	14.55	14.91
	32k	15.80	14.77	**16.51**	15.44	15.73	15.58	14.71	15.37	15.46
	64k	17.23	14.92	**17.35**	16.91	15.74	16.49	16.51	17.16	15.68
③	16k	31.86	30.46	**33.00**	32.54	31.64	31.42	32.77	31.42	30.86
	32k	32.58	28.81	32.24	31.39	31.64	**32.71**	31.57	31.19	31.02
	64k	25.24	24.10	**25.48**	24.77	24.49	25.14	24.32	24.13	23.69
④	16k	20.97	21.42	**22.18**	21.88	20.24	20.38	21.33	21.95	21.61
	32k	21.37	19.44	**21.08**	19.79	20.15	19.36	19.94	20.34	18.96
	64k	20.72	17.99	**20.69**	19.76	19.55	18.13	18.74	19.21	17.82
⑤	16k	27.38	24.21	**27.64**	25.41	26.78	25.51	26.91	26.91	24.34
	32k	26.19	21.41	**27.47**	27.24	26.73	23.54	22.41	22.56	22.47
	64k	26.72	23.32	26.77	25.21	**27.42**	22.14	21.45	21.66	23.37
⑥	16k	21.04	19.63	**20.99**	20.43	20.35	20.15	20.05	19.53	20.42
	32k	19.64	18.55	19.81	19.36	19.61	**20.06**	19.59	19.33	18.41
	64k	19.24	17.07	**18.95**	18.27	18.66	18.43	18.17	17.74	16.73
⑦	16k	58.47	56.57	**59.44**	58.60	56.78	59.18	56.01	59.37	57.50
	32k	57.72	54.13	**58.76**	57.91	57.13	56.88	53.29	56.41	55.37
	64k	55.09	51.39	56.18	55.79	**57.74**	54.47	52.13	55.46	54.47
⑧	16k	33.81	32.61	**34.40**	32.80	34.10	33.68	33.84	32.35	33.65
	32k	31.98	31.46	**32.65**	31.46	30.70	32.02	32.17	31.02	31.41
	64k	31.26	29.62	**33.00**	32.99	32.01	29.97	30.75	28.96	28.92
⑨	16k	41.92	42.80	45.63	45.18	42.75	44.78	44.05	**46.91**	44.50
	32k	42.31	42.48	**43.61**	41.92	41.47	43.08	42.60	42.63	41.29
	64k	43.98	41.30	42.95	42.79	42.68	42.87	42.73	**44.13**	40.29

maps connect those observations to the actual model representation: each selected region retains its tile coordinate and one stored format through the addressing scheme in Appendix E.1.

H.3.2 Tile-Size Ablation

We vary tile size to test the resolution needed for precision allocation. Using the unrounded measurements, 64×64 gives the lowest prefill and decode NLL in all six model-phase comparisons (Table 15). The finer 32×32 partition uses four times as many scales and precision labels, while 128×128 increases NLL in the tested settings. The selected geometry therefore preserves useful local resolution with substantially less metadata than the finer partition.

H.3.3 Selected-Weight Storage and W4 Budget

After choosing the tile geometry, we vary the W4 fraction to examine how the precision budget changes actual storage and runtime behavior. All configurations in Table 16 use the final *AlignQuant* runtime. Table 1 reports the method-comparison reference, while the uniform endpoints here are configurations of the *AlignQuant* runtime. The default row links the quality result in Table 2 (12.61 perplexity) to its measured storage and throughput. Packed storage follows the integer-weight relation $8 - 4\rho$ closely, with reconstruction scales and indexing metadata included in the reported values. Across the mixed configurations from 40% to 80% W4, packed storage decreases from 6.41 to 4.81 bits per weight and resident GPU memory decreases from 2.98 to 2.45 GiB. This progression agrees with the selected-representation accounting in equation 18: each additional W4 tile replaces its stored W8 values with a payload of half the size.

Execution changes much less across these mixed configurations. Throughput remains between 130.51 and 134.83 tokens/s, with the default 60% configuration reaching 132.28 tokens/s at 5.61 packed bits per weight. The representation therefore exposes a useful storage control while retaining

Table 13: Long-context question answering results for **Qwen3-14B** on LV-Eval across nine datasets and three context lengths. Scores are reported as percentages. Cell colors are relative to BF16: green for higher scores, blue for lower scores, and gray for BF16. Bold values indicate the best result among the non-reference methods.

Dataset	Len	Upper Ref.	Lower Ref.	Ours	Others				Ablation			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP	Output-row	Red.-col	Proj./layer	Random
①	16k	21.12	19.45	**21.11**	20.11	20.86	20.34	21.06	20.99	19.67	19.95	19.46
	32k	24.02	20.42	23.57	21.21	**24.00**	22.66	20.71	20.33	19.43	20.25	19.30
	64k	21.60	13.45	**20.96**	12.86	13.78	18.33	19.18	14.53	12.86	13.12	13.11
②	16k	19.04	16.34	**19.73**	16.90	17.02	18.13	18.58	17.27	17.46	18.50	17.97
	32k	21.10	16.28	19.72	**19.84**	18.69	17.34	19.62	18.61	18.91	18.33	17.46
	64k	17.68	14.87	**17.32**	16.54	16.41	15.44	15.84	16.02	16.49	16.77	16.53
③	16k	5.64	3.22	**5.35**	4.26	3.69	4.67	4.19	3.88	4.14	4.93	4.60
	32k	5.33	2.97	**4.76**	3.67	3.02	4.74	3.59	3.46	3.75	4.33	4.01
	64k	6.18	2.13	**5.04**	1.06	1.90	4.66	4.12	2.34	2.79	3.50	3.23
④	16k	22.47	20.91	**21.12**	18.84	20.92	17.75	20.32	20.03	18.96	20.34	20.90
	32k	19.66	17.84	**19.74**	19.43	18.12	18.51	18.47	16.88	14.94	17.31	17.89
	64k	18.33	12.22	17.20	12.81	14.89	15.19	**18.79**	12.09	11.14	12.36	12.98
⑤	16k	13.88	11.97	**13.38**	13.10	12.24	12.75	13.10	12.67	12.75	12.90	12.44
	32k	13.56	11.99	**12.34**	12.26	11.11	11.64	12.00	12.03	12.10	12.21	12.19
	64k	13.95	9.84	**12.41**	8.16	8.27	10.28	10.18	10.12	10.66	10.94	10.54
⑥	16k	11.36	8.63	**10.13**	9.49	9.13	9.49	9.18	9.23	9.01	8.76	8.64
	32k	10.05	7.01	**8.96**	7.96	7.42	8.82	7.71	8.23	8.14	7.88	7.78
	64k	8.93	4.12	**8.35**	4.44	4.88	7.82	7.99	5.26	5.01	4.90	4.63
⑦	16k	21.41	18.54	**21.51**	21.50	**21.51**	20.76	16.53	20.21	20.48	21.03	20.70
	32k	21.43	15.58	**18.96**	18.35	18.24	17.15	14.98	16.51	16.22	16.97	16.74
	64k	20.32	8.55	**17.86**	8.44	8.19	17.17	14.24	9.23	9.45	10.11	10.06
⑧	16k	17.49	14.84	15.91	15.83	14.94	14.66	14.97	13.97	14.79	**16.93**	14.84
	32k	15.96	11.21	**13.04**	11.76	11.89	11.74	11.87	10.33	11.43	12.61	11.15
	64k	14.66	5.89	**14.28**	6.33	7.15	12.56	12.28	5.11	6.23	7.19	6.47
⑨	16k	16.47	13.90	**16.52**	14.62	13.76	12.14	14.97	15.59	14.58	16.31	14.55
	32k	16.86	13.84	**15.84**	13.24	14.65	12.76	13.44	14.32	15.49	15.10	14.94
	64k	13.85	10.41	**14.04**	10.11	12.03	11.65	12.20	11.44	12.01	13.46	12.19

Table 14: Quantitative characterization of sensitivity localization and the resulting precision maps. Sensitivity CV measures within-unit variation in sensitivity scores, state entropy summarizes W4/W8 assignment diversity, and mixed-row fraction reports the percentage of output-tile rows containing both precisions. Nonzero variance is the share of units with varying scores.

Model	Sensitivity CV	State Entropy (bits)	Mixed-row (%)	Nonzero Variance (%)	W4 Tiles
Llama-3.2-3B	0.47	0.36	59.81	100.00	412,880
Qwen3-4B	2.10	0.52	87.37	100.00	532,288
Ministral-3-8B	1.42	0.44	90.17	100.00	1,086,240
Qwen3-14B	1.35	0.51	88.79	100.00	1,935,344

a similar generation rate over the tested mixed budgets. In the runtime described in Appendix E.3, both formats supply INT8 operands, and each tile retains the scale of its selected candidate. Changing the W4 fraction changes which numerical values are stored and how many bytes they occupy, while the arithmetic organization remains shared.

This distinction explains how the allocation and execution components fit together. The budget and sensitivity ranking determine the selected numerical model. Packing realizes that choice in bytes, and operand preparation presents those same selected values to the integer kernels. The sweep shows how one execution design supports several compact storage levels.

H.3.4 Calibration-Data Sensitivity

We vary calibration-set size and sampling seed to assess how the data used for allocation affects downstream quality. Table 17 covers 8, 16, 24, and 48 documents, with three seeds at each size. Within this sweep, the reported task-quality scores remain in narrow ranges for each model. The sample standard deviation across seeds ranges from 0.03 to 0.11 percentage points, and similar quality is retained across a sixfold range of document counts. Thus, the selected maps continue to support comparable downstream performance under the tested changes in calibration data.

Table 15: Tile-size ablation across three models. Prefill and decode quality are reported as held-out negative log-likelihood (NLL). Bold and underlined values indicate the best and worst results, respectively, according to the unrounded measurements for each model.

Tile Size	Llama-3.2-3B		Qwen3-4B		Ministral-3-8B	
	Prefill ↓	Decode ↓	Prefill ↓	Decode ↓	Prefill ↓	Decode ↓
32×32	2.24	2.36	2.38	2.50	2.02	2.13
$\mathbf{64 \times 64}$	**2.24**	**2.36**	**2.37**	**2.49**	**2.01**	**2.12**
128×128	2.26	2.39	2.38	2.49	2.02	2.14

Table 16: Effect of the W4 allocation budget on storage and end-to-end inference for **Llama-3.2-3B-Instruct** on an NVIDIA A100 with a 128-token prompt and 512 generated tokens. All configurations use the final *AlignQuant* runtime. The 60% W4 setting is the default operating point used in the main experiments. Bold and underlined values indicate the best and worst results, respectively, across the evaluated W4 fractions.

W4 Fraction	Packed BPW ↓	TTFT ↓ (ms)	TPOT ↓ (ms/token)	Throughput ↑ (tok/s)	Resident ↓ (GiB)	Peak ↓ (GiB)	Energy ↓ (J/1K tokens)
0%	8.01	**18.39**	**7.03**	**141.72**	3.48	3.48	1177.30
40%	6.41	19.33	7.39	134.83	2.98	2.98	1288.47
50%	6.01	19.36	7.48	133.26	2.84	2.84	1255.14
60%	5.61	19.67	7.54	132.28	2.72	2.72	1267.92
70%	5.21	19.80	7.58	131.47	2.58	2.58	1266.19
80%	4.81	19.84	7.64	130.51	2.45	2.45	1263.40
100%	**4.01**	19.60	7.17	138.96	**2.17**	**2.17**	**1145.29**

These measurements evaluate the sampling dependence of the procedure described in Appendix C. Each sampled record contributes prompt-processing and token-generation observations, which provide inputs and loss gradients for the same candidate comparisons. Phase normalization expresses their accumulated scores as relative tile importance, and the common 4-bit target fixes the number of selected substitutions. The experiment consequently varies the evidence guiding placement while preserving the representation and storage objective. The resulting quality stability supports obtaining useful local precision decisions from compact calibration collections.

H.4 Inference Efficiency

H.4.1 Workload Scaling

The throughput results in Figure 5 show that the execution advantage extends across prompt lengths, output lengths, and batch sizes. Figures 9 and 10 examine the same workload grid from the complementary perspectives of per-token latency and energy efficiency.

Per-token generation latency. Figure 9 reports TPOT speedup relative to BF16. *AlignQuant* provides the largest speedup among the displayed configurations, exceeding $2\times$ for several batch-one workloads. The advantage also extends to batched long-input generation, including the 16K-input settings at batch sizes 4 and 8. These results support adapting integer execution to the available parallelism while reusing the same selected weight representation.

Energy efficiency. Figure 10 shows lower energy per generated token than BF16 across the displayed workloads. For 8K and 16K prompts, the energy-efficiency gain generally increases with continuation length. Longer continuations spread prompt processing and weight preparation over more generated tokens, while generation reuses the compact selected weights. The results connect the shared representation and phase-specific execution to energy savings across request shapes.

Together with Figure 5, these measurements show that the same selected W4/W8 model improves three related aspects of generation: aggregate throughput, per-token latency, and energy efficiency. The magnitude of each gain changes with workload composition, while the advantage persists across the tested request shapes.

Table 17: Sensitivity of downstream quality to calibration-set size and sampling seed across three models. Each calibration size uses three seeds. General Avg is reported as the mean across seeds; SD is the sample standard deviation, and Min/Max give the observed range. Bold and underlined General Avg values indicate the highest and lowest mean, respectively, for each model.

Set Size	Llama-3.2-3B				Qwen3-4B				Ministral-3-8B			
	General Avg ↑	SD	Min	Max	General Avg ↑	SD	Min	Max	General Avg ↑	SD	Min	Max
8	**65.27**	0.04	65.22	65.30	**69.27**	0.04	69.24	69.31	75.38	0.06	75.33	75.44
16	64.97	0.11	64.87	65.08	69.02	0.10	68.94	69.13	**75.54**	0.06	75.47	75.57
24	65.02	0.03	65.00	65.06	69.10	0.06	69.05	69.16	75.47	0.07	75.43	75.55
48	64.98	0.04	64.95	65.02	69.12	0.07	69.06	69.19	75.37	0.03	75.34	75.40

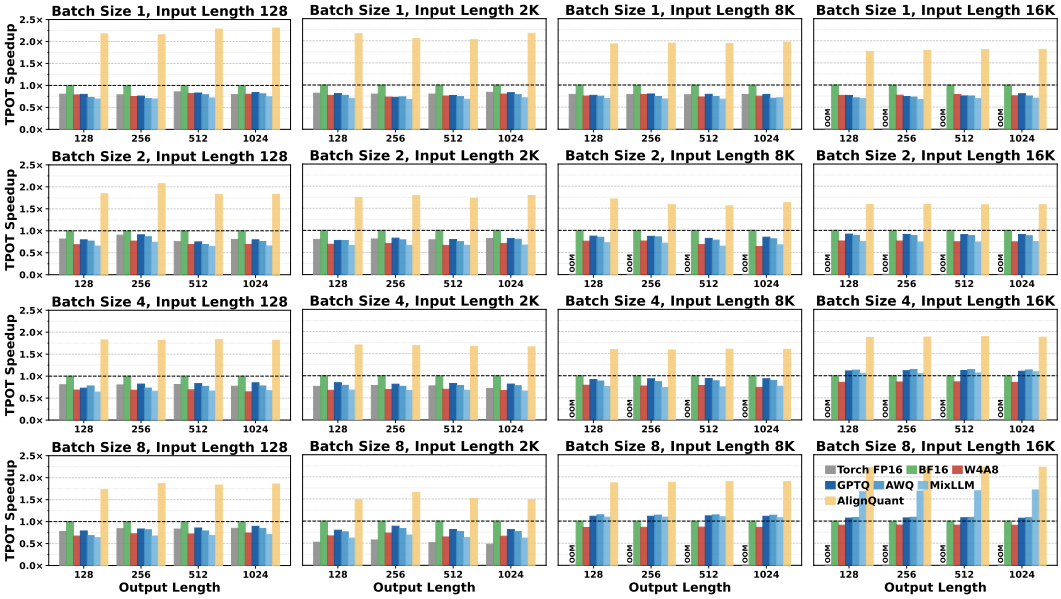

Figure 9: TPOT speedup relative to BF16 on Llama-3.2-3B. Rows vary batch size over 1, 2, 4, and 8; columns vary prompt length over 128, 2K, 8K, and 16K tokens. Each panel varies output length; higher values indicate faster generation.

H.4.2 Cross-Hardware and Cross-Model Results

The cross-model system experiment tests whether the quality-preserving allocations also deliver efficient generation as model dimensions and parameter counts change. All four models use one A100 GPU with batch size 1, a 128-token prompt, and 512 generated tokens. Table 20 shows lower TPOT and higher throughput for *AlignQuant* on every model, with TPOT speedups of $1.53\times$ to $2.50\times$ over BF16. On Qwen3-14B, TPOT decreases from 30.29 to 19.85 ms, and energy per 1,000 generated tokens decreases from 6972.05 to 4354.57 J. The largest model therefore combines the task-quality preservation reported earlier with lower time and energy costs for generation.

The common tile representation provides the connection across these model sizes. Each projection stores its own selected values and scales, while the execution organization in Appendix E.3 distributes work over output tiles and input-column groups. This lets the allocation follow each model's sensitivity distribution while reusing the same forms of integer computation. The cross-model measurements show that this combination remains effective across the evaluated projection geometries. Quality and efficiency are thus two properties of the same selected models.

To examine whether the execution design carries across GPUs, we repeat the Llama-3.2-3B workload on H100 NVL and RTX 4090. *AlignQuant* achieves the highest throughput and lowest TPOT among the compared configurations on both devices (Tables 19 and 18). Throughput reaches 167.49 tokens/s on H100 NVL and 180.64 tokens/s on RTX 4090, compared with their respective BF16 results of 81.54 and 108.05 tokens/s. Both measurements also report lower energy per generated token and a resident footprint of approximately 2.7 GiB.

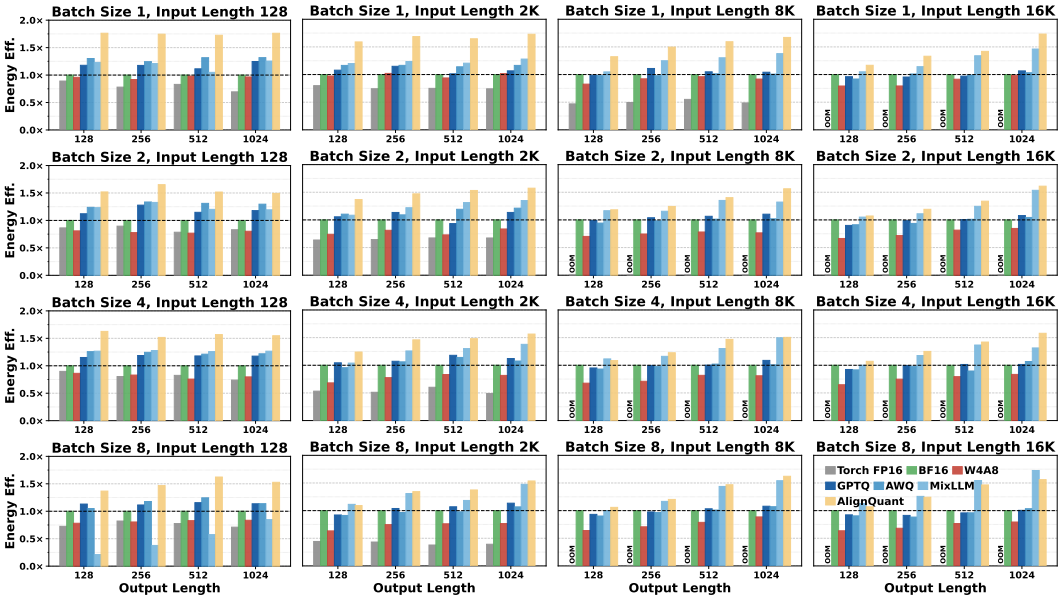

Figure 10: Generation energy efficiency relative to BF16 on Llama-3.2-3B. Rows vary batch size over 1, 2, 4, and 8; columns vary prompt length over 128, 2K, 8K, and 16K tokens. Each panel varies output length; higher values indicate greater energy efficiency.

Table 18: End-to-end inference efficiency for **Llama-3.2-3B-Instruct** on an **NVIDIA RTX 4090**. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold and underlined values indicate the best and worst results, respectively, among the compared methods excluding the reference columns.

Metric	Unit	Upper Ref.	Lower Ref.	Ours	Others			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP
TTFT ↓	ms	11.79	17.13	16.54	11.71	10.97	**8.35**	57.55
TPOT ↓	ms/token	9.25	7.60	**5.51**	8.13	8.68	7.32	53.12
Throughput ↑	tok/s	108.05	131.31	**180.64**	122.88	115.16	136.54	18.82
Resident GPU Mem. ↓	GiB	5.99	2.08	2.71	**2.11**	2.12	2.87	2.16
Peak VRAM ↓	GiB	6.09	2.16	2.71	**2.20**	2.21	2.92	2.37
Energy ↓	J/1K tokens	2537.20	1536.05	**1430.25**	1520.65	1549.37	1690.31	21103.45

The representation and operand preparation have complementary roles in these results. The selected W4 values provide compact storage, and their expansion preserves the integer values and reconstruction scales used by the common INT8 arithmetic. The same numerical representation can therefore be used by the phase-specific kernels on each evaluated platform. The measurements support carrying the local precision choices into efficient execution across these devices.

Persistent weight storage gives the model's base footprint, while longer requests add activations, key/value states, and temporary execution buffers. We therefore measure peak GPU allocation across prompt lengths to assess the representation within a complete workload. From 128 to 16K prompt tokens, *AlignQuant*'s peak increases from 2.72 to 4.44 GiB, while BF16 increases from 6.09 to 8.88 GiB (Table 21). At 16K, *AlignQuant* uses half the BF16 peak memory and records the lowest peak among the compared configurations. The compact model thus retains a substantial memory benefit as context-dependent state grows.

The distinction between persistent and temporary storage is specified in Appendices E.2 and E.3. For long prefill, a projection prepares selected weights in a temporary INT8 buffer for reuse across activation-row groups. The packed model remains the persistent representation, and the prepared operands belong to the current projection call. Peak-memory measurement includes these workload-dependent lifetimes alongside the model and context state. The observed footprint therefore evaluates compact storage together with the execution strategy that uses it.

Table 19: End-to-end inference efficiency for **Llama-3.2-3B-Instruct** on an **NVIDIA H100 NVL**. Cell colors are relative to BF16 under the corresponding metric direction: green for better than or equal to BF16, blue for worse than BF16, and gray for BF16. Bold and underlined values indicate the best and worst results, respectively, among the compared methods excluding the reference columns.

Metric	Unit	Upper Ref.	Lower Ref.	Ours	Others			
		BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP
TTFT ↓	ms	13.19	17.93	15.86	17.53	18.11	**15.79**	59.92
TPOT ↓	ms/token	12.26	15.31	**5.95**	16.52	16.91	14.06	56.84
Throughput ↑	tok/s	81.54	65.29	**167.49**	60.52	59.13	71.12	17.59
Resident GPU Mem. ↓	GiB	6.02	2.10	2.76	**2.13**	2.14	2.89	2.18
Peak VRAM ↓	GiB	6.11	2.19	2.76	**2.22**	2.23	2.94	2.40
Energy ↓	J/1K tokens	1829.26	1866.95	**1133.84**	1899.70	1945.35	1866.01	17611.37

Table 20: Cross-model decode efficiency on an **NVIDIA A100 40GB**. All measurements use batch size 1, a 128-token prompt, and 512 generated tokens. TPOT is reported in ms/token, throughput in tokens/s, and energy in J/1K generated tokens. Cell colors are relative to BF16 under the corresponding metric direction: green for better than BF16, blue for worse than BF16, and gray for BF16. Bold and underlined values indicate the best and worst results, respectively, among the compared methods excluding the reference columns.

Model	Metric	Unit	Upper Ref.	Lower Ref.	Ours	Others			
			BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP
Llama-3.2-3B	TPOT ↓	ms/token	17.19	21.12	**7.54**	20.77	21.78	24.05	82.33
	Throughput ↑	tok/s	58.18	47.33	**132.28**	48.13	45.91	41.58	12.15
	Energy ↓	J/1K tokens	2280.68	2322.34	**1267.92**	2047.43	1732.24	2172.71	22535.65
Qwen3-4B	TPOT ↓	ms/token	27.18	33.33	**10.88**	33.09	34.47	32.49	109.06
	Throughput ↑	tok/s	36.78	30.00	**91.62**	30.22	29.00	30.77	9.17
	Energy ↓	J/1K tokens	3540.00	3503.56	**1735.21**	2665.43	2586.02	2724.03	29045.23
Ministral-3-8B	TPOT ↓	ms/token	22.47	28.14	**13.15**	28.27	29.23	33.76	204.22
	Throughput ↑	tok/s	44.50	35.50	**75.72**	35.38	34.21	29.62	4.90
	Energy ↓	J/1K tokens	4285.26	3589.44	**2652.38**	3248.43	3132.54	3219.88	60742.08
Qwen3-14B	TPOT ↓	ms/token	30.29	36.24	**19.85**	36.45	37.55	35.17	350.49
	Throughput ↑	tok/s	32.99	27.55	**50.16**	27.43	26.63	28.42	2.85
	Energy ↓	J/1K tokens	6972.05	5364.27	**4354.57**	4747.57	4848.51	5805.63	111668.07

Table 21: Peak VRAM for **Llama-3.2-3B-Instruct** across prompt lengths. Peak VRAM is reported in GiB. Cell colors are relative to BF16: green for lower memory usage, blue for higher memory usage, and gray for BF16. Bold and underlined values indicate the best and worst results, respectively, among the compared methods excluding the reference columns.

Length	Upper Ref.	Lower Ref.	Ours	Others			
	BF16	W4A8	*AlignQuant*	GPTQ	AWQ	MixLLM	SFMP
128	6.09	2.16	2.72	**2.20**	2.21	2.92	2.37
512	6.10	2.20	2.80	**2.22**	2.23	2.99	2.41
2K	6.35	2.54	2.93	**2.47**	2.48	3.23	2.62
4K	6.71	2.99	3.13	**2.83**	2.84	3.62	2.95
8K	7.43	3.90	3.60	**3.55**	3.56	4.40	3.69
16K	8.88	5.72	**4.44**	4.99	5.00	5.94	5.18

H.5 Runtime-Control Results

Having established the full-model gains, we keep the selected weights fixed and vary the execution configuration to examine where runtime improvements arise. The CUDA Graph control measures a change in execution submission within *AlignQuant*, while the BF16 comparison evaluates the **complete deployed method**. Each runtime effect is conditional on the other components active in its comparison. Adaptive decode splitting lowers TPOT from 8.54 to 7.54 ms, while TTFT remains approximately 19.6 ms in both configurations (Table 22). The change is concentrated in the generation schedule. As described in Appendix E.3, single-sequence decode has one activation row, so the

Table 22: Runtime optimization ablation for **Llama-3.2-3B-Instruct** on an NVIDIA A100 40GB with a 128-token prompt and 512 generated tokens. Bold and underlined values indicate the best and worst results, respectively, across the four runtime configurations.

Metric	Unit	Full	Runtime Controls		
		AlignQuant	Fixed decode split	Separate preparation	Individual launches
TTFT ↓	ms	19.67	**19.62**	<u>29.80</u>	22.44
TPOT ↓	ms/token	**7.54**	8.54	<u>10.42</u>	<u>21.07</u>
Throughput ↑	tok/s	**132.28**	116.79	95.66	<u>47.45</u>
Peak VRAM ↓	GiB	2.72	**2.71**	**2.71**	<u>2.73</u>
Energy ↓	J/1K tokens	**1267.92**	1508.95	1820.18	<u>2234.22</u>

kernel exposes additional parallel work through output features and input-column groups. The split choice distributes reduction work, and measured TPOT supports adapting it to projection geometry.

Shared activation preparation has a broader phase effect. Relative to separate preparation, it reduces TTFT from 29.80 to 19.67 ms and TPOT from 10.42 to 7.54 ms. The implementation in Appendix E.4 prepares a common transformed 8-bit input for compatible query/key/value or gate/up projections. Each consumer then uses its own selected weights, scales, and output transform. The reductions in both metrics match this placement of the optimization: input preparation is reused across projections during both prompt processing and generation. The selected precision map stays attached to each projection while common input work is shared.

CUDA Graph execution addresses the submission of the repeated inference computation. Compared with individual kernel launches, it lowers TPOT from 21.07 to 7.54 ms and increases throughput from 47.45 to 132.28 tokens/s (Table 22). Peak GPU memory is 2.73 and 2.72 GiB, respectively, so the timing change occurs at a similar runtime footprint. The comparison holds the selected numerical model fixed and changes how its computation is submitted. It shows that efficient submission is an important part of converting fast low-bit projection work into full-model generation speed.

The controls identify improvements at three levels: reduction scheduling within a projection, input reuse across projections, and submission of repeated model computation. These execution choices operate on the values selected by the same calibration and packing procedure. Combined with the quality ablations, the results explain the roles of the design: local sensitivity guides where numerical precision is retained, compact packing realizes those choices in memory, and the runtime organizes their computation efficiently. The shared tile partition connects these roles in one deployed model.