BOOSTLORA: GROWING EFFECTIVE RANK BY BOOSTING ADAPTERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Parameter-efficient fine-tuning (PEFT) methods face a tradeoff between adapter size and expressivity: ultra-low-parameter adapters are confined to fixed low-rank subspaces, capping performance even with extended training. We propose BoostLoRA, a gradient-boosting framework that overcomes this limit by iteratively training and merging minimal adapters on the examples the current model gets wrong. A ROTATE SVD basis strategy assigns each round to an orthogonal subspace, so cumulative effective rank grows linearly with the number of rounds while each adapter remains ultra-low-rank. After merging, adapters are discarded, leaving zero inference overhead. On Qwen2.5-3B, BoostLoRA reaches 89.1% on GSM8K and 68.8% on MATH-500, surpassing both the best single-shot ultra-low-parameter adapter (TinyLoRA) and full fine-tuning; on code generation it reaches 57.2% on MBPP and 80.4% on HumanEval while full fine-tuning drops below the zero-shot baseline. We also demonstrate general applicability across domains via protein binding classification with ESM2-650M and cross-entropy training. BoostLoRA is, to our knowledge, the first ultra-low PEFT method whose effective rank grows with training, separating per-round parameter cost from total representational capacity.

1 INTRODUCTION

Parameter-efficient fine-tuning enables large language model adaptation without updating billions of weights. LoRA (Hu et al., 2021) injects trainable low-rank matrices into frozen layers; follow-up work has pushed the Pareto frontier further through shared random projections (Kopiczko et al., 2024), SVD-informed initialization (Meng et al., 2025; Zhang et al., 2023), and weight tying (Bałazy et al., 2025). TinyLoRA (Morris et al., 2026) is the lowest-parameter point on this frontier, reducing trainable parameters to as few as one per module by projecting a low-dimensional vector through fixed random matrices within the SVD subspace of each weight matrix. Combined with reinforcement learning (RL) (Shao et al., 2024), a 16-parameter TinyLoRA adapter improves Qwen2.5-3B from 76% to 81% on GSM8K, showing that RL is far more parameter-efficient than supervised fine-tuning (SFT) at this scale. However, individual ultra-low-parameter adapters hit an *expressivity limit* (Yin et al., 2025). Each adapter is confined to a fixed-rank subspace, and adding more parameters yields diminishing returns beyond a threshold. TinyLoRA's best configuration (8,064 parameters) plateaus at 87.2% on GSM8K, within a point of full fine-tuning (87.0%).

We propose **BoostLoRA**, a PEFT framework based on gradient boosting (Friedman, 2001). BoostLoRA trains T sequential TinyLoRA adapters, each with ultra-low parameter budgets, on the examples the current model gets wrong. After each round, the adapter is merged into the base weights and discarded, leaving no inference overhead. The core mechanism is a ROTATE SVD basis strategy that assigns each round's adapter to an orthogonal subspace. The cumulative weight update reaches effective rank $\min(rT, d, k)$ where each adapter is rank r. With $r = 2$ and $T = 20$ rounds, BoostLoRA builds the capacity of a rank-40 update while each round optimizes over a 2×2 *transformation matrix*.

On GSM8K, BoostLoRA reaches **89.1%**, beating TinyLoRA's best (87.2% at 8,064 parameters) and full fine-tuning (87.0% at 3.09B parameters). On MATH-500 it reaches 68.8%, and on code generation it achieves 57.2% on MBPP and 80.4% on HumanEval. We also test BoostLoRA outside decoder models and generation: on PPB-Affinity (Liu et al., 2024a) protein-protein binding classification with ESM2-650M (Lin et al., 2022) and cross-entropy training. The BoostLoRA framework improves

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.

over single-shot TinyLoRA throughout the ultra-low and mid-range parameter settings, while both degrade at the 129K-parameter setting.

We contribute:

1. **BoostLoRA**, a gradient-boosted PEFT framework that resolves expressivity limits of ultra-low-parameter adapters by sequentially training and merging TinyLoRA adapters on failures (§3).
2. A **ROTATE SVD basis strategy** that ensures linear effective rank growth ($\leq rT$) across boosting rounds, with empirical confirmation that the bound is tight (§3.3, §5).
3. An analysis of gradient isolation from correct examples (§3.2), exact rank growth (Theorem 1), and a non-vacuous generalization bound (Theorem 2), which we validate empirically (App. D).
4. Results across math, code, and protein tasks on both decoder and encoder models, with both GRPO and cross-entropy training (§5).

2 Related Work

Parameter-efficient fine-tuning. LoRA (Hu et al., 2021) adapts frozen weights with low-rank updates. AdaLoRA (Zhang et al., 2023) allocates rank by importance, LoRA-XS (Bałazy et al., 2025) learns a small core between frozen SVD factors, VeRA (Kopiczko et al., 2024) shares random projections, DoRA (Liu et al., 2024b) separates magnitude and direction, PiSSA (Meng et al., 2025) initializes principal SVD directions, and TinyLoRA (Morris et al., 2026) uses a trainable vector in a fixed SVD subspace. These methods use a single adapter whose update rank is fixed at creation; BoostLoRA instead accumulates disjoint low-rank updates across rounds.

Iterative and boosted adaptation. CoLA (Xia et al., 2024) is the closest prior method that sequentially merges LoRA adapters, and we compare against it directly in Appendix E. In our matched implementation, CoLA accuracy generally peaks within the first three merges and does not improve consistently with chain length (Table 6). Its configurations remain below BoostLoRA, suggesting that iterative merging alone saturates without explicitly disjoint subspaces.

MELoRA (Ren et al., 2024) increases effective rank by stacking mini-LoRA modules within one training round. Model Soups (Wortsman et al., 2022) and LoRA Soups (Prabhakar et al., 2024) average independently trained adapters for skill composition rather than sequential residual fitting. ReLoRA (Lialin et al., 2023) periodically merges and restarts standard-rank LoRA during pre-training, but uses millions of parameters per round, trains on the full dataset, and does not enforce orthogonality across rounds.

XGBLoRA (Zhang et al., 2024) also applies gradient boosting to rank-1 LoRA adapters. However, its adapters use approximately 2.4M parameters per round on LLaMA3-8B, compared with BoostLoRA's 12 parameters; it uses SFT, which TinyLoRA (Morris et al., 2026) found substantially less parameter-efficient than RL at scale; and it evaluates commonsense reasoning rather than math or code.

HRA (Yuan et al., 2024) parameterizes each frozen weight matrix with a chain of learnable Householder reflections, but is a single-run method rather than sequential failure-set adaptation. O-LoRA (Wang et al., 2023) keeps low-rank subspaces orthogonal across tasks to reduce forgetting. In contrast, BoostLoRA assigns disjoint SVD blocks across rounds of the same task, using orthogonality to grow within-task capacity.

RL for reasoning. GRPO (Shao et al., 2024) estimates advantages from within-group reward statistics without a PPO-style critic. DeepSeek R1 (Guo et al., 2025), SimpleRL (Zeng et al., 2025), and RLOO (Ahmadian et al., 2024) demonstrate that pure or simplified RL can produce strong reasoning behavior without an SFT stage or complex PPO variants. Under these dynamics, ultra-low-parameter adapters face an *expressivity limit*: VeRA and rank-1 LoRA fail to learn reasoning, while SVD-based initializations such as PiSSA can undergo spectral collapse (Yin et al., 2025). BoostLoRA addresses this limit by accumulating effective rank across sequential ultra-low-parameter adapters.

Gradient boosting theory. Gradient boosting (Friedman, 2001) greedily adds weak learners that fit the current residual. Its margin theory (Freund & Schapire, 1997; Schapire et al., 1997) relates generalization to the margin distribution rather than ensemble size. XGBoost (Chen & Guestrin, 2016)

scaled this principle to large-scale tree learning, while BoostResNet (Huang et al., 2018) established exponentially decreasing training error for sequential residual blocks under a weak-learning condition. BoostLoRA applies this framework to PEFT: TinyLoRA adapters are weak learners, and ROTATE grows their cumulative capacity linearly across rounds.

Protein language models and binding affinity. ESM2 (Lin et al., 2022) is a protein language model family trained on evolutionary sequences. PPB-Affinity (Liu et al., 2024a) aggregates protein-protein binding affinities from structural databases and has primarily been studied as a regression task (best Pearson $r = 0.701$ (Liu et al., 2024a); Spearman $\rho = 0.48$ (Alsamkary et al., 2025)). To evaluate BoostLoRA on an encoder-only model with cross-entropy training, we binarize the task at $K_D < 500\,\text{nM}$.

3 METHOD

BoostLoRA repeatedly trains and merges minimal adapters with ultra-low parameter budgets to incrementally improve a frozen underlying model. Each round (i) identifies the examples the current model gets wrong, (ii) trains a fresh TinyLoRA adapter on that failure set via GRPO (for generation tasks) or cross-entropy (for classification), and (iii) merges the adapter into the base weights.

3.1 TINYLORA ADAPTER PARAMETERIZATION

For each linear layer W that is $d \times k$ dimensions in the base model, we compute a rank-r truncated SVD, by $W \approx U\Sigma V^\top$. The adapter introduces a weight update

$$\Delta W = U\,\Sigma\,R\,V^\top, \qquad R = \sum_{i=1}^{u} v_i\,P_i, \tag{1}$$

where P_i are fixed $r \times r$ random projection matrices, and $v \in \mathbb{R}^u$ is the *only* trainable vector. v selects a linear combination of random rotations within the rank-r subspace of W, modulated by the singular values Σ.

Following Morris et al. (2026), we use block-level weight tying: the adapted modules (e.g., 252 for Qwen2.5-3B, 198 for ESM2-650M) are partitioned into g tying groups using a tiled assignment, and all modules within a group share the same v. With $g = 4$ groups and projection dimension $u = 3$, the total number of trainable parameters per boosting round is $g \times u = 12$.

3.2 GRADIENT BOOSTING WITH TINYLORA WEAK LEARNERS

Algorithm 1 summarizes the procedure. Let W_0 denote the original pretrained weights. At round t:

1. **Evaluate.** Run the current model $\mathcal{M}$ on the full training set and collect the *failure set*, $\mathcal{F}_t = \{(x, y) : \mathcal{M}(x) \neq y\}$.
2. **Train.** Initialize a fresh TinyLoRA adapter by setting $v \leftarrow 0$ and train it via GRPO or cross-entropy on $\mathcal{F}_t$.
3. **Merge.** Compute $\Delta_t = U\Sigma R_t V^\top$ and fold it into the base weights, $W_t \leftarrow W_{t-1} + \Delta_t$.
4. **Accumulate.** Track the cumulative update for effective rank measurement: $\bar{\Delta}_t \leftarrow \bar{\Delta}_{t-1} + \Delta_t$.

After merging, the adapter is discarded and the model runs at its original inference cost, with no adapter overhead. This parallels gradient boosting (Friedman, 2001), where each TinyLoRA adapter is a weak learner that fits the ensemble's current residual (the failure set).

Dynamic residuals. Merging Δ_t changes the underlying model and can both correct previous failures and introduce new ones. Consequently, we recompute $\mathcal{F}_t$ from the current model at every round. BoostLoRA follows the stage-wise additive structure of gradient boosting, while its residual is defined by the round's task-level correctness rather than a fixed output-space residual.

Algorithm 1 BoostLoRA: Gradient-boosted TinyLoRA

Require: Pretrained model $\mathcal{M}$ with weights W_0, training set $\mathcal{D}$, SVD rank r, rounds T, basis strategy $\in \{\text{TOP}, \text{ROTATE}\}$

Initialize cumulative delta $\bar{\Delta} \leftarrow 0$
for $t = 1$ **to** T **do**
  $\mathcal{F}_t \leftarrow \{(x, y) \in \mathcal{D} : \mathcal{M}(x) \neq y\}$ ▷ Identify failures
  if $|\mathcal{F}_t| <$ threshold **then**
    break ▷ Early stop
  end if
  Select SVD components (U_t, Σ_t, V_t) using basis strategy and round t
  $v_t \leftarrow \text{GRPO}(\mathcal{M}, \mathcal{F}_t, U_t, \Sigma_t, V_t)$ ▷ Train adapter
  $\Delta_t \leftarrow U_t \Sigma_t R(v_t) V_t^\top$ ▷ Compute weight update
  $W_t \leftarrow W_{t-1} + \Delta_t$ ▷ Merge into base
  $\bar{\Delta} \leftarrow \bar{\Delta} + \Delta_t$ ▷ Track cumulative change
end for
return $\mathcal{M}$ with weights W_T

Gradient isolation. Each round trains only on $\mathcal{F}_t$, so its gradient contains no direct contribution from currently correct examples. Merging can nevertheless change any prediction: a correct example regresses only if the update overcomes its logit margin. Empirically, the small per-round updates ($\|\Delta W_t\|_F \approx 0.01$) cause approximately 2% regression of previously correct examples per round.

3.3 SVD BASIS STRATEGIES AND EFFECTIVE RANK GROWTH

The choice of which singular vectors to use in each round determines the *effective rank* of the cumulative weight update $\sum_t \Delta_t$. We consider two strategies:

1. **Top basis.** Every round computes the top-r singular vectors of the current weight matrix W_{t-1}. Because the adapter norm $|\Delta_t|_F \approx 0.005$–$0.013$ is small relative to $|W|_F$, the top singular vectors change minimally between rounds. Successive adapters therefore operate in the same rank-r subspace, and the cumulative update remains approximately rank r regardless of T.
2. **Rotate basis.** Before training, we compute the full SVD on the pretrained weight matrix W_0 once. Round t then uses singular vectors $r(t-1)+1$ through rt, corresponding to the SVD offset $r \cdot (t-1)$. Because all rounds index disjoint blocks from the same fixed SVD, their left and right subspaces are mutually orthogonal. This ensures that

$$\text{rank}\Big(\textstyle\sum_{t=1}^{T} \Delta_t\Big) \;=\; \min(rT,\; d,\; k). \tag{2}$$

   Caching the SVD outside the boosting loop produced final performance within the same confidence interval and preserved an ε-rank of 40. Rollout generation dominates runtime, this optimization reduced wall-clock time by only 0.3-0.6%.

Effective rank measurement. We track the cumulative delta $\bar{\Delta}_t = \sum_{s=1}^{t} \Delta_s$ in float32 (to avoid bf16 rounding noise that inflates apparent rank) throughout training and report two rank measures: (i) the *participation ratio* $\rho = (\sum_i \sigma_i)^2 / \sum_i \sigma_i^2$, which equals 1 for a rank-1 matrix and $\min(d, k)$ for uniform singular values σ_i; and (ii) the *ε-rank*, the count of singular values exceeding $\varepsilon \cdot \sigma_1$ with $\varepsilon = 0.01$.

3.4 TRAINING PROCEDURE

Each round trains the adapter using GRPO (Shao et al., 2024) for generation tasks or cross-entropy for classification tasks. For GRPO, we sample a group of $G = 8$ completions per prompt x in the failure set, score them with a binary reward function (exact-match for math, sandboxed execution for code), and compute advantages via within-group normalization:

$$\hat{A}_i = \frac{r_i - \bar{r}}{\sigma_r + \epsilon}, \qquad \bar{r} = \frac{1}{G} \sum_{j=1}^{G} r_j. \tag{3}$$

The policy gradient uses clipped importance weights with $\epsilon_c = 0.2$, plus an optional KL penalty toward the reference policy. We use AdamW with cosine learning rate (LR) scheduling (warmup ratio 0.1) and gradient clipping at 1.0, training for three epochs per round.

3.5 THEORETICAL PROPERTIES

We establish two formal results: the ROTATE basis achieves exact rank growth (Theorem 1), and the generalization penalty from T rounds is bounded by the total adapter norm, which grows sublinearly due to self-limiting dynamics (Theorem 2).

Theorem 1 (Exact Rank Growth)**.** *For BoostLoRA with* ROTATE *basis, SVD rank r, and T rounds ($rT \leq \min(d, k)$), if each round's mixing matrix $R_t = \sum_i (v_t)_i P_i$ has rank r, the cumulative rank is:* $\operatorname{rank}\left(\sum_{t=1}^{T} \Delta_t\right) = rT$.

Proof. Round t uses left singular vectors $r(t-1)+1$ through rt as columns of U_t, and the corresponding right singular vectors as columns of V_t, from the full SVD of W_0. Since $\Delta_t = U_t \Sigma_t R_t V_t^\top$, we have $\operatorname{col}(\Delta_t) \subseteq \operatorname{col}(U_t)$ and $\operatorname{row}(\Delta_t) \subseteq \operatorname{col}(V_t)$. Because U_t and V_t use distinct columns of orthogonal matrices, both column and row spaces are mutually orthogonal across rounds, so ranks add: $\operatorname{rank}(\sum_t \Delta_t) = \sum_t \operatorname{rank}(\Delta_t) = rT$.

With TOP basis, all rounds reuse the same columns U_1, so $\operatorname{rank}(\sum_t \Delta_t) \leq r$ regardless of T. The empirical ε-rank in Figure 3 confirms this, showing ROTATE reaches rank 40 at round 20 while TOP stays at $\approx$2.

To prove generalization bounds, for each input x, choose a finite, fixed evaluation set of possible outputs $\mathcal{Y}(x) = \mathcal{Y}^+(x) \cup \mathcal{Y}^-(x)$, with $\mathcal{Y}^+(x)$ the outputs the task's reward marks correct and $\mathcal{Y}^-(x)$ those it marks incorrect. Let $s_T(x, y)$ be the model's log-probability of output y after T rounds. The margin $m_T(x) = \max_{y \in \mathcal{Y}^+} s_T(x, y) - \max_{y \in \mathcal{Y}^-} s_T(x, y)$ is the gap between the best correct output and the best incorrect one. Define a model error when $m_T(x) \leq 0$. Merging round t's adapter perturbs each adapted layer's output by $\Delta W_\ell \, h_\ell(x)$, where $h_\ell(x)$ is the input to layer ℓ. To first order, the resulting change in margin is $\Delta m(x) \approx \sum_\ell g_\ell(x)^\top \Delta W_\ell \, h_\ell(x)$, where $g_\ell(x)$ is the gradient of the margin with respect to layer ℓ's output. Substituting $\Delta W_\ell = U_\ell \Sigma_\ell R_t V_\ell^\top$ with $R_t = \sum_i (v_t)_i P_i$ from Eq. 1 gives $\Delta m(x) \approx \sum_i (v_t)_i \sum_\ell g_\ell(x)^\top U_\ell \Sigma_\ell P_i V_\ell^\top h_\ell(x) = \langle v_t, \phi_t(x) \rangle$, which defines the feature map $\phi_t(x) \in \mathbb{R}^u$. Each round is thus a linear predictor in its trainable vector v_t. Write $\|v_t\|_2 \leq B_t$, $B_{\text{total}} = \sum_t B_t$, and $X = \max_{t,x} \|\phi_t(x)\|_2$.

Theorem 2 (Generalization Bound)**.** *For any $\theta > 0$ and $\delta > 0$, with probability $\geq 1 - \delta$ over a training set of size n,*

$$\Pr_{\text{test}}[\text{error}] \;\leq\; \Pr_{\text{train}}[m_T \leq \theta] \;+\; \frac{2X B_{\text{total}}}{\theta \sqrt{n}} \;+\; \sqrt{\frac{\log(2/\delta)}{2n}}. \tag{4}$$

The proof in Appendix C relies on each round's adapter being a linear function of v_t, as shown above, and bounds the Rademacher complexity of the ensemble via subadditivity. The bound depends on B_{total}, not T directly. Because adapter norms decay as failures shrink (Figure 4), B_{total} grows sublinearly in T, so additional rounds do not significantly hurt generalization. Under ROTATE the orthogonal per-round blocks tighten the complexity term by up to $\sqrt{T}$ relative to TOP. See Appendix C for a proof and Appendix D for numerical evaluation.

4 EXPERIMENTS

Models. Math and code experiments use Qwen2.5-3B-Instruct (QwenTeam, 2024), a 36-layer decoder with hidden dimension 2,048 and grouped-query attention (16 query heads, 2 KV heads). All 252 linear projections are adapted. Protein experiments use ESM2-650M (Lin et al., 2022), a 33-layer encoder with 1,280 hidden dimension trained on evolutionary protein sequences; all 198 linear projections are adapted. Unless otherwise noted: SVD rank $r = 2$, $g = 4$ tying groups, projection dimension $u = 3$, yielding 12 trainable parameters per adapter per round for $T = 20$ boosting rounds.

Benchmarks and baselines. We evaluate on five benchmarks across three domains: (1) *Math*: GSM8K (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2023); (2) *Code*: MBPP (Austin et al., 2021) for training and evaluation, and HumanEval (Chen et al., 2021) as a held-out cross-benchmark, following the protocol of Dou et al. (2024); Jain et al. (2025); and (3) *Proteins*: PPB-Affinity (Liu et al., 2024a), a protein-protein binding dataset. We formulate it as binary classification using the common threshold $K_D < 500$ nM to denote binding.

We compare against four baselines: (i) the pretrained base model (zero-shot), (ii) TinyLoRA (Morris et al., 2026) at 16, 252, 8,064, and 129,024 parameters (single adapter, single round), (iii) full fine-tuning at 3.09B parameters for Qwen-Instruct and 651M for ESM2, and (iv) for protein only, a frozen-ESM2 linear probe (logistic regression on mean-pooled embeddings, 1,281 parameters). Math baselines are taken from Morris et al. (2026).

Ablations. We study three axes: (i) SVD basis (TOP vs. ROTATE) on GSM8K and MATH-500, shown in Table 3; (ii) a monolithic rank-40 adapter (r=40, T=1) that matches ROTATE's total subspace in one round; (iii) parameter scaling on PPB-Affinity for both TinyLoRA and BoostLoRA at 12 to 129,024 parameters, shown in Table 2.

Training details. For math and code, we use GRPO with group size $G = 8$, learning rate 5×10^{-4}, cosine schedule, and 3 epochs per round. For protein modeling, we use cross-entropy with two-phase training per round. Phase 1 trains the classification head on the full dataset. Phase 2 trains the adapter on the failure set with the head frozen. Models run in bf16 throughout. All experiments use 8 GPUs with distributed data parallelism (DDP). Accuracy figures include 95% bootstrap confidence intervals.

Learning rate and stopping. We use a fixed learning rate of 5×10^{-4} for the default 12-parameter adapter. For a per-round parameter budget $p > 12$, we use

$$\eta(p) = 5 \times 10^{-4} \sqrt{\frac{12}{p}}. \tag{5}$$

This scaling limits the magnitude of increasingly expressive adapters. We additionally monitor $\|v_t\|_2$: failure of the adapter norm to decrease as the failure set shrinks indicates that later learners may be too strong.

Training terminates when either (i) the failure set contains fewer than eight examples for math and protein tasks, or fewer than sixteen examples for MBPP, or (ii) accuracy fails to improve for five consecutive rounds. The second criterion is applied only after completing at least five rounds. These rules were fixed across the reported experiments rather than tuned independently for each dataset. Detailed training costs are reported in Appendix B.

5 MAIN RESULTS

Mathematical reasoning. Table 1 shows that BoostLoRA reaches **89.1%** on GSM8K, beating TinyLoRA's best at 8,064 parameters (87.2%) by 1.9 points and full fine-tuning at 3.09B parameters (87.0%) by 2.1 points. Figure 1 shows BoostLoRA reaching **68.8%** on MATH-500, exceeding TinyLoRA's best (67.8%) and approaching full fine-tuning (69.0%). The slower saturation relative to GSM8K (round 14 vs. round 12) reflects the greater difficulty of competition-level problems. The dual evaluation (Figure 1) shows concurrent improvement on both MATH-500 and GSM8K (76.0% $\rightarrow$ 89.1%) when training on SimpleRL (Zeng et al., 2025).

Code generation. Figure 2(c) presents results when training on the MBPP train split (374 problems) with code-execution reward. MBPP accuracy improves from 49.8% to **57.2%** over 20 rounds (15% relative improvement). HumanEval, evaluated as a held-out cross-benchmark, peaks at **80.4%** and stabilizes around 79%. The cross-benchmark transfer suggests that BoostLoRA learns general coding ability rather than MBPP-specific patterns. By contrast, full fine-tuning *degrades* HumanEval from 72.6% to 57.9%. This matches prior findings (Dou et al. (2024)) where fine-tuning DeepSeek-Coder on APPS+ caused a similar performance drop (HumanEval: 78.0% $\rightarrow$ 55.5%), attributing degraded generalization from SFT on the narrow-dataset.

Table 1: **Benchmark results on Qwen2.5-3B-Instruct.** TinyLoRA baselines trained with GRPO (Morris et al., 2026); math baselines from Morris et al. (2026); code baselines are our own TinyLoRA runs using the same codebase. †Full fine-tuning (math from Morris et al. (2026)). *Full FT degrades HumanEval from 72.6% to 57.9%, matching narrow-dataset SFT degradation reported by Dou et al. (2024). Parameter counts denote the maximum number of adapter parameters optimized simultaneously. BoostLoRA trains 12 parameters per round, optimizes $20 \times 12 = 240$ adapter coordinates over the complete sequence, and retains zero additional parameters after merging.

		Mathematical Reasoning		Code Generation	
Method	**Peak Trainable Adapter Params**	GSM8K	MATH-500	MBPP	HumanEval
Base model (zero-shot)	0	76.0	55.0	49.8	72.6
TinyLoRA	16	80.9	64.0	50.6	63.4
TinyLoRA	252	85.4	66.4	52.2	64.0
TinyLoRA	8,064	87.2	67.8	52.6	64.6
TinyLoRA	129,024	86.7	67.8	54.4	67.7
Full FT†	3.09B	87.0	69.0	50.4	57.9*
BoostLoRA (ours)	**12**	**89.1**	**68.8**	**57.2**	**80.4**

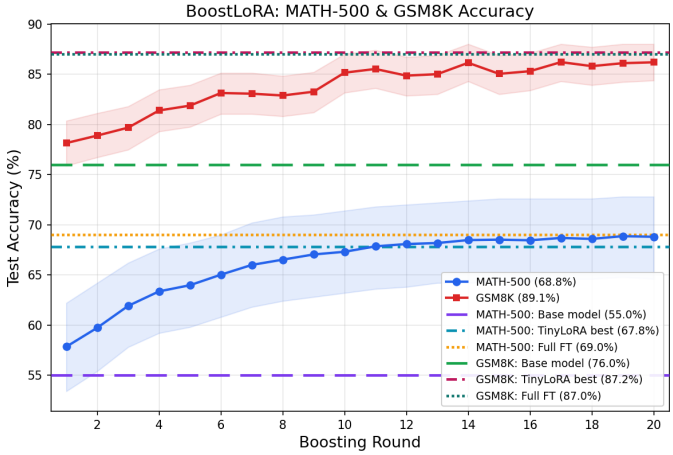

Figure 1: **Dual evaluation:** MATH-500 (blue) and GSM8K (red) improve concurrently when training on SimpleRL (Zeng et al., 2025), suggesting general reasoning gains. MATH-500 beats TinyLoRA best at round 12 and saturates near Full FT at round 14.

Protein modeling. Table 2 reports accuracy, F1, and AUC-ROC across parameter scales for both TinyLoRA (single round) and BoostLoRA (20 rounds) for PPB-Affinity. Boosting helps most at ultra-low parameters: at 12 parameters, BoostLoRA (67.9%) beats TinyLoRA (66.3%) by 1.6 points. We observe that TinyLoRA degrades with more parameters, increasing from 12 to 129,024 parameters *reduces* AUC from 68% to 50.6% (near random). The linear probe (frozen ESM2 + logistic regression, 1,281 parameters) reaches only 59.7% accuracy with AUC = 0.585. BoostLoRA at 12 parameters per round closes most of the gap to full fine-tuning (651M parameters, 69.4%). At 129,024 parameters, both methods struggle, with TinyLoRA's AUC dropping to 0.506 and BoostLoRA to 0.485 (random).

Effective rank growth with ROTATE. Figure 2(a) shows that TOP and ROTATE produce identical accuracy trajectories through round 8. After round 8, TOP saturates at 87.7% while ROTATE continues to 89.1%, a 1.4-point gap. This divergence aligns with the rank-capacity limit: by round 8, TOP has exhausted its fixed rank-2 subspace, while ROTATE has accumulated rank 16 and continues adding orthogonal directions. The tight match between observed ε-rank and the bound rT confirms that ROTATE produces rank growth that translates to accuracy gains. Figure 3 validates equation 2. With ROTATE, ε-rank grows linearly to 40 = rT, while the participation ratio reaches ≈25 (below the

Table 2: **PPB-Affinity binding classification (ESM2-650M).** Binary classification at $K_D = 500$ nM threshold. TinyLoRA: single round, trained on full dataset. BoostLoRA: 20 rounds with ROTATE basis, failure-focused training, LR scaled as $5\times10^{-4} \cdot \sqrt{12/p}$ for $p > 12$. No prior classification baselines exist for this dataset. ‡TinyLoRA and BoostLoRA both struggle at 129K params: large adapters disrupt ESM2's pretrained representations (see text).

Metric	**Linear Probe**	**Full FT**	**TinyLoRA**					**BoostLoRA (ours)**				
Params	*1,281*	*651M*	12	240	4,032	8,064	129K	12	240	4,032	8,064	129K
Acc (%)	59.7	69.4	66.3	64.3	65.1	65.3	65.3	**67.9**	68.5	**69.1**	68.1	58.2
F1	76.0	81.0	80.4	80.5	79.3	79.0	79.0	80.1	80.6	**81.0**	80.1	79.2
AUC	58.5	67.0	68.0	68.0	60.5	56.2	50.6‡	67.7	68.4	**69.0**	65.8	48.5‡

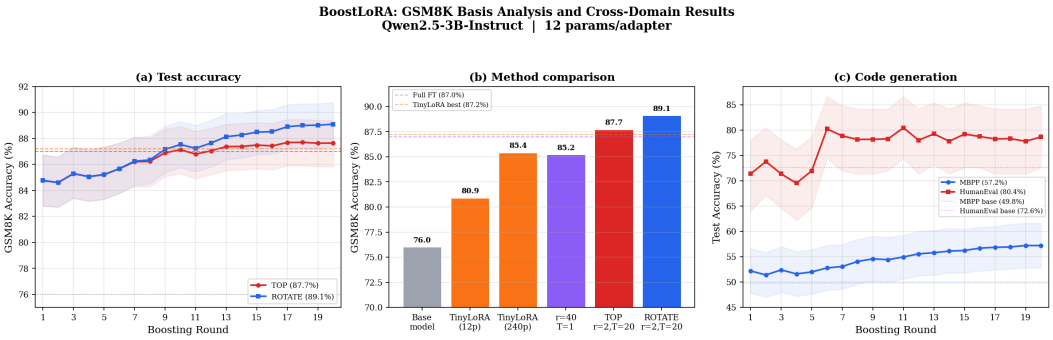

Figure 2: **BoostLoRA across domains. (a)** GSM8K: TOP (red) and ROTATE (blue) track identically through round 8, then diverge: TOP saturates at 87.7% while ROTATE reaches 89.1%. **(b)** GSM8K method comparison: BoostLoRA ROTATE (89.1%) beats TinyLoRA at all param counts, full FT (87.0%), and the r=40 monolithic ablation (85.2%). **(c)** Code generation: MBPP (blue) improves from 49.8% to 57.2%; HumanEval (red, cross-benchmark) peaks at 80.4%.

bound because early-round adapters contribute larger $\|\Delta_t\|_F$, creating non-uniform singular values). This rank gap explains the 1.4-point accuracy difference on GSM8K (Table 3) and motivates ROTATE as the default.

Rank-2 and monolithic rank-40. Figure 2(b) compares all methods. The monolithic $r = 40$, $T = 1$ adapter, which matches ROTATE's total subspace in one round, reaches only 85.2%, **3.9 points below** boosted ROTATE. With $r = 2$, each of the 12 parameters controls $1/4$ of the entries in a 2×2 matrix R, whereas with $r = 40$ the same 12 parameters must influence a 40×40 matrix, diluting the gradient signal.

Boosting dynamics. Figure 4 shows that failures decrease from 687 to 462 over 20 rounds while training accuracy rises from 90.8% to 93.8%. Despite approximately 2% average regression of previously correct examples per round, held-out accuracy increases from 76.0% to 89.1%, narrowing the train-test gap from 14.8 to 4.7 points and providing no evidence of failure-set memorization. Adapter norms decline from ≈ 0.013 in early rounds to 0.005 in later rounds, and later rounds require fewer optimizer steps (24 vs 33); the same patterns are qualitatively observed with ROTATE.

6 Conclusion

BoostLoRA overcomes the expressivity limit of ultra-low-parameter adapters by sequentially training and merging them on failure sets, with a ROTATE basis that grows effective rank to rT. It surpasses both TinyLoRA and full fine-tuning on math reasoning (89.1% on GSM8K with 12 parameters per round vs. 87.0% with 3.09B), avoids catastrophic degradation seen in full fine-tuning narrow

Table 3: **Ablation: SVD basis strategy and rank.** All BoostLoRA variants use 12 trainable parameters per adapter. The r=40 ablation matches ROTATE's total subspace in a single round.

Method	Params/Adapter	Rounds (T)	GSM8K	MATH-500
Base model (zero-shot)	—	—	76.0	55.0
TinyLoRA	12	1	80.9	64.0
TinyLoRA	240	1	85.4	66.4
BoostLoRA (r=40, ablation)	12	1	85.2	64.8
BoostLoRA TOP (r=2)	12	20	87.7	67.3
BoostLoRA ROTATE (r=2)	**12**	**20**	**89.1**	**68.8**

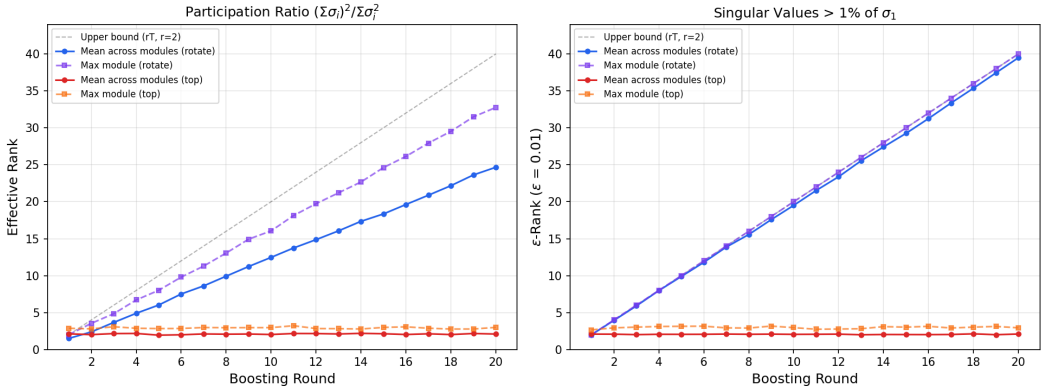

Figure 3: **Effective rank of cumulative weight update $\bar{\Delta} = \sum_t \Delta_t$.** Left: participation ratio. Right: ε-rank ($\varepsilon = 0.01$). TOP (red): rank stays flat at ≈ 2. ROTATE (blue): rank grows linearly to ≈ 25 (participation ratio) and ≈ 40 (ε-rank), matching $rT = 40$.

datasets (HumanEval: 80.4% vs. 57.9%), and transfers to protein binding classification on an encoder-only pLM. Boosting is most useful when individual adapters are weak; larger per-round budgets can overcorrect and require learning-rate scaling. The approach trades peak parameter efficiency for sequential, non-parallel rounds and full-dataset failure scans, and remains untested beyond Qwen2.5-3B, ESM2-650M, and the evaluated datasets.

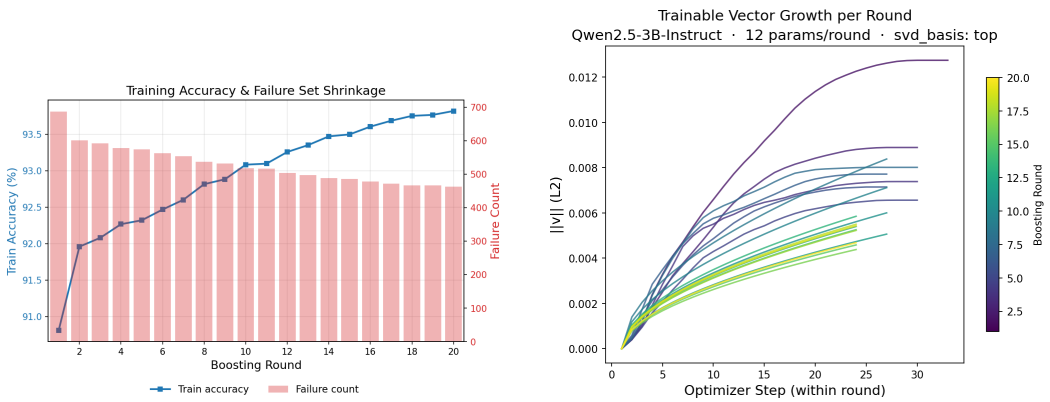

Figure 4: **Boosting dynamics on GSM8K with TOP.** Left: training accuracy rises from 90.8% to 93.8% as failures fall from 687 to 462. Right: later rounds converge to smaller trainable-vector norms as the failure set shrinks.

AI USE STATEMENT

In this work, we used generative AI tools solely for copy-editing of the manuscript-grammar, syntax, and clarity suggestions on author-written prose. We have not used generative AI tools for research ideation, the theoretical statements and proofs, experimental design, implementation, data analysis, or the conclusions; these were produced and verified by the authors. We have reviewed all AI-assisted edits. We take full responsibility for the final content of this work, including all text, claims, and artifacts.

ETHICS STATEMENT

This work uses only publicly available models (Qwen2.5-3B-Instruct, ESM2-650M) and public benchmark datasets (GSM8K, MATH-500, MBPP, HumanEval, and PPB-Affinity). It involves no human subjects, no personally identifiable information, and releases no new data. We are not aware of ethical concerns raised by the method or its evaluation beyond the general considerations that apply to parameter-efficient fine-tuning of large models.

REPRODUCIBILITY STATEMENT

The BoostLoRA procedure is fully specified in Algorithm 1 and Section 3, including the TinyLoRA parameterization (§3.1), the ROTATE SVD basis (§3.3), and the failure-set construction, training procedure, and stopping rules (§3.4, §4). All models, datasets, benchmarks, baselines, and training hyperparameters are listed in Section 4. Complete proofs of Theorems 1 and 2 appear in the main text and Appendix C; end-to-end training costs are reported in Appendix B; the controlled empirical validation of the theory is in Appendix D; and the details of our Chain-of-LoRA re-implementation are given in Appendix E.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.662. URL https://aclanthology.org/2024.acl-long.662/.

Hazem Alsamkary, Mohamed Elshaffei, Mohamed Soudy, Sara Ossman, Abdallah Amr, Nehal Adel Abdelsalam, Mohamed Elkerdawy, and Ahmed Elnaggar. Beyond simple concatenation: Fairly assessing plm architectures for multi-chain protein-protein interactions prediction, 2025. URL https://arxiv.org/abs/2505.20036.

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

Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters, 2025. URL https://arxiv.org/abs/2405.17604.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders,

Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

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.

Shihan Dou, Yan Liu, Haoxiang Jia, Limao Xiong, Enyu Zhou, Wei Shen, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, Yuhao Zhou, Tao Ji, Rui Zheng, Qi Zhang, Xuanjing Huang, and Tao Gui. Stepcoder: Improve code generation with reinforcement learning from compiler feedback, 2024. URL https://arxiv.org/abs/2402.01391.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.1997.1504. URL https://www.sciencedirect.com/science/article/pii/S002200009791504X.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, October 2001. doi: 10.1214/aos/1013203451.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

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, 2021. URL https://arxiv.org/abs/2106.09685.

Furong Huang, Jordan Ash, John Langford, and Robert Schapire. Learning deep ResNet blocks sequentially using boosting theory. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2058–2067. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/huang18b.html.

Arnav Kumar Jain, Gonzalo Gonzalez-Pumariega, Wayne Chen, Alexander M Rush, Wenting Zhao, and Sanjiban Choudhury. Multi-turn code generation through single-step rewards, 2025. URL https://arxiv.org/abs/2502.20380.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. Vera: Vector-based random matrix adaptation, 2024. URL https://arxiv.org/abs/2310.11454.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates, 2023. URL https://arxiv.org/abs/2307.05695.

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

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic level protein structure with a language model. *bioRxiv*, 2022. doi: 10.1101/2022.07.20.500902. URL https://www.biorxiv.org/content/early/2022/10/31/2022.07.20.500902.

Huaqing Liu, Peiyi Chen, Xiaochen Zhai, Ku-Geng Huo, Shuxian Zhou, Lanqing Han, and Guoxin Fan. Ppb-affinity: Protein-protein binding affinity dataset for ai-based protein drug discovery. *Scientific Data*, 11, 12 2024a. doi: 10.1038/s41597-024-03997-4.

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, 2024b. URL https://arxiv.org/abs/2402.09353.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models, 2025. URL https://arxiv.org/abs/2404.02948.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

John X. Morris, Niloofar Mireshghallah, Mark Ibrahim, and Saeed Mahloujifar. Learning to reason in 13 parameters, 2026. URL https://arxiv.org/abs/2602.04118.

Akshara Prabhakar, Yuanzhi Li, Karthik Narasimhan, Sham Kakade, Eran Malach, and Samy Jelassi. Lora soups: Merging loras for practical skill composition tasks, 2024. URL https://arxiv.org/abs/2410.13025.

QwenTeam. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Pengjie Ren, Chengshun Shi, Shiguang Wu, Mengqi Zhang, Zhaochun Ren, Maarten de Rijke, Zhumin Chen, and Jiahuan Pei. MELoRA: Mini-ensemble low-rank adapters for parameter-efficient fine-tuning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3052–3064, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.168. URL https://aclanthology.org/2024.acl-long.168/.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *International Conference on Machine Learning*, 1997. URL https://api.semanticscholar.org/CorpusID:573509.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.715. URL https://aclanthology.org/2023.findings-emnlp.715/.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022. URL https://arxiv.org/abs/2203.05482.

Wenhan Xia, Chengwei Qin, and Elad Hazan. Chain of lora: Efficient fine-tuning of language models via residual learning, 2024. URL https://arxiv.org/abs/2401.04151.

Qingyu Yin, Yulun Wu, Zhennan Shen, Sunbowen Li, Zhilin Wang, Yanshu Li, Chak Tou Leong, Jiale Kang, and Jinjin Gu. Evaluating parameter efficient methods for rlvr, 2025. URL https://arxiv.org/abs/2512.23165.

Shen Yuan, Haotian Liu, and Hongteng Xu. Bridging the gap between low-rank and orthogonal adaptation via householder reflection adaptation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 113484–113518. Curran Associates, Inc., 2024. doi: 10.52202/079017-3606. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/cdd0640218a27e9e2c0e52e324e25db0-Paper-Conference.pdf.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild, 2025. URL https://arxiv.org/abs/2503.18892.

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

Yifei Zhang, Hao Zhu, Aiwei Liu, Han Yu, Piotr Koniusz, and Irwin King. Less is more: Extreme gradient boost rank-1 adaption for efficient finetuning of llms, 2024. URL https://arxiv.org/abs/2410.19694.

Table 4: **Parameter complexity comparison.** n: layers, m: modules per layer, d: width, r: rank, u: projection dimension. All prior methods have fixed effective rank r. BoostLoRA's cumulative rank grows with T (up to $\min(rT, d, k)$ with ROTATE), even though each round's adapter is rank r. Extends Table 1 of Morris et al. (2026).

Method	Training Parameters	Minimum	Effective Rank
Full FT	$\mathcal{O}(nmd^2)$	nmd^2	d
LoRA	$\mathcal{O}(nmdr)$	$2nmd$	r
LoRA-XS	$\mathcal{O}(nmr^2)$	nm	r
VeRA	$\mathcal{O}(nm(d+r))$	$2nmd$	r
TinyLoRA	$\mathcal{O}(nmu)$	1	r
BoostLoRA	$\mathcal{O}(gu)$ peak $\mathcal{O}(Tgu)$ cumulative	T	$\leq \mathbf{rT}$

Table 5: **End-to-end BoostLoRA training cost.** All experiments use 20 boosting rounds. "Failure scan" is the full-training-set evaluation used to construct $\mathcal{F}_t$.

Dataset	Wall clock (h)	Failure scan	Adapter training
GSM8K	184.2	14.4%	80.4%
MATH-500	257.8	4.5%	79.2%
MBPP	14.0	4.5%	79.2%

A Complexity Comparison

Table 4 compares peak trainable parameters, cumulative training parameters, and effective rank across PEFT methods. BoostLoRA exposes $\mathcal{O}(gu)$ trainable parameters in each round and optimizes $\mathcal{O}(Tgu)$ distinct adapter coordinates over T rounds. After merging, the final model retains no additional adapter parameters. With ROTATE, the cumulative update has rank at most $\min(rT, d, k)$ (Theorem 1), decoupling peak trainable state from cumulative representational capacity.

B Training Cost

BoostLoRA introduces sequential training cost because each round requires identifying the current failure set before training a new adapter. Table 5 reports end-to-end wall-clock time and the measured fractions attributable to failure-set evaluation and adapter training. The reported percentages need not sum to 100%; the remainder contains held-out evaluation and other pipeline overheads.

The failure scan is measurable but does not dominate runtime; GRPO rollout generation and optimization account for most of the measured cost. Training becomes cheaper as failures shrink. On GSM8K, per-round runtime decreased from 536.8 minutes in round 1 to 469.5 minutes in round 20, a reduction of 67.3 minutes (12.5%). The rounds remain sequential and therefore cannot be parallelized across boosting depth.

C Proof of Theorem 2

We bound the generalization error of the BoostLoRA ensemble after T rounds using Rademacher complexity.

Function class per round. At round t, the adapter perturbs each adapted layer's output by $\Delta W_\ell\, h_\ell(x) = U_\ell \Sigma_\ell R_t V_\ell^\top h_\ell(x)$, where $h_\ell(x)$ is the input to layer ℓ. Under the linearization approximation (treating the network above each adapted layer as frozen at its pre-adaptation state), the margin perturbation is $\Delta m(x) \approx \sum_\ell g_\ell(x)^\top \Delta W_\ell\, h_\ell(x)$, where $g_\ell(x) \in \mathbb{R}^d$ is the gradient of the margin with respect to layer ℓ's output. Substituting $R_t = \sum_i (v_t)_i P_i$ gives

$$f_t(x) = \langle v_t,\, \phi_t(x) \rangle, \qquad \phi_t(x)_i = \sum_\ell g_\ell(x)^\top U_\ell\, \Sigma_\ell\, P_i\, V_\ell^\top h_\ell(x), \tag{6}$$

where all quantities except v_t are frozen. Since f_t is linear in v_t, the round-t function class is $\mathcal{F}_t = \{x \mapsto \langle v_t, \phi_t(x)\rangle : \|v_t\|_2 \le B_t\}$.

Rademacher complexity of one round. The empirical Rademacher complexity of $\mathcal{F}_t$ over n samples $x_1, \dots, x_n$ is

$$\hat{\mathcal{R}}_n(\mathcal{F}_t) = \frac{1}{n}\,\mathbb{E}_\sigma\left[\sup_{\|v_t\|\le B_t} \sum_{j=1}^{n} \sigma_j \langle v_t, \phi_t(x_j)\rangle\right] = \frac{B_t}{n}\,\mathbb{E}_\sigma\left[\left\|\textstyle\sum_{j=1}^{n} \sigma_j\, \phi_t(x_j)\right\|_2\right], \tag{7}$$

where σ_j are iid Rademacher variables and the supremum is attained at $v_t \propto \sum_j \sigma_j \phi_t(x_j)$. By Jensen's inequality,

$$\mathbb{E}_\sigma\left[\left\|\textstyle\sum_j \sigma_j\, \phi_t(x_j)\right\|\right] \;\le\; \sqrt{\sum_j \|\phi_t(x_j)\|^2} \;\le\; \sqrt{n}\,\max_x \|\phi_t(x)\| \;=\; \sqrt{n}\,X_t, \tag{8}$$

so $\hat{\mathcal{R}}_n(\mathcal{F}_t) \le B_t X_t/\sqrt{n}$.

Subadditivity across rounds. The cumulative perturbation after T rounds is $F(x) = \sum_{t=1}^{T} f_t(x)$, which lives in the Minkowski sum $\mathcal{F}_1 + \cdots + \mathcal{F}_T$. Rademacher complexity is subadditive over Minkowski sums:

$$\mathcal{R}_n(\textstyle\sum_t \mathcal{F}_t) \;\le\; \displaystyle\sum_t \mathcal{R}_n(\mathcal{F}_t) \;\le\; \sum_t \frac{B_t X_t}{\sqrt{n}} \;\le\; \frac{B_{\text{total}} X}{\sqrt{n}}, \tag{9}$$

where $X = \max_t X_t$ and $B_{\text{total}} = \sum_t B_t$.

When using the TOP basis, the Rademacher complexity matches the general bound of Eq. 9. Every round uses the same top-r singular vectors, so $\phi_t = \phi_1$ for all t and $X_t = X_1 = \max_x \|\phi_1(x)\|_2$. With uniform $B_t = B$ the subadditive bound above becomes $\mathcal{R}_n \le B X_1 T/\sqrt{n}$, which is linear in T.

When using the ROTATE basis, the Rademacher complexity is smaller than Eq. 9 by a factor of up to $\sqrt{T}$. Round t projects $g_\ell(x)$ and $h_\ell(x)$ onto its own block of singular directions, so the $\phi_t(x)$ are orthogonal components of one vector and $H^2 = \max_x \sum_t \|\phi_t(x)\|_2^2$ is bounded independently of T. Bounding the concatenated predictor $F(x) = \langle v, \Phi(x)\rangle$ directly, with $v = (v_1, \dots, v_T)$ and $\Phi(x) = (\phi_1(x), \dots, \phi_T(x))$, gives $\mathcal{R}_n \le \sqrt{\sum_t B_t^2}\, H/\sqrt{n}$, which for uniform $B_t = B$ is $BH\sqrt{T}/\sqrt{n}$. Under TOP the same argument gives $H^2 = T \max_x \|\phi_1(x)\|_2^2$ and recovers $B X_1 T/\sqrt{n}$, so the gain is due to the basis.

Margin bound. Applying the standard Rademacher generalization bound (Bartlett & Mendelson, 2002) (see Theorem 3.3 of Mohri et al. (2018)) to the margin loss $\ell_\theta(x) = \min(1, \max(0, 1 - m(x)/\theta))$, which maps to $[0, 1]$: for any margin $\theta > 0$, with probability $\ge 1 - \delta$,

$$\Pr_{\text{test}}[\text{error}] \;\le\; \Pr_{\text{train}}[m_T(x) \le \theta] \;+\; \frac{2}{\theta}\,\mathcal{R}_n(\textstyle\sum_t \mathcal{F}_t) \;+\; \sqrt{\dfrac{\log(2/\delta)}{2n}}, \tag{10}$$

which gives the stated bound after substituting the Rademacher estimate from Step 3.

Approximation. The proof treats $\phi_t(x)$ as fixed for each round, but in practice $h_\ell(x)$ changes after merging previous adapters. This is a reasonable approximation because the per-round weight perturbation is small ($\|\Delta W_t\|_F \approx 0.01$), so hidden states shift only slightly between rounds. The proof treats $g_\ell(x)$ and $h_\ell(x)$ as fixed at their pre-adaptation values, but both shift after merging previous adapters. The linearization error is $|F(x) - \tilde{F}(x)| = O(T\|\Delta W\|_F B_{\text{total}})$, which is negligible as $\|\Delta W_t\|_F \ll 1$ per round.

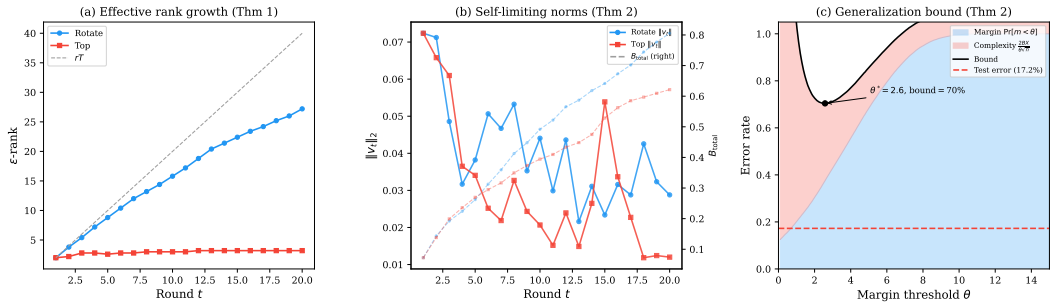

Figure 5: **Empirical validation of theoretical properties** on CIFAR-10 (CNN, r=2, T=20, 9 params/round). **(a)** ε-rank of the cumulative weight update $\bar{\Delta} = \sum_t \Delta_t$. ROTATE (blue) grows linearly, tracking the theoretical bound rT. TOP (red) saturates at ≈ 3, confirming Theorem 1. **(b)** Per-round adapter norms $\|v_t\|_2$ (solid) and cumulative norm $B_{\text{total}} = \sum_{s \le t} \|v_s\|$ (dashed, right axis). Both bases show declining per-round norms; B_{total} grows sublinearly, consistent with the self-limiting property in Theorem 2. **(c)** Generalization bound (Theorem 2) evaluated numerically. The bound (black curve) decomposes into a margin term (blue, fraction of training examples with margin below θ) and a complexity term (red, $\propto B_{\text{total}} \cdot X/\theta\sqrt{n}$). At the optimal $\theta^* = 2.6$, the bound is 70%, non-vacuous against the observed 17.2% test error.

D EMPIRICAL THEORY VALIDATION

We validate Theorems 1 and 2 on a controlled CIFAR-10 classification task using a small CNN with 5 adapted layers, r=2, T=20, and 9 trainable parameters per round. This isolates the boosting mechanism from LLM-specific confounders (tokenization, generation, reward noise). The CNN includes LayerNorm before each fully connected layer, matching the normalization structure of transformer architectures. Figure 5 summarizes the results.

The ε-rank of the cumulative update $\bar{\Delta}$ grows linearly with T for ROTATE, reaching ≈ 27 at round 20. The gap from the theoretical maximum $rT = 40$ reflects the self-limiting dynamics visible in panel (b): later rounds produce smaller adapters whose singular values fall below the ε-rank threshold, even though the subspaces are orthogonal. On Qwen2.5-3B, where adapter norms are more uniform across rounds, ε-rank tracks rT exactly (Figure 3). With TOP, ε-rank stays at ≈ 3 regardless of T, confirming that successive adapters overwrite the same subspace.

Per-round adapter norms decline over boosting rounds for both bases: from ≈ 0.07 in early rounds to $\approx 0.01{-}0.03$ in later rounds. TOP norms decay faster because the fixed subspace offers diminishing returns. The cumulative norm B_{total} grows sublinearly, consistent with the convergence argument in Theorem 2. As the failure set shrinks, the gradient signal weakens, producing smaller adapters.

We evaluate Theorem 2 numerically using the measured quantities: $B_{\text{total}} = 0.81$, feature norm $X = 115.6$ (computed from hidden-state activations at the adapted layers), $n = 50{,}000$, and the empirical margin distribution at round 20. The bound is minimized at $\theta^* = 2.6$, where the margin term (37.5%) and complexity term (32.3%) balance. The resulting bound of 70% is non-vacuous, though loose relative to the observed 17.2% test error. The looseness is driven by X: in architectures with smaller weight singular values or stronger normalization, X decreases and the bound tightens proportionally.

E CHAIN OF LORA COMPARISON

We implemented Chain of LoRA (CoLA) from the description in the original paper, as no public repository is available. We mirrored BoostLoRA's use of Qwen2.5-3B in both its base and instruct variants, LoRA rank r=8 and α=16 on the `q_proj` and `v_proj` projections, and a batch size of 8 (reduced only for MATH-500 to avoid OOM). Train/test splits were held identical to the BoostLoRA experiments. We ran five random seeds per configuration and report the best. We evaluated chain lengths $t \in \{1, 2, 3, 8, 10, 15, 20, 25, 30\}$, where t=1 recovers vanilla LoRA and each subsequent

stage trains a new adapter on the residual and merges it into the running weights. Training used the L4 GPU (the original CoLA experiments used A100s; we were constrained by availability). The learning rate was selected by grid search around the range reported in the CoLA paper. Full per-seed-best results are in Table 6.

Table 6: Chain of LoRA accuracy across chain lengths t. t=1 is vanilla LoRA.

Dataset	t=1	t=2	t=3	t=8	t=10	t=15	t=20	t=25	t=30
MBPP (base)	0.498	**0.526**	0.514	0.328	0.342	0.316	0.298	0.292	0.248
MBPP (instruct)	**0.502**	0.450	0.458	0.492	0.488	**0.502**	0.458	0.438	0.402
HumanEval (base)	0.4146	**0.4817**	0.4634	0.2866	0.2866	0.3171	0.2988	0.2134	0.2073
HumanEval (instruct)	**0.7500**	0.7195	0.7134	0.7073	0.7134	0.7073	0.6890	0.6707	0.6646
GSM8K (base)	0.2684	**0.3427**	0.2782	0.2934	0.2881	0.2964	0.2472	0.2252	0.2199
GSM8K (instruct)	0.0591	0.0614	0.0569	0.0697	0.0637	**0.0735**	0.0773	0.0538	0.0803