ADAPTER-FREE PARAMETER-EFFICIENT FINE-TUNING WITH TINY ROUTERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-experts fine-tuning (MoE-PEFT) routes tokens among experts that are newly trained or that span every singular component of the pretrained weights, so it adds computation on top of the frozen backbone. Can it instead use the pretrained model's own spectral components as experts, selecting only some of them for each token and learning how strongly to weight them? The frozen model already holds much of what adaptation needs, but a strict per-token budget over its spectral components does not train when candidates are ranked by the router's gate logit, which encodes only how far a gate departs from one, not how much its group contributes. Ranking them by spectral energy makes the budget trainable (untrained, it keeps the base accuracy where random selection scores zero), and gates that can amplify the selected groups improve its accuracy. **Execution Tuning** builds on these findings: it decomposes each MLP weight matrix into frozen spectral groups that exactly reconstruct it and trains only a tiny router, 0.055% of Qwen3-8B's parameters, that learns token-conditioned magnitudes over an energy-selected support. Against MoE-PEFT baselines on six LLMs and six tasks, it trains 23–100$\times$ fewer parameters than MixLoRA and MoORE (with an 11–25% larger frozen bank), counts 1.5–1.7$\times$ fewer prefill FLOPs than the cheapest baseline, fewer even than the dense model, and runs faster than all of them, partly through implementation efficiency. It stays within 2 points of the strongest baseline on most model–task pairs, although MixLoRA remains more accurate on three quarters of them, and in single runs it also applies to vision–language, vision, and speech models.

1 INTRODUCTION

Mixture-of-experts parameter-efficient fine-tuning (MoE-PEFT) adapts a frozen model by routing tokens among experts. MixLoRA, AdaMoLE, and PESC route among newly trained low-rank experts (Li et al., 2024; Liu & Luo, 2024; Wu et al., 2024), Sparse Upcycling trains copies of the dense MLPs (Komatsuzaki et al., 2023), and MoORE turns every singular component of the pretrained weights into an expert under a learned router (Yuan et al., 2025). In each case the model executes at least the full backbone, so adaptation adds computation: every such baseline in our experiments exceeds the FLOPs of the dense MLP (Section 5). This raises our central question: **Can an MoE-PEFT method adapt by executing only part of the pretrained computation and learning how strongly to weight it, without training new experts?**

There is reason to expect that it can. Frozen LLMs from three families reach 93–98% Pass@64 on GSM8K although their Pass@1 is 17.34–57.22%, so for most problems the unadapted model can already generate a correct solution (Section 2.1). Within MoORE, an SVD-based router architecture (Yuan et al., 2025), training only the router over frozen spectral experts raises Llama-3.1-8B on GSM8K from 9.02% to 55.27%, and also training the experts adds at most 0.61 points (one seed; Table 1b).

Dense routing, however, still executes every component, and making it sparse poses a different problem on a frozen spectral bank than for trainable adapters. For trainable LoRA experts, imposing top-one routing from the first step collapses accuracy, and a gradual dense-to-sparse path recovers 40.62 of the lost points (Section 2.2). On a frozen spectral bank, no schedule we tried helps: with candidates ranked by the router's own gate logit, which is zero at initialization and later encodes only the

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.

gate's departure from one, a strict per-token 25% budget stays below 1.1% on Qwen3-8B/GSM8K whether it is imposed from the first step, annealed, or applied only at evaluation (Section 5.2).

In our experiments, energy ranking makes a strict per-token budget over frozen spectral components trainable, and amplifying gates make it accurate. First, candidates are ranked by the spectral energy of each group's input coordinates rather than by the gate logit. The energy ranking is informative before any training: untrained energy top-25% selection retains the unadapted accuracy of Qwen3-8B on GSM8K (68.69% versus 66.11%, flexible extraction), and of Llama-3.1-8B on MATH (1.14% versus 1.00%). Second, the selected groups are rescaled by unbounded multiplicative gates. In single runs on Qwen3-8B/GSM8K under the same budget, binary gates collapse to 6.52%, gates clipped to $[0, 1]$ lose 6.8 points, and in earlier runs 28.5% of the selected gates exceed one. Finer groups further make aggressive sparsity far less destructive (Section 5.2). **Execution Tuning** combines these ingredients (Section 4). It decomposes each MLP weight matrix into frozen spectral groups that exactly reconstruct it and trains only a router that learns token-conditioned, non-negative magnitudes over an energy-selected top-k support and assigns exact zeros to the rest. The router is a tiny network, 4.48M parameters on Qwen3-8B, where a standard linear router would need 113M. On the checkpoint we analyzed, the support stays close to the energy ranking and nearly fixed across tokens, and after training it can be frozen: fixing each projection's most frequent groups keeps flexible-extraction accuracy but loses 18.0 strict-format points, whereas token-averaged magnitudes cost 16.8 and 79.7 points (Section 5.2).

Against MoE-based fine-tuning, Execution Tuning trains 0.055% of Qwen3-8B's parameters, 23–$59\times$ fewer than MixLoRA and 70–$100\times$ fewer than MoORE, and it is the only compared MoE-PEFT method that does not add prefill computation to the backbone: its counted prefill MLP cost is 0.59–$0.78\times$ the dense MLP. In prefill, its decoder block counts 1.5–$1.7\times$ fewer FLOPs than the cheapest MoE-based baseline, and a fused implementation runs 1.45–$2.52\times$ faster than the fastest, eagerly run baseline, partly through implementation efficiency; its prefill decoder block also counts 1.2–$1.6\times$ fewer FLOPs than the dense model. In accuracy, across 36 model–task pairs (six LLMs, six tasks), it stays within 2 points of the best baseline in 26, above MoORE in 29 and Sparse Upcycling in 23, and above MixLoRA in 9; on vision–language, vision, and speech models it trains 0.04–0.51% of the base parameters and trails the best baseline by 0.3–3.2 points (Section 5).

We do not claim that SVD experts, ReLU routing, dense-to-sparse schedules, or adaptation by sparsification are individually new. Relative to SVF and MoORE (Sun et al., 2025; Yuan et al., 2025), which rescale every singular component and execute the full backbone, Execution Tuning combines RaNA's energy criterion (Garcia et al., 2025) with token-conditioned scaling of singular components; our contribution is to show what a strict per-token budget over a frozen, reconstructive spectral bank needs in order to train, and to measure what it costs. Dense-cost PEFT such as LoRA is a secondary reference (Appendices F and G).

Our contributions are:

1. We show that a strict per-token budget over frozen spectral components is made trainable by ranking candidates with a contribution-aware spectral-energy prior rather than by the gate logit alone (Section 5.2). Spectral energy is such a ranking: untrained, its top-64 keeps the unadapted accuracy of Qwen3-8B on GSM8K (68.69%) where a random 64 scores 0%, and trained, the 25% budget reaches 80.44%, whereas the gate logit, which encodes only $\alpha_g - 1$, stays below 1.1% under every tested schedule; in our single-seed gate ablation, binary gates collapse and gates bounded by one lose 6.8 points; and finer groups tolerate more sparsity. Sparsity without a strict per-token budget is easier: a PI-controller variant with neither top-k nor energy prior trains under every tested schedule, but it controls activity only on average, whereas the strict budget bounds the groups selected for every token.
2. We introduce Execution Tuning, an adapter-free MoE-PEFT method (no new expert or adapter weights are trained) whose experts are frozen spectral groups that exactly reconstruct each MLP matrix and whose only trained module is a bottleneck router over the spectral coordinates the input factor already computes: 4.48M parameters on Qwen3-8B, $25\times$ fewer than a linear router (Section 4).
3. Against MoE-based fine-tuning, Execution Tuning trains 23–$100\times$ fewer parameters than MixLoRA and MoORE on six LLMs (4–$93\times$ on six non-LLM models) and, in LLM prefill, counts 1.5–$1.7\times$ fewer decoder-block FLOPs than the cheapest baseline and runs 1.45–

Table 1: Motivating findings on GSM8K (accuracy, %). (a) Frozen models: Pass@1 (mean accuracy over the samples) and empirical Pass@64 (64 samples at temperature 0.8, top-p 0.95; 1,319 problems). (b) One MoORE parameterization on Llama-3.1-8B (seed 42, greedy, 1,319 problems) with frozen or trainable experts under static, token, or token-plus-task routing. (c) MixLoRA with 16 trainable rank-one LoRA experts on a frozen Llama-3.1-8B (single run, 256-problem development split): dense routing and two paths to top-one routing (experts per token). Protocols and paired tests: Appendices C and D.

(a) Accessible capability

Model	Pass@1	Pass@64
Gemma-2-9B	19.32	93.48
Llama-3.1-8B	17.34	93.10
Qwen3-8B	57.22	98.41

(b) Frozen experts suffice (MoORE)

Experts	Static	Token	Token+task
Frozen	9.02	54.36	55.27
Trainable	40.11	54.97	53.30

(c) Path to top-one (MixLoRA)

Path	Experts	Acc.
Soft (dense)	16	53.52
Hard top-one	1	4.69
Controlled	1	45.31

$2.52\times$ faster than the fastest, partly through implementation efficiency, while staying within 2 points of the best baseline in 26 of 36 model–task pairs (Section 5).

2 Motivating Findings

Two observations shape Execution Tuning (Table 1): frozen pretrained components can carry adaptation when a router learns to use them, and making that routing sparse is an optimization problem whose solution differs between trainable and frozen experts.

2.1 Finding 1: Accessible Capability and Frozen-Component Adaptation

On GSM8K (Cobbe et al., 2021), frozen Gemma-2-9B, Llama-3.1-8B, and Qwen3-8B solve 93–98% of problems in at least one of 64 samples (coverage, as in repeated-sampling studies (Brown et al., 2024; Yue et al., 2025)), although their Pass@1 is 17–57% (Table 1a).

Within one MoORE parameterization (Yuan et al., 2025) on Llama-3.1-8B (Table 1b), learning the router over frozen experts raises accuracy from 9.02% (static) to 54.36% (token routing) and 55.27% (token-plus-task routing), and also training the experts changes it by $+0.61$ and -1.97 points (one seed; paired exact McNemar $p = 0.68$ and 0.078). Details are in Appendix C.

2.2 Finding 2: For Trainable Experts, the Path to Sparsity Matters

In the official MixLoRA implementation (Li et al., 2024), with a frozen Llama-3.1-8B and 16 trainable rank-one LoRA experts, routing to one expert from the first update (Hard) reaches 4.69%, whereas reaching the same top-one rule gradually from dense routing (Controlled) reaches 45.31%, against 53.52% for dense routing (Table 1c). There, sparsification removes adapter deltas on top of an always-executed backbone; on the frozen spectral bank it truncates the backbone itself, and this lesson does not carry over: a gradual schedule does not rescue a strict mask ranked by the gate logit (Section 5.2).

3 Related Work

Mixture-of-experts routing. Sparsely gated MoE and Switch Transformers condition computation on the token while training expert networks (Shazeer et al., 2017; Fedus et al., 2022). ReMoE replaces fixed top-k gating with ReLU routing and adaptive sparsity regularization (Wang et al., 2025d). Dense-to-sparse gating is also established: the DTS-Gate of EvoMoE starts dense and becomes sparser during training, and DS-MoE trains all experts densely and routes sparsely at inference (Nie et al., 2021; Pan et al., 2024). Thus, neither nonnegative sparse routing, controlling average activity, nor a dense-to-sparse transition is our novelty. Our setting fixes the pretrained spectral transformations and adapts their use under downstream supervision.

PEFT and mixtures of adapters. Adapters, prefix tuning, LoRA, and $(IA)^3$ train small added modules or scaling vectors on a frozen backbone (Houlsby et al., 2019; Li & Liang, 2021; Hu

et al., 2022; Liu et al., 2022). AdaLoRA adapts rank allocation, DoRA decomposes updates into magnitude and direction, PiSSA and MiLoRA initialize adapters from principal and minor singular components, KaSA rescales singular values by task relevance, and CorDA orients the decomposition with task context (Zhang et al., 2023; Liu et al., 2024; Meng et al., 2024; Wang et al., 2025b;a; Yang et al., 2024). MixLoRA, AdaMoLE, and PESC combine efficient adapters with expert routing (Li et al., 2024; Liu & Luo, 2024; Wu et al., 2024), and GateRA modulates PEFT updates per token (Ou et al., 2026). SMoRA routes among LoRA ranks, SoRA prunes adapter ranks, VeRA trains only scaling vectors, and Router-Tuning trains a small router that skips whole layers (Zhao et al., 2025; Ding et al., 2023; Kopiczko et al., 2024; He et al., 2025). Frozen-adapter composition is also established by AdapterFusion, X-LoRA, and LoRA-Flow (Pfeiffer et al., 2021; Buehler & Buehler, 2024; Wang et al., 2024). Execution Tuning instead builds its frozen bank from the base operators.

Adaptation in the singular basis of pretrained weights. Several methods adapt a model by changing the contribution of its own singular components. With $W = U\,\mathrm{diag}(\sigma)V^\top$, SVF, the fine-tuning stage of Transformer^2, learns a per-task vector z_t and uses $U\,\mathrm{diag}(\sigma \odot z_t)V^\top$ (Sun et al., 2025); SVFT trains sparse coefficients M in $U(\mathrm{diag}(\sigma) + M)V^\top$ (Lingam et al., 2024); Spectral Adapter fine-tunes the top spectral components (Zhang & Pilanci, 2024); and MoORE, designed to reduce conflict and forgetting in multi-task adaptation, computes $y = U\,\mathrm{diag}(\sigma + g(x,t))\,V^\top Hx$ with a linear input- and task-conditioned router g and learned Householder reflections H (Yuan et al., 2025). Mo-SARA adds to the full weight a truncated-SVD term whose singular values are mixed by a router that, like ours, reads the spectral coordinates through a narrow bottleneck (Gu et al., 2024). In the same notation, Execution Tuning computes $y = U\,\mathrm{diag}(\sigma \odot \bar{\alpha}(x))\,V^\top x$ (Equation 3), where $\bar{\alpha}$ expands group gates to columns. Table 6 (Appendix A.8) places the four side by side. SVF, SVFT, and MoORE rescale all components and Mo-SARA adds to the full weight, so none of them reduces backbone computation. Execution Tuning addresses a different question, which pretrained computation to skip: an energy-selected, near-static support decides what executes, and learned magnitudes carry the adaptation. Its top-k mask gives exact zeros, and a group that is zero for every token in a batch can be dropped from the output-factor product; an additive coefficient $\sigma_d + g_d$ vanishes only by exact cancellation. Groups rather than rank-one components keep the router output at G rather than r scores per token. The router reads $z = V^\top x$; the energy prior itself adds only $O(r)$ work, but it requires the full z, which keeps the input-factor product dense (Section 4).

Dense-to-MoE upcycling and adaptation by sparsification. Sparse Upcycling initializes experts from a dense checkpoint, while Drop-Upcycling uses partial reinitialization to promote differentiation (Komatsuzaki et al., 2023; Nakamura et al., 2025). Activation-aware factorization is established in ASVD and SVD-LLM; GOAT combines spectral structure with MoE low-rank adaptation (Yuan et al., 2023; Wang et al., 2025c; Fan et al., 2025). Our energy criterion is RaNA's: RaNA decomposes linear layers into low-rank factors and masks rank components whose squared contribution $(Bx)_i^2$ is small to reduce inference FLOPs, with a threshold set to meet an average FLOP budget, and then recovers quality by fine-tuning the adapted model with LoRA (Garcia et al., 2025); E_g aggregates that criterion over a group. What differs is the use: Execution Tuning adapts a frozen, exactly reconstructive bank to a downstream task with a strict per-token budget, learning only token-conditioned magnitudes over the energy-selected support and no adapter weights. SSMoE interpolates a weight-spectral score with a learned router score, but to route among the experts of pretrained sparse MoEs without training (Do et al., 2026). FineGates (Svirsky et al., 2026) adapts frozen weights by learning stochastic gates that sparsify their rows and columns during fine-tuning; its gates define a fixed structured pruning pattern learned during fine-tuning, whereas our support starts from the energy ranking of the pretrained operator and our gates are unbounded learned magnitudes. APT prunes while tuning, and LoRAPrune and LLM-Pruner recover structurally pruned models with LoRA (Zhao et al., 2024; Zhang et al., 2024; Ma et al., 2023), fixing the executed structure per model.

Contextual activation sparsity. Deja Vu, TEAL, CATS, R-Sparse, LTE, and D2DMoE reduce inference cost by activating fewer neurons, channels, or structured groups (Liu et al., 2023; 2025; Lee et al., 2024; Zhang et al., 2025; Zheng et al., 2024; Szatkowski et al., 2024), and Polar Sparsity studies how contextual sparsity scales to batched inference (Shrestha et al., 2025); they aim to preserve quality, whereas Execution Tuning adapts through learned magnitudes on an energy-selected support.

4 METHOD

4.1 NOTATION AND LEARNING GOAL

Let $f_{\theta_0,\phi}$ be a pretrained model with frozen backbone parameters θ_0 and trainable router parameters ϕ. The causal-LM configuration replaces only MLP gate, up, and down projections. Attention, embeddings, normalization, and the language-model head remain unchanged. We use a scheduled cardinality k_t, ending at $k^\star = p^\star G$. Exactly k_t candidates are selected when $0 < k_t < G$, but ReLU magnitudes may zero some of them, so the number of nonzero groups is at most k_t. A PI-controller variant without a top-k is described in Section 4.4.

4.2 A RECONSTRUCTIVE FROZEN SPECTRAL BANK

For $W \in \mathbb{R}^{d_o \times d_i}$, take its full thin SVD,

$$W = U\Sigma V^\top, \qquad A = U\Sigma, \qquad B = V, \qquad W = AB^\top, \tag{1}$$

where $r = \min(d_i, d_o)$ columns are retained. The factors depend only on the pretrained weights: no calibration data, activation statistics, or continued language-model optimization are used to construct them; one bank serves all tasks, and only the router is task-specific. Partition the columns into disjoint groups $J_1, \ldots, J_G$ and define $W_g = A_{J_g} B_{J_g}^\top$. Groups are contiguous blocks of the σ-ordered spectrum with equal width, r/G columns each; we store the column-to-group map with the checkpoint.

4.3 TINY ENERGY-PRIOR ROUTING AND LEARNED MAGNITUDES

Let $z = B^\top x$ and $\ell = h_\phi(z)$, where h_ϕ has affine widths $r \to 8 \to 32 \to G$, GELU activations, and biases. The router combines a spectral-energy prior with learned routing scores:

$$E_g(x) = \sum_{j \in J_g} \sigma_j^2 z_j^2, \qquad q_g(x) = s\,\ell_g + \beta \log \max(E_g, \epsilon_E), \tag{2}$$

$$S_t(x) = \mathrm{TopK}\big(q(x), k_t\big), \qquad \alpha_g(x) = \mathbf{1}[g \in S_t(x)]\,\mathrm{ReLU}(1 + s\ell_g), \tag{3}$$

$$y(x) = \sum_g \alpha_g(x)\, A_{J_g} z_{J_g} + b.$$

Here β is a trainable scalar per projection, initialized to one, and s is a stored logit scale. The prior selects candidates; it does not replace their learned magnitudes. Training centers log-energy, which leaves the ideal ranking unchanged (Appendix I.5).

A tiny router. The routing problem is large: $G = 256$ scores per token for each of the 108 MLP matrices of Qwen3-8B. A standard linear router over z would need rG weights per projection, 113M parameters in total, about MixLoRA's entire trainable budget on this model (118M). We instead use a tiny network built on three choices. (i) It reads $z = B^\top x$, which the output path computes anyway, so routing adds no product with x and its input is already expressed in the coordinates that define the groups. (ii) An eight-unit bottleneck, $r \to 8 \to 32 \to G$, makes the input layer $8r$ weights, so each projection has $8r + 8 + 8 \cdot 32 + 32 + 32G + G + 1 = 41{,}513$ parameters at $r = 4096$, or 4,483,404 in total: $25\times$ fewer than the linear router and 0.055% of the model. (iii) The output layer is zero-initialized, so $\ell = 0$ and, with $k_0 = G$, every gate equals one and the model is exactly the pretrained one at step one.

Why rank candidates by energy. The energy of a group is the squared norm of its contribution to the output, $E_g(x) = \|A_{J_g} z_{J_g}\|^2 = \|W_g x\|^2$, because the columns of U are orthonormal. With unit gates, dropping a set of groups changes the projection output by exactly the energy of the dropped groups (Proposition 2), so the energy top-k is the per-token subset that deviates least from the pretrained projection (Proposition 4), though not necessarily the most accurate end to end (Section 5.2). With learned gates, dropping a set of selected groups changes the output by $\sum_g \alpha_g^2 E_g$ over the dropped groups, by the same argument. Because $s\ell_g = \alpha_g - 1$ for a selected group with

a positive gate, the score is $q_g = (\alpha_g - 1) + \beta \log E_g$, which, for gates near one and $\beta \approx 1$, approximately ranks groups by $\alpha_g E_g$, a proxy for their gated contribution $\alpha_g^2 E_g$; the learned score can therefore reorder candidates only as far as its gate departs from one.

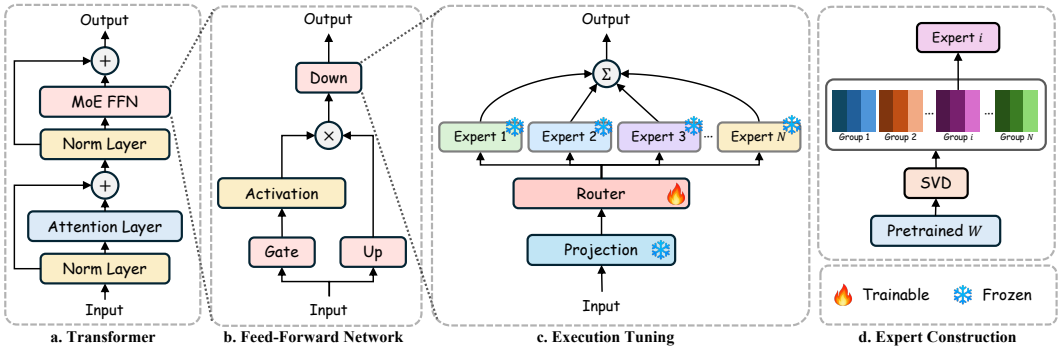


Figure 1: Execution Tuning in a transformer MLP. (a) Only the feed-forward block is replaced. (b) Its gate, up, and down projections are factorized separately. (c) One projection: a frozen input map, the tiny trainable router, which keeps the top-k groups ranked with the energy prior, and a sum of frozen experts. (d) Expert i is group i of the SVD of W. Flame: trainable; snowflake: frozen.

In Figure 1, the input projection still runs at full rank, so a zero gate is not a skipped multiplication. Keeping all groups active after training does not preserve W, because gates are neither normalized nor upper-bounded. Non-causal-LM task heads are accounted for separately.

4.4 Candidate Budget Schedule and PI-Controller Variant

The candidate budget follows the cubic sparsity schedule of gradual magnitude pruning (Zhu & Gupta, 2017), applied to the number of candidate groups rather than to individual weights:

$$u_t = \mathrm{clip}\left(\frac{t}{\max(1, 0.7T)}, 0, 1\right), \qquad k_t = \mathrm{round}\left[k^\star + (G - k^\star)(1 - u_t)^3\right], \tag{4}$$

clipped to $[k^\star, G]$. The default $G = 256$, $k^\star = 64$ reaches its final candidate budget by 70% of optimizer steps and holds it thereafter. The schedule begins decreasing immediately, and the all-on (100% density) setting bypasses selection.

The objective is the supervised task loss plus a load-balancing term of coefficient 0.01; set membership is not differentiated, so gate magnitudes and soft routing propensity carry gradients (Appendix B.2).

PI-controller variant. To train sparsity without a per-token budget, in the spirit of ReMoE's adaptive regularization (Wang et al., 2025d), this variant drops the top-k and the energy prior, applies $g_g(x) = \mathrm{ReLU}(1 + \ell_g(x))$ to every group, and steers the fraction a_t of positive gates toward a scheduled target p_t with a nonnegative multiplier:

$$\lambda_{t+1} = \left[\lambda_t + \eta_{\text{leak}}\,(a_t - p_t)\right]_+, \qquad L_t = L_{\text{task}} + \lambda_t\, L_{\text{density}}(g, p_t). \tag{5}$$

Activity is controlled only on average over tokens; the target schedule, which replaces Equation 4, and the gains are in Table 8.

4.5 Logical Activity versus Executed Work

For active rank $r_{\text{act}}(x) = \sum_g |J_g|\,\mathbf{1}[\alpha_g(x) > 0]$, $2r_{\text{act}}(d_i + d_o)$ counts the two selected factor products as if both could skip inactive columns. The implemented backend computes the full input factor, then fuses routing, selection, and scaling; large-prefill output GEMMs can drop groups unused by every token in the batch (a union near 25%, as the support is nearly static; Appendix I). Decode retains a dense masked output GEMM. Because the energy prior needs the full latent, the input factor

Table 2: What makes a strict 25% budget trainable (accuracy, %). (a) Candidate ranking on Qwen3-8B/GSM8K: the gate-logit row spans six single-run schedules (Table 33); the energy row is the main recipe (three seeds, Table 13). (b) Gate form under the same budget, seed 0; only the map from the gate logit to the gate changes (Appendix J.2). (c) Strict-format accuracy of the reported seed-0 checkpoint, re-evaluated with the eager router (80.36; 80.44 with the fused kernel, as in Table 14), with a fixed support or mean gates (flexible extraction in Table 32). (d) Group count G at fixed density on Qwen3-8B/GSM8K, three seeds (Table 36). The unadapted Qwen3-8B scores 66.11% on GSM8K. Shaded rows are the main recipe.

(a) Candidate ranking

Ranking	Untrained	Trained
Gate logit only	0.00	0.23–0.83
Energy prior	68.69	**80.44**

(b) Gate form

Gate on selected groups	Qwen / GSM8K	Llama / MATH
ReLU, $\mathrm{ReLU}(1 + s\ell)$	**80.44**	**17.82**
Binary, trained selection	6.52	1.04
Clip to $[0, 1]$	73.62	13.12
Energy top-k, untrained	68.69	1.14

(c) Trained checkpoint, evaluation only

Support and gates at evaluation	Strict format
Per-token support, learned gates	80.36
Fixed frequent support, learned gates	62.40
Fixed top-σ support, learned gates	13.80
Per-token support, mean gates	0.68

(d) Group count G

Density	32	64	128	256
25%	68.01	75.82	79.38	**80.44**
12.5%	10.24	30.63	56.41	**68.46**

alone costs 0.42–$0.58\times$ the dense MLP, and at 25% density `down_proj` costs 1.03–$1.09\times$ its dense counterpart; the implemented count for the MLP is 0.59–$0.78\times$ dense when the output factor skips unselected groups (Table 5; closed forms in Proposition 9). A fixed per-projection support would let both factors skip, but imposed at evaluation it loses strict-format accuracy (Section 5.2).

5 Experiments and Results

5.1 Evaluation Scope and Comparisons

Our LLM evaluation covers six model instances from Qwen3, Llama, and Gemma families (Yang et al., 2025; Grattafiori et al., 2024; Gemma Team, 2024), with Sparse Upcycling, MixLoRA, MoORE, and Execution Tuning. The six tasks are GSM8K, MATH, MBPP, ARC-Challenge, BoolQ, and PIQA (Cobbe et al., 2021; Hendrycks et al., 2021; Austin et al., 2021; Clark et al., 2018; 2019; Bisk et al., 2020); MBPP reports Pass@1. Every model–method–task cell is trained independently on a single task, Execution Tuning with the main recipe of Section 4 (top-64 of 256 groups, cubic schedule, energy prior) and each baseline with its own recipe (Appendix E.2); every cell is trained with seeds 0, 1, and 42, and we report the mean and the standard deviation across seeds. For each run, the checkpoint with the lowest held-out validation loss is evaluated once on the official evaluation split (validation for BoolQ and PIQA, whose test labels are not public), which is never used for selection. Accuracy, FLOP counts, and timing come from the same trained checkpoints (seed 0 for timing), model configurations, and fused kernels, so every cost is paired with the model whose accuracy is reported; the only exception is Sparse Upcycling on Qwen3-14B and Llama-3.1-8B, timed at replacement interval 2 but trained at 4 (Table 18). The protocol is in Appendix E.2.

5.2 What Makes a Strict Per-Token Budget Trainable

Section 2.2 showed that, for trainable LoRA experts, a gradual path recovers most of the accuracy lost to hard top-one routing. We now ask what the frozen spectral bank needs (Table 2).

Gates that can amplify. Under the same recipe and budget, binary gates trained with a straight-through estimator collapse to 6.52% on Qwen3-8B/GSM8K and 1.04% on Llama-3.1-8B/MATH, and gates clipped to $[0, 1]$ lose 6.8 and 4.7 points relative to the ReLU gate (Table 2b). In this ablation, selection without learned magnitudes does not adapt the frozen bank; because Binary and Clip also free the selection score from the gate magnitude, part of their loss may come from selection drift. The share of selected ReLU gates that exceed one rises from 8.5% at 100% density to 28.5% at 25% on Qwen3-8B (earlier runs of the same configuration; Table 31), and clipping costs most at the 25% budget: 6.8 points, versus 1.9 at 100% and 0.15 at 50% (Table 30).

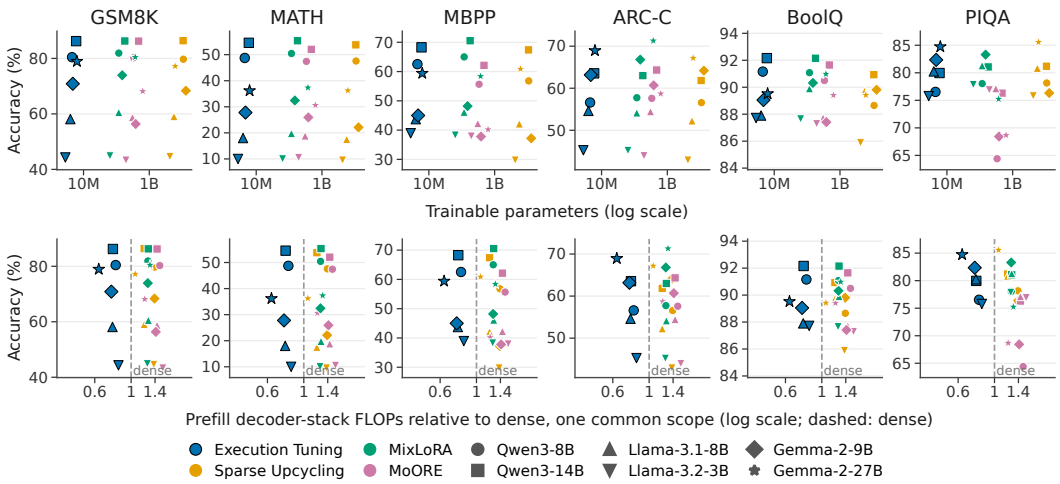


Figure 2: All 36 LLM model–task results against trainable parameters (top) and prefill decoder-stack FLOPs relative to the dense model under one common scope (bottom; Table 28); upper left is better, MBPP is Pass@1. Execution Tuning's count charges the full input factor and 25% of the output factor; Sparse Upcycling uses the replacement interval of its accuracy runs. Values with seed standard deviations: Table 13.

Energy ranking, not gate-logit ranking. With candidates ranked by the gate logit ℓ alone, a strict top-64 mask stays below 1.1% on Qwen3-8B/GSM8K whether it is imposed from the first step, annealed cubically, switched on after 50–80% of training, or applied only at evaluation (Table 2a), and it collapses on all three models and both tasks of Table 29. The same budget with the energy prior reaches 80.44% (three seeds). Logit-only ranking is a weak comparator by construction: the gate logit is zero for every group at initialization and, for a selected group, later encodes only $\alpha_g - 1$. The contrast therefore indicates that a strict budget becomes trainable once candidates are ranked by a contribution-aware prior rather than by the gate logit alone, which energy provides: untrained energy top-25% selection retains the unadapted accuracy on Qwen3-8B/GSM8K (68.69% versus 66.11%, flexible extraction; a random top-64 scores 0%), and on Llama-3.1-8B/MATH (1.14% versus 1.00%).

What the router learns. On a separately trained Qwen3-8B/GSM8K checkpoint, the learned top-64 agrees with the energy top-64 (Jaccard 0.967), β stays near one, and the groups selected over a 64×512 prefill batch cover 26.7% of all groups, so selection is essentially the energy top-k and nearly fixed across tokens; the router's learned contribution is the magnitudes. On the reported seed-0 checkpoint (Table 2c), strict-format accuracy falls from 80.36% with per-token support and learned gates to 62.40% when each projection's support is fixed to its 64 most frequent groups, 13.80% with a fixed top-σ support, and 0.68% when gates are replaced by their means over 512 training prompts. Under flexible extraction, the fixed frequent support keeps its accuracy (81.27%; Table 32). A fixed support thus keeps the extracted answers but not their format, which per-token selection and token-conditioned magnitudes maintain; in training, one global magnitude per group instead of token-conditioned ones raises held-out loss from 0.63 to 2.96 (Appendix J.2).

Schedule sensitivity. Without a strict per-token budget, sparsity is easier to train: the PI-controller variant (Section 4.4), with neither top-k nor energy prior, reaches 81.1–82.2% on Qwen3-8B/GSM8K whether its 25% target is held from the first step, annealed cubically, or switched on after 40–70% of training (single runs; about 21% realized density for the cubic arm; Table 34).

Finer groups tolerate more sparsity. On the three-seed Qwen3-8B/GSM8K grid, 25% density costs 1.52 points relative to 100% at $G = 256$, and 12.5% density costs 13.50 points (Table 36). At 25% density, raising G from 32 to 256 improves accuracy from 68.01% to 80.44%, and at 12.5% from 10.24% to 68.46% (Table 2d); the 100% column spans 0.60 points. Group count also changes

Table 3: Non-LLM models (single run; main recipe). Quality is accuracy (%) on ScienceQA and CIFAR-100 and word error rate (%, lower is better) on FLEURS; Train/base: trainable parameters, % of the base model (Upcycling trains MLP copies). Bold is best per row; Tables 22–23.

		Quality				Train/base (%)			
Task	Model	ET	Upcycling	MixLoRA	MoORE	ET	Upcycling	MixLoRA	MoORE
ScienceQA	Qwen2.5-VL-3B	93.89	**97.10**	97.01	91.98	**0.07**	259.40	0.63	5.02
	Qwen2.5-VL-7B	96.32	**97.90**	97.71	97.67	**0.04**	275.12	0.39	3.09
CIFAR-100	ViT-B	91.90	90.38	**92.66**	92.32	**0.51**	298.01	2.02	25.59
	ViT-L	93.48	88.20	**94.16**	93.90	**0.30**	299.22	1.49	19.28
FLEURS	Whisper-small	8.59	8.28	**7.45**	7.89	**0.30**	240.21	1.43	23.12
	Whisper-medium	6.42	6.28	**6.12**	**6.12**	**0.21**	258.06	1.21	19.51

router capacity (3.69M to 4.48M; Table 38), yet the 100% column stays within 0.60 points, suggesting that capacity alone does not explain the sparse-density gains.

5.3 LLM Quality at a Fraction of the Cost

Figure 2 places every model–task result against cost. Execution Tuning occupies the low-cost corner, with 23–59× fewer trainable parameters than MixLoRA and the only counted prefill MLP cost below the dense MLP. At that cost it stays within 2 points (gap at most 2.00) of the best baseline in 26 of the 36 model–task pairs and within 1 point in 13; it is above MoORE in 29 pairs and above Sparse Upcycling in 23, and it trails MixLoRA, the strongest baseline, in 27 (Table 13). The largest gaps are on Gemma-2-9B (4.64 points on MATH, 3.64 on ARC-C, 3.20 on MBPP, 3.06 on GSM8K). Two baseline effects qualify these counts: the one large win over MixLoRA (Gemma-2-27B/PIQA, $+9.49$) meets a MixLoRA cell far below its Gemma-2-9B score (75.24 vs. 83.30), and 12 of the 29 wins over MoORE are on Gemma. Seed SDs are at most 0.57 (one baseline cell: 2.74).

5.4 Beyond Causal Language Models

In single runs, the same trade-off holds for Qwen2.5-VL, ViT, and Whisper (Table 3; Appendix H). Execution Tuning trains 4–10× fewer parameters than MixLoRA and 50–93× fewer than MoORE; it trails the best baseline by 0.68–3.21 accuracy points or 0.30–1.14 WER points. In a separate single-seed protocol, it is within 0.25 points of LoRA ($r = 4$) (Appendices F, G).

5.5 Cost and Timing

In prefill, the counted Execution Tuning MLP is 0.59–$0.78\times$ the dense MLP (Table 5). At one consistent scope including attention, its prefill decoder block counts 1.5–$1.7\times$ fewer FLOPs than the cheapest baseline and 1.19–$1.58\times$ fewer than the dense model (Table 28). On one H200 (batch 64, prompt 512, 128 new tokens), fused prefill runs 1.45–$2.52\times$ faster than the fastest-prefill baseline and delivers 1.44–$3.34\times$ the end-to-end throughput of the highest-throughput one (Table 26). Table 27 adds the CUDA-graphed dense model as a reference: time to first token is 1.34–$1.59\times$ lower, beyond the counted FLOP ratio on five of six models. The baselines run eagerly, so part of these gains is implementation efficiency; with MoORE graph-captured, Execution Tuning still delivers 1.47–$1.50\times$ its throughput on Qwen3-8B, and it is the fastest MoE-PEFT method in every regime measured (Appendix I).

6 Discussion and Conclusion

With a strict per-token budget made trainable by energy ranking and accurate by amplifying gates and fine groups, a model adapts by reweighting energy-selected spectral components without adding prefill computation to the backbone, unlike the compared MoE-PEFT methods.

Limitations. The energy prior needs the full spectral latent, so the input factor is always computed and savings come only from the output factor in prefill; decode runs every output column and costs 1.13–$1.38\times$ the dense MLP (Proposition 9); the bank stores 11–25% more parameters; the fused kernel needs equal-width groups; and part of the speedups is implementation efficiency.

Reproducibility Statement

The appendix specifies the operator, router, budget schedule and controller, numerical identities, training and evaluation protocol, and implementation limitations. Main-campaign checkpoints are selected on held-out validation loss; the schedule ablations of Appendix J.3 use the final checkpoint without a holdout. Each table states its seed count. Accuracy, FLOP counts, and timing use the same trained checkpoints, configurations, and fused kernels (Section 5). Anonymized code, configurations, and evaluation scripts are provided as supplementary material and will be released publicly upon publication.

Ethics and Limitations of Use

This work changes pretrained model computation rather than establishing new safety guarantees. Changes in routing may alter behavior outside the evaluated tasks, including robustness, calibration, and safety. The present evaluations do not establish deployment safety, fairness, or preservation of pretrained safeguards. All datasets and models are used under their public licenses. No claims about private training-data membership or the causal origin of model knowledge are made from the capability-coverage analysis.

AI Use Statement

We used generative AI tools to assist literature discovery, source-code inspection, the exposition of mathematical derivations, manuscript revision, LaTeX preparation, table generation from our result files, and an internal reviewer-style audit. We did not use them to generate experimental results. We have reviewed all AI-assisted work, checked every derivation and every reported number against our result files, and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025. URL https://arxiv.org/abs/2502.13923.

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

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024. URL https://arxiv.org/abs/2407.21787.

Eric L. Buehler and Markus J. Buehler. X-LoRA: Mixture of low-rank adapter experts, a flexible framework for large language models with applications in protein mechanics and molecular design. *APL Machine Learning*, 2024. URL https://arxiv.org/abs/2402.07148.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

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 (NAACL)*, 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.

Alexis Conneau, Min Ma, Simran Khanuja, Yu Zhang, Vera Axelrod, Siddharth Dalmia, Jason Riesa, Clara Rivera, and Ankur Bapna. FLEURS: Few-shot learning evaluation of universal representations of speech. In *IEEE Spoken Language Technology Workshop (SLT)*, 2023. URL https://arxiv.org/abs/2205.12446.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://arxiv.org/abs/2311.11696.

Giang Do, Hung Le, and Truyen Tran. Eigenvectors of experts are training-free non-collapsing routers. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.30992.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.11929.

Chenghao Fan, Zhenyi Lu, Sichen Liu, Chengfeng Gu, Xiaoye Qu, Wei Wei, and Yu Cheng. Make LoRA great again: Boosting LoRA with adaptive singular values and mixture-of-experts optimization alignment. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2502.16894.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022. URL https://arxiv.org/abs/2101.03961.

Roberto Garcia, Jerry Liu, Daniel Sorvisto, and Sabri Eyuboglu. Adaptive rank allocation: Speeding up modern transformers with RaNA adapters. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2503.18216.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024. URL https://arxiv.org/abs/2408.00118.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Jihao Gu, Shuai Chen, Zelin Wang, Yibo Zhang, and Ping Gong. SARA: Singular-value based adaptive low-rank adaption. *arXiv preprint arXiv:2408.03290*, 2024. URL https://arxiv.org/abs/2408.03290.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2111.09543.

Shwai He, Tao Ge, Guoheng Sun, Bowei Tian, Xiaoyang Wang, and Dong Yu. Router-tuning: A simple and effective approach for enabling dynamic-depth in transformers. In *Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://arxiv.org/abs/2410.13184.

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*, 2021. URL https://arxiv.org/abs/2103.03874.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, 2019. URL https://arxiv.org/abs/1902.00751.

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://arxiv.org/abs/2106.09685.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2212.05055.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. VeRA: Vector-based random matrix adaptation. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.11454.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://cave.cs.toronto.edu/kriz/cifar.html.

Donghyun Lee, Je-Yong Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. CATS: Contextually-aware thresholding for sparsity in large language models. *arXiv preprint arXiv:2404.08763*, 2024.

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhengmao Ye, Zhiyuan Cheng, Yinghao Tang, Yan Zhang, Lei Duan, Jie Zuo, Cal Yang, and Mingjie Tang. MixLoRA: Enhancing large language models fine-tuning with LoRA-based mixture of experts. *arXiv preprint arXiv:2404.15159*, 2024. URL https://arxiv.org/abs/2404.15159.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics*, 2021. URL https://arxiv.org/abs/2101.00190.

Vijay Lingam, Atula Tejaswi, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Alex Dimakis, Eunsol Choi, Aleksandar Bojchevski, and Sujay Sanghavi. SVFT: Parameter-efficient fine-tuning with singular vectors. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.19597.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2205.05638.

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. Training-free activation sparsity in large language models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.14690.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.09353.

Zefang Liu and Jiahua Luo. AdaMoLE: Fine-tuning large language models with adaptive mixture of low-rank adaptation experts. In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2405.00361.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2310.17157.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2209.09513.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.11627.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2404.02948.

Taishi Nakamura, Takuya Akiba, Kazuki Fujii, Yusuke Oda, Rio Yokota, and Jun Suzuki. Drop-upcycling: Training sparse mixture of experts with partial re-initialization. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2502.19261.

Xiaonan Nie, Xupeng Miao, Shijie Cao, Lingxiao Ma, Qibin Liu, Jilong Xue, Youshan Miao, Yi Liu, Zhi Yang, and Bin Cui. EvoMoE: An evolutional mixture-of-experts training framework via dense-to-sparse gate. *arXiv preprint arXiv:2112.14397*, 2021. URL https://arxiv.org/abs/2112.14397.

Jie Ou, Shuaihong Jiang, Yingjun Du, and Cees G. M. Snoek. GateRA: Token-aware modulation for parameter-efficient fine-tuning. In *AAAI Conference on Artificial Intelligence*, 2026. URL https://arxiv.org/abs/2511.17582.

Bowen Pan, Yikang Shen, Haokun Liu, Mayank Mishra, Gaoyuan Zhang, Aude Oliva, Colin Raffel, and Rameswar Panda. Dense training, sparse inference: Rethinking training of mixture-of-experts language models. *arXiv preprint arXiv:2404.05567*, 2024. URL https://arxiv.org/abs/2404.05567.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2021. URL https://arxiv.org/abs/2005.00247.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2212.04356.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1701.06538.

Susav Shrestha, Brad Settlemyer, Nikoli Dryden, and Narasimha Reddy. Polar sparsity: High throughput batched LLM inferencing with scalable contextual sparsity. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2505.14884.

Qi Sun, Edoardo Cetin, and Yujin Tang. Transformer-squared: Self-adaptive LLMs. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2501.06252.

Jonathan Svirsky, Yehonathan Refael, and Ofir Lindenbaum. Train less, infer faster: Efficient model finetuning and compression via structured sparsity. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2026. URL https://arxiv.org/abs/2602.09169.

Filip Szatkowski, Bartosz Wójcik, Mikołaj Piórczyński, and Simone Scardapane. Exploiting activation sparsity with dense to dynamic-k mixture-of-experts conversion. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2310.04361.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1804.07461.

Fan Wang, Juyong Jiang, Chansung Park, Sunghun Kim, and Jing Tang. KaSA: Knowledge-aware singular-value adaptation of large language models. In *International Conference on Learning Representations*, 2025a. URL https://arxiv.org/abs/2412.06071.

Hanqing Wang, Bowen Ping, Shuo Wang, Xu Han, Yun Chen, Zhiyuan Liu, and Maosong Sun. LoRA-Flow: Dynamic LoRA fusion for large language models in generative tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2402.11455.

Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. MiLoRA: Harnessing minor singular components for parameter-efficient LLM finetuning. In *Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, 2025b. URL https://aclanthology.org/2025.naacl-long.248/.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *International Conference on Learning Representations*, 2025c. URL https://arxiv.org/abs/2403.07378.

Ziteng Wang, Jun Zhu, and Jianfei Chen. ReMoE: Fully differentiable mixture-of-experts with ReLU routing. In *International Conference on Learning Representations*, 2025d. URL https://arxiv.org/abs/2412.14711.

Haoyuan Wu, Haisheng Zheng, Zhuolun He, and Bei Yu. Parameter-efficient sparsity crafting from dense to mixture-of-experts for instruction tuning on general tasks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. URL https://arxiv.org/abs/2401.02731.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Yibo Yang, Xiaojie Li, Zhongzhu Zhou, Shuaiwen Leon Song, Jianlong Wu, Liqiang Nie, and Bernard Ghanem. CorDA: Context-oriented decomposition adaptation of large language models for task-aware parameter-efficient fine-tuning. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.05223.

Shen Yuan, Yin Zheng, Taifeng Wang, Binbin Liu, and Hongteng Xu. MoORE: SVD-based model MoE-ization for conflict- and oblivion-resistant multi-task adaptation. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2506.14436.

Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. ASVD: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*, 2023. URL https://arxiv.org/abs/2312.05821.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2504.13837.

Fangzhao Zhang and Mert Pilanci. Spectral adapter: Fine-tuning in spectral space. *arXiv preprint arXiv:2405.13952*, 2024.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. LoRAPrune: Structured pruning meets low-rank parameter-efficient fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. URL https://arxiv.org/abs/2305.18403.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2303.10512.

Zhenyu Zhang, Zechun Liu, Yuandong Tian, Harshit Khaitan, Zhangyang Wang, and Steven Li. R-Sparse: Rank-aware activation sparsity for efficient LLM inference. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9VMW4iXfKt.

Bowen Zhao, Hannaneh Hajishirzi, and Qingqing Cao. APT: Adaptive pruning and tuning pretrained language models for efficient training and inference. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2401.12200.

Ziyu Zhao, Yixiao Zhou, Zhi Zhang, Didi Zhu, Tao Shen, Zexi Li, Jinluan Yang, Xuwu Wang, Jing Su, Kun Kuang, Zhongyu Wei, Fei Wu, and Yu Cheng. Each rank could be an expert: Single-ranked mixture of experts LoRA for multi-task learning. *arXiv preprint arXiv:2501.15103*, 2025. URL https://arxiv.org/abs/2501.15103.

Haizhong Zheng, Xiaoyan Bai, Xueshen Liu, Z. Morley Mao, Beidi Chen, Fan Lai, and Atul Prakash. Learn to be efficient: Build structured sparsity in large language models. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2402.06126.

Michael Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. URL https://arxiv.org/abs/1710.01878.

Appendix

The appendix is organized as follows. Appendix A collects the formal results with proofs: exact reconstruction (Proposition 1), the error identity (Proposition 2), optimality of energy top-k selection (Propositions 4 and 5), what the selection score ranks (Lemma 6), task gradients under a strict mask (Proposition 7), and the counted cost in closed form (Proposition 9).

- Appendix A: Theory: Reconstruction, Selection, and Cost (p. 16).
- Appendix B: Implementation and Complete Training Procedure (p. 20).
- Appendix C: Finding 1: Protocols, Controls, and Statistical Meaning (p. 22).
- Appendix D: Finding 2: An Independent Continuation Intervention (p. 24).
- Appendix E: Complete Six-Task Language-Model Results (p. 25).
- Appendix F: Dense-Cost PEFT on LLMs under a Separate Protocol (p. 28).
- Appendix G: GLUE: Encoder Transfer with Explicit Head and Target-Module Scope (p. 29).
- Appendix H: VLM, Vision, and Speech: Complete Results (p. 30).
- Appendix I: Efficiency: Formulas, Faithful Execution, and Measurements (p. 31).
- Appendix J: Method-Specific Ablations (p. 34).

A Theory: Reconstruction, Selection, and Cost

A.1 Notation and Factorization Conventions

Table 4 records the notation used below. Indices g, j, and t refer to groups, spectral columns, and optimizer steps, respectively; ℓ denotes router logits.

Table 4: Notation.

Symbol	Definition
$W \in \mathbb{R}^{d_o \times d_i}$	Frozen pretrained linear operator; b is its original bias.
U, Σ, V	Full thin SVD of W; $r = \min(d_i, d_o)$.
A, B	Factors $A = U\Sigma$, $B = V$, so $W = AB^\top$.
J_g, r_g	Disjoint column group and its actual rank $r_g = \lvert J_g \rvert$; q_g denotes a routing score.
$z, \alpha, \bar{\alpha}$	Basis coordinates $B^\top x$, group gates, and gates expanded to columns.
m, G, p_t	Number of adapted projections, groups per projection, and scheduled target activity.
a_t, r_{act}	Observed positive-gate fraction (PI-controller variant) and actual rank selected per token.
ϕ, λ_t	Trainable router weights/biases and prior scalars; non-gradient controller state (PI-controller variant).

The factorization is computed once per pretrained projection and uses no data. For rank-deficient W, the thin factorization can contain zero singular values. The algebra below still holds, but zero-energy columns do not provide useful expert capacity. The column-to-group map is stored with each checkpoint.

A.2 Exact Reconstruction Requires Unit Gates

Proposition 1 (Reconstructive initialization)**.** *Suppose all thin-SVD columns are retained, $\{J_g\}$ partitions them, and the original bias is kept. If $\alpha_g(x) = 1$ for every g and every input, then the adapted linear operator equals $Wx + b$. If every replaced projection satisfies these conditions, the entire model has the same function in exact arithmetic.*

Proof. By definition, $W = U\Sigma V^\top = AB^\top$. The partition implies $\sum_g A_{J_g} B_{J_g}^\top = AB^\top$. Substituting unit gates into Equation 3 proves the linear statement. Replacing each linear operator with

the identical function leaves all intervening nonlinearities, residual connections, and attention computations unchanged, proving the model statement by composition. □

Full activity alone is insufficient: positive gates that differ from one generally alter the function. Nor is exact arithmetic equality a statement about bitwise identity after BF16 storage, SVD numerical error, fused kernels, or different accumulation order.

A.3 A Local, Input-Specific Error Identity

Proposition 2 (Error relative to the original linear operator)**.** *Let $g(j)$ denote the group containing column j, $z = V^\top x$, and σ_j the singular values of W. Then*

$$\left\| Wx - \sum_g \alpha_g(x)\, W_g x \right\|^2 = \sum_{j=1}^{r} \big(1 - \alpha_{g(j)}(x)\big)^2 \sigma_j^2 z_j^2. \tag{6}$$

Proof. The error is $U\Sigma\,\mathrm{diag}(1-\bar{\alpha})\,z$. The columns of U are orthonormal, so left multiplication by U preserves the Euclidean norm of this r-dimensional vector. Expanding its squared norm yields Equation 6. □

A.4 Consequences for Strict Selection

Throughout, fix one adapted projection and one input x, write $E_g = E_g(x) = \sum_{j\in J_g} \sigma_j^2 z_j^2 = \|W_g x\|^2$, and let $\tilde{\alpha}_g = \mathrm{ReLU}(1+s\ell_g)$ denote the gate that group g would receive if it were selected, so that $\alpha_g = \mathbf{1}[g \in S]\,\tilde{\alpha}_g$ in Equation 3. Because the columns of U are orthonormal, $\sum_g E_g = \|Wx\|^2$.

Corollary 3 (Identity at the first step)**.** *If the output layer of h_ϕ is zero-initialized and $k_0 = G$, then $\ell_g(x) = 0$ and $\alpha_g(x) = 1$ for every group and input, and the adapted model equals the pretrained model in exact arithmetic.*

Proof. A zero output layer gives $\ell \equiv 0$, hence $\tilde{\alpha}_g = \mathrm{ReLU}(1) = 1$; with $k_0 = G$ every group is selected, so $\alpha_g = 1$. Proposition 1 applies. □

Proposition 4 (Energy top-k is the least-deviation strict support)**.** *Fix a budget $k \le G$. Among all sets S with $|S| = k$ and unit gates on S and zero gates elsewhere,*

$$\left\| Wx - \sum_{g\in S} W_g x \right\|^2 = \sum_{g\notin S} E_g = \|Wx\|^2 - \sum_{g\in S} E_g, \tag{7}$$

and the minimizers are exactly the sets consisting of k groups with the largest E_g (ties broken arbitrarily).

Proof. With $\alpha_g = \mathbf{1}[g \in S]$, the weight $(1-\alpha_{g(j)})^2$ in Proposition 2 equals $\mathbf{1}[g(j) \notin S]$, so the right-hand side of Equation 6 is $\sum_{g\notin S}\sum_{j\in J_g}\sigma_j^2 z_j^2 = \sum_{g\notin S} E_g$; the second equality in Equation 7 uses $\sum_g E_g = \|Wx\|^2$. Minimizing the deviation is therefore equivalent to maximizing $\sum_{g\in S} E_g$ over k-subsets. If S omits a group h with $E_h > E_g$ for some $g \in S$, exchanging g for h strictly increases the sum, so every maximizer contains k largest values; conversely, all such sets attain the same sum. □

Proposition 5 (Deviation-optimal support with learned gates)**.** *Let $y_{\mathrm{all}} = \sum_g \tilde{\alpha}_g W_g x$ be the output with every group selected and $y_S = \sum_{g\in S} \tilde{\alpha}_g W_g x$. Then*

$$\|y_{\mathrm{all}} - y_S\|^2 = \sum_{g\notin S} \tilde{\alpha}_g^2 E_g, \qquad \|Wx - y_S\|^2 = \sum_{g\in S}(1-\tilde{\alpha}_g)^2 E_g + \sum_{g\notin S} E_g. \tag{8}$$

Among k-subsets, the first quantity is minimized exactly by the k groups with the largest gated energy $\tilde{\alpha}_g^2 E_g$.

Proof. $y_{\text{all}} - y_S = U\Sigma\,\mathrm{diag}(v)\,z$ with $v_j = \tilde{\alpha}_{g(j)}\mathbf{1}[g(j) \notin S]$. Orthonormality of the columns of U gives $\|y_{\text{all}} - y_S\|^2 = \sum_{g\notin S}\tilde{\alpha}_g^2 \sum_{j\in J_g}\sigma_j^2 z_j^2$. The second identity is Proposition 2 with $\alpha_g = \mathbf{1}[g \in S]\tilde{\alpha}_g$. The minimizers follow from the exchange argument of Proposition 4 with weights $\tilde{\alpha}_g^2 E_g$. □

Lemma 6 (What the selection score ranks)**.** *Suppose $1+s\ell_g > 0$ and $E_g > \epsilon_E$, so that $\tilde{\alpha}_g = 1+s\ell_g$. Then the score of Equation 2 satisfies*

$$q_g = (\tilde{\alpha}_g - 1) + \beta \log E_g \quad \textit{and} \quad q_g - \log(\tilde{\alpha}_g E_g) = (\beta - 1)\log E_g + \varphi(\tilde{\alpha}_g), \tag{9}$$

where $\varphi(a) = a - 1 - \log a$ satisfies $0 \le \varphi(a) \le (a-1)^2/\big(2\min(a,1)^2\big)$. Subtracting a per-token constant from every $\log E_g$, as the centered training score does, shifts all q_g equally and leaves the ranking unchanged.

Proof. The first identity is Equation 2 with $s\ell_g = \tilde{\alpha}_g - 1$; subtracting $\log\tilde{\alpha}_g + \log E_g$ gives the second. The function φ is convex with $\varphi(1) = \varphi'(1) = 0$, so $\varphi \ge 0$. By Taylor's theorem, $\varphi(a) = \varphi''(\xi)(a-1)^2/2$ for some ξ between a and 1, and $\varphi''(\xi) = \xi^{-2} \le \min(a,1)^{-2}$. □

Remark. With $\beta = 1$, the score therefore ranks groups by the gated energy $\tilde{\alpha}_g E_g$ up to a term of second order in $\tilde{\alpha}_g - 1$, whereas the deviation-optimal ranking of Proposition 5 uses $\tilde{\alpha}_g^2 E_g = \tilde{\alpha}_g \cdot \tilde{\alpha}_g E_g$. The two coincide when all gates equal one (Proposition 4) and remain close for gates near one; mean selected gates are 1.00–1.06, although individual gates reach 4.09–17.44 (Table 31); the learned score can reorder candidates only to the extent that $\tilde{\alpha}_g$ departs from one. Without the energy prior, the score is $s\ell_g = \tilde{\alpha}_g - 1$, which does not involve $E_g(x)$ at all.

Proposition 7 (Task gradients under a strict mask)**.** *Write the output layer of the router as $\ell_g(x) = w_g^\top h(x) + c_g$, where $h(x)$ is its last hidden layer, and treat the membership indicator $m_g(x) = \mathbf{1}[g \in S_t(x)]$ as a constant, as in the implementation. If $m_g(x) = 0$ or $1 + s\ell_g(x) \le 0$, then $\partial y(x)/\partial \ell_g(x) = 0$, and token x contributes nothing to the task-loss gradient with respect to (w_g, c_g). Consequently, a group that is not selected with a positive gate for any token of a step receives, in that step, gradient in (w_g, c_g) only from the balance term of Equation 10.*

Proof. By Equation 3, $y(x) = \sum_g m_g(x)\,\mathrm{ReLU}\big(1 + s\ell_g(x)\big)\,A_{J_g} z_{J_g} + b$, and ℓ_g enters y only through the g-th term; it also enters q_g, but q affects y only through m, which is not differentiated. Hence $\partial y/\partial \ell_g = m_g\, s\,\mathbf{1}[1 + s\ell_g > 0]\,A_{J_g} z_{J_g}$, which vanishes under either condition. By the chain rule, the task-loss gradient with respect to (w_g, c_g) is a sum over tokens of $(\partial L_{\text{task}}/\partial y(x))^\top (\partial y(x)/\partial \ell_g(x))\,(h(x), 1)$, and each such term vanishes. The balance term depends on ℓ_g through $\mathrm{softmax}(q)$ and is the only remaining source. □

Corollary 8 (Logit-only ranking at the first strict step)**.** *Without the energy prior, $q_g = s\ell_g$. If a strict budget $k < G$ is imposed while the output layer is still zero, all scores tie and the same tie-broken set S_0 is selected for every token, independently of x; its deviation $\sum_{g\notin S_0} E_g(x)$ (Proposition 4) is then not controlled by the input, and by Proposition 7 the output rows of groups outside S_0 receive no task gradient. With the energy prior and $\beta > 0$, the same step selects the least-deviation set of Proposition 4.*

Proof. With $\ell \equiv 0$, every $q_g = 0$, so the selection is determined by the tie-breaking rule alone. The deviation statement is Equation 7, and the gradient statement is Proposition 7. With the prior, $q_g = \beta \log\max(E_g, \epsilon_E)$, which is increasing in E_g for $\beta > 0$, so the top-k by q is a top-k by energy. □

These statements concern one projection's deviation from its pretrained output and the flow of task gradients; they do not by themselves predict end-to-end accuracy, which Section 5.2 measures.

A.5 Counted Cost in Closed Form

Proposition 9 (Cost relative to the dense MLP)**.** *Consider a gated MLP whose gate and up projections map $d \to f$ and whose down projection maps $f \to d$, with $f > d$, so that $r = d$ for all three. Let the groups have equal width, let $\rho \in [0,1]$ be the fraction of groups whose output-factor columns are executed, count two FLOPs per multiply–add, and ignore router, elementwise, and bias work. With $\gamma = d/f$ and the dense MLP cost $6df$ per token as the unit:*

1. *the three input factors cost* $(1 + 2\gamma)/3$*;*
2. *computing the full input factors and only the executed output columns costs* $C_{\text{impl}}(\rho) = (1{+}2\gamma{+}2\rho{+}\rho\gamma)/3$*, and the down projection alone costs* $1{+}\rho\gamma$ *times its dense counterpart;*
3. *executing every output column (*$\rho = 1$*) costs* $1 + \gamma$*;*
4. *skipping unselected columns in both factors costs* $\rho(1 + \gamma)$*;*
5. $C_{\text{impl}}(\rho) < 1$ *if and only if* $\rho < (2 - 2\gamma)/(2 + \gamma)$.

Proof. For a projection $d_i \to d_o$ with rank $r = d$, the input factor $B^\top x$ costs $2rd_i$ and the output factor restricted to a fraction ρ of its columns costs $2\rho r d_o$. The gate and up projections ($d_i = d$, $d_o = f$) each cost $2d^2 + 2\rho df$, and the down projection ($d_i = f$, $d_o = d$) costs $2df + 2\rho d^2$. The total $4d^2 + 2df + 4\rho df + 2\rho d^2$ divided by $6df$ is $C_{\text{impl}}(\rho)$; setting $\rho = 0$ gives item 1, the down projection divided by $2df$ gives $1 + \rho\gamma$, and $\rho = 1$ gives item 3. Restricting both factors multiplies every term by ρ, giving $\rho(6d^2 + 6df)/(6df) = \rho(1 + \gamma)$. Item 5 rearranges $1 + 2\gamma + 2\rho + \rho\gamma < 3$. □

Table 5 evaluates the proposition at $\rho = k/G = 1/4$ with the public hidden and intermediate sizes of the checkpoints in Table 17. It reproduces the ranges used in the paper: input factors 0.42–$0.58\times$, `down_proj` 1.03–$1.09\times$, implemented prefill 0.59–$0.78\times$ (Section 4, Table 25), and both factors skipped 0.28–$0.34\times$. A batch union larger than k/G raises C_{impl} linearly with slope $(2 + \gamma)/3$; the implemented MLP stays below dense while the executed fraction is below 0.53–0.82, depending on the model.

Table 5: Proposition 9 at $\rho = 1/4$ (costs relative to the dense MLP). d and f are the hidden and intermediate sizes; the last column is the executed output fraction ρ at which the implemented MLP reaches the dense cost (item 5).

Model	d	f	γ	Input factors $(1{+}2\gamma)/3$	`down` $1{+}\gamma/4$	Implemented $C_{\text{impl}}(1/4)$	Both skipped $(1{+}\gamma)/4$	Break-even ρ $(2{-}2\gamma)/(2{+}\gamma)$
Qwen3-8B	4,096	12,288	0.333	0.56	1.08	0.75	0.33	0.57
Qwen3-14B	5,120	17,408	0.294	0.53	1.07	0.72	0.32	0.62
Llama-3.1-8B	4,096	14,336	0.286	0.52	1.07	0.71	0.32	0.63
Llama-3.2-3B	3,072	8,192	0.375	0.58	1.09	0.78	0.34	0.53
Gemma-2-9B	3,584	14,336	0.250	0.50	1.06	0.69	0.31	0.67
Gemma-2-27B	4,608	36,864	0.125	0.42	1.03	0.59	0.28	0.82

A.6 Grouping Is Not a Collection of Independent Nonlinear FFNs

Each group is a linear factor slice of one pretrained projection. Summing selected groups reconstructs a modified linear map; it does not insert a new nonlinearity between experts. The only learned nonlinearity in the router is inside h_ϕ, which produces gates rather than an expert output. This distinction matters when comparing Execution Tuning with MoE layers whose experts are separate MLPs.

A.7 What "Adapter-Free" and "Frozen" Mean

"Adapter-free" means that no new expert transformation is trained. The router, including the per-projection energy-prior scalar β, is trainable. "Frozen" means A, B, the group map, and the original bias are not updated by the task loss. It does not mean that the deployed function equals W, because learned gates need not be identically one. Task heads, when present, are a separate trainable regime.

A.8 Spectral Adaptation Methods in a Common Notation

Table 6: Spectral adaptation methods in a common notation, $W = U\,\mathrm{diag}(\sigma)V^\top$. Among these fine-tuning methods, only Execution Tuning has token-level exact zeros over groups, which a backend can use to skip output-factor work; the input factor remains dense. RaNA's inference-time masker (Section 3) also produces per-token exact zeros.

Method	Adapted operator	Conditioning	Gate form	Exact zeros
SVF	$U\,\mathrm{diag}(\sigma \odot z_t)V^\top$	task	mult., dense	no
SVFT	$U(\mathrm{diag}(\sigma) + M)V^\top$	none	additive, static	no
MoORE	$U\,\mathrm{diag}(\sigma + g(x,t))V^\top H$	token, task; linear in x	additive, dense	no
Execution Tuning	$U\,\mathrm{diag}(\sigma \odot \bar{\alpha}(x))V^\top$	token; MLP on z + energy	mult., top-k	yes

B Implementation and Complete Training Procedure

B.1 Main Recipe and the Alternative PI-Controller Variant

The main configuration uses a candidate target $k^\star = 64$, the cubic budget schedule of Equation 4, and no budget dropout. The strict schedule disables the PI controller used by the PI-controller variant. Consequently the scheduled candidate budget and the PI-controlled positive-gate fraction are different experimental regimes, and results from the PI variant are labeled as such.

Table 7: Strict-budget causal-LM recipe used for all causal-LM results in Table 13.

Component	Current setting
Frozen bank	Gate/up/down MLP projections only; data-free full thin SVD of each weight; 256 equal-width σ-ordered groups; no residual bank.
Router	Coordinates z; MLP $r \to 8 \to 32 \to 256$; GELU; learned energy-prior scalar β initialized to one.
Magnitude	$\mathrm{ReLU}(1 + s\ell_g)$ on selected candidates; stored logit scale $s = 96$.
Initialization	Hidden affine weights $\mathrm{Normal}(0, 0.022)$; biases zero; final affine weights zero; full candidate set at step zero.
Budget schedule	Cubic $256 \to 64$ over the first $0.7T$ optimizer steps; final 30% at 64; no budget dropout or maturity gate.
Auxiliary losses	Load balance coefficient 0.01; evaluated every fourth microbatch and multiplied by four; no self-distillation; no PI/L1 penalty.
Optimization	Three epochs; cosine LR; 5% LR warm-up; gradient clipping 1.0; effective batch depends on task (Table 16).
Selection	Internal held-out validation loss every half epoch; evaluate the selected checkpoint on the official benchmark once.
Seeds	Three independent training seeds (0, 1, 42) per model–task cell; tables report the mean.

The schedule computes k_t from the optimizer's global step, not from a measured entropy or expert-maturity threshold. It has no separate dense-routing warm-up: $k_0 = G$, followed immediately by the cubic decrease. Resuming requires the schedule state and global step to be restored. The all-on case $p = 1$ bypasses the schedule, which is defined only for $0 < k^\star < G$.

B.2 Literal Strict-Budget Loss and Its Gradients

For one projection and the token positions seen by its forward pass, define the detached candidate frequency and mean soft propensity

$$f_g = \frac{1}{n}\sum_a \mathbf{1}[g \in S(x_a)], \qquad P_g = \frac{1}{n}\sum_a \mathrm{softmax}\big(q(x_a)\big)_g, \qquad L_{\mathrm{bal}} = G\sum_g f_g P_g. \tag{10}$$

The code uses candidate membership in f_g, not positive-gate membership and not f_g/k. Thus $\sum_g f_g = k$ at a fixed candidate budget. With uniform propensity, $L_{\mathrm{bal}} = k$; its uniform value

equals one only under top-one normalization. We subtract one, which is a constant offset with respect to trainable parameters, but makes logged auxiliary magnitudes budget-dependent.

Let $a_j = \mathbf{1}[j \bmod 4 = 0]$ for microbatch index j and $L_{\mathrm{bal},j}$ be the mean of the terms actually collected across projections. Where that collection is nonempty, the literal loss is

$$L_j = L_{\mathrm{task},j} + 4(0.01)\, a_j\, (L_{\mathrm{bal},j} - 1). \tag{11}$$

No auxiliary term is added when the collection is empty, including all-on forwards that bypass the partial-budget branch. This periodic rescaling is not an exact equality of gradients to computing the loss on every microbatch. The deterministic every-fourth rule is also not, without an assumption on data ordering, an unbiased random-sampling estimator. It is a training-efficiency choice.

Hard sorting and set membership do not provide derivatives. Supervised gradients reach selected ReLU magnitudes and propagate through earlier network states; the soft propensity loss supplies an additional gradient for selection scores, including the energy-prior scalar. The forward still differentiates through z into earlier trainable routers even though its own A and B are frozen. Optional residual-LoRA, L_0, and teacher-distillation components are disabled in this recipe. Utilization statistics logged during training are sampled from one microbatch per window; they are diagnostics, not a substitute for measuring positive activity on all evaluation tokens. Selection count k_t, positive gate count, union support, and hardware execution are therefore distinct quantities. These distinctions also prevent a strict candidate schedule from being misreported as an exact 25% executed-FLOP guarantee.

Update order. Initialize final router layers to zero and prior scalars to one. For each optimizer step $t = 0, \ldots, T-1$: set k_t using Equation 4 and keep this budget during accumulation; for each accumulation microbatch with persistent index j, compute full z, router logits, training energy-prior scores, and the top-k_t mask; apply selected ReLU magnitudes and the output factor; form the supervised task loss; if $j \bmod 4 = 0$ and the auxiliary collection is nonempty, add Equation 11; accumulate gradients. Then apply the configured optimizer, gradient clipping, and learning-rate scheduler. Check, save, and evaluate at the configured step intervals, and retain the lowest internal validation-loss checkpoint. Export the selected checkpoint with its frozen-factor, group, router, prior, and scale identities.

B.3 PI-Controller Variant

Table 8 specifies the PI-controller variant. It is an alternative to the strict-budget recipe and is not combined with it.

Table 8: PI-controller variant: causal-LM configuration.

Component	Setting
Adapted modules	MLP `gate_proj`, `up_proj`, `down_proj`; attention unchanged.
Spectral construction	Data-free full thin SVD; $G = 256$; equal-width σ-ordered groups (stored column-to-group map).
Always-on residual	Zero fraction in the main configuration.
Controller selection	No fixed candidate budget; hard-concrete gating and self-distillation off; residual-LoRA rank zero.
Router	Effective coordinates z; affine widths $r \to 8 \to 32 \to 256$; GELU hidden activations.
Gate / initialization	$\mathrm{ReLU}(1 + h_\phi)$; hidden weights $\mathrm{Normal}(0, 0.022)$; all biases zero; final weights zero.
Target density	$p_0 = 0.95$ to $p^\star = 0.25$; default warm-up $0.1T$; cubic transition ending at $0.6T$.
PI state	$\lambda_0 = 0$, $e_{\mathrm{prev}} = 0$; $\eta = 0.1$, $K_I = K_P = 1$, $\rho = 0.005$, $\lambda_{\max} = 16$.
Dead-module safeguard	Halve the updated multiplier when observed activity is below $2/G$.
Collapse diagnostic	Pooled normalized utilization entropy threshold 0.2; default patience 6 optimizer checks after a grace interval.
Example Qwen/GSM8K	3 epochs; learning rate 2×10^{-4}; effective batch 64; seed 0; four GPUs; length grouping; checkpoint interval 0.5 epoch.

In this variant, explicit zeros arise from the ReLU gate alone, without a fixed top-k. The controller and loss are those of Equation 5, with observed activity a_t the fraction of positive gates at step t; p_t is the scheduled activity target, ending at $p^\star = 0.25$, and the gains, leak, cap, and dead-module safeguard are those of Table 8. There is no candidate ranking, so the energy prior of Equation 2 does not enter; the variant follows the adaptive-regularization idea of ReMoE (Wang et al., 2025d) on the frozen spectral bank. Activity is controlled on average over tokens, not per token.

B.4 Supervised Examples, Labels, and Auxiliary Gradients

Math SFT formats a GSM8K question or MATH problem with a fixed prompt template and uses the reference rationale/answer or solution as its target. It tokenizes prompt and answer separately, with special tokens enabled only for the prompt, and appends the tokenizer's actual EOS ID explicitly. Prompt labels are masked with -100; answer and EOS labels remain supervised. One token slot is reserved for EOS. Long answers are truncated to the remaining budget. Gradient checkpointing is disabled for ReLU routing so the auxiliary loss retains its gradient.

B.5 Numerical Factor Construction, Grouping, and Caching

Factors are built from a full thin SVD of each pretrained weight matrix. Groups are contiguous, σ-ordered, and of equal width r/G; the stored column-to-group map records them. Factors are cached once per model, and exported checkpoints include the exact factors rather than recomputing them, because trained router coordinates depend on exact SVD signs and orientation.

B.6 PI-Controller Variant: Loss, Scheduling, and Update Order

The PI controller updates a leaky PI multiplier from the gap between observed positive-gate activity and the scheduled target p_t. It is used only as an ablation. It is not combined with Equation 4. Its warm-up fraction, cubic endpoint, gains, and dead-module safeguard are those in Table 8. Update order is part of the method definition, and the PI variant is distinct from the strict candidate schedule.

B.7 Collapse Diagnostics and Distributed Operation

A pooled normalized utilization-entropy diagnostic can flag collapsed routing after a grace interval. It is a monitor, not a proof that routing remained diverse. On resume, distributed training restores both the optimizer step and the schedule state; microbatch index j in Equation 11 is persistent across accumulation, not restarted inside each rank without a defined rule.

B.8 Parameter and Serialization Accounting

Trainable counts include router weights, biases, and one learned β per projection. Non-causal-LM task heads are additional trainable state and are not included in the 4,483,404 causal-LM router count.

C Finding 1: Protocols, Controls, and Statistical Meaning

C.1 Frozen-Model Sample Coverage

For problem i, let c_i be the number of correct responses among 64 sampled outputs under the specified extractor. The reported empirical Pass@64 is $n^{-1}\sum_i \mathbf{1}[c_i > 0]$. For $k < 64$, a finite-sample Pass@k estimator can use $1 - \binom{64-c_i}{k} / \binom{64}{k}$ (Chen et al., 2021). Sampling uses temperature 0.8 and top-p 0.95.

The three GSM8K coverage values in Section 2.1 come from untouched pretrained checkpoints. This is a claim about reachable correct outputs under that sampling protocol. It is weaker than saying the model reliably solves those problems, and different from evaluating a budgeted single-shot adapted model. The 64-sample cost is an analysis cost, not included in the proposed deployment path.

C.2 Controls and the Overlap Bridge

In an answer-mismatch control, MATH empirical Pass@16 decreases from 53.52% to at most 3.52% when outputs/answers are paired with the wrong questions. Permuting unchanged MLP blocks across transformer layers reduces Qwen3-8B GSM8K coverage from 98.41% to 0%. These controls argue against trivial question-independent answer matching and show that the original organization matters. They do not isolate a unique reasoning mechanism or establish that spectral routing implements a particular sampled reasoning trace.

Of the 1,056 correct predictions of one adapted Qwen3-8B/GSM8K checkpoint (80.06%), 1,051 had at least one correct pretrained sample, a conditional overlap of 99.53%. This is not a "99.53% knowledge-reuse" causal estimate. Overall pretrained coverage is already high, and the complement consists of questions with no witness in a finite sample, not questions the model could never solve. The number is therefore a descriptive overlap; a coverage-conditioned baseline and seed uncertainty would be needed for a causal reading.

C.3 Expert × Router Intervention

The six cells in Table 9 use one MoORE parameterization, Llama-3.1-8B, full GSM8K test, seed 42, and greedy decoding. Static means no learned router. Token-only trains the token gate and gate-up projection; token+task additionally trains the task embedding and task projection. "Expert trainable" denotes the intervention's enabled expert-side parameters, not an assumption that all base-model weights were fully fine-tuned. These cells are not the MoORE baseline runs of Table 13, so the trainable token+task cell (53.30%) differs from that table's seed-42 MoORE result (58.76%).

Table 9: Finding 1: a single MoORE parameterization on Llama-3.1-8B with static, token, or token-plus-task routing (GSM8K accuracy, %, seed 42, greedy evaluation on 1,319 problems); cells are trainability interventions, not separately implemented named baselines. Also training the experts changes accuracy by $+0.61$ (token) and -1.97 (token + task) points (paired McNemar $p = 0.68$ and 0.078; Table 10).

Experts	Static	Token	Token + task
Frozen	9.02	54.36	55.27
Trainable	40.11	54.97	53.30

Table 10: Paired exact McNemar comparisons for Finding 1. Differences are percentage points, not relative percentages.

Contrast	Difference	Two-sided p
Frozen, token-only minus static	$+45.34$	2.54×10^{-140}
Frozen, token+task minus static	$+46.25$	1.26×10^{-143}
Trainable minus frozen, token-only	$+0.61$	0.677
Trainable minus frozen, token+task	-1.97	0.078

The paired test addresses disagreement over evaluation items for the trained instances, not variability over training seeds. Failure to reject a difference is not an equivalence or noninferiority test. The conclusion is limited to effective frozen-component adaptation in this setting. It does not compare six independently tuned methods, and its rows are trainability interventions rather than LoRA, MixLoRA, AdapterFusion, or X-LoRA baselines. Nor should the greedy static 9.02% be directly equated with the stochastic Pass@1 of 17.34%; the protocols differ.

D FINDING 2: AN INDEPENDENT CONTINUATION INTERVENTION

D.1 TRAINING AND HELD-OUT EVALUATION

This experiment is independent of our spectral construction and uses the official MoE-PEFT/MixLoRA code on a frozen Llama-3.1-8B backbone. Each of 32 layers has 16 rank-one LoRA experts on gate/up/down MLP projections. Training uses 7,217 GSM8K examples and a fixed 256-example validation tail, one epoch, seed 0, and greedy evaluation. The official test split is not used in this experiment. Data, optimizer, adapter capacity, and router parameterization are identical across paths.

Let q denote the nucleus routing threshold, to distinguish it from the activity target p_t of the PI-controller variant (the main recipe uses the candidate budget k_t). Soft keeps $q = 1$ and all 16 experts. Hard uses $q = 0.01$ from the beginning and selects exactly one expert. Controlled warms up at full routing through 10% of training and decreases q with an early-cubic schedule to 0.01 by 80%. Final Hard and Controlled both execute one expert.

Table 11: Finding 2: matched sparse deployment, not matched training compute. MixLoRA, seed 0, one epoch. FLOPs include expert adapters and their router, but exclude the unchanged dense backbone.

Path	Correct / 256	Accuracy (%)	Experts/token	MFLOPs/token
Soft	137	53.52	16	60.82
Hard	12	4.69	1	7.73
Controlled	116	45.31	1	7.73

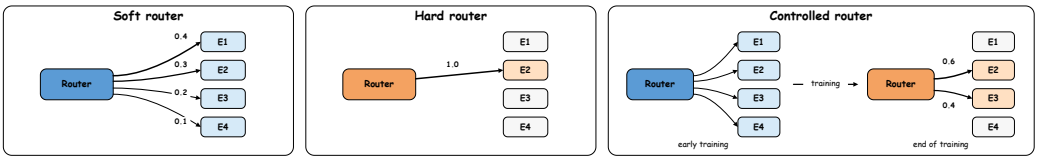


Figure 3: Soft, Hard, and Controlled training paths. Soft gives all experts nonzero weights; Hard uses only the selected route from the start; Controlled transitions from dense access to sparse routing. The illustration is qualitative. The MixLoRA test uses a scheduled nucleus threshold and a final top-one rule, whereas the current Execution Tuning recipe combines ReLU magnitudes with a scheduled top-k candidate set (Section 4). The PI-controller variant is described in Appendix B.

D.2 LOGICAL ADAPTER COMPUTATION

A rank-one LoRA expert on a projection $d_i \to d_o$ costs $2(d_i + d_o)$ multiply/add FLOPs, omitting small elementwise operations. For Llama-3.1-8B's gate/up/down shapes with hidden width 4,096, intermediate width 14,336, and 32 layers, this gives $32 \cdot 3 \cdot 2(4096 + 14336) = 3{,}538{,}944$ FLOPs/token for one expert at every adapted projection. The 4.19 MFLOPs/token router count corresponds to one shared $H \to G$ router per FFN: $2LHG = 2 \cdot 32 \cdot 4096 \cdot 16 = 4{,}194{,}304$. Under this accounting convention, Soft costs $16 \cdot 3.538944 + 4.194304 = 60.817408$ MFLOPs/token and Hard/Controlled cost $3.538944 + 4.194304 = 7.733248$ MFLOPs/token. The resulting reduction is approximately 87.3%. The expert-only reduction is $1 - 1/16 = 93.75\%$.

These counts assume one router shared by the three projections of each FFN; a variant with separate projection-input routers would require a different count. Neither calculation includes the unchanged dense Llama backbone, small routing elementwise operations, tensor dispatch, or measured device FLOPs. The numbers describe logical adapter-side work, not total-model compression.

D.3 Paired Outcomes and Scope

Controlled has 107 unique successes relative to Hard, while Hard has 3 relative to Controlled. Against Soft, Controlled has 23 unique successes and Soft has 44. An exact two-sided paired McNemar test conditions on the discordant pair count, giving the reported 3.42×10^{-28} and 0.0139 values. Accuracy differences computed from the integer counts are 40.625 and -8.203125 percentage points, respectively.

A Soft-minus-2-point margin is not met. Moreover, it is not enough to run an ordinary difference test and call a nonsignificant result noninferiority: a prespecified one-sided margin and corresponding uncertainty would be needed. No multiplicity-adjusted family-wide significance claim is made for the exploratory capacity sweep.

D.4 Mechanistic Interpretation Is Not Causal Identification

Failed outputs include repetitive arithmetic fragments, repeated answer delimiters, and empty responses. They demonstrate sequence-level failures under the tested generation protocol. Fragmented gradient exposure, feedback between a trainable router and trainable expert deltas, and autoregressive amplification are plausible explanations, but the experiment does not separately intervene on those mechanisms. Gradient-starvation explanations for trainable LoRA experts do not transfer directly to frozen spectral experts, whose weights receive no updates.

Table 12: Expert-count sweep for the independent MixLoRA intervention. Same top-one endpoint for Hard/Controlled; validation accuracy (%). These are not Execution Tuning spectral experiments.

Experts	Soft	Hard	Controlled	Interpretation
12	55.08	0.00	45.70	Severe hard-start failure
16	53.52	4.69	45.31	Primary reported cell
32	52.73	1.56	30.86	Recovery is less complete

This intervention uses a single seed. Delayed one-shot pruning, a fixed-target activity controller, and a ReMoE-style controller would further separate smooth continuation from merely spending longer in a dense regime.

E Complete Six-Task Language-Model Results

E.1 Task Definitions and Aggregation

GSM8K and MATH evaluate mathematical answers; MBPP evaluates generated programs using task-specific tests; ARC-Challenge, BoolQ, and PIQA evaluate multiple-choice or binary decisions (Cobbe et al., 2021; Hendrycks et al., 2021; Austin et al., 2021; Clark et al., 2018; 2019; Bisk et al., 2020). Their percentage scales support a descriptive unweighted macro average, but they do not have identical difficulty, uncertainty, or scoring semantics. MBPP is reported as Pass@1 rather than generic classification accuracy. Prompts, answer extraction, and scoring are fixed per task and shared across methods (Appendix E.2).

Table 13 reports all 24 model–method rows over the six tasks. Every entry is the mean over training seeds 0, 1, and 42, and the macro average is the mean of the six task means. Seed-to-seed standard deviations are at most 0.57 points, except Qwen3-8B/BoolQ for MoORE (2.74), whose seed-0 run is lower. Evaluation-set sizes are 1,319 (GSM8K), 5,000 (MATH), 500 (MBPP), 1,172 (ARC-C), 3,270 (BoolQ), and 1,838 (PIQA); the corresponding single-run evaluation standard errors are 0.4–2.2 points per task.

Table 13: Complete LLM task matrix (%). Four methods, six model instances, six benchmarks. MBPP uses Pass@1, and GSM8K uses flexible exact match. The macro average is the last column. Every value is the mean over training seeds 0, 1, and 42, and ± is the standard deviation across the three seeds (for the average, of the per-seed macro averages). Conversion and training budgets are not matched across methods. The Avg column is descriptive and is not used in any claim. Within each model, bold is best and underline is second best; the shaded row is Execution Tuning. Per-seed values are in Table 14.

Model	Method	GSM8K	MATH	MBPP	ARC-C	BoolQ	PIQA	Avg.
Qwen3-8B	Sparse Upcycling	79.71±0.12	47.57±0.08	56.80±0.20	56.57±0.17	88.65±0.24	**78.15**±0.19	67.91±0.10
	MixLoRA	**81.91**±0.23	**50.45**±0.12	**65.00**±0.20	**57.74**±0.27	91.08±0.57	78.04±0.13	**70.70**±0.18
	MoORE	80.26±0.24	47.43±0.27	55.67±0.31	57.59±0.23	90.50±2.74	64.40±0.27	65.97±0.49
	Execution Tuning	80.44±0.08	48.73±0.25	62.53±0.31	56.60±0.32	**91.16**±0.21	76.53±0.06	69.33±0.04
Qwen3-14B	Sparse Upcycling	**86.38**±0.27	53.77±0.30	67.40±0.20	61.83±0.10	90.93±0.25	**81.19**±0.30	73.58±0.04
	MixLoRA	86.28±0.20	**55.35**±0.28	**70.53**±0.31	63.00±0.21	92.15±0.29	81.07±0.20	**74.73**±0.16
	MoORE	86.23±0.23	52.05±0.18	62.13±0.31	**64.33**±0.26	91.65±0.25	76.30±0.22	72.11±0.04
	Execution Tuning	86.23±0.09	54.55±0.27	68.27±0.23	63.54±0.13	**92.16**±0.39	79.96±0.36	74.12±0.02
Llama-3.1-8B	Sparse Upcycling	59.06±0.15	17.51±0.27	42.13±0.31	52.25±0.21	89.72±0.19	80.83±0.11	56.92±0.05
	MixLoRA	**60.55**±0.19	**19.77**±0.25	**46.13**±0.31	54.15±0.10	**89.90**±0.10	**81.30**±0.17	**58.63**±0.05
	MoORE	58.71±0.09	18.83±0.16	42.27±0.31	54.44±0.23	87.72±0.12	77.13±0.08	56.51±0.06
	Execution Tuning	58.12±0.19	18.01±0.17	43.73±0.31	**54.61**±0.37	87.90±0.12	80.25±0.33	57.10±0.24
Llama-3.2-3B	Sparse Upcycling	44.60±0.16	9.52±0.29	29.73±0.31	42.89±0.21	85.88±0.12	75.81±0.22	48.07±0.04
	MixLoRA	**44.86**±0.22	10.01±0.17	38.27±0.23	45.19±0.30	87.65±0.19	**77.86**±0.20	**50.64**±0.12
	MoORE	43.32±0.32	**10.52**±0.14	37.93±0.31	43.94±0.23	87.29±0.08	76.90±0.14	49.98±0.10
	Execution Tuning	44.30±0.16	9.99±0.26	**38.93**±0.42	**45.25**±0.39	**87.72**±0.35	75.77±0.37	50.33±0.06
Gemma-2-9B	Sparse Upcycling	68.36±0.32	22.13±0.31	37.20±0.20	64.19±0.21	89.81±0.17	76.33±0.19	59.67±0.08
	MixLoRA	**73.92**±0.13	**32.43**±0.21	**48.20**±0.20	**66.81**±0.31	**90.32**±0.27	**83.30**±0.14	**65.83**±0.09
	MoORE	56.36±0.29	25.93±0.29	37.80±0.20	60.69±0.32	87.41±0.34	68.43±0.31	56.10±0.09
	Execution Tuning	70.86±0.24	27.79±0.44	45.00±0.53	63.17±0.13	89.05±0.31	82.35±0.32	63.04±0.07
Gemma-2-27B	Sparse Upcycling	77.20±0.27	36.24±0.21	**60.87**±0.31	67.18±0.18	89.42±0.30	**85.56**±0.31	69.41±0.15
	MixLoRA	**80.46**±0.19	**37.33**±0.23	58.40±0.20	**71.30**±0.18	**90.98**±0.14	75.24±0.24	68.95±0.08
	MoORE	68.16±0.23	30.62±0.22	40.20±0.20	58.70±0.31	89.41±0.26	68.68±0.17	59.29±0.14
	Execution Tuning	78.92±0.33	36.17±0.17	59.40±0.40	68.94±0.37	89.52±0.19	84.73±0.27	**69.61**±0.14

Table 14: Per-seed results (%) behind Table 13: seeds 0, 1, and 42 for each task.

		GSM8K			MATH			MBPP			ARC-C			BoolQ			PIQA		
Model	Method	0	1	42	0	1	42	0	1	42	0	1	42	0	1	42	0	1	42
Qwen3-8B	Sparse Upc.	79.83	79.68	79.61	47.66	47.52	47.52	57.00	56.60	56.80	56.74	56.57	56.40	88.93	88.56	88.47	77.97	78.13	78.35
	MixLoRA	82.11	81.65	81.96	50.32	50.56	50.46	64.80	65.00	65.20	57.42	57.94	57.85	90.43	91.35	91.47	77.97	77.97	78.18
	MoORE	79.98	80.36	80.44	47.74	47.30	47.24	56.00	55.40	55.60	57.34	57.68	57.76	87.34	92.02	92.14	64.09	64.53	64.58
	Exec. Tuning	80.44	80.36	80.52	48.86	48.90	48.44	62.60	62.20	62.80	56.23	56.83	56.74	91.25	90.92	91.31	76.50	76.61	76.50
Qwen3-14B	Sparse Upc.	86.66	86.13	86.35	54.02	53.84	53.44	67.20	67.40	67.60	61.77	61.95	61.77	90.67	91.16	90.95	80.90	81.18	81.50
	MixLoRA	86.43	86.05	86.35	55.58	55.04	55.44	70.80	70.20	70.60	63.23	62.97	62.80	91.83	92.20	92.42	81.23	80.85	81.12
	MoORE	85.97	86.43	86.28	51.84	52.14	52.16	61.80	62.40	62.20	64.59	64.08	64.33	91.87	91.71	91.38	76.55	76.17	76.17
	Exec. Tuning	86.13	86.28	86.28	54.40	54.86	54.38	68.40	68.00	68.40	63.40	63.65	63.57	92.35	91.71	92.42	80.14	80.20	79.54
Llama-3.1-8B	Sparse Upc.	59.21	59.06	58.91	17.76	17.56	17.22	41.80	42.40	42.20	52.47	52.05	52.22	89.51	89.82	89.85	80.96	80.74	80.79
	MixLoRA	60.73	60.35	60.58	19.94	19.88	19.48	46.40	46.20	45.80	54.10	54.10	54.27	89.79	89.97	89.94	81.12	81.34	81.45
	MoORE	58.61	58.76	58.76	18.64	18.90	18.94	42.60	42.20	42.00	54.61	54.52	54.18	87.86	87.65	87.65	77.04	77.20	77.15
	Exec. Tuning	57.92	58.15	58.30	17.82	18.04	18.16	43.40	43.80	44.00	54.18	54.78	54.86	87.77	87.95	87.98	79.87	80.47	80.41
Llama-3.2-3B	Sparse Upc.	44.43	44.73	44.66	9.28	9.84	9.44	30.00	29.40	29.80	42.66	42.92	43.09	85.75	85.90	85.99	76.06	75.68	75.68
	MixLoRA	44.73	44.73	45.11	10.16	10.06	9.82	38.40	38.40	38.00	45.48	45.22	44.88	87.80	87.43	87.71	78.07	77.80	77.69
	MoORE	43.06	43.67	43.21	10.36	10.56	10.64	37.60	38.00	38.20	44.11	44.03	43.69	87.37	87.28	87.22	76.77	76.88	77.04
	Exec. Tuning	44.12	44.35	44.43	10.10	10.18	9.70	38.80	38.60	39.40	44.80	45.48	45.48	87.89	87.31	87.95	75.90	76.06	75.35
Gemma-2-9B	Sparse Upc.	68.61	68.01	68.46	22.48	21.92	21.98	37.40	37.20	37.00	63.99	64.16	64.42	89.63	89.97	89.82	76.44	76.44	76.12
	MixLoRA	73.84	73.84	74.07	32.64	32.22	32.44	48.00	48.40	48.20	67.06	66.89	66.47	90.61	90.24	90.09	83.41	83.13	83.35
	MoORE	56.03	56.56	56.48	26.18	25.62	26.00	38.00	37.60	37.80	60.32	60.92	60.84	87.80	87.25	87.19	68.61	68.06	68.61
	Exec. Tuning	70.58	71.04	70.96	28.04	28.04	27.28	45.20	44.40	45.40	63.31	63.14	63.05	89.11	88.72	89.33	82.48	82.59	81.99
Gemma-2-27B	Sparse Upc.	76.95	77.18	77.48	36.44	36.26	36.02	60.60	60.80	61.20	66.98	67.24	67.32	89.17	89.76	89.33	85.31	85.47	85.91
	MixLoRA	80.29	80.44	80.67	37.10	37.56	37.34	58.60	58.40	58.20	71.16	71.25	71.50	91.10	90.83	91.01	74.97	75.35	75.41
	MoORE	68.39	67.93	68.16	30.40	30.84	30.62	40.00	40.40	40.20	58.36	58.79	58.96	89.14	89.42	89.66	68.50	68.72	68.82
	Exec. Tuning	78.54	79.08	79.15	36.08	36.36	36.06	59.40	59.00	59.80	68.52	69.11	69.20	89.63	89.30	89.63	84.60	85.04	84.55

E.2 Training and Evaluation Protocol

Each of the six models is trained separately on each of the six tasks with each of the four methods, giving 144 model–method–task configurations. All configurations are trained with seeds 0, 1, and 42, and no checkpoint trained on a task mixture is used.

Table 15: Which protocol produced which table. All protocols use the frozen spectral bank with $G = 256$ and MLP-only adaptation unless stated.

Tables	Protocol
13, 14, 2a (energy row) and d, 36	Main campaign: seeds 0, 1, and 42; internal holdout; lowest held-out loss; three epochs; learning rates listed below; vLLM with the fused Execution Tuning kernels.
2b, 30, 31	Gate ablation, seed 0, main recipe; the 25% ReLU cells are main-campaign seed-0 runs; Table 31 uses earlier runs of the same configuration.
2c, 32	Evaluation only, on the main-campaign seed-0 Qwen3-8B/GSM8K checkpoint.
2a (gate-logit row), 33, 34, 35	Qwen3-8B/GSM8K, seed 0, all training examples without a holdout, final checkpoint (step 351).
29	Single run per cell with its own training settings; lowest held-out loss, except the Qwen3-8B/GSM8K cells, which follow the previous row.
19	Separate dense-cost PEFT run: seed 0, three epochs, learning rate 3×10^{-4}, batch 64, maximum length 2,048, $8 \rightarrow 32$ router, lowest held-out loss.
1, Appendices C–D	Finding 1: one MoORE parameterization on Llama-3.1-8B, seed 42, greedy decoding. Finding 2: official MixLoRA code, one epoch, seed 0, 256-example development split.
3, 22, 23, 24	Non-LLM models, single run, main recipe.
25, 26, 27	Timing: seed-0 checkpoints; one H200, BF16, batch 64, prompt length 512, 128 generated tokens; decoder-block harness.
Section 5.2 router statistics	A separately trained Qwen3-8B/GSM8K checkpoint.

Table 16: Task data accounting. Training examples per epoch exclude a fixed internal holdout. Official evaluation examples are not used for checkpoint selection.

Task	Source train	Holdout	Train/epoch	Max length / effective batch
GSM8K	7,473	128	7,345	512 / 64
MATH	7,499	128	7,371	1,024 / 64
MBPP	374	32	342	1,024 / 32
ARC-C	1,119	128	991	512 / 32
BoolQ	9,427	128	9,299	512 / 64
PIQA	16,113	128	15,985	512 / 64

The MATH training set is a leakage-filtered reconstruction, evaluated on the 5,000-example test split. Each retained training row is used once per epoch for three epochs, without task mixing, oversampling, or cycling. All configurations are trained with seeds 0, 1, and 42. MoORE's seed-0 runs were trained on a different machine than its seed-1 and seed-42 runs. Validation-loss checkpoints are compared every half epoch; the official evaluation split (test for GSM8K, MATH, MBPP, and ARC-C; validation for BoolQ and PIQA) is reserved for a final metric on the selected checkpoint.

We use fused AdamW, except for Sparse Upcycling, which uses Adafactor. Learning rates are 2×10^{-5} for Sparse Upcycling, 10^{-4} for MixLoRA, 3×10^{-4} for MoORE, and 10^{-4} for Execution Tuning except Qwen3-8B's non-MBPP tasks at 2×10^{-4}; these Execution Tuning values form a locked optimization configuration, identical across seeds. Learning rates are not swept: Execution Tuning uses one of three standard values (10^{-4}, 2×10^{-4}, and 3×10^{-4}, the last in Appendix F), and each baseline starts from the best-practice configuration of its source paper or official repository. Qwen3 models are evaluated without a chat template, using the raw training prompt

(`Question: ...\nAnswer:`), so the thinking mode is never invoked. Shared settings include cosine decay, 5% LR warm-up, clip norm 1.0, right padding, and length grouping. Both Llama models and Gemma-2-9B are trained in FP32/TF32 because BF16 produced non-finite gradients; Qwen and Gemma-2-27B use BF16. This training precision distinction does not change the separate H200 latency table's BF16 timing contract.

MixLoRA uses the top-k example configuration of the official repository (TUDB-Labs/MixLoRA, commit `d59d09b`) unchanged, without sweeping expert count or rank: top-two routing over eight LoRA experts (rank 8, $\alpha = 16$, dropout 0.05) with one router shared by the gate, up, and down projections (initialization std 0.02, no jitter, load-balancing coefficient 0.01), plus separate rank-eight LoRA adapters on the query and value projections. Model repositories and pinned revisions are listed in Table 17. MoORE retains seven projection targets, router width 128, and eight Householder reflections. Our compact-WY execution of those reflections is an implementation optimization, not a reduced MoORE configuration. A single-task export also fixes its task ID. MBPP uses a fixed three-shot prompt and a network-isolated execution scorer.

Table 17: LLM checkpoints: upstream repository, the repository actually loaded, and its pinned revision. The Qwen3 repositories are the released post-trained models; the Llama and Gemma repositories are pretrained base models.

Model	Upstream	Loaded from	Revision
Qwen3-8B	Qwen/Qwen3-8B	same	`b968826d9c46dd6066d109eabc6255188de91218`
Qwen3-14B	Qwen/Qwen3-14B	same	`40c069824f4251a91eefaf281ebe4c544efd3e18`
Llama-3.1-8B	meta-llama/Llama-3.1-8B	unsloth/Meta-Llama-3.1-8B (byte-identical)	`e9a141a2091ea561b96483212645a2a05e6f99fc`
Llama-3.2-3B	meta-llama/Llama-3.2-3B	unsloth/Llama-3.2-3B (byte-identical)	`d4446454d87d51aa42e1fb174f25acc5f8762331`
Gemma-2-9B	google/gemma-2-9b	unsloth/gemma-2-9b (BF16 reshard)	`9145841c83add11c48155cd7bb87d8656703f997`
Gemma-2-27B	google/gemma-2-27b	unsloth/gemma-2-27b (BF16 reshard)	`ae96b425538ec2e31f9111dbe83f4a705f1eacaa`

Table 18: Model settings for training and for the latency benchmark. The interval denotes Sparse Upcycling replacement frequency, starting at layer index one. A mismatch forbids treating that timing and training configuration as one joint point.

Model	Layers	Rank	Train precision	Upcycling interval: train / time
Qwen3-8B	36	4,096	BF16	2 / 2
Qwen3-14B	40	5,120	BF16	4 / 2
Llama-3.1-8B	32	4,096	FP32/TF32	4 / 2
Llama-3.2-3B	28	3,072	FP32/TF32	2 / 2
Gemma-2-9B	42	3,584	FP32/TF32	2 / 2
Gemma-2-27B	46	4,608	BF16	16 / 16

F Dense-Cost PEFT on LLMs under a Separate Protocol

The MoE-PEFT campaign of Table 13 uses each method's own recipe. To compare with dense-cost PEFT, we ran a separate experiment in which Execution Tuning, LoRA, DoRA, and AdaLoRA (Hu et al., 2022; Liu et al., 2024; Zhang et al., 2023) are trained on Llama-3.1-8B and Qwen3-8B for GSM8K and MATH. All methods are trained for 3 epochs with learning rate 3×10^{-4}, effective batch size 64, maximum sequence length 2,048, and a single seed (0). LoRA and DoRA ($r = 4$, $\alpha = 8$) and AdaLoRA ($r = 8 \rightarrow 4$) adapt all seven attention and MLP projections, whereas Execution Tuning adapts only the MLP projections, with the main $8 \rightarrow 32$ router. Checkpoints are saved every half epoch, and for every method we report the checkpoint with the lowest held-out evaluation loss. Because this protocol differs from the MoE-PEFT campaign (learning rate 3×10^{-4} rather than 10^{-4} or 2×10^{-4}, maximum length 2,048 rather than 512 or 1,024, and a single seed), its Execution Tuning numbers differ from those of Table 13 for the same pairs, and the two tables should not be compared directly. Under this protocol, Execution Tuning is within 0.25 points of LoRA ($r = 4$) on all four cells and within 0.91 points of DoRA ($r = 4$), and it exceeds AdaLoRA on three of the four cells; single-run evaluation standard errors are 1.1–1.3 points on GSM8K and 0.6–0.7 points on MATH, so these LoRA and DoRA differences are smaller than one single-run evaluation standard error. Execution Tuning trains 2.4–4.9$\times$ fewer parameters than these baselines on Qwen3-8B, and

LoRA, DoRA, and AdaLoRA execute at dense cost after merging, whereas the Execution Tuning prefill decoder stack counts 1.19–1.58× fewer FLOPs than dense (Table 28): it reaches LoRA-level accuracy at lower prefill cost.

Table 19: Execution Tuning versus dense-cost PEFT under a separate protocol from the MoE-PEFT campaign of Table 13 (accuracy, %; seed 0; lowest held-out evaluation-loss checkpoint per method). Trainable parameters are given as a percentage of Qwen3-8B. Bold is best and underline second best per column.

	GSM8K		MATH		
Method	Llama-3.1-8B	Qwen3-8B	Llama-3.1-8B	Qwen3-8B	Trainable (%)
LoRA ($r = 4$)	64.29	80.97	**19.34**	**51.22**	0.133
DoRA ($r = 4$)	**64.67**	<u>82.11</u>	19.20	50.86	0.150
AdaLoRA ($r = 8 \rightarrow 4$)	51.25	**82.26**	7.96	40.82	0.267
Execution Tuning	<u>64.44</u>	81.20	<u>19.26</u>	<u>51.18</u>	**0.055**

G GLUE: Encoder Transfer with Explicit Head and Target-Module Scope

We evaluate DeBERTa-v3-base (He et al., 2023) on the eight GLUE tasks (Wang et al., 2019) against LoRA, PiSSA, and AdaLoRA (Hu et al., 2022; Meng et al., 2024; Zhang et al., 2023). Each score is the mean over training seeds 0, 1, and 42 at the dev-selected checkpoint; as is common for GLUE, checkpoints are selected and scores reported on the same dev set, so all methods are evaluated with the same optimistic protocol. We report Matthews correlation for CoLA, Pearson correlation for STS-B, and accuracy for the other tasks, with MNLI evaluated on the matched validation split; the average is the unweighted mean of the eight tasks. Full fine-tuning and the all-matrix LoRA ($r = 2$) and AdaLoRA rows are taken from Zhang et al. (2023).

Execution Tuning reaches the same eight-task average as our query/key/value LoRA run (87.66) with fewer trainable parameters (0.36M versus 0.44M), is 0.11 points below PiSSA, and is 0.41 points above our AdaLoRA run. The published all-matrix LoRA and AdaLoRA results, with 0.33M and 0.32M parameters, average 88.32 and 89.03 over the eight listed tasks (the source's all-metric averages are 88.15 and 88.86), above every run we reproduce, including Execution Tuning; they use five seeds and adapt all weight matrices. It gives the best MRPC score and ties PiSSA on RTE, but trails the LoRA family on STS-B, MNLI, and QNLI. Parameter counts in Table 20 exclude the classification head; the NLU setting also trains the score, classifier, and pooler layers where applicable, for every method. The encoder results are therefore router-plus-head adaptation, not strict router-only adaptation of every model component.

Table 20: GLUE dev results on DeBERTa-v3-base (percentage scale; mean over seeds 0, 1, 42). CoLA: Matthews correlation; STS-B: Pearson correlation; others: accuracy. Params exclude the classification head. †Reported by Zhang et al. (2023): mean of five seeds, adapters on all weight matrices, STS-B as the average of Pearson and Spearman correlation; the Avg. column is our unweighted mean of the eight listed values. Among our runs, bold is best and underline is second best; the shaded row is Execution Tuning.

Method	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg.	Params
Full FT†	69.19	95.63	89.46	92.40	91.60	89.90	94.03	83.75	88.25	184M
LoRA, r=2, all matrices†	68.71	94.95	89.71	91.61	91.68	90.30	94.03	85.56	88.32	0.33M
AdaLoRA, all matrices†	70.04	95.80	90.44	91.78	91.63	90.66	94.49	87.36	89.03	0.32M
LoRA	67.76	**95.64**	<u>89.46</u>	89.61	**91.59**	<u>89.96</u>	92.79	84.48	<u>87.66</u>	0.44M
PiSSA	**68.05**	**95.64**	88.48	**89.97**	91.35	**90.15**	**93.30**	**85.20**	**87.77**	0.44M
AdaLoRA	67.88	94.15	87.34	<u>89.79</u>	<u>91.44</u>	89.33	<u>93.23</u>	<u>84.84</u>	87.25	0.66M
Execution Tuning	<u>67.90</u>	<u>95.18</u>	**90.69**	89.74	90.96	88.89	92.75	**85.20**	<u>87.66</u>	0.36M

Target modules. Execution Tuning adapts the FFN projections, whereas LoRA, PiSSA, and AdaLoRA adapt the attention query/key/value projections; Table 20 therefore compares the methods at similar trainable-parameter budgets (0.36M versus 0.44M) rather than on identical target modules.

H VLM, VISION, AND SPEECH: COMPLETE RESULTS

H.1 MODELS, TASKS, AND UNIT CONVENTIONS

For each non-LLM model we report quality, trainable/base parameter percentage, and registered evaluation GFLOPs/example. All three cover the same six models and four methods and are the final results, each from a single training seed; Execution Tuning uses the main recipe (strict top-64 of 256 candidate groups, cubic schedule, energy prior).

Table 21: Cross-architecture model identities. The suffixes 3B/7B are display labels; the recorded parameter denominators are listed explicitly.

Short label	Family	Model ID / benchmark	Base params (M)
VLM-3B	VLM	Qwen2.5-VL-3B-Instruct; ScienceQA test (4,241)	3754.62
VLM-7B	VLM	Qwen2.5-VL-7B-Instruct; ScienceQA test (4,241)	8292.17
ViT-B	ViT	vit-base-patch16-224-in21k; CIFAR-100	85.88
ViT-L	ViT	vit-large-patch16-224-in21k; CIFAR-100	303.40
Whisper-small	ASR	whisper-small; FLEURS English (`en_us`)	241.73
Whisper-medium	ASR	whisper-medium; FLEURS English (`en_us`)	763.86

These references identify the public models and tasks (Bai et al., 2025; Dosovitskiy et al., 2021; Radford et al., 2023; Lu et al., 2022; Krizhevsky, 2009; Conneau et al., 2023).

Table 22: Cross-architecture quality: accuracy (%) ↑ for VLM/ViT; WER (%) ↓ for Whisper. Final results from a single training seed. Bold is best and underline second best per row; the shaded column is Execution Tuning.

Model	Execution Tuning	Upcycling	MixLoRA	MoORE
Qwen2.5-VL-3B	93.89	**97.10**	<u>97.01</u>	91.98
Qwen2.5-VL-7B	96.32	**97.90**	<u>97.71</u>	97.67
ViT-B	91.90	90.38	**92.66**	<u>92.32</u>
ViT-L	93.48	88.20	**94.16**	<u>93.90</u>
Whisper-small	8.59	8.28	**7.45**	<u>7.89</u>
Whisper-medium	6.42	<u>6.28</u>	**6.12**	**6.12**

Table 23: Trainable parameters divided by the recorded original base-model parameter count (%). Values above 100% are possible when upcycling adds multiple trainable experts; they are not percentages of the expanded model's total. Bold is best and underline second best per row; the shaded column is Execution Tuning.

Model	Execution Tuning	Upcycling	MixLoRA	MoORE
Qwen2.5-VL-3B	**0.07**	259.40	<u>0.63</u>	5.02
Qwen2.5-VL-7B	**0.04**	275.12	<u>0.39</u>	3.09
ViT-B	**0.51**	298.01	<u>2.02</u>	25.59
ViT-L	**0.30**	299.22	<u>1.49</u>	19.28
Whisper-small	**0.30**	240.21	<u>1.43</u>	23.12
Whisper-medium	**0.21**	258.06	<u>1.21</u>	19.51

Table 24: Registered evaluation GFLOPs/example. A dagger marks partial operation coverage. For Execution Tuning the counter charges each adapted projection $2r_{\text{act}}(d_i + d_o)$, the nominal selected-factor cost; the implemented path also computes the full input factor (Appendix I), so these values are lower bounds on its executed work. The shaded column is Execution Tuning.

Model	Execution Tuning	Upcycling	MixLoRA	MoORE
Qwen2.5-VL-3B	3085.82	6084.38	6094.60	$5068.77^{\dagger}$
Qwen2.5-VL-7B	4964.13	11983.32	11997.05	9494.95
ViT-B	19.92	57.45	57.80	$58.83^{\dagger}$
ViT-L	68.88	202.47	203.40	$201.19^{\dagger}$
Whisper-small	282.20	575.04	577.97	$644.10^{\dagger}$
Whisper-medium	906.15	1949.70	1957.80	$2133.98^{\dagger}$

H.2 What These Results Support

In single runs, the method also applies beyond the causal-LM examples, with trainable/base percentages of 0.04–0.51%. They also show explicit quality costs: the best competing VLM values exceed the Execution Tuning values, MixLoRA gives higher ViT accuracy, and all listed ASR competitors have lower WER in these two rows. These outcomes are not hidden by inverting WER or reporting only a favorable aggregate. Model-scale changes are comparisons between distinct pretrained systems, not controlled interventions on parameter count alone. Registered compute values are smaller for Execution Tuning, but they use the nominal selected-factor count, and partial counter coverage for several baselines further limits their interpretation as physical computation savings. A dagger is not a statistical uncertainty interval. No FLOPs-only comparison demonstrates latency, throughput, energy reduction, or lower peak memory.

I Efficiency: Formulas, Faithful Execution, and Measurements

I.1 Cost Definitions and Their Denominators

For one projection and one input token, dense multiplication uses approximately $2d_i d_o$ FLOPs under a multiply-plus-add-equals-two convention. With full thin factors, the current reference uses approximately $2r(d_i + d_o)$, plus router and elementwise work. Its positive-gate fraction does not change the shapes of those matrix multiplications. If a backend knows a selected set of rank columns of cardinality k and faithfully computes only those columns with their original nonzero gates, the factor products cost $2k(d_i + d_o)$ before routing and dispatch. If the router first computes the full z and only the output factor is skipped, the count is instead approximately $2d_i r + 2d_o k + C_{\text{router}}$. Unequal group ranks require $k = \sum_g |J_g| \, \mathbf{1}[\alpha_g > 0]$, not $k = r$ times the fraction of active groups.

For $d_i = H$, $d_o = 3H$, and $k = H/4$, the two selected factor products have one third of the dense projection's arithmetic. Adding unchanged attention projections reduces the model-shaped projection-stack saving. Because the energy prior reads the full latent, the implemented path computes the full input factor and skips only output-factor columns; Table 25 reports both counts.

I.2 Folding the Router's First Layer

For a router-only selector without the spectral-energy prior, let its first affine map be $R_1 z + b_1$, where $R_1 \in \mathbb{R}^{h_1 \times r}$ and $h_1 = 8$. For fixed inference weights, define $C = BR_1^{\top}$. Then

$$R_1 B^{\top} x + b_1 = C^{\top} x + b_1. \tag{12}$$

This is matrix associativity, not a newly trained approximation. Equation 12 is not used in the timed implementation, and it does not compute $E_g = \sum_{j \in J_g} \sigma_j^2 z_j^2$. The current energy-prior selector still reads the full latent, so applying the fold alone would not justify selected input-factor FLOPs.

I.3 Optimized Dynamic Decoder-Block Benchmark

This path retains the dynamic energy-prior selector with 64 candidates among 256 groups. The measurement uses one H200, BF16, tensor parallelism one, batch $B = 64$, prompt length $L = 512$,

and output-length convention $O = 128$. It is not a vLLM serving run. We denote the prefill wall time by t_{prefill}. Throughput is a derived fixed-workload quantity,

$$Q_{128} = \frac{64 \cdot 128}{t_{\text{prefill}} + 127\, t_{\text{decode}}}, \tag{13}$$

with times in seconds. It assumes the reported decode-step time represents each of the 127 subsequent steps. All Execution Tuning accuracy, FLOP, and latency numbers come from the same checkpoints and the same fused execution path: every checkpoint uses equal-width partitions ($r/256$ rank columns per group, 12–20 across our models; 64 of 256 groups is 25% of the rank), which the fused scale kernel requires, and accuracy is evaluated in vLLM with the same fused Execution Tuning kernels used for timing. Execution Tuning decode is CUDA-graphed while all three baselines are eager. All methods use the same harness, shapes, precision, and workload, and each timed model is the trained checkpoint (seed 0) with the configuration whose accuracy Table 13 reports; Sparse Upcycling on Qwen3-14B and Llama-3.1-8B is the one exception (Table 18); timed at its training interval of four, it reaches 1.34–$1.35\times$ higher throughput and 1.19–$1.20\times$ lower TTFT on these two models, and Execution Tuning remains faster on both. With MoORE captured in CUDA graphs on Qwen3-8B (batch 64), Execution Tuning's end-to-end throughput is $1.47\times$ MoORE's at 512-token prompts and $1.50\times$ at 2,048; across the additional batch sizes and prompt lengths we measured, Execution Tuning is the fastest MoE-PEFT method in every setting. Execution Tuning rows come from the dense-comparison run of Table 27; baseline rows come from separate runs of the same harness.

Table 25: Complete H200 decoder-block results. Execution Tuning prefill times are recorded to the millisecond and its throughput to the token per second. Nominal GF is logical GFLOPs/token as counted for each method; for Execution Tuning it assumes both factors skip unselected groups. Impl. GF adds the full input-factor product that the energy prior requires, excluding the router; it is given for Execution Tuning only. Prefill is wall-clock prefill time and Q_{128} is derived throughput. Prefill and decode are milliseconds; Q_{128} is Equation 13. Execution Tuning rows are from the run of Table 27; baseline rows use the same harness and workload. Within each model, bold and underline mark the best and second-best trainable-parameter and timing values (lower is better except Q_{128}); GF columns are not ranked because their counting scopes differ; shaded rows are Execution Tuning, which is fastest among the four methods timed here; the baselines run eagerly (Section 5), and nominal counts use method-specific scopes.

Model	Method	Trainable params	Nominal GF	Impl. GF	Prefill	Decode	Q_{128}
Qwen3-8B	Sparse Upcycling	10,871,930,880	16.308	–	1288.38	23.789	1,900.9
	MixLoRA	118,259,712	16.072	–	1536.12	114.032	511.4
	MoORE	336,328,704	20.776	–	1283.08	54.702	995.3
	Execution Tuning	**4,483,404**	3.624	8.155	**607**	**7.923**	**5,078**
Qwen3-14B†	Sparse Upcycling	10,695,680,000	32.087	–	2351.28	31.975	1,277.6
	MixLoRA	179,896,320	31.138	–	2656.52	126.774	436.7
	MoORE	474,296,320	38.057	–	2136.73	64.138	796.7
	Execution Tuning	**5,964,600**	6.921	15.413	**1158**	**11.096**	**3,191**
Llama-3.1-8B‡	Sparse Upcycling	5,637,275,648	16.912	–	1267.86	21.372	2,057.2
	MixLoRA	117,702,656	16.440	–	1505.85	99.111	581.3
	MoORE	307,871,744	20.097	–	1201.28	50.704	1,072.2
	Execution Tuning	**3,985,248**	3.624	8.053	**587**	**7.233**	**5,441**
Llama-3.2-3B	Sparse Upcycling	4,228,030,464	6.342	–	559.18	17.411	2,957.0
	MixLoRA	63,537,152	6.378	–	711.00	84.591	715.2
	MoORE	195,069,952	8.799	–	592.89	42.018	1,381.7
	Execution Tuning	**2,798,964**	1.453	3.303	**222**	**4.773**	**9,891**
Gemma-2-9B	Sparse Upcycling	12,948,117,504	19.422	–	1518.44	27.721	1,625.7
	MixLoRA	150,183,936	19.197	–	1844.98	132.295	439.3
	MoORE	385,330,176	23.542	–	1467.70	69.185	798.9
	Execution Tuning	**4,714,542**	4.046	8.899	**738**	**9.511**	**4,210**
Gemma-2-27B	Sparse Upcycling	6,115,350,528	49.942	–	3249.56	27.703	1,210.4
	MixLoRA	373,628,928	65.314	–	5170.27	155.907	328.1
	MoORE	631,170,048	63.122	–	3597.75	71.820	644.1
	Execution Tuning	**6,294,042**	13.186	27.838	**2242**	**19.333**	**1,744**

All Execution Tuning decode entries use graph replay; all prior-method entries are eager. †Qwen3-14B Sparse Upcycling uses every-second-layer replacement for latency and FLOPs, while its

trainable-count cell is for every-fourth-layer replacement. ‡Llama-3.1-8B Sparse Upcycling likewise uses every-second-layer replacement for latency and FLOPs, while its trainable-count cell is for the every-fourth-layer replacement used in training.

Table 26: Execution Tuning speedup ratios. The first three columns compare with Sparse Upcycling; the last divides the fastest baseline prefill on each model (MoORE for Qwen, Llama-3.1-8B, and Gemma-2-9B; Sparse Upcycling otherwise) by Execution Tuning prefill. Values greater than one favor Execution Tuning. The same configuration qualifications as Table 25 apply. Green marks a speedup.

	vs. Sparse Upcycling			vs. fastest
Model	Prefill	Decode step	Q_{128}	Prefill
Qwen3-8B	2.12	3.00	2.67	2.11
Qwen3-14B	2.03	2.88	2.50	1.85
Llama-3.1-8B	2.16	2.95	2.64	2.05
Llama-3.2-3B	2.52	3.65	3.34	2.52
Gemma-2-9B	2.06	2.91	2.59	1.99
Gemma-2-27B	1.45	1.43	1.44	1.45

Table 27: Execution Tuning versus the dense pretrained model. Whole decoder block, batch 64, prompt length 512, 128 generated tokens, BF16, one H200; main recipe (64 of 256 candidate groups). TTFT is prefill wall time. Execution Tuning is the seed-0 checkpoint whose accuracy Table 13 reports, with the same bank, group count, and budget; dense and Execution Tuning both run with CUDA graphs in the same harness. Ratios greater than one favor Execution Tuning. The Execution Tuning rows of Table 25 are taken from this run. Green marks a speedup. FLOP bound is the ratio of dense to implemented decoder-block FLOPs (attention plus the MLP factors, router excluded), the prefill speedup that reduced arithmetic alone would give.

	TTFT (ms) ↓			
Model	Dense	Ours	Ratio	FLOP bound
Qwen3-8B	860	607	1.42×	1.24
Qwen3-14B	1,593	1,158	1.38×	1.29
Llama-3.1-8B	830	587	1.41×	1.30
Llama-3.2-3B	352	222	1.59×	1.19
Gemma-2-9B	1,018	738	1.38×	1.32
Gemma-2-27B	3,014	2,242	1.34×	1.58

Against Sparse Upcycling (Tables 25 and 26), the Qwen3-8B pair shows lower prefill time (1,288.38 to 607.00 ms) and lower decode-step time (23.789 to 7.923 ms), producing the $2.67\times$ amortized throughput ratio. This reflects a reduction in execution overhead while preserving dynamic routing. The gain combines reduced arithmetic with fusion and graph capture. Gemma-2-27B shows the smallest gains, $1.45\times$ in prefill and $1.43\times$ per decode step.

What the reported FLOP columns count. Sparse Upcycling counts dense MLP layers at $6HI$ and replaced layers at two full FFNs plus a four-way router; replacement intervals vary by model (Table 18). MixLoRA counts a dense gate/up once and a dense down for both selected experts, top-two rank-eight LoRA gate/up/down experts, an eight-way router, and rank-eight query/value adapters. MoORE counts all seven projections as full thin factors, eight Householder reflections, a 128-wide sample router, and 64-dimensional task-router terms. Execution Tuning counts three MLP selected-factor products at pr in both factors. Router MLP, full-input selection work, attention, and dispatch are omitted from that column.

One common scope. Table 28 counts every method under the same rule: all linear projections of a decoder layer (query, key, value, output, MLP or experts, adapters, and routers), two FLOPs per multiply–add, per token, with attention scores and elementwise work excluded because they are

identical across methods or negligible. Sparse Upcycling uses the replacement interval of its accuracy runs (Table 18), and Execution Tuning computes the full input factor, 25% of the output-factor columns, and its router. The counter reproduces the method-specific counts of Table 25 for Qwen3-8B (16.308, 16.072, and 20.776 GFLOPs for Sparse Upcycling, MixLoRA with its query/value projections, and MoORE) and the FLOP bound of Table 27.

Table 28: Prefill decoder-stack GFLOPs per token under one common scope (all linear projections; attention scores and elementwise work excluded). The last three columns divide dense, the cheapest MoE-based baseline, and the fastest baseline in prefill (Table 26) by Execution Tuning.

Model	Dense	Upcycling	MixLoRA	MoORE	ET	Dense/ET	Cheapest/ET	Fastest/ET
Qwen3-8B	13.89	19.33	17.58	20.78	11.18	1.24	1.57	1.86
Qwen3-14B	26.42	31.77	33.65	38.06	20.46	1.29	1.55	1.86
Llama-3.1-8B	13.96	16.78	17.78	20.10	10.75	1.30	1.56	1.87
Llama-3.2-3B	5.64	7.75	7.08	8.80	4.72	1.19	1.50	1.64
Gemma-2-9B	16.65	23.12	21.05	23.54	12.61	1.32	1.67	1.87
Gemma-2-27B	52.09	55.15	67.92	63.12	33.06	1.58	1.67	1.67

I.4 Optimized Projection Pipeline

Large-prefill output GEMMs can drop groups unused by every token in the batch. Decode retains a dense masked output GEMM. On actual GSM8K prompts, the Qwen3-8B $k = 64$ prefill reads 24.85% of the output-factor columns, matching the nominal $\rho = 25\%$ of Table 28; at $k = 128$ and 146 it reads 49.97% and 56.94%, so the executed fraction tracks k/G, whereas the decode batch union covers about 233 of 256 groups. Dispatch thresholds select between these paths; semantic zero gates alone do not make the reference GEMMs sparse. A faithful skipping backend preserves token-wise selection and nonzero gate magnitudes.

I.5 Numerical Compatibility and Preservation Checks

A common score shift preserves ideal rankings, but not necessarily the scale of an index perturbation. Training and deployment energy floors differ, so parity between backends is defined on selected sets and gate magnitudes, not only on task accuracy. Near-zero logits can change sign under different accumulation orders. These checks are required before treating two backends as the same checkpoint; Table 32 reports them for the fused and eager routers.

I.6 Implementation Optimizations

The H200 implementation uses fusion, vectorized energy/selection/scale kernels, conditional batch-union output GEMMs, and CUDA-graph decode. Alternatives that did not improve speed were not adopted. Graph replay is applied to Execution Tuning and to the dense model; the MoE baselines run eagerly, and the available Sparse Upcycling and MixLoRA implementations synchronize on token-dependent expert loops.

J Method-Specific Ablations

J.1 Full-to-Sparse Ablations

Table 29 reports Soft/Hard/PI-cubic training paths on the Execution Tuning spectral bank, distinct from the MixLoRA intervention of Section 2.2. Soft keeps every group active; Hard applies a strict top-64 mask from the first step, selecting by the gate logit without the energy prior; PI cubic uses the PI-controller variant with a cubic target annealed to 25% activity. The PI-cubic Qwen3-8B/GSM8K run (81.12%) is a separate ablation run, so it differs from the three-seed mean in Table 13 (80.44%); its observed activity is approximately 21%. Appendix J.3 separates the mask from the controller.

Table 29: Spectral-bank training-path ablations (accuracy, %). Soft = all groups active; Hard = strict top-64 mask from the first step, without the energy prior; PI cubic = PI-controller variant with a cubic target to 25% activity; PI−Hard is their gap. Single run per cell; checkpoint with the lowest held-out loss, except the Qwen3-8B/GSM8K cells, which are the final-checkpoint runs without a holdout of Appendix J.3. These runs use their own training settings, which differ from the main runs of Table 13 (for example, Soft is below the 25% main recipe on Llama-3.1-8B and Gemma-2-9B), so compare columns within a row, not with Table 13. Bold is best and underline second best per row.

Task	Model	Soft	Hard	PI cubic	PI−Hard
GSM8K	Qwen3-8B	<u>80.36</u>	0.83	**81.12**	+80.29
GSM8K	Llama-3.1-8B	<u>45.79</u>	2.12	**49.73**	+47.61
GSM8K	Gemma-2-9B	<u>63.46</u>	0.08	**67.63**	+67.55
MATH	Qwen3-8B	**49.86**	0.60	<u>46.06</u>	+45.46
MATH	Llama-3.1-8B	<u>14.88</u>	0.02	**15.64**	+15.62
MATH	Gemma-2-9B	**31.12**	0.04	<u>29.54</u>	+29.50

These runs use training settings that differ from the main runs, so their absolute values are not comparable with Table 13. Hard ranks candidates by the gate logit without the energy prior and PI cubic uses the PI-controller variant, so the PI−Hard gap reflects selection score and controller together, not the path alone (Appendix J.3). PI cubic remains below Soft for Qwen and Gemma on MATH.

J.2 Gate Ablation: Selection versus Reweighting

To separate selection from reweighting, we keep the training recipe and the top-k candidate budget fixed and change only the map from the gate logit to the gate. (a) ReLU is the main gate, $\alpha_g = \mathrm{ReLU}(1 + s\ell_g)$, so a selected group can be amplified. (b) Binary sets $\alpha_g = 1$ on the selected set, with a straight-through gradient for the selection score. (c) Clip uses $\alpha_g = \mathrm{clip}(1 + s\ell_g, 0, 1)$, so a selected group can only shrink. (d) Energy selects the energy top-k with $\alpha_g = 1$ and trains no router. At 100% density, Binary and Energy both equal the pretrained model, so these two cells are a single evaluation of the base model. We run Qwen3-8B/GSM8K at 100%, 50%, and 25% density ($k = 256, 128, 64$) and Llama-3.1-8B/MATH at 25%, with one training seed (0). Qwen uses BF16, learning rate 2×10^{-4}; Llama uses FP32, learning rate 10^{-4}; both train for three epochs at effective batch 64, select the checkpoint with the lowest held-out loss, and evaluate zero-shot on the full test set. Gate statistics are measured on the selected groups of the first 64 training rows. All gate variants in a column are trained with this same recipe; the 25% ReLU cells are the seed-0 runs of the main campaign (Table 14), so the Binary and Clip cells are compared against the same checkpoints reported there. The unadapted Llama-3.1-8B scores 1.00% on MATH.

Table 30: Gate ablation accuracy (%). Only the gate map differs; the candidate budget is the same. *At 100% density, Binary and Energy are the unadapted pretrained model. Bold is best and underline second best per column; the shaded row is the main gate.

	Qwen3-8B / GSM8K			Llama-3.1-8B / MATH
Gate	100%	50%	25%	25%
(a) ReLU (main)	**82.11**	**80.82**	**80.44**	**17.82**
(b) Binary, trained selection	66.11*	63.08	6.52	1.04
(c) Clip to $[0, 1]$	<u>80.21</u>	<u>80.67</u>	<u>73.62</u>	<u>13.12</u>
(d) Energy top-k, no training	66.11*	68.54	68.69	1.14

Table 31: Gate magnitudes on selected groups, measured on the first 64 training rows of an earlier single run of each configuration; the 25% ReLU rows therefore come from runs other than the seed-0 checkpoints of Tables 30 and 14, whose magnitudes were not re-measured. Binary and Energy gates equal one by construction.

Setting	Gate	Mean α	Max α	Fraction $\alpha > 1$
Qwen3-8B, 100%	ReLU	1.002	6.49	8.5%
Qwen3-8B, 50%	ReLU	1.020	17.44	15.6%
Qwen3-8B, 25%	ReLU	1.064	13.95	28.5%
Llama-3.1-8B, 25%	ReLU	1.006	4.09	37.0%
Qwen3-8B, 100% / 50% / 25%	Clip	0.988 / 0.982 / 0.966	1.00	0
Llama-3.1-8B, 25%	Clip	0.990	1.00	0

Binary gates fall far below ReLU in every sparse cell: by 17.7 points at 50% and 73.9 at 25% on Qwen3-8B/GSM8K, and by 16.8 points on Llama-3.1-8B/MATH; at 100% density the binary cell is the unadapted model, 16.0 points below ReLU. Selection alone therefore does not adapt the model. At 100% density the support is the full set, so the 16.0-point gain of ReLU over the pretrained model comes entirely from magnitudes; 8.5% of its gates exceed one. Clip nearly matches ReLU at 50% density (-0.15) and trails by 1.9 points at 100%, but the gap widens at 25%: 6.8 points on GSM8K and 4.7 on MATH. Shrinking suffices at half density, whereas the tight budget benefits from amplification, and the fraction of amplified ReLU gates grows as the budget tightens. The untrained energy top-k is a weak floor, and trained binary selection falls below it at 25% on GSM8K. In this single-seed ablation, adaptation therefore comes mainly from reweighting the selected spectral groups, and bounded or binary gates perform worse than the unbounded ReLU gate.

Evaluation-only checks. Table 32 probes the reported Qwen3-8B/GSM8K seed-0 checkpoint without retraining. Fixed supports keep the learned gates; the mean-gate arm keeps the per-token support and replaces each gate by its mean over 512 training prompts, which also define the most frequent groups.

Table 32: Evaluation-only checks on the reported Qwen3-8B/GSM8K seed-0 checkpoint (main-campaign run, step 345; 1,319 problems; vLLM, zero-shot), with flexible and strict-format answer extraction. No arm is retrained. Table 14 reports the fused-kernel evaluation of this checkpoint (1,061, 80.44%); the reference row re-evaluates it with the eager router. The untrained energy row is likewise a re-evaluation of the arm reported as 906 correct (68.69%) in Table 30. The shaded row is the deployed configuration.

	Flexible		Strict format	
Arm	Correct	%	Correct	%
Per-token energy top-64, learned gates (reference)	1,060	80.36	1,060	80.36
Same, fused router kernel	1,061	80.44	1,061	80.44
Fixed support: 64 most frequent groups per projection	1,072	81.27	823	62.40
Fixed support: 64 top-σ groups per projection	548	41.55	182	13.80
Per-token support, gates replaced by their mean	838	63.53	9	0.68
Untrained: energy top-64, unit gates	905	68.61	0	0.00
Untrained: random 64 groups, unit gates	0	0.00	0	0.00

Against the reference, the frequent-group support is right on 93 problems where the reference is wrong and wrong on 81 where it is right ($\Delta = +0.91$ points, paired 95% CI $[-1.05, 2.87]$), so no loss is detected under flexible extraction, but equivalence within one point is not established. Under strict-format extraction the same arm keeps 823 correct answers against 1,060 for the reference: the frozen support preserves the extracted answers but not the answer format. The mean-gate arm keeps 9 strict-format answers, so its 63.53% flexible score does not mean that its outputs remain well formed. The top-σ support and the mean gates have paired confidence intervals entirely below zero: the learned support is not a static singular-value truncation, and the magnitudes depend on the token.

Trained token versus global magnitudes. The mean-gate arm replaces magnitudes only at evaluation. As a trained control, we fix a static support of 64 groups per projection before fine-tuning, chosen from the frozen base model on 512 calibration examples (no trained checkpoint is used), slice both factors to it, and train either token-conditioned magnitudes, with a router that reads only the selected coordinates, or one global magnitude per group, as in SVF (Sun et al., 2025) (Qwen3-8B/GSM8K, single run). With a base-energy support, the best held-out loss is 0.629 with token-conditioned magnitudes and 2.955 with global ones; with a top-σ support, 0.631 and 2.983. With the dynamic energy support, global magnitudes reach 1.143. Magnitudes that do not depend on the token therefore do not fit the task, whichever support is used.

The fused router kernel matches the eager router's logits to about 10^{-8} and selects the same top-64 on the projections checked; over GSM8K it changes 49 answers in both directions, for a net difference of one problem, so greedy re-evaluation alone moves individual answers and single-evaluation differences of about one point should not be read as effects. The untrained energy selection scores 905 here and 906 in Table 30, a one-problem difference from nondeterministic evaluation; random selection produces no well-formed answers.

J.3 Training Schedule: Strict Mask versus PI-Controller Variant

These runs use Qwen3-8B/GSM8K, seed 0, all 7,473 training examples without an internal holdout, and the final checkpoint (step 351), evaluated on the full 1,319-problem test set with flexible and strict-format answer extraction. The main campaign instead holds out 128 examples, trains for 345 steps, and selects the checkpoint on held-out loss (Table 16); the Soft run here (80.36%) is therefore a different run from the all-on ReLU cell of Table 30 (82.11%), which selects on held-out loss. They share MLP-only adaptation, $G = 256$, the $r \to 8 \to 32 \to 256$ router, learning rate 2×10^{-4} with cosine decay, effective batch 64, and three epochs (351 steps) unless stated. Table 33 applies a strict top-64 mask with a ReLU gate, selecting groups by the gate logit alone without the energy prior: from the first step (Hard), with a cubic schedule (Strict, cubic), or after all groups stay active for a fraction of training (Abrupt); Prune cuts the trained Soft model to top-64 only at evaluation. Table 34 instead uses the PI-controller variant, whose target activity stays at 0.95 and jumps to 0.25 at the listed fraction of steps; Full→Sparse anneals the target cubically to 0.25 by 60% of training, and Fixed-target holds 0.25 from the first step.

Table 33: Strict top-64 mask (GSM8K, %).

Arm	Flexible	Strict-format
Soft, all groups	80.36	80.44
Hard, from step 0	0.83	1.06
Strict, cubic	0.38	0.99
Abrupt at 50%	0.30	0.00
Abrupt at 70%	0.76	0.00
Abrupt at 80%	0.76	0.00
Prune at eval	0.23	0.00
Hard at step 0, untrained	0.00	0.00

Table 34: PI-controller variant (GSM8K, %). Bold is best, underline second.

Arm	Flexible	Strict-format
Jump at 40%	**82.18**	**82.18**
Jump at 50%	<u>81.65</u>	<u>81.65</u>
Jump at 70%	81.58	<u>81.65</u>
Full→Sparse, cubic	81.12	81.05
Fixed-target	81.20	81.20
Soft, all groups	80.36	80.44

A strict top-64 mask stays below 1.1% whether it is applied from the first step, annealed, or delayed until 80% of training, and cutting the trained Soft model at evaluation gives 0.23%; the untrained network under the mask already scores 0/1,319. At initialization the gate logit is zero for every group, so the mask selects no informative groups; with the energy prior, the same untrained top-64 selection scores 68.69% (Table 30). With the PI-controller variant, which uses no energy prior, every schedule reaches 81.1–82.2%, within evaluation noise of Soft (80.36%), and the best arm, a jump at 40% of training, reaches 82.18%. With one seed the evaluation standard error is about 1.1 points, so the jump, cubic, and fixed-target arms are level. Ranking candidates by the gate logit alone therefore fails under every schedule, whereas adding the energy prior makes the strict budget trainable: the main recipe reaches 80.44% (Tables 30 and 13). Sparsity reached through the PI-controller variant trains well under every tested schedule.

Table 35: Variations of the jump-at-40% PI-controller arm (Qwen3-8B/GSM8K, seed 0, final checkpoint without a holdout, %). Residual LoRA adds an always-on low-rank term to each projection, initialized to zero. Bold is best, underline second; the shaded row is the reference arm.

Change	Steps	Flexible	Strict-format
None (router $8 \to 32$)	351	**82.18**	**82.18**
Router $16 \to 32$	351	<u>81.05</u>	81.05
Residual LoRA, rank 8	351	<u>81.05</u>	<u>81.20</u>
Residual LoRA, rank 16	351	80.97	80.97
Router $16 \to 32$, residual rank 16	351	80.82	80.82
Six epochs, router $32 \to 64$	702	80.67	80.67
Residual LoRA, rank 32	351	79.38	79.38
Six epochs, router $16 \to 32$	702	79.53	79.53
Six epochs, router $16 \to 32$, cubic	702	79.45	79.45
Eight epochs, router $16 \to 32$	936	78.01	78.01

No variation improves on the three-epoch jump arm. Longer training lowers accuracy from 82.18% at three epochs to 79.53% at six and 78.01% at eight, and a wider router or an added residual LoRA stays within 79–81%. In this single-seed test on Qwen3-8B/GSM8K, no variation improved on the reference arm, so we see no sign that the small router or the absence of new adapter weights limits capacity in this setting.

J.4 Complete Density-by-Group-Count Ablation

The ablation evaluates Qwen3-8B on all 1,319 GSM8K test problems with zero-shot flexible exact match. It uses three epochs, BF16, learning rate 2×10^{-4}, effective batch 64, MLP-only adaptation, and energy-prior deployment. Every cell is the mean over three training seeds (0, 1, 42); each run uses the checkpoint with lowest held-out validation loss and then one final test evaluation. The $G = 256$, $p = 25\%$ cell is the Table 13 result; the other 15 cells are trained separately with the same recipe.

With equal-rank groups, fixed $r = 4096$, and candidate count $k = pG$, the selected rank is $k(r/G) = pr$. The analytic selected-factor count is

$$F_{\text{selected}}(p) = \frac{6p\,(4096)\,(4096 + 12288)\,(36)}{10^9} = 14.495514624\,p \tag{14}$$

GFLOPs/token. It is the same for different G at a fixed p. It omits router, energy-selection, output-union, and attention costs. A 75% reduction from $p = 1$ to $p = 0.25$ in this selected-factor count is not a 75% reduction from the original dense transformer's executed FLOPs.

Table 36: Full Qwen3-8B/GSM8K density × group-count grid. Entries are accuracy percentages, averaged over three training seeds. *Same runs as Table 13. Darker cells are more accurate; bold is the best cell.

Groups G	100%	50%	25%	12.5%
32	81.58	78.85	68.01	10.24
64	81.80	79.30	75.82	30.63
128	**82.18**	81.73	79.38	56.41
256	81.96	81.58	80.44*	68.46

Table 37: One-dimensional slices. Left fixes $G = 256$; right fixes density at 25%. GF denotes GFLOPs/token from Equation 14, not measured GPU work.

Density	Accuracy	GF	Groups	Accuracy	GF
100%	81.96	14.496	32	68.01	3.624
50%	81.58	7.248	64	75.82	3.624
25%	80.44*	3.624	128	79.38	3.624
12.5%	68.46	1.812	256	80.44*	3.624

At $G = 256$, reducing density from 100% to 50% changes accuracy by -0.38 points; reducing it to 25% changes it by -1.52 points. The further reduction from 25% to 12.5% loses 11.98 points. At 25% density, moving from 32 to 256 groups increases accuracy by 12.43 points, whereas moving from 128 to 256 adds only 1.06 points. At 12.5%, the corresponding 32-to-256 increase is 58.22 points. By comparison, the 100% column spans only 0.60 points. These observations are consistent with finer groups making severe selection less destructive. They do not prove that increased expert specialization is the cause: changing G also changes router capacity (Table 38). The trained 100% case is all-on reweighting, not necessarily the unadapted dense model. The 80.44% cell comes from the main recipe, whereas the 81.12% PI-cubic result in Table 29 is a separate ablation run.

Table 38: Analytical quantities that change in the group-count sweep, derived from the stated $r \to 8 \to 32 \to G$ router and 108 projections, including one learned prior scalar per projection.

Groups	Rank/group	Candidates at 25%	Selected rank	Router parameters
32	128	8	1,024	3,685,068
64	64	16	1,024	3,799,116
128	32	32	1,024	4,027,212
256	16	64	1,024	4,483,404