SALTQ: SALIENCY-ALLOCATED LOW-BIT TRAINABILITY FOR QUANTIZED LLM FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Fine-tuning for deployment below 4 bits should directly produce a uniform low-bit weight-only checkpoint of integer codes and per-group quantization parameters, without an adapter branch or FP16 columns. Existing approaches fall short in different ways: adapter-based methods require post-merge re-quantization, mergeable variants freeze integer codes, weight-updating QAT computes the full weight gradient, and selective methods keep salient columns in FP16. To address this gap, we propose SALTQ, which allocates trainability by saliency within the deployed representation. A second-order analysis of the task loss shows that the columns where rounding costs the most loss are also those where a fine-tuning update moves the output the most, namely the few input columns with the largest activation energy, so that repairing the quantization error and adapting to the task can share one set of trainable columns. SALTQ makes these salient columns fully trainable at the deployment bit-width. In every other quantization group it trains only the zero-point, whose gradient pools within the group and is therefore almost free, whereas training the scales or the integer codes would require the full weight gradient. A channel permutation gathers the salient columns into complete groups; it is shared within contiguous segments of layers rather than across the whole network, which leaves one to three residual-stream reorderings at inference. Controlled comparisons show that the salient columns need to be trained but not in FP16: training them at the deployment bit-width retains nearly all of the gain of training them in FP16, with a gap of at most 0.8 points. At INT2 on Llama-2-7B and Qwen2.5-32B, SALTQ outperforms the compared mergeable and mixed-precision baselines by at least 0.7 points on the Commonsense-170k average and 3.7 points on GSM8K. At INT3, it reaches the level of the FP16 QLoRA reference on commonsense reasoning and GSM8K. It trains in about the time of QLoRA and in less than a third of the time of LR-QAT, which updates all weights and reaches a similar accuracy. The code will be released.

1 INTRODUCTION

A widely supported deliverable for low-bit deployment is a uniform low-bit weight-only checkpoint, in which every linear layer consists of b-bit integer codes and per-group quantization parameters and runs directly on a weight-only kernel, with neither an adapter branch nor FP16 columns. Below 4 bits, obtaining this checkpoint directly without a full weight gradient remains challenging (Figure 1 and Table 1). QLoRA-style methods (Dettmers et al., 2023) train a high-precision low-rank branch BA, but with frozen integer codes, group quantization $\widehat{W} = s(q - z)$ absorbs only per-group affine changes (Section 3.1), so a free-form BA must be merged and quantized a second time. Below 4 bits this step is costly: on Llama-2-7B, re-quantizing the merged QLoRA model to INT2 leaves less than 40% of its GSM8K accuracy (Section 4). The mergeable QA-LoRA (Xu et al., 2024) and PEQA (Kim et al., 2023) avoid this step but train only the per-group fields of the format, a low-rank shift of the group zero-points and the group scales, respectively. Every integer code stays frozen, and at INT2 they regain only about half of the accuracy lost by re-quantization. At the other end, quantization-aware training (QAT) such as LR-QAT (Bondarenko et al., 2024), EfficientQAT (Chen et al., 2025b) and LLM-QAT (Liu et al., 2024) treats all weights equally, computing the full weight gradient ($8N$ per token per layer, against $6N$ for QLoRA) and updating the entire dense weight at every step, so it trains about 3 to 5 times slower than QLoRA.

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.

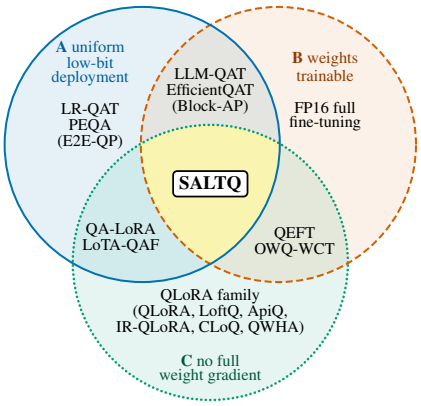

Figure 1: The problem that each approach leaves open. Circles A–C are the first three conditions of Table 1; each region lists the approaches meeting exactly those conditions.

Table 1: Design conditions satisfied by each approach. Uniform low-bit deployment: no adapter branch and no FP16 columns. Weights trainable: integer codes or latent weights receive element-wise, full-rank updates. No full weight gradient: the $6N$ tier of Table 6. No re-quantization: training takes place within the deployed representation and export does not re-quantize. SALTQ itself needs at most $S-1$ residual reorderings in the full model.

Method	Uniform low-bit deployment	Weights trainable	No full weight gradient	No re-quantization
FP16 full fine-tuning	✗	✓	✗	✗
QLoRA family	✗	✗	✓	✗
OWQ-WCT, QEFT	✗	✓	✓	✓
QA-LoRA	✓	✗	✓	✓
LR-QAT, PEQA	✓	✗	✗	✓
Block-AP, LLM-QAT	✓	✓	✗	✓
SALTQ	✓	✓	✓	✓

This uniform treatment is not justified, because the quantization loss is itself unevenly distributed over the weights. If the weight of a linear layer $y = Wx$ is perturbed by ΔW, either by rounding or by a fine-tuning update, its output changes by $(W + \Delta W)x - Wx = \Delta Wx$. For approximately uncorrelated input channels, the energy (expected squared norm) of this change is $\mathbb{E}\|\Delta Wx\|^2 \approx \sum_j \mathbb{E}[x_j^2]\|\Delta w_{:,j}\|^2$, with $\Delta w_{:,j}$ the j-th column of ΔW. Each column is thus weighted by the activation second moment $\mathbb{E}[x_j^2]$ of its input channel, as in the second-order expansion of the task loss (Section 3.2). In large language models $\mathbb{E}[x_j^2]$ is heavy-tailed, and a few channels carry one to two orders of magnitude more energy than the rest. A rounding error in these columns costs the most loss and an update there moves the output the most, so the columns whose quantization error most needs repair are those in which fine-tuning is most effective, and quantization-aware training and parameter-efficient adaptation can take place on one set of columns. OWQ-WCT (Lee et al., 2024b) and QEFT (Lee et al., 2024a) identify these columns but store and train them in FP16, which needs a mixed-precision kernel. On Llama-2-7B with all other weights static, training these columns at the deployment bit-width instead of in FP16 lowers the eight-task Commonsense-170k average by only 0.24 points at INT3 and 0.53 points at INT2 (Table 4). Keeping fine-tuned columns in FP16 while the rest is quantized again, in contrast, regains only about half of the gap to the FP16 reference even with nearly 40% of the weights in FP16, so the columns have to be trained within the quantized model. *Salient columns need to be trained, but not in FP16.*

These problems point to one design: train under the deployed quantization rule and allocate trainability by saliency. Under this rule each row is split into groups of g consecutive input columns (group size g), and each group shares one scale and one zero-point. SALTQ makes only the columns with the largest $\mathbb{E}[x^2]$ fully trainable (128 or 256 per layer, 2%–5% of the weights), by quantization-aware training at the deployment bit-width with a learned step size, in which latent weights are rounded in the forward pass and updated through the straight-through estimator (Esser et al., 2020; Bhalgat et al., 2020). In every other group it trains only the zero-point, the same field that QA-LoRA reaches through a low-rank factor. Trained directly, this field is almost free: its gradient pools within each group at $1/g$ of the cost of the forward general matrix multiplication (GEMM), whereas a per-group scale or an integer code needs the full weight gradient, another $2N$ per token per layer (Section 3.2). The full weight gradient is thus paid only on the salient columns, and the total cost of $\approx 6.1N$ is on the level of QLoRA. Because a group shares its scale and zero-point, the salient columns must fill whole groups of their own, and since they are scattered over the input dimension, this requires an input-channel permutation. The single global permutation of QEFT assumes that all layers share their salient channels, yet on Llama-2-7B the mean Jaccard similarity of the salient sets of two layers falls from 0.66 for adjacent layers to 0.15 for the first and the last layer, so salient sets are shared only locally. AutoSegment therefore refines this single ordering by dividing the network into $S \leq 4$ contiguous layer segments that each share one permutation, with boundaries placed by dynamic programming on calibration statistics. Within a segment the permutation is folded into the weights, so the full model adds at most $S-1$ residual reorderings. Each permuted layer is then quantized once, trained as above and exported bit-exactly as a uniform b-bit checkpoint.

Building on this design, this paper makes the following three contributions.

1. We propose SALTQ, a low-bit fine-tuning framework that trains directly within the deployed representation and allocates trainability by saliency. It directly delivers a uniform low-bit weight-only checkpoint, which existing approaches obtain only with an adapter branch, FP16 columns, a second quantization of the merged weights or a full weight gradient over all columns.
2. We introduce a two-level trainability allocation, which spends the full weight gradient only on the salient columns and adapts every other quantization group through its zero-point, whose gradient is almost free, so that training costs about as much as QLoRA. We further introduce AutoSegment, which replaces the single global ordering of the salient columns with orderings shared within layer segments.
3. We show on Llama-2-7B and Qwen2.5-32B that SALTQ outperforms all compared mergeable and mixed-precision baselines at INT2, with the largest margins on mathematical reasoning, and is on par with the FP16 reference at INT3, while training in about the time of QLoRA and in less than a third of the time of LR-QAT at a similar accuracy.

2 Related Work

Quantization-aware parameter-efficient fine-tuning. QLoRA (Dettmers et al., 2023) freezes the quantized weights and trains a high-precision low-rank adapter (Hu et al., 2022) in a side branch. LoftQ (Li et al., 2024), ApiQ (Liao et al., 2024), IR-QLoRA (Qin et al., 2024), CLoQ (Deng et al., 2025) and QWHA (Jeon et al., 2026) improve the initialization or the adapter form, but all deliver a high-precision branch. A mergeable line keeps the update within the deployed fields: QA-LoRA (Xu et al., 2024) pools the low-rank update within each group into the per-group zero-points, LoTA-QAF (Chen et al., 2025a) restricts it to a ternary low-rank increment on the integer grid, and PEQA (Kim et al., 2023) and AlphaTuning (Kwon et al., 2022) train quantization scales, with all integer codes frozen. SALTQ trains the zero-point field of QA-LoRA directly, since its gradient is almost free, and spends the full weight gradient on regenerating the integer codes of the salient columns, which these methods keep frozen.

Saliency-aware quantization and selective training. AWQ (Lin et al., 2024) ranks channels by activation statistics, and OWQ (Lee et al., 2024b), SpQR (Dettmers et al., 2024) and SqueezeLLM (Kim et al., 2024) isolate sensitive weights in higher precision, while OWQ-WCT and QEFT (Lee et al., 2024a) further train the weak columns kept in FP16. Our work differs in two respects: the salient columns are trained at the deployment bit-width rather than in FP16, at little cost in accuracy (Section 4.3), and the remaining columns train their zero-points rather than stay static. EfQAT (Ashkboos et al., 2024) applies quantization-aware training only to important weight blocks to reduce the backward cost, and PB-LLM (Shang et al., 2024) freezes the salient weights and trains the rest; their selection metric, granularity or direction of allocation differ from ours.

Quantization-aware training and reordering. LLM-QAT (Liu et al., 2024) and BitDistiller (Du et al., 2024) train all weights, LR-QAT (Bondarenko et al., 2024) constructs a low-rank product inside the quantizer, and EfficientQAT (Chen et al., 2025b) first optimizes all parameters block by block (Block-AP) and then fine-tunes the step sizes end to end (E2E-QP). These methods pay for the full weight gradient and dense per-step passes, whereas our question is which weights are worth this cost. RPTQ (Yuan et al., 2023), Atom (Zhao et al., 2024) and QEFT reorder channels per layer or globally to obtain contiguous layouts, and AutoSegment refines the global ordering of QEFT by sharing an ordering only among neighboring layers, whose salient sets overlap.

3 Method

3.1 Deployed-Representation Training

SALTQ fine-tunes the model directly in the quantized form in which it will be deployed, so that export only writes out the trained values and adds no quantization step. This form is weight-only, asymmetric uniform group quantization: every linear layer $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ stores one b-bit integer code $q \in \{0, \dots, 2^b - 1\}$ per weight, and activations stay in high precision. Each row of W is

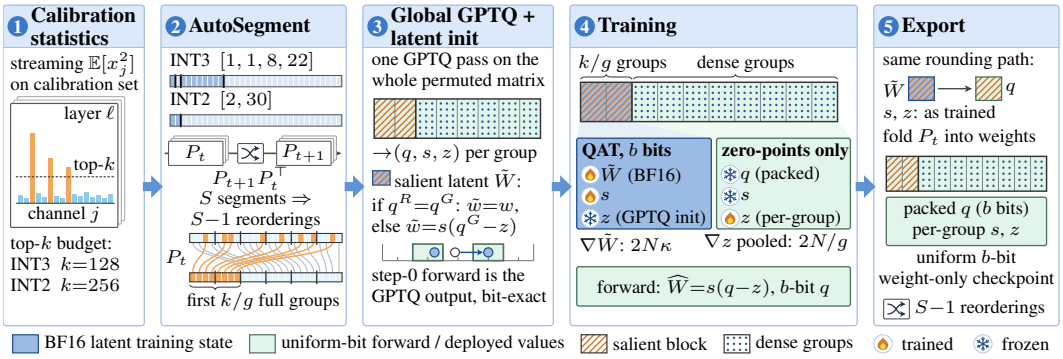

Figure 2: Overview of SALTQ. From left to right: $\mathbb{E}[x^2]$ statistics and top-k selection; the AutoSegment partition with one residual permutation per segment; global GPTQ after permutation and the latent initialization of the salient block; training, where the salient block is trained by quantization-aware training at b bits with BF16 latent weights while the dense groups keep packed frozen codes and train only zero-points; export along the same rounding path into uniform-bit codes and per-group parameters.

divided into groups of g consecutive input columns, where g is the group size, and all weights of a group share one scale s and one zero-point z, so that each weight is represented as

$$\widehat{W} = s\,(q - z). \tag{1}$$

This representation also determines which updates a checkpoint can absorb without a second quantization. Where the integer codes are frozen, training can change only the scale and the zero-point of each group, and these rescale or shift all weights of the group together. Individual weights can move only in the columns whose integer codes are regenerated during training, and Appendix A.1 states this set of feasible updates formally. A free-form low-rank update BA changes weights individually across all columns, so after merging it can be stored only by quantizing the model a second time. Within this constraint, the two kinds of trainability differ in cost and reach: regenerating integer codes moves individual weights but needs the full weight gradient, whereas the per-group zero-points can be trained far more cheaply but move whole groups at once. SALTQ regenerates integer codes only in the few columns where quantization error and fine-tuning updates matter most and trains the zero-point of every other group (Section 3.2), and it gathers the selected columns into complete groups with a permutation shared by contiguous segments of layers (Section 3.3). In the resulting pipeline (Figure 2), one GPTQ pass over the permuted weights initializes every field, training updates only the fields that the deployed checkpoint stores, and export regenerates the salient codes along the same rounding path and folds the permutations into the weight order, giving a uniform b-bit checkpoint that is bit-exact with the training forward pass.

3.2 Two-Level Trainability Allocation

Salient columns. Quantization-aware training asks where in a linear layer rounding costs the most loss, and task adaptation asks where an update moves the output the most. Both act on the layer through a weight change ΔW, and for the columns of W the two questions have the same answer. For the rounding error, the answer follows from a second-order expansion of the task loss in the weight perturbation. As in optimal brain damage, the first-order term is neglected (LeCun et al., 1990), and the Hessian is approximated by the Fisher information (Martens, 2020). For $y = Wx$ with output index o and input index j, so that $W_{o,j}$ connects input channel j to output o, and for $\delta = \partial\ell/\partial y$, the gradient $\nabla_W \ell = \delta x^\top$ lets K-FAC (Martens & Grosse, 2015) approximate the Fisher matrix by the Kronecker product $\mathbb{E}[\delta\delta^\top] \otimes \mathbb{E}[xx^\top]$, whose diagonal yields

$$H_{(o,j),(o,j)} \approx \mathbb{E}[\delta_o^2]\,\mathbb{E}[x_j^2], \qquad \Delta\mathcal{L} \approx \tfrac{1}{2}\sum_{o,j} \mathbb{E}[\delta_o^2]\,\mathbb{E}[x_j^2]\,\Delta W_{o,j}^2. \tag{2}$$

The row factor $\mathbb{E}[\delta_o^2]$ acts on a whole row and is largely absorbed by the scales of that row. The column factor $\mathbb{E}[x_j^2]$ acts on a whole column and is heavy-tailed in large language models (Lin et al., 2024; Xiao et al., 2023), so the same rounding error costs one to two orders of magnitude more loss in a few high-energy columns. For the update, the same column factor reappears: $\mathbb{E}\|\Delta W x\|^2 \approx$

$\sum_j \mathbb{E}[x_j^2]\|\Delta w_{:,j}\|^2$ for approximately uncorrelated input channels, so an update of a given size moves the output the most in the same high-energy columns. The two analyses thus select one set of columns: those whose rounding error most needs repair are those in which adaptation is most effective, so quantization-aware training and parameter-efficient adaptation can take place on the same few columns through one mechanism, and the full weight gradient is best spent there. Elsewhere sensitivity and leverage are both small, and per-group affine fields suffice. Both analyses rest on diagonal approximations, so Section 4.3 tests the resulting rule against random columns. SALTQ therefore estimates $\mathbb{E}[x_j^2]$ on a small calibration set, selects k channels of the largest second moment and moves them to the first k positions with a permutation P of the input dimension, which leaves the network function unchanged as shown in Appendix A.3. With k chosen as a multiple of g, these channels fill exactly the first k/g groups, which hold only a few percent of the weights. The projections $W_q, W_k, W_v, W_{\text{up}}$ and W_{gate} all read the residual stream and therefore share one permutation, whose k channels are chosen per segment (Section 3.3). W_{down} reads the MLP activations and takes its own k channels of the largest $\mathbb{E}[x_j^2]$, with a permutation that the output rows of W_{up} and W_{gate} absorb. W_o has no salient block, since a permutation of its input would have to act within each head with W_v. In each of these layers the first k/g groups form the salient block and the rest form the dense groups.

Dense groups. The dense groups hold more than 95% of the weights, so any field trained on them must be cheap. A field that needs the full weight gradient would bring back the cost of QAT over the whole matrix, and the three fields of Eq. (1), integer codes, scales and zero-points, differ precisely in this respect. Let $Y = X\widehat{W}^\top$, $G_Y = \partial\mathcal{L}/\partial Y$ and $N = d_{\text{in}}d_{\text{out}}$, the number of multiply-adds in the forward pass of one token. Writing γ for a group, that is, a set of g input columns, and $s_{o,\gamma}$, $z_{o,\gamma}$ for its scale and zero-point in output row o, the zero-point gradient is

$$\frac{\partial\mathcal{L}}{\partial z_{o,\gamma}} = -s_{o,\gamma}\sum_{j\in\gamma} G_{W,o,j} = -s_{o,\gamma}\Big[G_Y^\top\Big(\sum_{j\in\gamma} X_{:,j}\Big)\Big]_o, \tag{3}$$

whose within-group coefficient does not depend on the input position. X can thus be summed within each group first and then contracted with the output gradient at a dimension reduced by g, at a cost of about $2N/g$ and without forming the full weight gradient $G_W = G_Y^\top X$. The gradient of a per-group scale has no such pooled form: Appendix A.4 shows that it needs either the partial output of every group or G_W itself, both at the cost of a forward GEMM, and regenerating integer codes needs G_W as well. The zero-point is therefore the only dense field that reaches every group almost for free, while a scale or a code would add another $2N$. QA-LoRA trains the same field through a low-rank factor, which at rank 64 already spans all zero-point shifts of a 4096-wide layer at $g = 64$ (Appendix A.2); what matters here is that, trained directly, the field costs only $2N/g$, which leaves the full weight gradient to the salient columns.

Training paths. Following this cost structure, the salient block keeps BF16 latent weights $\tilde{W}$ and trains them by quantization-aware training at the deployment bit-width with a learned step size (Esser et al., 2020; Bhalgat et al., 2020): the latent weights are rounded in the forward pass, $q = \text{clamp}(\text{round}_{\text{STE}}(\tilde{W}/s + z), 0, 2^b - 1)$ and $\widehat{W} = (q - z)s$, and updated through the straight-through estimator. Its latent weights and scales are trained, and its zero-points stay at their initial values. The dense groups freeze integer codes and scales and train only the per-group zero-point z, which enters the forward pass and the exported checkpoint in the same form. Both paths start from one GPTQ pass (Frantar et al., 2023) over the whole permuted matrix. GPTQ quantizes the salient block first, and its column-wise error feedback has already made the dense codes compensate for the rounding error of the salient codes, so the latent weights of the salient block should start from this solution rather than from the original weights. Let q^R be the code obtained by directly rounding the original weight under the same group scale and zero-point and q^G the GPTQ code. We set $\tilde{w} = w$ if $q^R = q^G$ and $\tilde{w} = s(q^G - z)$ otherwise. The first forward pass of training then reproduces the GPTQ model exactly, while the latent weights keep the original sub-grid information where the codes agree.

3.3 AutoSegment Segmented Permutation

The permutation that gathers the salient channels acts on the residual dimension, so layers that read the same residual stream must either share it or reorder the stream between them. A single

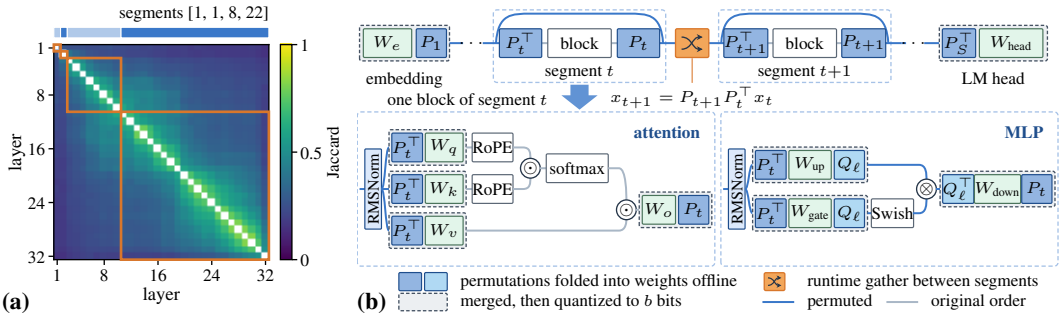

Figure 3: AutoSegment. (a) Jaccard similarity between the per-layer salient sets ($k = 128$) of Llama-2-7B at INT3: neighboring layers share most of their salient channels (mean 0.66 at distance 1) and the overlap decays with layer distance (0.15 at distance 31), so one permutation cannot serve the whole network. The diagonal is left empty, and the orange blocks mark the AutoSegment segments $[1, 1, 8, 22]$. (b) Where the permutation goes. Top: the network as segments; the embedding rows take P_1 and the LM-head columns $P_S^\top$ offline. Bottom: one block of segment t. Layers that read the residual stream take $P_t^\top$ on their input columns, layers that write it take P_t on their output rows, the RMSNorm gains follow, and the output rows of W_{up} and W_{gate} absorb Q_ℓ of W_{down}. Each segment boundary gathers the residual stream once, $x_{t+1} = P_{t+1} P_t^\top x_t$.

global permutation, as in QEFT (Lee et al., 2024a), discards the salient channels on which layers disagree, whereas a separate permutation per layer needs either a reordering of the residual stream before every block or kernels that permute the input of every layer. Figure 3(a) shows that neither extreme matches how salient channels are distributed across depth: neighboring layers share most of their salient channels, and this overlap decays with layer distance. AutoSegment therefore refines the global ordering of QEFT: it divides the network into contiguous layer segments and shares one residual permutation P_t within each segment. Within a segment the permutation is folded into the weights offline, as Figure 3(b) illustrates. Only the change of basis between two segments remains as a runtime gather of the residual stream, because the skip connection carries no weight into which it could be folded. The first permutation is folded into the embedding and the last into the LM head, so S segments cost at most $S - 1$ reorderings.

Using calibration statistics alone, AutoSegment marks as candidates in every layer the channels whose log second moment at the attention or MLP input lies more than σ standard deviations above the layer mean ($\sigma = 2.5$ on Llama-2-7B and 3 on Qwen2.5-32B), at most k per layer, and it scores a segment by how far the union of its candidates exceeds the budget of k channels. Dynamic programming then finds the partition with the least total overflow and takes the fewest segments that attain it (Algorithm 1 in Appendix B). On Llama-2-7B this gives two segments at INT2, whose budget of 256 channels holds all candidates from the third layer on, and four at INT3; both partitions split off the first two layers, whose outlier channels share little with the later ones.

4 Experiments

4.1 Setup

We fine-tune Llama-2-7B (Touvron et al., 2023) and Qwen2.5-32B (Qwen Team, 2024) on Commonsense-170k (Hu et al., 2023), evaluated on eight commonsense reasoning tasks, and on MetaMathQA (Yu et al., 2024), evaluated on GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021b). On Llama-2-7B we also fine-tune on Alpaca (Taori et al., 2023), evaluated by 5-shot MMLU (Hendrycks et al., 2021a), and on WikiText-2 (Merity et al., 2017), evaluated by test perplexity. The FP16 reference is the QLoRA adapter merged into the dequantized NF4 weights without re-quantization, and the floor (the GPTQ rows of the tables) is this checkpoint quantized by GPTQ to the target bit-width. At INT2 we also report how much of the gap between the floor and the reference a method recovers, while at INT3 the floor is only 1.70 points below the reference. The baselines are QEFT, which keeps k trainable FP16 columns and is written b+16, QA-LoRA, and PEQA with per-group scales, which trains the same parameters as E2E-QP of EfficientQAT. On the QAT side, LR-QAT and the full EfficientQAT pipeline, Block-AP followed by E2E-QP, are compared on Llama-2-7B in accuracy and training cost. All methods share the group size, the evaluation

Table 2: Commonsense reasoning (Commonsense-170k fine-tuning, eight tasks) **and mathematical reasoning** (MetaMathQA fine-tuning; GSM8K and MATH, greedy decoding) at INT3 $g64$ and INT2 $g32$. FP16 reference: QLoRA merged, not re-quantized; GPTQ: this reference re-quantized to the target bit-width (the floor). QEFT keeps k FP16 columns (b+16). Best value per block in bold.

Method	Bits	Commonsense-170k									MetaMathQA	
		BoolQ	PIQA	SIQA	HellaS	WinoG	ARC-e	ARC-c	OBQA	Avg	GSM8K	MATH
Llama-2-7B												
FP16 reference	16	69.27	81.12	77.53	92.42	76.56	82.70	64.42	78.00	77.75	58.07	10.64
GPTQ	3	67.25	79.11	76.56	91.32	74.66	80.51	63.82	75.20	76.05	52.69	8.58
QEFT k=128	3+16	69.51	80.20	77.33	91.87	77.11	82.20	65.61	77.60	77.68	56.41	9.50
QA-LoRA	3	69.27	80.09	79.17	92.09	77.35	82.79	64.33	76.60	77.71	57.09	8.58
PEQA	3	69.63	79.82	**79.79**	92.21	76.95	82.66	63.82	77.60	77.81	56.86	8.90
SALTQ k=128	3	**71.10**	**81.66**	78.76	**92.23**	**78.61**	**83.12**	**65.87**	**79.40**	**78.84**	**58.30**	**12.56**
GPTQ	2	58.78	71.76	71.90	78.90	64.96	68.77	53.07	61.60	66.22	22.06	2.16
QEFT k=256	2+16	65.54	**77.91**	76.82	87.58	73.32	75.63	59.64	73.20	73.71	48.52	7.44
QA-LoRA	2	65.08	77.86	75.44	86.46	73.01	73.78	56.83	71.20	72.46	52.54	10.04
PEQA	2	64.37	77.64	75.69	85.85	71.11	73.99	58.36	70.40	72.18	52.69	10.58
SALTQ k=256	2	**66.48**	**77.91**	**77.58**	**87.74**	**76.56**	**76.22**	**59.73**	**76.60**	**74.85**	**56.48**	**11.52**
Qwen2.5-32B												
FP16 reference	16	76.24	92.82	82.24	95.98	89.50	98.06	95.05	94.80	90.59	91.81	64.30
GPTQ	3	75.32	91.78	81.68	95.19	88.16	98.06	94.11	95.40	89.96	87.95	60.48
QEFT k=256	3+16	76.94	93.04	81.53	94.66	89.19	97.77	93.43	95.00	90.20	91.51	63.14
QA-LoRA	3	75.47	92.49	**82.04**	94.78	89.27	97.98	94.03	95.20	90.16	91.28	62.98
PEQA	3	76.39	92.27	81.78	95.08	**89.42**	**98.02**	93.77	94.60	90.17	**91.89**	63.28
SALTQ k=256	3	**77.65**	**93.42**	81.93	**96.50**	**89.42**	97.85	**94.11**	**95.40**	**90.79**	**91.89**	**63.82**
GPTQ	2	70.76	87.87	77.48	92.93	79.01	96.00	88.57	89.00	85.20	61.49	33.38
QEFT k=256	2+16	74.65	91.02	81.73	95.24	87.92	95.83	90.27	92.20	88.61	82.41	51.60
QA-LoRA	2	72.94	90.37	80.45	94.79	86.42	95.83	90.10	91.80	87.84	83.02	54.14
PEQA	2	72.48	90.15	80.76	94.61	86.98	95.33	89.85	91.40	87.70	83.62	53.62
SALTQ k=256	2	**75.32**	**91.19**	**82.14**	**95.60**	**89.74**	**96.76**	**90.96**	**93.00**	**89.34**	**88.70**	**57.02**

scripts and a task-balanced GPTQ calibration set drawn from the training data (Appendix E.5), and the learning rates of the baselines are tuned on a doubling grid (Appendix D). SALTQ and QEFT use $k = 128$ at INT3 $g64$ and $k = 256$ at INT2 $g32$ on Llama-2-7B, chosen with the zero-point learning rate on the Commonsense-170k validation split, and $k = 256$ at both bit-widths on Qwen2.5-32B.

4.2 Main Results

Table 2 shows a consistent pattern across the two backbones: the advantage of SALTQ grows as the bit-width decreases and is largest on GSM8K, where an answer depends on many exact generation steps. At INT3 every compared method stays within about one point of the FP16 reference on commonsense reasoning, and SALTQ reaches the reference on the commonsense average and on GSM8K for both models, about one point above the best baseline on the commonsense average of Llama-2-7B. Because the reference comes from a single recipe (QLoRA), we read reaching it as no measurable loss from quantization and do not interpret the margin above it. On Qwen2.5-32B at INT3 all methods lie within a point of each other on mathematics and are not ranked.

The methods separate at INT2, most clearly on GSM8K, where the mergeable baselines, whose integer codes all stay frozen, fall five to nine points below the reference while SALTQ stays within about three points on both models. The mixed-precision QEFT is the strongest baseline on commonsense reasoning at INT2 but falls behind the mergeable baselines on GSM8K for both backbones. This contrast suggests that mathematical reasoning needs updates that reach every quantization group, which the static dense part of QEFT cannot provide, and not only a few precise columns. SALTQ provides both, and it is ahead of QEFT in every block of the table without storing any FP16 column. On commonsense reasoning at INT2 it recovers 75% and 77% of the gap between the floor and the reference on the two backbones, against at most 65% for the baselines. The larger backbone is more tolerant to quantization, yet at INT2 the methods rank in the same order on both models, and SALTQ leads on both benchmarks in each of three training seeds, as Appendix E.8 shows.

The margins of SALTQ shrink on evaluations that measure general ability rather than the fine-tuned task (Table 3). After Alpaca fine-tuning, MMLU tests knowledge that the instruction data does not contain, and every method regains at most about half of the MMLU gap between the floor and

Table 3: MMLU after Alpaca fine-tuning (5-shot) **and WikiText-2 perplexity after WikiText-2 fine-tuning**, Llama-2-7B.

Method	Bits	MMLU	PPL
FP16, not fine-tuned	16	45.97	5.47
FP16 reference	16	45.84	4.84
GPTQ	3	40.96	5.27
QEFT k=128	3+16	43.17	5.14
QA-LoRA	3	42.20	5.15
PEQA	3	42.74	**5.12**
SALTQ k=128	3	**43.60**	**5.12**
GPTQ	2	27.15	8.87
QEFT k=256	2+16	34.25	7.02
QA-LoRA	2	31.93	7.30
PEQA	2	31.89	7.18
SALTQ k=256	2	**35.67**	**7.00**

Table 5: Comparison with QAT of all weights ($8N$), Llama-2-7B, as in Tables 2 and 6; SALTQ with the packed path. EfficientQAT: Block-AP, then E2E-QP.

Method	Bits	Time	CS-Avg	GSM8K	MATH
LR-QAT	3	5:24	78.31	**58.45**	11.86
EfficientQAT	3	7:28	77.54	57.85	9.94
SALTQ	3	1:39	**78.84**	58.30	**12.56**
LR-QAT	2	6:06	**75.19**	**56.56**	11.02
EfficientQAT	2	8:17	73.33	55.72	10.02
SALTQ	2	1:44	74.85	56.48	**11.52**

Table 4: 3×2 **comparison**: storage and training of the salient columns (rows) $\times$ state of the dense part (columns), eight-task means, Llama-2-7B. Before GPTQ: the columns fine-tuned with the QLoRA reference stay in FP16 and the rest is re-quantized. After GPTQ: the columns are trained on the quantized model in the setup of SALTQ.

Salient columns		Dense static		Dense z trained	
Stored	Trained	INT3	INT2	INT3	INT2
FP16	before GPTQ	76.32	69.99	77.43	71.39
FP16	after GPTQ	**77.84**	**74.12**	78.82	**75.65**
b-bit (SALTQ)	after GPTQ	77.60	73.59	**78.84**	74.85

Table 6: Training cost and accuracy at INT2 $g32$, Llama-2-7B, one Commonsense-170k epoch on 4×A100. Arithmetic: multiply-adds per token and linear layer ($N = d_{\text{in}}d_{\text{out}}$): $6N$ shared, $+2N$ full weight gradient, $+2N\kappa$ trainable columns ($\kappa = 4.86\%$), $+2N/g$ pooled zero-points. SALTQ: BF16-buffer or packed dense groups (Appendix C.3). Time in h:min. Params: trainable. CS-Avg: Commonsense-170k mean (QLoRA re-quantized).

Method	Arith.	Params	Time	Samp./s	CS-Avg
QEFT	$\approx 6.10N$	0.31B	1:06	37.14	73.71
QA-LoRA	$\approx 6N$	0.09B	1:13	33.54	72.46
QLoRA	$\approx 6N$	0.16B	1:37	25.25	66.22
PEQA (E2E-QP)	$8N$	0.20B	3:37	11.33	72.18
SALTQ (buffered)	$\approx 6.16N$	0.52B	1:39	24.87	**74.85**
SALTQ (packed)	$\approx 6.16N$	0.52B	1:44	23.64	**74.85**

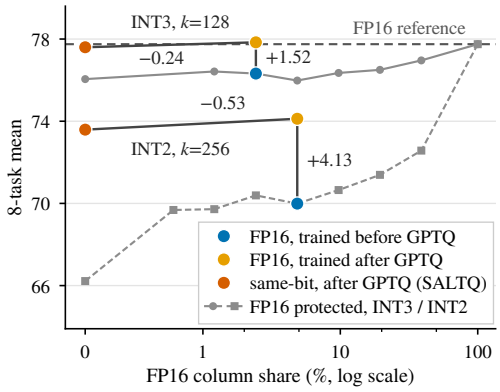

Figure 4: Accuracy versus trainability. Gray: fine-tuned columns kept in FP16, the rest re-quantized (Appendix E.2). Circles: dense-static cells of Table 4, joined per bit-width. Dashed: FP16 reference.

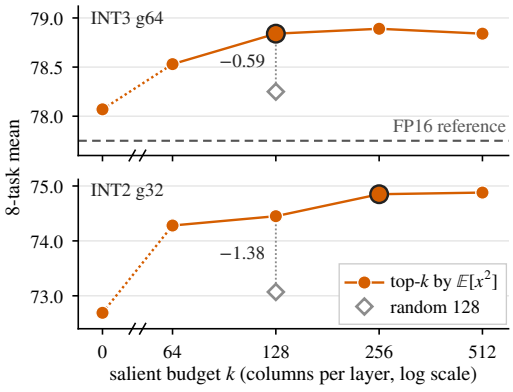

Figure 5: Accuracy versus salient budget. Top-k columns by $\mathbb{E}[x^2]$ (orange; up to 9.72% of weights) and 128 random ones (diamonds); ringed: default k; $k = 0$: zero-points only; dashed: FP16 reference.

the reference, with SALTQ ahead by at most one and a half points. WikiText-2 perplexity likewise measures how well the model restores the pretrained distribution over general text, and there SALTQ is on par with the baselines. We attribute both results to the kind of loss they measure: the knowledge removed by quantization is plausibly spread over the integer codes of all columns, which stay frozen in the dense part of every compared method, and data without this knowledge cannot restore it.

4.3 Ablations

All ablations fine-tune Llama-2-7B on Commonsense-170k, and Table 4 separates how the salient columns are stored from when they are trained. Whether they need FP16 is decided by its last two

rows, which train them on the quantized model: with a static dense part, same-bit training stays within 0.24 points of FP16 training at INT3 and 0.53 at INT2, so it retains nearly all of the gain of FP16 training. Keeping the columns fine-tuned with the reference in FP16 is no substitute: the first row is four points below the second at INT2, and even with nearly 40% of the weights in FP16 this protection recovers only about half of the gap to the reference (Figure 4). Training the dense zero-points adds one to one and a half points in every row. The two levels are complementary: trained alone ($k = 0$, Figure 5), the zero-points reach about the accuracy of QA-LoRA, which trains the same field, and stay more than two points below SALTQ at INT2. With both levels trained, same-bit salient columns cost nothing at INT3 and 0.80 points at INT2 against FP16 columns, which is the price of a uniform bit-width.

Segment-shared permutations recover part of the accuracy that the global ordering of QEFT loses: AutoSegment is 0.42 and 0.49 points above it at INT3 and INT2 and close to per-layer permutation, and within run-to-run noise of a uniform split with as many segments (Table 9, Appendix B).

Among the remaining choices (Appendix E.3), the initialization of the salient block has the largest effect. Starting from the original weights, which breaks the error compensation of GPTQ, costs more than three points at INT2, while keeping their sub-grid information where the codes agree adds a smaller gain over the grid centroids, within the seed-to-seed variation. Figure 5 shows that the budget saturates at 128 columns at INT3 and grows slowly up to 256 at INT2, so the codes need to be regenerated in very few columns. The ranking by $\mathbb{E}[x^2]$ carries most of this gain: 128 random columns improve on none ($k = 0$) by only 0.18 and 0.38 points, against 0.77 and 1.76 for the top 128, so what matters is where the trainable columns are placed, not how many parameters they add.

4.4 Training Cost and Deployment

Table 5 compares SALTQ with QAT methods that, like it, deliver a uniform low-bit checkpoint but compute the full weight gradient of every column. LR-QAT is within about half a point of SALTQ on commonsense reasoning and on par on GSM8K at 3.3 to 3.7 times the training time of SALTQ, depending on its implementation (Tables 6 and 18), and EfficientQAT, whose two stages together take 4.5 to 5 times as long, is less accurate on every benchmark, so a full weight gradient on every column buys little over one on the salient columns.

As Table 6 shows, SALTQ trains at INT2 in about the time of QLoRA, 1.02 to 1.07 times as long depending on the implementation, but delivers the INT2 checkpoint directly, whereas the QLoRA model falls to the floor once it is re-quantized. PEQA (E2E-QP), which trains every per-group scale, takes twice as long and is less accurate because every integer code stays frozen. These gaps exceed the ratio of $8N$ to $6N$ because the $8N$ methods also make the equivalent of 12 to 24 element-wise passes over the whole dense weight per step, against about three for SALTQ and QLoRA (Appendix C.4). QEFT and QA-LoRA are 1.4 to 1.6 times faster but less accurate, and QEFT stays mixed precision. The packed path of SALTQ lowers the INT2 peak training memory from 26.48 to 15.86 GiB at 5% less throughput; Appendix E.7 lists the INT3, Qwen2.5-32B and one-time costs.

At deployment, every quantized linear layer of SALTQ holds uniform b-bit integer codes and per-group quantization parameters and runs on a weight-only kernel with neither a mixed-precision kernel nor an adapter branch. The full model additionally reorders the residual stream at most $S - 1$ times, as a gather at each segment boundary fused with the adjacent operator.

5 Discussion and Conclusion

SALTQ allocates trainability by the quantization damage and update leverage that $\mathbb{E}[x^2]$ characterizes: the full weight gradient goes only to the salient columns, trained at the deployment bit-width, and every other group trains its zero-point at almost no cost. Under a uniform low bit-width, re-quantization after training rewrites the learned solution, whereas training within the deployed representation makes SALTQ on par with or more accurate than the compared mergeable methods at INT3 and more accurate at INT2, at the training time of QLoRA. If mixed precision is allowed, trainable FP16 salient columns remain about 0.8 points better at INT2. This paper is limited to weight-only task adaptation with high-precision activations; the frozen dense codes may limit the INT2 accuracy, and the initialization relies on task-balanced calibration data (Appendix E.5).

Reproducibility Statement

The method section defines the quantized representation and its trainable fields, the salient channel selection, the two-level trainability allocation, the AutoSegment objective, the GPTQ-consistent latent initialization and the training–export consistency; derivations and the algorithm are in Appendices A and B, implementation details in Appendix C, and the experimental field list and baseline notes in Appendix D. We plan to release the second-moment collection, AutoSegment, the post-permutation GPTQ initialization, the training kernels of both dense paths, the export, the execution support for segmented models and the evaluation code, with per-seed logs and environment versions.

Ethics Statement

This work proposes a model-compression and fine-tuning method, uses publicly available models and datasets, and involves no human subjects or private data.

Statement on the Use of Large Language Models

In this work, we used generative AI tools (LLM-based writing and coding assistants) for the following tasks that require disclosure: to translate the authors' Chinese drafts into the English manuscript, to assist in writing the derivations of Appendix A, to provide feedback on the research methodology and experiments through simulated reviews, and to help interpret the measured results. We have not used generative AI tools to generate synthetic data sets, to develop the conceptual framework, to formulate the mathematical claims or provide critical ingredients for their proofs, to propose hypotheses, to implement the method or the baselines, or to clean and reformat datasets, and qualitative or thematic data analysis is not applicable to this work. Additionally, we used generative AI tools to suggest the structure of the paper, to edit its text for readability, to create and modify the figures, to write the scripts that run the experiments and draw the figures, and to search for, check and format references. We have reviewed all AI-assisted work: the authors checked every number in the paper against the experiment logs, checked the derivations and the description of Algorithm 1 against the implementation, ran the AI-written scripts and verified their outputs, and checked every citation against its source. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

Saleh Ashkboos, Bram Verhoef, Torsten Hoefler, Evangelos Eleftheriou, and Martino Dazzi. EfQAT: An efficient framework for quantization-aware training. *arXiv preprint arXiv:2411.11038*, 2024. URL https://arxiv.org/abs/2411.11038.

Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. LSQ+: Improving low-bit quantization through learnable offsets and better initialization. In *CVPR Workshops*, 2020. URL https://arxiv.org/abs/2004.09576.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1911.11641.

Yelysei Bondarenko, Riccardo Del Chiaro, and Markus Nagel. Low-rank quantization-aware training for LLMs. *arXiv preprint arXiv:2406.06385*, 2024. URL https://arxiv.org/abs/2406.06385. A short version appeared at the ICML 2024 Workshop on Efficient Systems for Foundation Models (ES-FoMo II).

Junyu Chen, Junzhuo Li, Zhen Peng, Wenjie Wang, Yuxiang Ren, Long Shi, and Xuming Hu. LoTA-QAF: Lossless ternary adaptation for quantization-aware fine-tuning. In *Advances in Neural Information Processing Systems*, 2025a. URL https://arxiv.org/abs/2505.18724.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. EfficientQAT: Efficient quantization-aware training for large language models. In *Annual Meeting*

of the Association for Computational Linguistics, 2025b. URL https://arxiv.org/abs/2407.11062.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2019. URL https://arxiv.org/abs/1905.10044.

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. URL https://arxiv.org/abs/1803.05457.

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

Yanxia Deng, Aozhong Zhang, Selcuk Gurses, Naigang Wang, Zi Yang, and Penghang Yin. CLoQ: Enhancing fine-tuning of quantized LLMs via calibrated LoRA initialization. *Transactions on Machine Learning Research*, 2025. URL https://arxiv.org/abs/2501.18475.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.14314.

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. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Q1u25ahSuy.

Dayou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. BitDistiller: Unleashing the potential of sub-4-bit LLMs via self-distillation. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2402.10631.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgO66VKDS.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.17323.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://arxiv.org/abs/2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021b. URL https://arxiv.org/abs/2103.03874.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://arxiv.org/abs/2304.01933.

Hyesung Jeon, Seojune Lee, Beomseok Kang, Yulhwa Kim, and Jae-Joon Kim. QWHA: Quantization-aware Walsh-Hadamard adaptation for parameter-efficient fine-tuning on large language models. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.17428.

Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.14152.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2306.07629.

Se Jung Kwon, Jeonghoon Kim, Jeongin Bae, Kang Min Yoo, Jin-Hwa Kim, Baeseong Park, Byeongwook Kim, Jung-Woo Ha, Nako Sung, and Dongsoo Lee. AlphaTuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models. In *Findings of the Association for Computational Linguistics: EMNLP*, 2022. URL https://arxiv.org/abs/2210.03858.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM SIGOPS Symposium on Operating Systems Principles*, 2023. URL https://arxiv.org/abs/2309.06180.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1990. URL https://proceedings.neurips.cc/paper/1989/hash/6c9882bbac1c7093bd25041881277658-Abstract.html.

Changhun Lee, Jungyu Jin, Younghyun Cho, and Eunhyeok Park. QEFT: Quantization for efficient fine-tuning of LLMs. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024a. URL https://arxiv.org/abs/2410.08661.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. OWQ: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024b. URL https://arxiv.org/abs/2306.02272.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. LoftQ: LoRA-fine-tuning-aware quantization for large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.08659.

Baohao Liao, Christian Herold, Shahram Khadivi, and Christof Monz. ApiQ: Finetuning of 2-bit quantized large language model. In *Conference on Empirical Methods in Natural Language Processing*, pp. 20996–21020, 2024. URL https://aclanthology.org/2024.emnlp-main.1168/.

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. In *Proceedings of Machine Learning and Systems*, 2024. URL https://arxiv.org/abs/2306.00978.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. LLM-QAT: Data-free quantization aware training for large language models. In *Findings of the Association for Computational Linguistics: ACL*, pp. 467–484, 2024. URL https://aclanthology.org/2024.findings-acl.26/.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020. URL https://arxiv.org/abs/1412.1193.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*, 2015. URL https://arxiv.org/abs/1503.05671.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1609.07843.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *Conference on Empirical Methods in Natural Language Processing*, 2018. URL https://arxiv.org/abs/1809.02789.

Haotong Qin, Xudong Ma, Xingyu Zheng, Xiaoyang Li, Yang Zhang, Shouda Liu, Jie Luo, Xianglong Liu, and Michele Magno. Accurate LoRA-finetuning quantization of LLMs via information retention. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.05445.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. *Communications of the ACM*, 2021. URL https://arxiv.org/abs/1907.10641.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQA: Commonsense reasoning about social interactions. In *Conference on Empirical Methods in Natural Language Processing*, 2019. URL https://arxiv.org/abs/1904.09728.

Yuzhang Shang, Zhihang Yuan, Qiang Wu, and Zhen Dong. PB-LLM: Partially binarized large language models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=BifeBRhikU.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. URL https://arxiv.org/abs/2307.09288.

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*, 2023. URL https://arxiv.org/abs/2211.10438.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. QA-LoRA: Quantization-aware low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.14717.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.12284.

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. RPTQ: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*, 2023. URL https://arxiv.org/abs/2304.01089.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://arxiv.org/abs/1905.07830.

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. In *Proceedings of Machine Learning and Systems*, 2024. URL https://arxiv.org/abs/2310.19102.

A Derivations and Permutation Invariance

A.1 Feasible Update Set

With indices, Eq. (1) reads $\widehat{W}_{o,j} = s_{o,\gamma(j)}\big(q_{o,j} - z_{o,\gamma(j)}\big)$ with $q_{o,j} \in \{0, \dots, 2^b - 1\}$, where o indexes output rows, j input columns and $\gamma(j)$ is the group of g consecutive columns that contains j. Let $\mathcal{C}$ be the columns whose integer codes the quantizer regenerates during training (the salient block) and $\bar{\mathcal{C}}$ its complement, on which the codes are frozen. Fix an output o, a group γ and a column $j \in \gamma \cap \bar{\mathcal{C}}$. The old and new checkpoints are $\widehat{W}_{o,j} = s_{o,\gamma}(q_{o,j} - z_{o,\gamma})$ and $\widehat{W}'_{o,j} = s'_{o,\gamma}(q_{o,j} - z'_{o,\gamma})$ with the same $q_{o,j}$. Writing $s' = s + \Delta s$ and $z' = z + \Delta z$ and subtracting gives $\widehat{W}'_{o,j} - \widehat{W}_{o,j} = \Delta s_{o,\gamma}(q_{o,j} - z_{o,\gamma}) - s'_{o,\gamma}\Delta z_{o,\gamma}$; the converse follows by writing any (s', z') back into the checkpoint. On $\bar{\mathcal{C}}$, two valid checkpoints therefore differ only by $\Delta W = \Delta s \odot (q - z) - s' \odot \Delta z$, a per-group rescaling and shift. On the salient columns the latent update passes through the same quantizer in training and at export, and the relevant degrees of freedom are the changes of integer codes and scales that this quantizer can represent. The set $\mathcal{M}$ of updates representable without a second quantization thus consists of per-group scale and zero-point changes on $\bar{\mathcal{C}}$ and of what the quantizer can represent on $\mathcal{C}$. A free-form low-rank update BA changes individual weights in $\bar{\mathcal{C}}$ and lies outside $\mathcal{M}$, so after merging it must be quantized again. The characterization is premised on "integer codes frozen on $\bar{\mathcal{C}}$" and does not cover a re-quantization that rewrites all integer codes.

A.2 QA-LoRA and the Zero-Point Term

Let $R \in \{0,1\}^{G \times d_{\text{in}}}$ be the input group-pooling matrix. A group-pooled low-rank adapter gives $\Delta y = BA(Rx) = (BAR)x$, so $\Delta W_{o,j} = (BA)_{o,\gamma(j)}$ is constant within a group. If s and q are fixed, the update of Appendix A.1 reduces to $\Delta W = -s \odot \Delta z$, and taking $\Delta z_{o,\gamma} = -(BA)_{o,\gamma}/s_{o,\gamma}$ absorbs it exactly; this is why QA-LoRA merges without loss. The rank-r factorization restricts Δz to a scale-weighted rank-r matrix, whereas SALTQ optimizes the zero-point of every group directly. Because the pooled input has d_{in}/g dimensions, this restriction is void once $r \geq \min(d_{\text{out}}, d_{\text{in}}/g)$: with $r = 64$ it holds at $g = 64$ for every layer of Llama-2-7B with $d_{\text{in}} = 4096$, at $g = 32$ the rank of these layers is halved, and W_{down} ($d_{\text{in}} = 11008$) stays rank-limited at both group sizes. QA-LoRA and the dense part of SALTQ therefore train the same field and differ mainly in parameterization; what SALTQ adds is the observation that this field, trained directly, costs only $2N/g$ (Section 3.2), which leaves the full weight gradient to the salient columns.

A.3 Residual-Basis Equivariance

Let P be a permutation of the residual dimension and $x' = Px$. If the linear maps that read the residual stream use $W'_{\text{read}} = W_{\text{read}}P^\top$, the maps that write it use $W'_{\text{write}} = PW_{\text{write}}$, and the RMSNorm gain is $\omega' = P\omega$, then a pre-norm Transformer block computes the same function in the permuted residual basis. The reasons are that a permutation preserves the Euclidean norm, so RMSNorm is equivariant with P; $(WP^\top)(Px) = Wx$, so the coordinates entering the attention heads and RoPE are unchanged; in the gated MLP, $W_{\text{up}}P^\top$ and $W_{\text{gate}}P^\top$ cancel the input permutation, Swish and the element-wise product are unchanged, and PW_{down} writes back; residual addition is equivariant in the same basis. The exact basis change at a segment boundary is $x_{t+1} = P_{t+1}P_t^\top x_t$.

A.4 Gradient of a Per-Group Scale

Write $Y = X\widehat{W}^\top$ and $G_Y = \partial\mathcal{L}/\partial Y$. The gradient of a per-group scale is

$$\frac{\partial \mathcal{L}}{\partial s_{o,\gamma}} = \sum_t G_{Y,t,o}\, H_{t,o,\gamma}, \qquad H_{t,o,\gamma} = \sum_{j \in \gamma} X_{t,j}\,(q_{o,j} - z_{o,\gamma}), \tag{4}$$

where H is the partial forward output of group γ for output o. It has no pooled form independent of the output: one must either store or recompute the partial outputs of all groups (computation of the same order as the forward GEMM, $2N$, plus $T\, d_{\text{out}}\, d_{\text{in}}/g$ of storage) or materialize G_W ($2N$). In the degenerate case of a single scale per output channel, H is the forward output itself and the scale gradient is a per-channel reduction of negligible cost, but there are only d_{out} parameters per layer. By contrast, the zero-point gradient $-s_{o,\gamma}\sum_{j \in \gamma} G_{W,o,j}$ has a within-group coefficient that does not

depend on the input position, so X can be summed within each group first and then contracted with G_Y, at a cost of $2N/g$.

B AUTOSEGMENT ALGORITHM

Algorithm 1 AutoSegment: segment-shared permutations from calibration statistics

Require: second-moment vectors $E_{\ell s} \in \mathbb{R}^d$ at the attention ($s = a$) and MLP ($s = m$) inputs of layers $\ell = 0, \dots, L-1$; budget k; group size g; outlier threshold σ; maximum number of segments $S_{\max}$

$O_\ell \leftarrow \mathrm{trunc}_k\big(\bigcup_{s\in\{a,m\}} \{\, j : \log E_{\ell sj} > \mu_{\ell s} + \sigma\, \varsigma_{\ell s} \,\}\big)$ ▷ candidate channels: σ-outliers of log energy, at most k per layer; μ, ς = mean and std of $\log E_{\ell s}$

$\hat{E}_{\ell s} \leftarrow E_{\ell s} / \max_j E_{\ell sj}$ ▷ normalize each vector before aggregation

$U[a,b] \leftarrow \bigcup_{\ell=a}^{b-1} O_\ell$; $c[a,b] \leftarrow \max(0, |U[a,b]| - k)$ for all $0 \le a < b \le L$ ▷ interval cost: candidates beyond the budget

$D[0,0] \leftarrow 0$; $D[s,b] \leftarrow \min_{a<b}\big(D[s-1,a] + c[a,b]\big)$ for $s = 1..S_{\max}$, $b = 1..L$ ▷ minimum total overflow of s segments over the first b layers

$S^\star \leftarrow \min\{S \le S_{\max} : D[S,L] = \min_{s\le S_{\max}} D[s,L]\}$ ▷ fewest segments attaining the minimum overflow

$\{[a_t, b_t)\}_{t=1}^{S^\star} \leftarrow \textsc{Backtrack}(D, S^\star)$

for each segment t **do**

  $A_t \leftarrow \sum_{\ell=a_t}^{b_t-1} \sum_s \hat{E}_{\ell s}$ ▷ energy aggregated over the segment

  $\mathcal{K}_t \leftarrow$ first k entries of $[\, U[a_t,b_t) \,\|\, \overline{U}[a_t,b_t) \,]$, each part sorted by A_t descending ▷ candidates first, then the rest

  $P_t \leftarrow [\, \mathrm{sort}(\mathcal{K}_t) \,\|\, \text{remaining channels} \,]$ ▷ $\mathcal{K}_t$ fills groups $0, \dots, k/g - 1$

end for

return $\{[a_t, b_t)\}, \{\mathcal{K}_t\}, \{P_t\}$; warn if $D[S^\star, L] > 0$ ▷ some segment's candidates exceed the budget

Notation. d is the residual dimension and L the number of blocks; $E_{\ell s} \in \mathbb{R}^d$ holds $\mathbb{E}[x_j^2]$ over the calibration set at the input of the attention projections ($s = a$) or of the MLP projections ($s = m$) of layer ℓ; $\mu_{\ell s}$ and $\varsigma_{\ell s}$ are the mean and standard deviation of $\log E_{\ell s}$ over channels; O_ℓ is the candidate set of layer ℓ and trunc_k keeps at most its k channels of largest second moment; $U[a,b]$ is the union of the candidates of layers $a, \dots, b-1$ and $\overline{U}[a,b]$ its complement; $c[a,b]$ the overflow cost of that interval; $D[s,b]$ the dynamic-programming table; $[a_t, b_t)$ the t-th segment; A_t the aggregated energy; $\mathcal{K}_t$ the k channels kept by segment t; P_t its permutation; $\|$ concatenation.

Input. The second-moment vectors $E_{\ell s}$ at the attention ($s = a$) and MLP ($s = m$) inputs of every layer $\ell = 0, \dots, L-1$, the budget k, the group size g, the outlier threshold σ (2.5 on Llama-2-7B, 3 on Qwen2.5-32B) and the maximum number of segments $S_{\max} = 4$.

Step 1 (candidates). O_ℓ is the union over s of the channels whose log second moment exceeds the mean of $\log E_{\ell s}$ by σ standard deviations, truncated to at most k channels of largest second moment, so that no single layer exceeds the budget; $\hat{E}_{\ell s} = E_{\ell s} / \max_j E_{\ell sj}$ normalizes each vector for later aggregation.

Step 2 (costs). For every $0 \le a < b \le L$, $U[a,b] = \bigcup_{\ell=a}^{b-1} O_\ell$ and $c[a,b] = \max(0, |U[a,b]| - k)$.

Step 3 (partition). $D[0,0] = 0$, all other entries $+\infty$; $D[s,b] = \min_{a<b}\{D[s-1,a] + c[a,b]\}$ with predecessors stored. $D[S,L]$ is non-increasing in S; the number of segments is $S^\star = \min\{S \le S_{\max} : D[S,L] = \min_s D[s,L]\}$, the fewest segments that attain the minimum total overflow, and backtracking from $D[S^\star, L]$ gives the intervals $[a_t, b_t)$.

Step 4 (channels). In segment t, $A_t = \sum_{\ell=a_t}^{b_t-1} \sum_s \hat{E}_{\ell s}$ aggregates the normalized second moments; $\mathcal{K}_t$ is the first k of the list that puts the union $U[a_t, b_t)$ before the remaining channels, each part sorted by A_t.

Step 5 (permutation). P_t places $\mathcal{K}_t$ (sorted) first and the remaining channels after it, so $\mathcal{K}_t$ fills groups $0, \dots, k/g - 1$. A warning is raised if $D[S^\star, L] > 0$, i.e. if some segment's candidates exceed the budget; on Llama-2-7B the INT2 partition has zero overflow ($S^\star = 2$), while the INT3 partition leaves 65 channels ($S^\star = 4$), whose candidates beyond the budget are dropped in favor of the k channels with the largest aggregated energy.

Other backbones. On Qwen2.5-32B ($d = 5120$, $L = 64$) we run the same procedure with $\sigma = 3$ on the same kind of calibration set (3500 balanced Commonsense-170k examples, 400,305 tokens) and $S_{\max} = 4$. Table 7 and Figure 6 give the dynamic-programming objective of both backbones for every number of segments. Both Qwen2.5-32B runs use $k = 256$ (Section 4.1), and the partition depends only on σ and k, not on the bit-width. Unlike Llama-2-7B at INT2, no partition with at most four segments removes the overflow on Qwen2.5-32B, because the deep layers carry many candidates: before truncation the last layer alone has 272, more than $k = 256$, and with $k = 128$, shown for comparison, 14 deep layers (layer 49 and layers 52–64) have more than 128 each, so any segment that joins several deep layers exceeds the budget. With four segments the overflow falls to 10 for $k = 256$ and 104 for $k = 128$, and both partitions give the last layer, whose truncated candidate set fills the budget, a segment of its own. The candidate sets of neighboring layers also overlap slightly less than on Llama-2-7B (Jaccard 0.72 against 0.80 at distance 1 for $\sigma = 2.5$), and the overlap decays with layer distance (Table 8); the salient channels of the 32B model are spread more widely across layers and concentrate in the deepest ones.

Table 7: AutoSegment partitions for every number of segments ($S_{\max} = 4$): Llama-2-7B with $\sigma = 2.5$ and Qwen2.5-32B with $\sigma = 3$. $D[S, L]$: minimum total overflow with S segments; $|U|$: candidate union of each segment, after truncating every layer to at most k candidates. The rows marked $\star$ are the chosen partitions; the partition depends on σ and k only, and both Qwen2.5-32B runs use $k = 256$.

Setting	S	$D[S, L]$	Segments (layers, 1-based)	$\lvert U\rvert$ per segment
Llama-2-7B, $\sigma = 2.5$				
INT3 $g64$, $k = 128$	1	233	1–32	361
	2	172	1–10, 11–32	293, 135
	3	115	1, 2–10, 11–32	111, 236, 135
	4*	65	1, 2, 3–10, 11–32	111, 159, 155, 135
INT2 $g32$, $k = 256$	1	105	1–32	361
	2*	0	1–2, 3–32	217, 228
	3	0	1, 2, 3–32	111, 159, 228
	4	0	1, 2, 3, 4–32	111, 159, 105, 202
Qwen2.5-32B, $\sigma = 3$				
$k = 128$	1	183	1–64	311
	2	140	1–4, 5–64	103, 268
	3	117	1–3, 4–10, 11–64	83, 128, 245
	4*	104	1–3, 4–10, 11–63, 64	83, 128, 232, 128
$k = 256$ (INT3, INT2)	1	124	1–64	380
	2	44	1–35, 36–64	259, 297
	3	28	1, 2–53, 54–64	47, 249, 284
	4*	10	1–27, 28–57, 58–63, 64	246, 256, 266, 256

Table 8: Overlap of the per-layer candidate sets on Qwen2.5-32B: mean Jaccard similarity of O_ℓ (union of the attention and MLP inputs, before truncation to k) at selected layer distances, for two thresholds; over all distances the decay is not monotone.

σ	$\lvert O_\ell\rvert$ range (mean)	1	2	4	8	16	32	63
3.0	8–272 (101.5)	0.736	0.693	0.634	0.533	0.441	0.295	0.026
2.5	25–316 (141.9)	0.719	0.688	0.638	0.552	0.458	0.336	0.092

Permutation granularity. Table 9 compares AutoSegment on Llama-2-7B with the global ordering of QEFT, uniform splits into as many segments and a separate permutation per layer, which serves as a reference; counted as residual reorderings, it would need one before every block.

Complexity. Interval unions $O(L^2 d)$, dynamic programming $O(S_{\max} L^2)$. All inputs come from calibration statistics only.

C Implementation Details

C.1 GPTQ-Consistent Latent Initialization

For every linear layer that carries a salient block, on the permuted weight W_π: run GPTQ to obtain q^G, s, z; take the salient block $W_\pi[:, :k]$ and its fields; compute $q^R = \mathrm{clamp}(\mathrm{round}(W_\pi[:, :k]/s +$

Table 9: Permutation granularity: eight-task means on Llama-2-7B against the number of residual reorderings (Reord.).

Configuration	Reord.	INT3	INT2
Global (as in QEFT)	0	78.42	74.36
Uniform, 2 segments	1	78.49	74.60
Uniform, 4 segments	3	78.54	74.68
AutoSegment	3 / 1	**78.84**	74.85
Per-layer	31	**78.84**	**75.02**

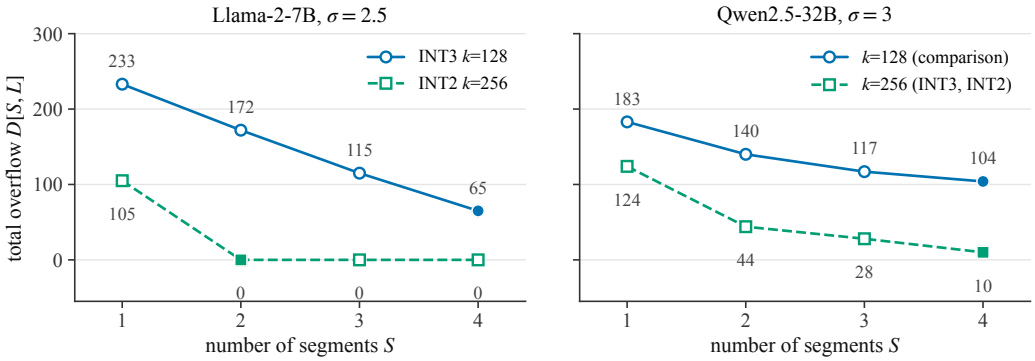

Figure 6: Dynamic-programming objective against the number of segments: total overflow $D[S, L]$ for $S = 1, \ldots, 4$ on Llama-2-7B ($\sigma = 2.5$, left; INT3 $k = 128$, INT2 $k = 256$) and Qwen2.5-32B ($\sigma = 3$, right; $k = 256$ as used by both runs, $k = 128$ for comparison). Filled markers: the chosen $S^\star$, the fewest segments attaining the minimum overflow. Per-segment details in Table 7.

$z), 0, 2^b - 1)$; form the mask $m = (q^R = q^G)$; set $\tilde{W} = m \odot W_\pi[:, :k] + (1 - m) \odot s(q^G - z)$; assert $\mathrm{fq}(\tilde{W}) = s(q^G - z)$; pack q^G for the dense block, freeze s, and take the zero-point z from GPTQ; the zero-point of the salient block is not trained. The two controls are taking the original weights directly (the step-0 forward pass differs from the GPTQ solution) and taking the GPTQ grid centroids everywhere (every latent weight sits at a cell center); results in Table 14. (Code labels: `gptq_latent`, `gptq`, `rtn`.)

C.2 Absorbing the Permutations

The residual-reading layers $W_q, W_k, W_v, W_{\text{up}}, W_{\text{gate}}$ right-multiply by $P_t^\top$; the residual-writing layers W_o, W_{down} left-multiply by P_t; RMSNorm gains follow; the output rows of the embedding are reordered offline in the order of the first segment and the input columns of the LM head in the order of the last segment; at a segment boundary the residual stream receives the index map $P_{t+1}P_t^\top$, fused with the adjacent operator. The input permutation of W_{down} is absorbed by the output rows of W_{up} and W_{gate}; W_o carries no salient block.

C.3 Two Dense Paths

BF16-buffer path. The dense weights are stored as fake-quantized values in a BF16 buffer, rebuilt at every step from (q, s, z); the forward pass is an ordinary GEMM. *Packed path.* Integer codes stay packed throughout; the forward kernel dequantizes in registers and accumulates, the input gradient reuses the packed representation, and the zero-point backward pass first reduces X within each group and then multiplies by G_Y; the salient block is a contiguous $d_{\text{out}} \times k$ block executed by a narrow QAT kernel, and the groups handled by the two paths do not overlap. Measured (Llama-2-7B, one Commonsense-170k epoch): at INT2 $g32$ with $k = 256$, the BF16-buffer path has a resident frozen state of 12.3 GB, wall-clock 5934 s, throughput 24.87 samples/s and peak 26.48 GiB; the packed path 1.5 GB, 6242 s, 23.64 and 15.86 GiB. At INT3 $g64$ with $k = 128$, the former is 12.6 GB, 5667 s, 26.04 and 21.18 GiB; the latter 2.4 GB, 5933 s, 24.87 and 12.06 GiB. The peak is dominated

by trainable state rather than frozen weights: at INT2 there are about 0.2 billion dense zero-points and about 0.3 billion salient latent weights, which with their gradients and Adam states account for roughly 6–7 GB by parameter count, plus activations and workspace.

C.4 Per-Step Time Model, Pass Counts and Measured Equivalents

The arithmetic column of Table 6 counts multiply-adds per token and predicts that the $8N$ methods train about one third longer than QLoRA, yet PEQA, Block-AP and LR-QAT take two to almost four times as long, and the two stages of EfficientQAT together four and a half to five times as long (Tables 5, 6 and 18). The difference comes from work that is done once per step on the whole dense weight and does not scale with the number of tokens. With $\bar{T}$ tokens per micro-step and N_{tot} linear-layer weights in the whole model, the time of one step is approximately

$$t_{\text{step}} \approx \frac{c_{\text{gemm}}\, N_{\text{tot}}\, \bar{T}}{R_c} + \frac{n_{\text{pass}}\, \beta\, N_{\text{tot}}}{R_m} + t_{\text{other}}, \tag{5}$$

where the first term is the arithmetic column of Table 6, the second is the per-step element-wise pass over the dense weights (n_{pass} passes, β bytes read and written per element, R_m the effective bandwidth of element-wise kernels), and the third is independent of the method. The second term does not grow with the number of tokens and dominates when fine-tuning samples are short. Table 10 counts, for each method under activation recomputation, the number of element-wise passes over the dense weights per step, roughly following the computation path of its published implementation (F1 first forward, F2 recomputation, B backward, Opt optimizer), together with the equivalent inferred from the measured training times. The inference uses the INT3 runs of Table 18: it takes the per-step time of QEFT as the zero-pass anchor and the amount by which QLoRA exceeds QEFT as the cost of three NF4 dequantization passes (about 69 ms each over the 6.5 B weights of the whole model), and converts each method's per-step time above QEFT into that equivalent; it folds the increase in arithmetic in as well and is therefore an upper bound. The INT2 runs of Table 6 give the same ordering of the methods.

Table 10: Per-step passes over the dense weights by counting (published computation paths) and by inference from measured per-step time.

Method	F1	F2	B	Opt	Count	Inferred
QEFT (dense BF16, static)	0	0	0	0	0	0 (anchor)
QA-LoRA (fused dequantize–GEMM kernel)	fused	fused	fused	0	≈ 0	0.6
QLoRA (NF4 materialized to BF16)	1	1	1	0	3	3.0 (anchor)
SALTQ, BF16 buffer	1	1	0	0	2	2.7
SALTQ, packed + fused kernel	fused	fused	fused	0	≈ 0 (fused kernel slower than cuBLAS)	3.1
PEQA ($W = s \odot q$, fp32 intermediates)	1	1	G_W 1 + reduction 1–2	0	4–5 (fp32 doubles)	11.9
EfficientQAT Block-AP	fake-quant chain ≈ 4	≈ 4	STE $\approx 3 + G_W$ 1	Adam ≈ 4	≈ 16	17.9
LR-QAT	$W_0 + sBA$ 2 + 6 fake-quant chain 4		STE 3 + G_W 1 + A, B contraction 2	0	≈ 18 (fp32 doubles)	24.2

These are the costs of the implementations measured, not lower bounds of the parameterizations: the $8N$ methods rebuild the whole dense weight at every step (by fake quantization or, in PEQA, by rescaling the codes) and form its full gradient, and Block-AP also updates it with Adam, whereas SALTQ and QLoRA only dequantize it two or three times per step.

D Experimental Details and Baseline Notes

Data. Commonsense-170k: 147,580 training examples, the remainder used as the validation split; the eight evaluation tasks are BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy and ARC-challenge (Clark et al., 2018) and OpenBookQA (Mihaylov et al., 2018), scored by generative answer matching with an evaluation window of 2048. MetaMathQA: GSM8K 1319 problems and

MATH 5000 problems, greedy decoding with vLLM (Kwon et al., 2023). Alpaca: 3 epochs, 5-shot MMLU, question-weighted overall mean. WikiText-2: 3 epochs on the training split, test split evaluated with non-overlapping windows and loss on all tokens. Commonsense-170k and MetaMathQA are trained for 1 epoch.

Hardware and batching. Llama-2-7B: 4×A100, micro-batch 4 per GPU, gradient accumulation 5, sequence length 2048. Qwen2.5-32B: 3×A6000, micro-batch 3 per GPU, gradient accumulation 9, sequence length 2048.

Quantization and initialization. Asymmetric INT3 $g64$ / INT2 $g32$; task-balanced GPTQ calibration drawn from the training data (commonsense: balanced 3500 examples, about 471k tokens; mathematics: balanced 1000 examples); k for SALTQ: 128 at INT3 and 256 at INT2 on Llama-2-7B, 256 at both bit-widths on Qwen2.5-32B; QEFT keeps the same number of FP16 columns; saliency is the top-k by second moment, shared within a segment by the layers that read the residual stream, with statistics collected on the calibration set; the salient block is initialized to the GPTQ-consistent latent weights (Appendix C.1); the salient block trains latent weights and scales and does not train zero-points, and the dense groups train only their zero-points (learning rates in Table 11).

Training recipe. All methods use a cosine schedule with 3% warm-up, gradient clipping at 0.3, AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$; the paged 8-bit variant on Qwen2.5-32B) and BF16 training, with an effective batch of 80 sequences (81 on Qwen2.5-32B, see above); on Commonsense-170k and MetaMathQA the loss covers instruction and response together. Apart from the optimizer variant, the same settings are used on both backbones and all fine-tuning tasks. Table 11 lists the learning rates and weight decays, which differ between methods and bit-widths; the low-rank methods use rank 64 ($\alpha = 16$ for QLoRA). The learning rate of every baseline is tuned per bit-width on the Commonsense-170k validation split with a doubling grid: starting from 10^{-4}, it takes the values 1, 2, 4 and 8×10^{-4}, and while the accuracy still improves the search continues with 1, 2, 4 and 8×10^{-3}, until a larger rate lowers the accuracy; the best rate is kept. All rates in Table 11 lie on this grid.

Table 11: Learning rates and weight decay per method, parameter group and bit-width; all other settings are shared (see text).

		Learning rate		
Method	Parameter group	INT3 $g64$	INT2 $g32$	Weight decay
QLoRA ($r = 64$, $\alpha = 16$)	adapter	2×10^{-4}	2×10^{-4}	0.01
QA-LoRA ($r = 64$)	adapter	4×10^{-3}	8×10^{-3}	0.01
PEQA	per-group scale	1×10^{-4}	2×10^{-4}	0.01
QEFT	FP16 columns	1×10^{-4}	2×10^{-4}	0.01
LR-QAT ($r = 64$)	low-rank factors	1×10^{-3}	4×10^{-4}	0.01
	scale	2×10^{-4}	2×10^{-4}	0
SALTQ	salient latent weights	1×10^{-4}	2×10^{-4}	0.01
	salient scales	2×10^{-4}	2×10^{-4}	0
	dense zero-points	2×10^{-3}	4×10^{-3}	0

Baseline notes. PEQA trains per-group scales. In terms of the trainable parameterization, E2E-QP of EfficientQAT is per-group-scale PEQA (integer codes and zero-points fixed, only per-group scales optimized); among the mergeable baselines we cover this mechanism with per-group-scale PEQA, and on the QAT side we run the full EfficientQAT pipeline, Block-AP followed by E2E-QP, whose training time is the sum of the two stages (Table 5). LoTA-QAF: we have not yet reproduced results close to those reported in its paper, and we found no other work that reports a reproduction, so it is not listed as a baseline.

Seeds. The main tables and all ablations report seed 0; Appendix E.8 repeats the main comparison with two further training seeds.

Table 12: LR-QAT and SALTQ at INT3 g64 and INT2 g32, Llama-2-7B, same protocol as Table 2 (seed 0). LR-QAT trains all weights through a low-rank term inside the quantizer ($8N$) and takes 3.3 to 3.7 times the training time of SALTQ, depending on its implementation (Tables 5, 6 and 18).

Method	BoolQ	PIQA	SIQA	HellaS	WinoG	ARC-e	ARC-c	OBQA	Avg	GSM8K	MATH
INT3 g64											
LR-QAT	71.77	81.88	77.12	92.25	79.32	82.62	63.48	78.00	78.31	58.45	11.86
SALTQ k=128	71.10	81.66	78.76	92.23	78.61	83.12	65.87	79.40	78.84	58.30	12.56
INT2 g32											
LR-QAT	67.16	76.88	76.61	87.69	79.48	76.01	62.46	75.20	75.19	56.56	11.02
SALTQ k=256	66.48	77.91	77.58	87.74	76.56	76.22	59.73	76.60	74.85	56.48	11.52

Table 13: FP16-protection sweep, Llama-2-7B, eight-task means; no further training after quantization. —: not measured.

k	FP16 column share	INT3 g64	INT2 g32
0 (floor)	0	76.05	66.22
32	0.61%	—	69.68
64	1.21%	76.42	69.72
128	2.43%	76.32	70.39
256	4.86%	75.98	69.99
512	9.72%	76.35	70.65
1024	19.43%	76.50	71.39
2048	38.86%	76.96	72.57
all	100%	77.75	77.75

E Additional Results

E.1 Comparison with Full-Weight QAT

Table 5 in Section 4.4 compares the averages and training times, and Table 12 gives the per-task scores of LR-QAT, which trains all weights through a low-rank term inside the quantizer, and SALTQ on Llama-2-7B. LR-QAT is 0.34 points higher on the commonsense average at INT2 and SALTQ 0.53 points higher at INT3, while GSM8K is on par at both bit-widths. One Commonsense-170k epoch takes LR-QAT 5 h 24 min at INT3 and 6 h 06 min at INT2, against 1 h 39 min and 1 h 44 min for the packed implementation of SALTQ (Tables 5 and 18).

E.2 FP16-Protection Sweep

Table 13 keeps the top-k columns by second moment in FP16 on the FP16 reference, that is, the fine-tuned QLoRA adapter merged into the dequantized NF4 weights, and quantizes the rest by GPTQ without further training; its curves are drawn in Figure 4. The cells of Table 4 relate to the other tables as follows. The FP16 row trained before GPTQ keeps the salient columns of the fine-tuned reference in FP16; with a static dense part it equals the sweep at $k = 128$ / 256, and its INT2 value with trained zero-points, 71.39, is the best of a learning-rate sweep and lies below the 72.69 of $k = 0$ (Table 14): with $k = 0$ the high-energy columns sit in ordinary groups and move with their trained zero-points, as they do in QA-LoRA, whereas the fixed FP16 columns receive no further update. The FP16 row trained after GPTQ is a separate run in the configuration of SALTQ (segmented permutation, GPTQ-consistent initialization, same recipe) with the salient columns stored and trained in FP16, so it differs from QEFT, which uses a global permutation. The same-bit static cell equals the zero-point-frozen arm of Table 14.

E.3 Salient-Block Initialization, Zero-Point Learning Rate and Budget

Table 14 gives the initialization and zero-point learning-rate arms (INT2 g32, $k = 256$) and the budget sweep; the additional arm with the salient-block learning rate doubled scores 74.36, below

Table 14: Salient-block initialization and zero-point learning rate (left; INT2 $g32$, $k = 256$; "zp lr" is the dense zero-point learning rate as a multiple of 2×10^{-3}, so $\times 2$ is the 4×10^{-3} of Table 11) **and salient budget** k (right), eight-task means, Llama-2-7B.

Initialization / zp lr	INT2	Budget k	INT3 $g64$	INT2 $g32$
Original weights (RTN), $\times 1$	70.33	0 (zero-points only)	78.07	72.69
GPTQ grid centroid, $\times 1$	73.91	64	78.53	74.28
GPTQ grid centroid, $\times 2$	74.42	128	78.84	74.45
GPTQ grid centroid, $\times 4$	73.13	256	**78.89**	74.85
GPTQ-consistent latent, $\times 2$ (SALTQ)	**74.85**	512	78.84	**74.88**
GPTQ-consistent latent, zp frozen	73.59	Random 128	78.25	73.07

Table 15: Initialization and zero-point learning-rate ablation per task (INT2 $g32$, $k = 256$, one Commonsense-170k epoch, Llama-2-7B). Each row names the initialization of the salient block and the learning rates that differ from the default; the SALTQ row is the main run of Table 2.

Arm	BoolQ	PIQA	SIQA	HellaS	WinoG	ARC-e	ARC-c	OBQA	Avg
Original weights (RTN), zp lr $\times 1$	65.05	76.55	75.64	81.77	71.27	69.99	53.33	69.00	70.33
Grid centroid, zp lr $\times 1$	66.15	78.45	76.05	87.32	73.32	76.26	59.73	74.00	73.91
Grid centroid, zp lr $\times 2$	66.64	78.62	77.33	87.39	75.14	76.52	59.13	74.60	74.42
Grid centroid, zp lr $\times 4$	66.21	77.80	76.10	86.37	74.66	72.98	56.31	74.60	73.13
Grid centroid, zp lr $\times 2$, salient lr $\times 2$	66.70	77.75	77.53	86.96	75.14	75.34	59.04	76.40	74.36
GPTQ-consistent latent, zp lr $\times 2$ (SALTQ)	66.48	77.91	77.58	87.74	76.56	76.22	59.73	76.60	**74.85**
GPTQ-consistent latent, zp frozen	65.66	77.37	75.79	86.62	74.03	74.75	59.73	74.80	73.59

Table 16: MMLU by category after Alpaca fine-tuning with non-fine-tuned anchors, Llama-2-7B, 5-shot; Avg is the question-weighted overall mean. GPTQ: the FP16 reference re-quantized (the floor).

Method	Bits	Hums	STEM	Social	Other	Avg
FP16, not fine-tuned	16	43.04	37.39	52.03	53.11	45.97
FP16 reference (QLoRA merged)	16	43.27	37.14	51.54	52.91	45.84
GPTQ of the pretrained model	3	37.87	34.60	45.53	47.51	40.95
GPTQ	3	39.28	33.90	44.13	47.54	40.96
QEFT, not fine-tuned	3+16	40.34	37.17	48.36	49.60	43.43
QEFT k=128	3+16	**40.60**	35.68	47.74	**50.14**	43.17
QA-LoRA	3	39.70	34.32	46.96	49.28	42.20
PEQA	3	40.13	35.01	47.74	49.49	42.74
SALTQ k=128	3	40.02	**37.65**	**49.17**	49.57	**43.60**
GPTQ of the pretrained model	2	24.65	27.50	28.96	27.04	26.76
GPTQ	2	25.84	28.42	29.25	25.78	27.15
QEFT, not fine-tuned	2+16	30.56	26.32	30.91	33.22	30.27
QEFT k=256	2+16	33.77	30.51	34.64	**38.40**	34.25
QA-LoRA	2	32.16	27.85	33.90	33.79	31.93
PEQA	2	32.11	27.38	34.53	33.30	31.89
SALTQ k=256	2	**34.97**	**33.83**	**35.94**	38.34	**35.67**

the 74.42 of the arm without doubling. Table 15 gives the per-task scores of the initialization and learning-rate arms.

E.4 MMLU Category Scores and Non-Fine-Tuned Anchors

Table 16 splits the MMLU scores of Table 3 into the four categories and adds three models that are not fine-tuned: the FP16 pretrained model, its GPTQ quantization and the QEFT starting point.

E.5 Calibration Ablation

Table 17 quantizes the same FP16 reference by INT2 $g32$ GPTQ and changes only the calibration set; the probe is the answer loss and answer accuracy on 128 examples of each of the eight tasks. Each row changes the corpus source, the task coverage and the token count at the same time, so the

Table 17: Calibration ablation, Llama-2-7B, INT2 $g32$ GPTQ of the same FP16 reference without further training. CS-Avg: eight-task mean of the quantized model; answer loss and accuracy: probe on 128 examples per task. The last balanced row is the calibration set of all other experiments.

Calibration set	Tokens	CS-Avg	Answer loss	Answer accuracy
BoolQ only, 128 examples	9.5k	36.64	1.17	0.41
BoolQ only, 3500 examples	256k	44.93	1.15	0.45
Balanced, 128 examples	17k	59.33	0.88	0.59
Balanced, 3500 examples	471k	**66.22**	**0.72**	**0.67**
C4, 128×2048	262k	30.67	1.10	0.45

Table 18: Training cost and accuracy at INT3 $g64$ on Llama-2-7B, same protocol as Table 6 ($\kappa = 2.43\%$). The QLoRA mean is the one after re-quantization to INT3 (77.75 before).

Method	Arithmetic	Params	Wall-clock (s)	Samples/s	CS-Avg
QEFT	$6N + 2N\kappa \approx 6.05N$	0.16B	3919	37.66	77.68
QA-LoRA	$\approx 6N$	0.09B	4286	34.43	77.71
QLoRA (NF4 base)	$\approx 6N$	0.16B	5840	25.27	76.05
PEQA (E2E-QP)	$8N$	0.10B	11512	12.82	77.81
EfficientQAT Block-AP	$8N$	6.68B	15372	9.60	n/a
LR-QAT	$8N$	0.16B	19445	7.59	78.31
SALTQ, BF16 buffer	$6N + 2N\kappa + 2N/g \approx 6.08N$	0.26B	5667	26.04	78.84
SALTQ, packed + fused	$6N + 2N\kappa + 2N/g \approx 6.08N$	0.26B	5933	24.87	78.84

table supports "the calibration configuration strongly affects the INT2 starting point of this pipeline" without attributing the effect to task balance alone.

E.6 Non-Fine-Tuned Anchors and Sanity Rows

WikiText-2 perplexity of non-fine-tuned bases: FP16 Llama-2-7B 5.47; QEFT base ($k = 256$ columns in FP16) 7.54; pure INT2 $g32$ GPTQ (0.5M in-domain calibration tokens) 13.44; round-to-nearest without GPTQ 217447 at INT2 and 99518 at INT3.

E.7 Additional Training Cost Results

Table 18 gives the INT3 counterpart of Table 6. Trainable parameters of SALTQ on Llama-2-7B: at INT2 $g32$ with $k = 256$, 314.6M salient latent weights, 9.8M salient scales and 192.5M dense zero-points (0.52B, 8.0% of the 6.48B linear weights); at INT3 $g64$ with $k = 128$, 157.3M, 2.5M and 98.7M (0.26B, 4.0%). QEFT trains the same salient columns in FP16, PEQA one scale per group, and Block-AP all linear weights with their per-group scales and zero-points, one block at a time; QLoRA, QA-LoRA and LR-QAT place rank-64 factors on every linear layer except the embedding and the LM head, 159.9M parameters for QLoRA and LR-QAT, and 89.3M at $g32$ and 88.2M at $g64$ for QA-LoRA, whose input factor acts on the group-pooled input. The peak training memory of SALTQ at INT3 is 12.06 GiB for the packed path and 21.18 GiB for the BF16-buffer path (Appendix C.3).

Qwen2.5-32B and one-time costs. The steps before training are the second-moment statistics, the AutoSegment partition, the selection of the salient channels and the post-permutation GPTQ initialization (Table 19); they run once per model and bit-width and are not included in the fine-tuning wall-clock times above. The statistics dominate together with GPTQ, while the partition and the channel selection take well under a second. The calibration set has 470,882 tokens on Llama-2-7B and 400,305 on Qwen2.5-32B. On Qwen2.5-32B (3×A6000, micro-batch 3 per GPU, gradient accumulation 9, paged 8-bit AdamW), one Commonsense-170k epoch of SALTQ at INT2 with the packed path takes 10.5 h at 3.89 samples/s with a peak training memory of 38.89 GiB per GPU; the BF16-buffer path does not fit in memory on this hardware.

Table 19: One-time steps before training. Second-moment statistics were re-measured by streaming one layer at a time on a single RTX 6000 Ada; the training runs collect them with a whole-model forward pass (Llama-2-7B batch 4; Qwen2.5-32B sharded over three GPUs, batch 1). GPTQ quantizes the layers one after another. Training-run times are those of the INT3 runs; the INT2 runs take about as long.

Step	Llama-2-7B	Qwen2.5-32B
Second-moment statistics, layer streaming	2 min 06 s	8 min 31 s
Second-moment statistics, training run	6 min 54 s	29 min 56 s
AutoSegment partition ($S = 1, \ldots, 4$)	0.06–0.07 s	0.32–0.34 s
Salient-channel selection	0.01–0.02 s	0.02–0.03 s
GPTQ initialization, training run	12 min 05 s	1 h 01 min 13 s

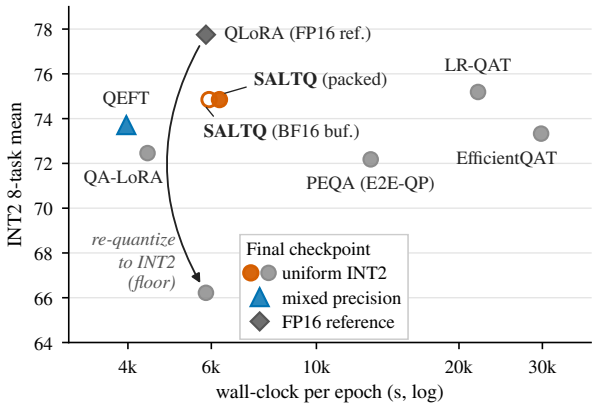

Figure 7: Training cost and accuracy. Horizontal axis: wall-clock time of one Commonsense-170k epoch (log scale); vertical axis: INT2 eight-task mean. Each point is annotated with the representation of its final checkpoint (uniform INT2; mixed precision, i.e. INT2 plus 256 FP16 columns; FP16 reference, merged and not re-quantized); the arrow from the QLoRA reference leads to its INT2 re-quantization (floor). EfficientQAT: Block-AP followed by E2E-QP, with the time of both stages.

Other tasks. Training cost at INT2 $g32$ on 4×A100. MetaMathQA, one epoch, wall-clock and throughput: QEFT 16840 s / 23.45, QA-LoRA 18610 / 21.22, SALTQ 21600 / 18.29, QLoRA 24030 / 16.44. WikiText-2, one epoch: QEFT 205.1 / 13.68, QA-LoRA 229.3 / 12.24, SALTQ 275.6 / 10.19, QLoRA 300.2 / 9.35; three epochs: QEFT 610.7 / 13.79, QA-LoRA 682.7 / 12.34, SALTQ 821.8 / 10.25, QLoRA 888.0 / 9.48.

E.8 Robustness over Training Seeds

Table 2 reports seed 0. Seeds 1 and 2 repeat the whole pipeline with another training seed, including the QLoRA reference and its GPTQ re-quantization, and Table 20 gives the mean and standard deviation over the three runs. Table 21 compares SALTQ with the strongest of QEFT, QA-LoRA and PEQA within each seed, which removes the variation that the methods share through the reference and the calibration. At INT2, SALTQ leads in every seed on both backbones, on the commonsense average by at least 0.64 points and on GSM8K by at least 2.04 points, and the mean margins are larger than the standard deviations of the methods involved. At INT3 it leads on the commonsense average in every seed and stays at the level of the FP16 reference on GSM8K, where the strongest baseline is within the seed-to-seed variation on both backbones; this is why the main text ranks the methods at INT3 only on the commonsense average.

Table 20: Three training seeds: mean ± standard deviation of the eight-task commonsense average (CS-Avg), GSM8K and MATH, same protocol as Table 2. The FP16 reference is shared by both bit-widths. Best mean per block among the compared methods in bold.

	INT3 $g64$			INT2 $g32$		
Method	CS-Avg	GSM8K	MATH	CS-Avg	GSM8K	MATH
Llama-2-7B						
FP16 reference	78.08 ± 0.32	57.42 ± 0.68	10.63 ± 0.44	78.08 ± 0.32	57.42 ± 0.68	10.63 ± 0.44
GPTQ	76.26 ± 0.18	50.77 ± 2.02	8.38 ± 0.42	66.68 ± 0.44	21.00 ± 0.99	2.19 ± 0.13
QEFT (b+16)	77.67 ± 0.17	55.72 ± 1.38	9.51 ± 0.11	73.73 ± 0.33	48.65 ± 0.58	7.71 ± 0.39
QA-LoRA	77.65 ± 0.39	$\mathbf{57.29 \pm 0.19}$	8.55 ± 0.08	72.12 ± 0.46	53.22 ± 0.99	10.49 ± 0.39
PEQA	77.64 ± 0.15	56.53 ± 2.48	8.51 ± 0.34	72.42 ± 0.33	52.74 ± 0.05	10.64 ± 0.05
SALTQ	$\mathbf{78.61 \pm 0.30}$	57.19 ± 1.14	$\mathbf{12.41 \pm 0.32}$	$\mathbf{74.73 \pm 0.44}$	$\mathbf{56.40 \pm 1.56}$	$\mathbf{11.46 \pm 0.61}$
Qwen2.5-32B						
FP16 reference	90.61 ± 0.23	92.04 ± 0.46	64.50 ± 0.80	90.61 ± 0.23	92.04 ± 0.46	64.50 ± 0.80
GPTQ	89.93 ± 0.29	87.97 ± 0.04	60.99 ± 0.48	85.21 ± 0.25	61.51 ± 1.94	33.81 ± 0.45
QEFT (b+16)	90.18 ± 0.06	91.48 ± 0.80	63.53 ± 0.52	88.58 ± 0.07	82.61 ± 0.71	51.35 ± 0.50
QA-LoRA	90.01 ± 0.19	90.93 ± 0.68	62.96 ± 0.19	87.64 ± 0.24	83.32 ± 0.27	54.53 ± 0.39
PEQA	90.15 ± 0.38	$\mathbf{92.19 \pm 0.30}$	63.48 ± 0.31	87.58 ± 0.12	83.19 ± 0.56	53.42 ± 0.62
SALTQ	$\mathbf{90.77 \pm 0.06}$	91.99 ± 0.25	$\mathbf{64.10 \pm 0.24}$	$\mathbf{89.28 \pm 0.06}$	$\mathbf{89.38 \pm 0.59}$	$\mathbf{57.23 \pm 0.34}$

Table 21: Margin of SALTQ over the strongest baseline in each seed (the best of QEFT, QA-LoRA and PEQA in that seed): mean and minimum over the three seeds, and wins / ties / losses.

	CS-Avg			GSM8K			MATH		
Setting	Mean	Min	W/T/L	Mean	Min	W/T/L	Mean	Min	W/T/L
Llama-2-7B, INT3 $g64$	$+0.80$	$+0.26$	3/0/0	-0.56	-2.80	1/0/2	$+2.90$	$+2.64$	3/0/0
Llama-2-7B, INT2 $g32$	$+1.01$	$+0.84$	3/0/0	$+3.13$	$+2.04$	3/0/0	$+0.79$	$+0.04$	3/0/0
Qwen2.5-32B, INT3 $g64$	$+0.49$	$+0.30$	3/0/0	-0.23	-0.68	0/2/1	$+0.35$	$+0.14$	3/0/0
Qwen2.5-32B, INT2 $g32$	$+0.70$	$+0.64$	3/0/0	$+5.86$	$+5.08$	3/0/0	$+2.69$	$+2.50$	3/0/0