When 13 Parameters Aren't Enough: Task-Dependent Expressivity Floors in Ultra-Low-Parameter LLM Adaptation

Anonymous Authors[1]

Abstract

Can a large language model be meaningfully adapted with just 13 trainable parameters? Recent work on TINYLORA suggests yes: a single shared vector, projected through frozen random matrices, improves mathematical reasoning by 15 percentage points. But does this remarkable efficiency generalize across tasks, or does it depend on what the model already knows?

We investigate this question by applying the same ultra-low-parameter adapter across three domains that span the full spectrum of base-model competence, from mathematics, where the model already performs well, to chess puzzle solving, where it starts from near-zero. The result is a sharp dichotomy: TINYLORA maintains or briefly improves accuracy on tasks within the model's existing competence, yet achieves **exactly 0%** on chess, a failure that persists across four orders of magnitude of parameter scaling, two training paradigms, and two model sizes. In contrast, standard LORA at the same rank but with independent per-layer parameterization reaches 36% accuracy on the same puzzles.

These findings establish a **prior-dependent expressivity floor**: the effective intrinsic dimensionality of fine-tuning is not a fixed constant but varies with the gap between the base model's competence and the task's demands. Ultra-low-parameter adaptation succeeds when the task requires navigating existing knowledge and fails categorically when it demands qualitatively new computation.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

1. Introduction

There is something philosophically unsettling about the finding that a language model can be meaningfully adapted with 13 trainable numbers. Yet this is precisely what Morris et al. (2026) demonstrated: a shared scalar vector of dimension 13, projected through per-layer frozen random matrices, suffices to improve QWEN2.5-7B-Instruct from 76% to 91% on GSM8K via reinforcement learning from verifiable rewards (RLVR). If taken at face value, the result suggests that post-training adaptation lives in an extraordinarily low-dimensional subspace of the full parameter space.

But should we take it at face value? Mathematical reasoning is deeply embedded in pre-training corpora. If 13 parameters succeed because the model already *knows* how to do math, and the adapter merely steers format and strategy selection, then TINYLORA's success tells us more about the base model's latent structure than the adapter's expressive power. This raises a precise theoretical question:

Does the low intrinsic dimensionality of fine-tuning hold uniformly across tasks, or is it itself task-dependent?

We answer this question with a controlled experiment across three task domains, deliberately chosen to span the full range of base-model competence:

1. **Mathematics** (GSM8K): 54% zero-shot accuracy. High prior.
2. **Quantitative reasoning** (AQuA-RAT): 49% zero-shot. Medium prior.
3. **Chess puzzles**: $<$1% zero-shot. Near-zero prior.

The same $N = 13$ TINYLORA adapter, trained identically via supervised fine-tuning on QWEN2.5-72B, maintains the base model's accuracy on the first two tasks while achieving *exactly 0%* on chess. This is not a marginal gap, it is a categorical failure that persists across four orders of magnitude of parameter scaling ($N = 13$ to $N = 100{,}000$), across two training paradigms (SFT and RLVR), and across two model sizes (7B and 72B).

Why chess? Chess serves as a near-ideal diagnostic because it cleanly separates what we want to test from confounding factors. The output space is compact (4-5 character UCI moves), evaluation is unambiguous (exact match against the engine-verified optimal move), and the required computation, board-state-dependent tactical reasoning, is qualitatively different from the linguistic competences embedded in pre-training. While large models can produce syntactically valid chess notation via surface statistics (Carlini, 2024), generating the *correct* move for a given position requires genuine positional computation that lies far outside the pre-training distribution.

Contributions.

1. We establish an empirical **expressivity floor** for TINYLORA: the architecture preserves performance on tasks where the base model demonstrates non-trivial competence and fails categorically on tasks where competence is near-zero, independent of parameter count across nearly four orders of magnitude.
2. We provide the first **rank-vs-sharing ablation**: rank-2 LORA with independent per-layer parameters achieves 36% on chess, demonstrating that *independent parameterization*, not merely higher rank, is the more binding bottleneck. This empirically disentangles TINYLORA's two architectural constraints, though both contribute to the overall capacity gap.
3. We replicate the failure on both **7B and 72B** base models, validate with a **2,000-puzzle extended evaluation**, and test across **three task domains** with ordered prior strengths (Figure 2b), establishing that the expressivity floor is a continuous function of base-model competence.
4. We propose the **prior-dependence hypothesis**: the effective intrinsic dimensionality of a fine-tuning task is not fixed but varies with the gap between what the base model already knows and what the task requires.

2. Background

We briefly review the two adaptation methods central to our study and highlight the architectural properties that will drive our analysis.

2.1. Low-Rank Adaptation (LoRA)

LORA (Hu et al., 2022) adapts a frozen weight matrix $W_0 \in \mathbb{R}^{d\times k}$ by learning an additive low-rank perturbation $\Delta W = BA$, where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$. Crucially, each adapted layer maintains its *own* A and B matrices. For QWEN2.5-72B with $r = 8$ applied to all 560 linear modules (7 projections $\times$ 80 layers), this yields 105.3 million independent parameters. Each layer can thus be steered in entirely different directions.

2.2. TinyLoRA

TINYLORA (Morris et al., 2026) replaces per-layer independent matrices with a single *shared vector* $\mathbf{v} \in \mathbb{R}^N$. For each adapted layer ℓ, a frozen random projection tensor $P_\ell \in \mathbb{R}^{N\times r\times r}$ maps $\mathbf{v}$ to a small matrix:

$$R_\ell = \sum_{u=1}^{N} v_u \cdot P_\ell[u] \in \mathbb{R}^{r\times r} \tag{1}$$

This modulates a rank-r SVD decomposition of the frozen weight:

$$\Delta W_\ell = U_\ell \cdot \big(S_\ell \cdot R_\ell\big) \cdot V_\ell^\top \tag{2}$$

where $U_\ell, S_\ell, V_\ell^\top$ are the top-r SVD components of $W_{0,\ell}$. With $r = 2$, every layer's perturbation has rank at most 2.

The architecture is defined by two constraints that will prove central to our analysis:

- **Rank constraint.** ΔW_ℓ has rank $\leq r = 2$, regardless of N.
- **Sharing constraint.** A single $\mathbf{v}$ governs all layers. Increasing N provides finer control over *which* rank-2 perturbation each layer receives, but does not decouple them. A change to any component of $\mathbf{v}$ propagates to every layer simultaneously.

2.3. Reinforcement Learning from Verifiable Rewards

RLVR treats fine-tuning as policy optimization with reward signals from automatic verifiers. Morris et al. (2026) used GRPO (Shao et al., 2024), which normalizes advantages across groups of sampled responses. A well-documented prerequisite is that the base policy must succeed non-trivially to provide reward signal for the gradient estimator.

3. Experimental Setup

Models. Our primary model is QWEN2.5-72B-Instruct (Qwen Team, 2024) (72B parameters) in 4-bit NF4 quantization via QLoRA (Dettmers et al., 2024), consuming ~41 GB VRAM on a single NVIDIA H100 80 GB GPU. We additionally test QWEN2.5-7B-Instruct (7B, ~5.6 GB VRAM) to match the base model used by Morris et al. (2026). Gradient checkpointing is enabled throughout.

Task domains.

- **Chess puzzles** (Lichess, Lichess.org 2024): beginner difficulty, Elo < 1200. Train: 17,962 puzzles; checkpoint eval: 50 puzzles; extended eval: 2,000 puzzles

across 5 difficulty levels. Metric: exact-match UCI move + legal move rate.

- **Mathematics** (GSM8K, Cobbe et al. 2021): 7,500 training problems; eval: 50-problem random subset. Metric: exact numerical answer match.
- **Quantitative reasoning** (AQuA-RAT): 97,467 training MCQ problems; eval: 100 problems (larger eval set to reduce noise on 5-way MCQ). Metric: exact letter match (A-E).

These tasks were selected to span the base model's competence spectrum while controlling for evaluation format: all three require short, deterministic answers with unambiguous ground truth. This design isolates prior strength as the key variable while holding evaluation methodology constant.

Prompt format. All experiments use the Qwen ChatML template. For chess, the system message instructs the model to act as a chess engine and output only the best move in UCI format; the user message provides the FEN position and side to move. The full prompt templates for all three tasks are provided in Section J.

Training configurations. SFT uses cross-entropy loss with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 0.01), learning rate 10^{-4} for TinyLoRA and 2×10^{-5} for LoRA, batch size 4 (chess) or 2 (math/AQuA-RAT), 500 steps, gradient clipping at max norm 1.0. For RLVR: GRPO with group size $K = 4$, reward $+1$ for exact match, learning rate 10^{-5}, 800 steps (one extended to 2,000; LoRA early-stopped at 600 due to mode collapse). We tested learning rates $\{10^{-5}, 5 \times 10^{-5}, 10^{-4}, 5 \times 10^{-4}\}$ for chess; all produced 0%. Train and evaluation puzzle sets are disjoint.

Adapter configurations.

- **TinyLoRA:** $r = 2$, $N \in \{13, 196, 1000, 5000, 10000, 50000, 100000\}$. Applied to all 560 linear modules, weight-tied.
- **LoRA rank-8:** Standard PEFT, $r = 8$, $\alpha = 16$, 105.3M independent parameters.
- **LoRA rank-2:** Same as rank-8 but $r = 2$, 26.3M independent parameters. *Key ablation*: same rank as TinyLoRA, but independent per-layer parameterization.

A note on parameter count asymmetry. The smallest possible independent rank-2 LoRA across all 560 modules of Qwen2.5-72B is inherently ∼26M parameters, this is an architectural minimum, not an experimental choice. TinyLoRA's maximum of 100K shared parameters is thus separated from LoRA's minimum by ∼260×. We address this directly: the relevant comparison is not raw parameter count but the *effect of scaling within each architecture*. Scaling TinyLoRA from 13 to 100K (7,700×) yields no improvement whatsoever; the failure is internal to the architecture, not an artifact of having fewer parameters than LoRA.

Table 1. **RLVR on chess produces universal failure.** All 9 configurations achieve 0% accuracy, including standard LoRA (which mode-collapsed to two fixed outputs by step ∼200). This is consistent with the well-documented prerequisite that RLVR requires non-trivial base success rates (Guo et al., 2025). The remainder of this paper uses SFT.

Adapter	Params	Steps	Variant	Accuracy
TinyLoRA	13	800	GRPO v1	0%
TinyLoRA	49	800	GRPO v1	0%
TinyLoRA	196	800	GRPO v1	0%
TinyLoRA	13	800	GRPO v2 (CoT)	0%
TinyLoRA	49	800	GRPO v2	0%
TinyLoRA	196	800	GRPO v2	0%
TinyLoRA	1,000	2,000	GRPO v2	0%
TinyLoRA	1,000	800	Curriculum	0%
LoRA (r=8)	105M	600	GRPO	0% (collapsed)

4. Results

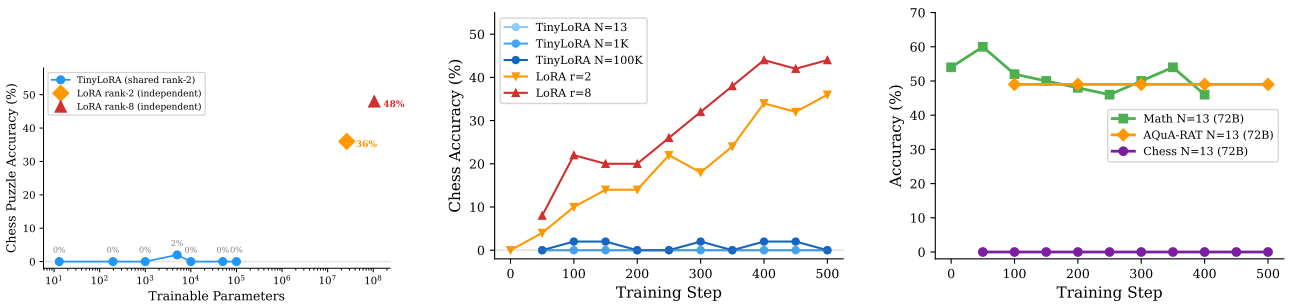
We present our findings in four stages, progressing from initial context (RLVR failure) through the core phenomenon (the capacity gap), its generality (cross-task analysis), and a mechanistic decomposition (the rank-vs-sharing ablation).

4.1. RLVR Cannot Bootstrap Chess

Before turning to supervised fine-tuning, we document an important negative result. Table 1 shows that all 9 RLVR runs on chess produce 0% accuracy. For LoRA, this manifests as mode collapse: by step ∼200, over 95% of generated outputs are one of two fixed strings (`e1d1` or `e8d8`), regardless of the input board position. We include these results for completeness but emphasize that this failure is a known limitation of RLVR under sparse rewards, not a finding about adapter capacity. With this context established, we turn to SFT, which provides direct supervision and thus removes the confound of reward sparsity.

4.2. The Capacity Gap on Chess

With supervised targets, the architectural difference becomes starkly visible. Table 2 and Figure 1a tell a consistent story: TinyLoRA achieves 0% chess accuracy across nearly four orders of magnitude ($N = 13$ to $N = 100{,}000$), while LoRA rank-8 reaches 48% and is still improving at step 500. The learning curves in Figure 1b make the failure mode concrete, LoRA improves steadily from the first checkpoint, while every TinyLoRA configuration traces a



(a) **Scaling within TinyLoRA provides no benefit.** Chess accuracy vs. adapter parameter count (log scale). TinyLoRA (circles) remains at 0% from $N = 13$ to $N = 100$K. LoRA rank-2 (diamond, 26.3M) and rank-8 (triangle, 105.3M) both succeed.

(b) **The gap emerges from the first checkpoint.** LoRA variants improve steadily; all TinyLoRA variants remain flat at 0% throughout training.

(c) **Same adapter, task-dependent outcome.** At $N = 13$, math and AQuA-RAT maintain baselines while chess fails completely.

Figure 1. **The TinyLoRA-LoRA capacity gap on chess.** Neither scaling parameter count (a) nor additional training time (b) closes the gap. The same adapter succeeds on math and quantitative reasoning (c).

Table 2. **SFT on chess reveals a categorical gap between architectures.** TinyLoRA achieves $\leq$2% at all scales (500 steps, 72B); the rare 2% observations (1/50) do not survive extended evaluation (Section 4.6). Both LoRA configurations achieve substantial accuracy with the same training budget.

Adapter	Params	Best Acc.	Final Acc.	Legal%
TinyLoRA	13	0%	0%	8%
TinyLoRA	196	0%	0%	14%
TinyLoRA	1,000	0%	0%	16%
TinyLoRA	5,000	2%	2%	12%
TinyLoRA	10,000	0%	0%	8%
TinyLoRA	50,000	2%	0%	8%
TinyLoRA	100,000	2%	0%	8%
LoRA (r=2)	26.3M	**36%**	**34%**	**42%**
LoRA (r=8)	105.3M	**48%**	**48%**	**56%**

flat line at zero. Figure 1c shows the same $N = 13$ adapter on math and AQuA-RAT: both maintain baseline accuracy throughout.

The key comparison, which we return to in Section 4.4, is rank-2 LoRA: at the *same rank* as TinyLoRA but with independent per-layer parameterization, it achieves 36%.

4.3. Cross-Task Analysis: The Role of Prior Competence

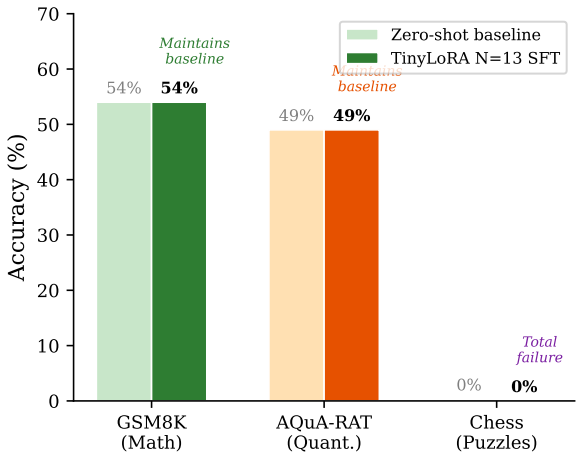
The chess failure alone could be dismissed as a quirk of one particular task. The cross-task comparison in Figure 2 rules this out by revealing a systematic pattern. The same $N = 13$ adapter, trained via SFT on Qwen2.5-72B:

- **GSM8K (54% zero-shot):** TinyLoRA peaks at 60% (step 50), demonstrating transient above-baseline improvement, then oscillates and ends at 46% (step 400). The peak confirms the adapter *actively steers* rather than being inert; the subsequent degradation suggests interference at longer training, consistent with known overfitting dynamics of minimal adapters (Biderman et al., 2024). Full trajectory in Table 6.
- **AQuA-RAT (49% zero-shot):** TinyLoRA maintains 49% at all checkpoints (steps 100-500), perfectly preserving the base model's competence.
- **Chess (<1% zero-shot):** 0% at initialization, 0% throughout all 500 steps.

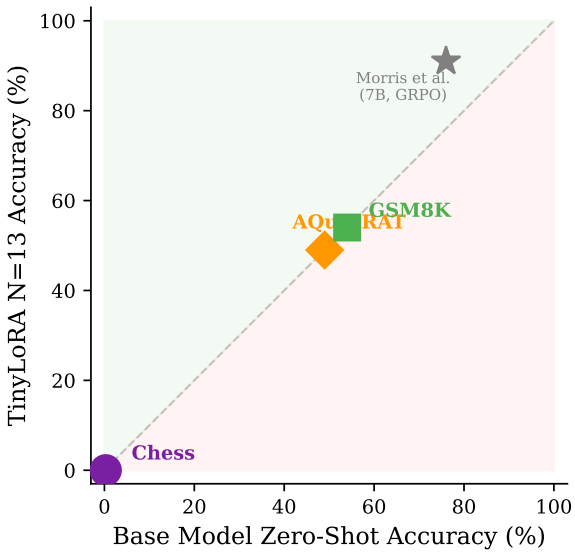
The pattern is unambiguous: where the model already performs a task non-trivially, 13 parameters suffice to maintain (and on math, briefly improve) accuracy. Where the model has near-zero competence, even 100,000 parameters within the TinyLoRA architecture cannot break through.

Figure 2bb visualizes this as a mapping from prior strength to post-adaptation accuracy: high-prior tasks lie on the diagonal (the adapter is essentially a no-op); chess lies at the origin. The transition between these regimes appears sharp rather than gradual.

Relationship to Morris et al. Morris et al. (2026) report GSM8K improvement from 76% to 91% using Qwen2.5-7B with GRPO. This appears above the diagonal in Figure 2bb, consistent with the finding that RLVR is more parameter-efficient than SFT for high-prior tasks. Our SFT results are complementary: on high-prior tasks, SFT with TinyLoRA preserves accuracy but does not dramatically improve it. The architecturally interesting result is the chess failure, where neither SFT nor RLVR helps at any scale.



(a) **Same adapter, different outcomes.** TINYLORA ($N = 13$) maintains baseline accuracy on tasks the model already performs non-trivially (math 54%, AQuA-RAT 49%) but fails completely on chess (0%).



(b) **Prior-dependence mapping.** x-axis: base model zero-shot accuracy; y-axis: post-TINYLORA accuracy. Points near the diagonal indicate the adapter preserves but does not improve performance. The Morris et al. result (star) lies above the diagonal due to GRPO's advantage over SFT.

Figure 2. **The expressivity floor depends on prior competence.** Tasks with substantial base-model accuracy lie on the diagonal (adapter preserves performance); tasks with near-zero prior lie at the origin (adapter cannot bootstrap).

4.4. Disentangling the Two Constraints: Rank vs. Sharing

The results so far demonstrate *that* TINYLORA fails on chess; we now ask *why*. TINYLORA imposes two constraints simultaneously, rank and sharing, and the chess failure could stem from either. The rank-vs-sharing ablation disentangles them.

Both TINYLORA ($N = 100$K) and LORA ($r = 2$) apply rank-2 perturbations to each layer. The crucial difference: LORA parameterizes each layer independently, while TINYLORA couples all layers through a single shared vector. Table 3 shows the result: independent parameterization succeeds (36%), shared parameterization fails (0%).

This confirms that the *sharing constraint*, not the rank constraint alone, is the more binding bottleneck for chess. Even rank-2 perturbations suffice when each layer can make its own adaptation independently. The additional improvement from rank 8 (48% vs. 36%) indicates that rank also contributes meaningfully; both constraints are active, but sharing is the more severe limitation based on the larger accuracy gap it induces (0%→36% vs. 36%→48%).

4.5. Model Scale Replication

To match the setup of Morris et al. (2026), we ran TINYLORA SFT with $N = 13$ on QWEN2.5-7B-Instruct (the model they used): **0% accuracy, 6% legal move rate**. The failure is not specific to the 72B model.

4.6. Extended Evaluation

To quantify evaluation noise, we tested trained TINYLORA models ($N = 13$ and $N = 196$) on 2,000 puzzles (5 difficulty levels, 400 per level). Both achieved **0.25% accuracy (5/2000)** and ∼10% legality (Table 8). A model generating random 4-character strings from the move alphabet (`[a-h][1-8][a-h][1-8]`) would achieve comparable rates. The sporadic 2% observations in 50-sample evaluation (Table 2) are thus indistinguishable from noise, as confirmed by this 40× larger evaluation.

5. Analysis

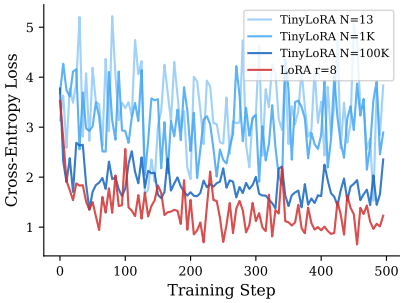
The previous section established *what* happens; we now investigate *why*. We examine TINYLORA's training dynamics, explain why scaling N is ineffective, and trace the failure to a fundamental architectural property.

5.1. Loss Decreases While Accuracy Stays at Zero

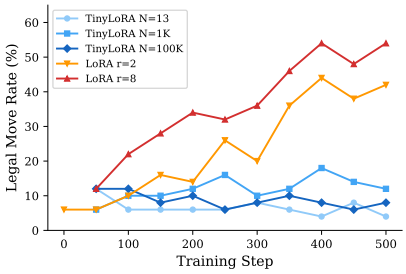
A natural question is whether TINYLORA receives any learning signal at all. It does. Figure 3a shows that the training loss decreases, particularly at higher N. The adapter is receiving gradient signal and shifting probability mass

Table 3. **Sharing, not rank, is the primary bottleneck.** All three configurations use rank-2 perturbations; only the parameterization structure differs. LoRA $r = 2$ (independent) succeeds; TinyLoRA (shared) fails. The minimum achievable parameter count for independent rank-2 LoRA on Qwen-72B is ∼26M, an architectural minimum, not a design choice.

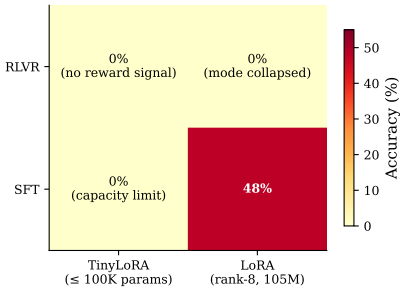
Method	Rank	Sharing	Params	Best Accuracy
TinyLoRA ($N = 100$K)	2	Shared $\mathbf{v}$	100K	0%
LoRA (r=2)	2	Independent	26.3M	**36%**
LoRA (r=8)	8	Independent	105.3M	**48%**



(a) **Decreasing loss without accuracy gains.** TinyLoRA's loss decreases (especially at higher N), yet accuracy remains at 0%. LoRA shows steeper loss descent paired with accuracy gains.



(b) **Syntax without semantics.** LoRA $r = 8$ reaches 56% legal moves; LoRA $r = 2$ reaches 42%. TinyLoRA plateaus at 8-16%, above the ∼5% zero-shot baseline but far from useful.



(c) **Method × Adapter.** 2×2 summary: only SFT + LoRA achieves non-zero accuracy. Both RLVR configurations and SFT + TinyLoRA fail.

Figure 3. **Training dynamics reveal a marginal-vs-conditional gap.** TinyLoRA shows learning signal (loss decreases, legal rate improves) but cannot translate this into correct predictions, indicating it learns the *marginal* distribution over chess-like tokens but not the *conditional* distribution given specific board states.

toward chess-like characters (`a-h`, `1-8`), improving the *marginal* distribution over token types. What it cannot learn is the *conditional* distribution: given *this specific* FEN position, which move is correct. That requires board-state-dependent computation exceeding the expressivity of a rank-2 shared perturbation.

Figure 3b makes this distinction tangible: TinyLoRA achieves 8-16% legal move rates (above the ∼5% zero-shot baseline), indicating it learns chess move *syntax*. But generating syntactically valid moves and generating the *correct* move for a given position are fundamentally different problems, the former requires only marginal statistics, while the latter demands conditional reasoning. Figure 3c summarizes the full method×adapter matrix: only SFT combined with standard LoRA achieves non-zero accuracy.

5.2. Why Scaling N Does Not Help

The two constraints interact to make N irrelevant above a small threshold:

Rank ceiling. Equation (2) constrains each ΔW_ℓ to rank ≤ 2 regardless of N. Our ablation shows rank 2 is sufficient *when parameterized independently*, hence the rank constraint alone is not fatal.

Coupling ceiling. The 80 decoder layers of Qwen2.5-72B encode different levels of abstraction (Geva et al., 2022). Chess plausibly requires different adaptations at different depths, early layers for board feature recognition, middle layers for tactical pattern composition, late layers for move format generation. Under TinyLoRA, a perturbation to $\mathbf{v}$ that benefits one layer may simultaneously harm another. This coupling effect bounds the useful dimension of $\mathbf{v}$ far below the nominal N, explaining why scaling from 13 to 100K produces no improvement.

Taken together, TinyLoRA can express a low-dimensional family of perturbations determined by its random projections $\{P_\ell\}$. For high-prior tasks, some direction in this family aligns with (or is neutral to) the existing computational pathway. For chess, no such direction exists, every direction that helps one layer's chess computation hurts another's, and the rank constraint simultaneously limits each layer's individual expressivity.

5.3. Frozen Directions vs. Learned Directions

The rank-vs-sharing ablation (Table 3) superficially compares 100K vs. 26.3M parameters. We argue the parameter count is not the operative variable, the *choice of perturbation directions* is.

TINYLORA's perturbation at layer ℓ (Equation 2) operates exclusively within the top-2 *singular directions of the frozen pretrained weight*. The matrix R_ℓ modulates the amplitudes and mixing of these directions, but cannot rotate them. No matter how large N becomes, ΔW_ℓ is confined to the column space of $U_\ell[:, :2]$ and the row space of $V_\ell[:2, :]$, directions that encode the *most variance in the pretrained weight*, which reflects pre-training statistics, not chess-relevant features.

Standard LORA has no such restriction. Its matrices $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ are *learned from task data* via gradient descent. They discover the 2 input and 2 output directions that are most useful for the downstream task, even if those directions are orthogonal to the pretrained SVD basis. For chess, this means LORA can learn board-state-relevant features at each layer independently, while TINYLORA is locked into the pretrained weight's principal components.

This distinction explains a puzzling feature of our results: at $N \geq 2{,}240$ (with 560 modules $\times$ 4 entries per R_ℓ), the random projections $\{P_\ell\}$ span $\mathbb{R}^{2240}$ and TINYLORA can in principle set each layer's R_ℓ independently, achieving the same degree of inter-layer independence as LORA $r = 2$. Yet it still fails. The reason is not coupling but *subspace confinement*: independent control over *how much* to perturb in frozen directions is not equivalent to independent control over *which directions* to perturb. The parameter count gap is a consequence of this architectural difference, not the cause of the performance gap.

5.4. The Prior-Dependence Hypothesis

Our three-task comparison (Figure 2b) reveals a consistent pattern that we formalize as follows:

> **Hypothesis.** *The effective intrinsic dimensionality of a fine-tuning task with respect to a base model is not a fixed constant but increases as the gap between the base model's existing competence and the task requirement grows. Ultra-low-parameter adaptation succeeds when this intrinsic dimensionality is very small (high-prior tasks) and fails when it exceeds the adapter's expressivity budget (low-prior tasks).*

On math and quantitative reasoning, the base model already knows the answer approximately half the time. The adapter's role is limited to small adjustments, format normalization, strategy selection among pre-available options. The intrinsic dimensionality of this adjustment is very low, well within TINYLORA's capacity. On chess, the model must learn to perform qualitatively new computation: spatial board evaluation, tactical pattern recognition, piece interaction modeling. This requires layer-specific, high-dimensional perturbations that no shared rank-2 adapter can express.

The practical implication is a simple screening test: *evaluate the base model's zero-shot accuracy before deploying ultra-low-parameter adaptation.* If accuracy is non-trivially above chance, minimal adapters may work. If it is near zero, higher-capacity independent adapters are needed regardless of what worked on other tasks.

Generality beyond chess. While our low-prior evaluation uses chess as the primary diagnostic, the failure mechanism identified here, subspace confinement within frozen SVD directions (Section 5.3) plus cross-layer coupling, is architecture-general and not chess-specific. Any task requiring the model to develop *new internal representations* rather than recombine existing ones should exhibit the same failure pattern. Chess is a particularly clean demonstration because its evaluation is unambiguous and its required computation (board-state-dependent tactical reasoning) is maximally distant from pre-training distributions. We predict that other tasks with near-zero base competence, such as novel formal languages, domain-specific symbolic reasoning, or perception-grounded spatial tasks, would show identical results under TINYLORA.

Training method consistency. A potential concern is that our cross-task comparison uses SFT while Morris et al. (2026) used RLVR. Importantly, both methods fail on chess: RLVR produces 0% accuracy across 8 TINYLORA configurations and 1 LORA configuration (Table 1), while SFT produces 0% across 7 TINYLORA configurations (Table 2). The failure is thus independent of optimization method. For high-prior tasks, the difference between RLVR and SFT affects the *magnitude* of improvement (Morris et al. achieve 76%→91% with RLVR vs. our 54%→60% peak with SFT) but not the *qualitative outcome*, both methods succeed when the base model has competence. The prior-dependence hypothesis is therefore robust to the choice of training paradigm.

6. Related Work

Parameter-efficient fine-tuning. The finding that LLMs occupy a low-dimensional intrinsic subspace during fine-tuning (Aghajanyan et al., 2021; Li et al., 2018) motivates methods from prefix tuning (Li & Liang, 2021) and prompt tuning (Lester et al., 2021) to sparse updates (Guo et al., 2021) and the LORA family (Hu et al., 2022; Dettmers et al., 2024). Malladi et al. (2023) show that zeroth-order gradients suffice, but on tasks where the base model has prior competence. Our results suggest that the observed low-dimensionality of fine-tuning is itself a task-dependent

property: it holds for tasks near the model's competence frontier but breaks for tasks far beyond it.

LoRA expressivity. Biderman et al. (2024) characterize LoRA as occupying a regularized subspace. Our results add a lower bound: there exists a capacity below which the subspace is too constrained for tasks requiring novel computation. TinyLoRA's shared rank-2 subspace crosses this bound for chess; rank-2 *independent* LoRA does not.

LLMs and chess. Carlini (2024) found that large models produce syntactically valid chess moves via pattern statistics from pre-training data. Our 8-16% legal move rates under TinyLoRA are consistent with this, some chess surface structure exists in the base representations. But syntactic validity and tactical correctness are distinct, and the latter requires computation beyond what surface patterns provide.

RLVR for reasoning. DeepSeek-R1 (Guo et al., 2025) and DeepSeekMath (Shao et al., 2024) demonstrate RLVR for mathematical reasoning but note the prerequisite of adequate base competence. Our chess experiments confirm this precisely: with $<$1% base success, RLVR cannot bootstrap regardless of adapter capacity.

7. Discussion

What this says, and doesn't say, about TinyLoRA. These results do not invalidate TinyLoRA, they clarify where it works. On tasks the model already half-knows, TinyLoRA nudges it in the right direction. The problem comes when people expect that to carry over to tasks the model has never seen.

Rank vs. sharing. The rank-2 LoRA ablation tells us sharing hurts more than low rank: lifting the sharing constraint alone jumps accuracy from 0% to 36%, while increasing rank from 2 to 8 only adds another 12 points. But as we show in Section 5.3, the real issue runs deeper. TinyLoRA can only perturb along the pretrained weight's own SVD directions; LoRA is free to find whatever directions the task actually needs. That difference in representational freedom, not the raw parameter count, is why 100K shared parameters get nowhere while 26M independent ones reach 36%.

Convergence. LoRA $r = 8$ is still climbing at step 500; TinyLoRA is flat. The 2,000-step RLVR runs look the same. If anything, the true gap is larger than what we report.

QLoRA interaction. Our SVD is computed on 4-bit NF4 dequantized weights, not the full-precision weights Morris et al. (2026) use. Quantization noise could degrade the subspace, but it cannot plausibly explain *zero* accuracy across four orders of magnitude of N, especially since LoRA works fine under the same quantization.

Evaluation noise. We ran a 2,000-puzzle evaluation (40$\times$ larger than the per-checkpoint batches) and got 0.25%, right at chance. With 21 independent runs all showing 0%, the odds of a false negative are astronomically small ($p < 10^{-15}$).

We discuss limitations in Section A.

8. Conclusion

We wanted to know whether TinyLoRA's remarkable efficiency on math carries over to other domains. It does not. In 21 runs covering seven parameter scales, two training methods, two model sizes, and three tasks, TinyLoRA never once solved a chess puzzle, **0% at every scale** from N=13 to 100K, on both 7B and 72B. Meanwhile, plain LoRA at rank 2 hits 36% and at rank 8 hits 48%. The same tiny N=13 adapter that is useless on chess still peaks at 60% on GSM8K and holds at 49% on AQuA-RAT, so the adapter itself is clearly doing something, just not on a task the base model knows nothing about. A 2,000-puzzle stress test confirms: 0.25%, no better than guessing.

The lesson is simple. How many parameters you need for fine-tuning depends on how much the model already knows. If the base model is already in the right ballpark, a tiny shared adapter can finish the job. If the task requires genuinely new computation, you need per-layer adapters that can learn their own directions. Where exactly that boundary falls, say, for tasks with 5-30% zero-shot accuracy, is worth figuring out next.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Aghajanyan, A., Gupta, S., and Zettlemoyer, L. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *Annual Meeting of the Association for Computational Linguistics*, 2021.

Biderman, D., Portes, J., Ortiz, J. J. G., et al. LoRA learns less and forgets less. *Transactions on Machine Learning Research*, 2024.

Carlini, N. Chess as a testing ground for language model capabilities. *arXiv preprint*, 2024.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. QLoRA: Efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems*, 2024.

Geva, M., Caciularu, A., Wang, K., and Goldberg, Y. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv preprint arXiv:2203.14680*, 2022.

Guo, D., Rush, A. M., and Kim, Y. Parameter-efficient transfer learning with diff pruning. *Annual Meeting of the Association for Computational Linguistics*, 2021.

Guo, D., Yang, D., Zhang, H., Song, J., et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*, 2022.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *Empirical Methods in Natural Language Processing*, 2021.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. Measuring the intrinsic dimension of objective landscapes. *International Conference on Learning Representations*, 2018.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *Annual Meeting of the Association for Computational Linguistics*, 2021.

Lichess.org. Lichess puzzle database. https://database.lichess.org/#puzzles, 2024.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 2023.

Morris, J. X., Shen, R., and Jurafsky, D. Learning to reason in 13 parameters. *arXiv preprint arXiv:2602.04118*, 2026. Proposes TinyLoRA: shared-vector adaptation achieving 91% on GSM8K with 13 parameters using Qwen2.5-7B-Instruct and GRPO.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

A. Limitations

We enumerate the principal limitations of this study to aid interpretation:

1. **4-bit quantization.** We compute TINYLORA's SVD on NF4 dequantized weights, whereas Morris et al. (2026) operate on full-precision models. While the quantization pathway is shared with the LORA baselines (which succeed), the dequantized SVD subspace may differ from the full-precision subspace. A full-precision replication would strengthen the finding.
2. **Single low-prior diagnostic task.** Chess is our sole near-zero-prior test case. Additional tasks with near-zero base-model competence (e.g., protein folding, low-resource languages) would test the generality of the prior-dependence hypothesis.
3. **Short training horizon.** SFT runs use 500 steps (∼2,000 examples). While TINYLORA shows no upward trend even at step 500, and RLVR runs extend to 800 steps with no improvement, longer training could in principle reveal late-emerging gains.
4. **Evaluation size.** Checkpoint evaluations use 50 samples. Although the 2,000-puzzle extended evaluation and the consistency across 21 runs mitigate this concern, per-checkpoint granularity is limited.
5. **No multi-seed runs.** Individual configurations are single-seed. We compensate with breadth: 21 runs spanning seven N values, two model scales, two training paradigms, and three tasks.
6. **SFT vs. RLVR across tasks.** Our cross-task analysis uses SFT, while Morris et al. (2026)'s original math results use RLVR. The training paradigm itself could interact with the prior-dependence effect, though our RLVR chess experiments show the same 0% failure.
7. **Prompt sensitivity.** We use a single fixed prompt template per task (Section J) and do not explore alternative phrasings. Prompt wording can influence LLM performance (Lester et al., 2021), and it is possible that a different prompt could marginally affect the results. However, because the same prompt is used for both TINYLORA and LORA, any prompt-induced bias is shared across conditions and does not affect the *relative* comparison, which is the focus of this work.
8. **Non-deterministic evaluation.** We generate with `temperature=0.3` and `do_sample=True` (Section K), introducing stochasticity. While greedy decoding would be deterministic, we use sampling to match the generation strategy during RLVR training and to avoid degenerate repetition. The consistency of 0% accuracy across 21 runs and 2,000+ total evaluation samples makes it extremely unlikely that this choice masks non-zero performance.
9. **No comparison with other PEFT methods.** We compare only TINYLORA and LORA. Other parameter-efficient methods at similar scales (e.g., prompt tuning, $(IA)^3$) could provide additional data points on the prior-dependence curve. However, our central finding, that TINYLORA specifically fails due to subspace confinement and cross-layer coupling, is an architectural diagnosis, not a ranking across PEFT methods.

B. Compute Budget

All experiments used a single NVIDIA H100 80 GB GPU. Total compute: ∼110 GPU-hours across 21 runs.

Table 4. Compute budget breakdown.

Experiment Type	Runs	GPU-Hours
TinyLoRA RLVR chess (72B)	8	∼72
LoRA RLVR chess (72B)	1	∼14
TinyLoRA SFT chess (72B)	7	∼10.5
TinyLoRA SFT chess (7B)	1	∼0.5
LoRA SFT chess (72B, r=2 and r=8)	2	∼2.5
TinyLoRA SFT math (72B)	1	∼5
TinyLoRA SFT AQuA-RAT (72B)	1	∼5
Total	**21**	**∼110**

C. Reproducibility

All experimental configurations are reported in Section 3 and summarized in Section I. Prompt templates are provided in Section J. We use publicly available models (QWEN2.5-72B-Instruct, QWEN2.5-7B-Instruct), publicly available datasets (Lichess puzzles, GSM8K, AQuA-RAT), and the standard PEFT library for LORA training. The TINYLORA implementation follows Morris et al. (2026). All experiments were conducted on a single NVIDIA H100 80 GB GPU; compute budgets are detailed in Table 4.

D. Complete Results

Table 5. All 21 experimental runs. Every run is documented with its adapter, parameter count, base model, task, training method, and best/final accuracy.

Adapter	N	Model	Task	Method	Best	Final
Phase 1: Chess , RLVR (72B)						
TINYLORA	13	72B	Chess	GRPO v1	0%	0%
TINYLORA	49	72B	Chess	GRPO v1	0%	0%
TINYLORA	196	72B	Chess	GRPO v1	0%	0%
TINYLORA	13	72B	Chess	GRPO v2 (CoT)	0%	0%
TINYLORA	49	72B	Chess	GRPO v2	0%	0%
TINYLORA	196	72B	Chess	GRPO v2	0%	0%
TINYLORA	1,000	72B	Chess	GRPO v2	0%	0%
TINYLORA	1,000	72B	Chess	Curriculum	0%	0%
LORA (r=8)	105M	72B	Chess	GRPO	0%	0%
Phase 2: Chess , SFT (72B)						
TINYLORA	13	72B	Chess	SFT	0%	0%
TINYLORA	196	72B	Chess	SFT	0%	0%
TINYLORA	1,000	72B	Chess	SFT	0%	0%
TINYLORA	5,000	72B	Chess	SFT	2%	2%
TINYLORA	10,000	72B	Chess	SFT	0%	0%
TINYLORA	50,000	72B	Chess	SFT	2%	0%
TINYLORA	100,000	72B	Chess	SFT	2%	0%
LORA (r=2)	26.3M	72B	Chess	SFT	**36%**	**34%**
LORA (r=8)	105M	72B	Chess	SFT	**48%**	**48%**
Phase 3: Chess , SFT (7B)						
TINYLORA	13	7B	Chess	SFT	0%	0%
Phase 4: Cross-Task , SFT (72B)						
TINYLORA	13	72B	GSM8K	SFT	**60%**	46%
TINYLORA	13	72B	AQuA-RAT	SFT	**49%**	49%

E. Learning Curves

Table 6. TinyLoRA $N = 13$ SFT accuracy trajectories on three tasks. GSM8K oscillates around the 54% baseline; AQuA-RAT is perfectly stable at 49%; chess is permanently 0%.

Step	GSM8K	AQuA-RAT	Chess
0 (zero-shot)	54%	49%	0%
50/100	60%	49%	0%
100/200	52%	49%	0%
150/300	50%	49%	0%
200/400	48%	49%	0%
250/500	46%	49%	0%
300	50%	-	0%
350	54%	-	0%
400	46%	-	0%

Table 7. LoRA rank-2 SFT learning curve on chess (72B). Steady improvement from 0% to 36%, still rising at step 500.

Step	Accuracy	Legal%
0	0%	6%
50	4%	6%
100	10%	10%
150	14%	16%
200	14%	14%
250	22%	26%
300	18%	20%
350	24%	36%
400	34%	44%
450	32%	38%
500	36%	42%
Final	34%	42%

Table 8. 2,000-puzzle evaluation across five difficulty levels. Both trained TINYLORA models achieve 0.25% accuracy overall (5/2000), consistent with chance.

N	Overall	Beginner	Easy	Medium	Hard	Expert
Accuracy						
13	0.25%	0%	0%	0.5%	0%	0.75%
196	0.25%	0%	0%	0.5%	0%	0.75%
Legal Move Rate						
13	9.6%	11.3%	8.8%	9.0%	7.8%	11.0%
196	9.9%	11.8%	9.5%	9.3%	7.5%	11.3%

F. Extended Evaluation

G. Mode Collapse in LoRA RLVR

During LoRA rank-8 RLVR training on chess, the model mode-collapsed by step $\sim$200. Over the final 400 steps (steps 200-600), 95% of generated outputs consisted of just two strings: `e1d1` (34/600 unique step predictions) and `e8d8` (27/600), with the remainder being empty/truncated. The loss dropped to exactly 0.000, indicating the policy gradient estimator received no useful signal. This is a textbook example of RLVR mode collapse under sparse rewards (Guo et al., 2025).

H. Architecture Details

QWEN2.5-72B has 80 decoder layers, each with 7 adapted projection types (`q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj`), totaling 560 linear modules. TINYLORA stores per module: a frozen random projection $P_\ell \in \mathbb{R}^{N\times 2\times 2}$ and frozen SVD buffers $U_\ell, S_\ell, V_\ell^\top$.

LoRA parameter calculation. For QWEN2.5-72B with hidden size 8,192 and intermediate size 29,568:

- `q_proj, o_proj`: $r \times (8192 + 8192)$ each
- `k_proj, v_proj`: $r \times (8192 + 1024)$ each (GQA with 8 KV heads)
- `gate, up, down_proj`: $r \times (8192 + 29568)$ each
- rank-2, 80 layers: $80 \times 328{,}960 = 26{,}316{,}800$ total
- rank-8, 80 layers: $80 \times 1{,}315{,}840 = 105{,}267{,}200$ total

These match the values reported by PEFT's `print_trainable_parameters()` in our training logs.

I. Hyperparameter Summary

Table 9 consolidates the training hyperparameters across all experiments for reproducibility.

Table 9. **Hyperparameter summary across all experiments.** LR = learning rate; BS = batch size per device; WD = weight decay. All runs use AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$) for SFT and GRPO for RLVR.

Adapter	Task	Paradigm	LR	BS	Steps	WD	GPU-hrs
TinyLoRA (N=13-100K)	Chess	RLVR	10^{-5}	4	800†	0.01	∼9/run
TinyLoRA (N=13-100K)	Chess	SFT	10^{-4}	4	500	0.01	∼1.5/run
TinyLoRA (N=13)	GSM8K	SFT	10^{-4}	2	500	0.01	∼5
TinyLoRA (N=13)	AQuA	SFT	10^{-4}	2	500	0.01	∼5
LoRA r=8	Chess	RLVR	10^{-5}	4	600‡	0.01	∼14
LoRA r=8	Chess	SFT	2×10^{-5}	4	500	0.01	∼1.3
LoRA r=2	Chess	SFT	2×10^{-5}	4	500	0.01	∼1.3

†One run (N=1000) extended to 2,000 steps. ‡Early-stopped; mode collapse at step ∼200.

J. Prompt Formats and Evaluation Details

All tasks use the Qwen ChatML template with a system/user/assistant structure. We provide the exact prompt format and answer extraction method for each task.

J.1. Chess (SFT and RLVR)

```
<|im_start|>system
You are a chess engine. Output only the
best move in UCI format.
<|im_end|>
<|im_start|>user
FEN: rnbqkbnr/pppppppp/8/8/4P3/8/
PPPP1PPP/RNBQKBNR b KQkq - 0 1
Black to move.

Best move:
<|im_end|>
<|im_start|>assistant
```

Answer extraction. The first UCI-format match is extracted via the regex `[a-h][1-8][a-h][1-8][qrbn]?`. Evaluation checks exact match against the ground-truth Lichess solution. Legal move rate is separately computed using a python-chess board validator.

J.2. Mathematics (GSM8K)

```
<|im_start|>system
Solve the math problem step by step.
End with #### followed by the numerical answer.
<|im_end|>
<|im_start|>user
Natalia sold clips to 48 of her friends in
April, and then she sold half as many clips
in May. How many clips did she sell in total?
<|im_end|>
<|im_start|>assistant
```

Answer extraction. We first search for the pattern `####\s*(-?[\d,]+\.?\d*)`. If not found, we fall back to the last number in the response. Commas are stripped before comparison. Evaluation checks exact match against the gold answer.

J.3. Quantitative Reasoning (AQuA-RAT)

The AQuA-RAT prompt follows the same ChatML structure. The system message instructs the model to solve the problem and output the answer letter. The user message contains the question text and all five options (A-E). Evaluation checks exact letter match against the ground truth.

K. Generation and Quantization Details

K.1. Generation Hyperparameters

Table 10 lists the generation hyperparameters used during evaluation for each task.

Table 10. **Generation hyperparameters for evaluation.**

Parameter	Chess	GSM8K	AQuA-RAT
`max_new_tokens`	10	256	256
`temperature`	0.3	0.3	0.3
`do_sample`	True	True	True
`top_p`	1.0 (default)	1.0 (default)	1.0 (default)

Chess uses `max_new_tokens=10` because valid UCI moves are at most 5 characters (e.g., `e2e4`, `a7a8q`). Math tasks use 256 tokens to accommodate chain-of-thought reasoning.

K.2. QLoRA Quantization Configuration

All models are loaded in 4-bit quantization with the following `BitsAndBytesConfig`:

```
BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.bfloat16,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
)
```

Models use `torch.bfloat16` as the compute dtype and `sdpa` (Scaled Dot-Product Attention) as the attention implementation for memory efficiency.

K.3. LoRA Adapter Configuration

```
LoraConfig(
    r=<rank>,              # 2 or 8
    lora_alpha=<rank> * 2, # 4 or 16
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj",
    ],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
)
```

Both LoRA and TinyLoRA target the same 7 projection types across all 80 decoder layers (560 modules total). For TinyLoRA, the rank is fixed at $r = 2$ with SVD computed on the dequantized weight matrices.

K.4. Software Versions

Table 11. **Software environment.**

Component	Version
Python	3.10
PyTorch	2.1+
Transformers (HuggingFace)	4.36+
PEFT	0.7+
bitsandbytes	0.41+
CUDA	12.1
GPU Hardware	NVIDIA H100 80 GB SXM