MIND LAB

On the Scaling of PEFT: Towards Million Personal Models of Trillion Parameters

Mind Lab

Parameter-efficient fine-tuning (PEFT) is usually evaluated as a cheaper alternative to full fine-tuning. This paper studies a broader possibility: whether small trainable adapters can serve as persistent local state on top of strong shared foundation models. In this view, a base model supplies common competence, while adapters may carry part of an instance-specific behavioral state, such as preferences, skills, tool habits, or memory-like updates. This framing is deliberately bounded: PEFT does not store the whole person or replace retrieval, but it may provide a compact unit of adaptive state that can be trained, evaluated, served, and composed at population scale.

We study this possibility through three coupled scaling problems that must reinforce one another. **Scale Up** asks whether a stronger shared base model makes small local updates more useful. We study large-prior LoRA reinforcement learning, routing-aware correction, and training–serving consistency at trillion-scale MoE. **Scale Down** asks how small the local adaptive state can become while still learning reliably. We analyze rank regimes, low-rank instability, RL-native initialization, hyperparameter transfer, and memory-oriented adapter designs such as δ-mem. **Scale Out** asks what becomes possible when many persistent adapted instances coexist. We examine LoRA memory capacity, context-to-parameter learning, per-user adapter simulation, and diversity-based majority voting. MinT (Mind Lab, 2026) provides one concrete infrastructure example for supporting all three axes: adapter identity, policy revision, training provenance, evaluation, and serving residency are the mechanisms that let large shared priors, small local adapters, and large adapter populations coexist.

Taken together, these results suggest that PEFT is more than a budget-conscious substitute for full fine-tuning. A personal model built on a strong shared prior can preserve continuity across repeated interaction, serve as a stable user simulator for agents that treat the user as part of their environment, and contribute to collective performance through diversity-based aggregation. The broader ambition is a world where strong foundation models support not one universal assistant, but millions of persistent personal ones.

Correspondence: contact@mindlab.ltd
Date: May 2026

1 Introduction

Frontier models can now write production code, operate tools, and reason across long contexts (OpenAI, 2025; GLM-5 Team, 2026; Kimi Team, 2025; Qwen Team, 2025; Anthropic, 2025a). Agentic systems built on these models resolve real-world software engineering tasks autonomously (Anthropic, 2025b; Wang et al., 2024; Jimenez et al., 2024). But a capable assistant is not automatically a personal one. It may answer more questions and call more tools, and still fail to preserve continuity with one person over time. Long context, retrieval (Lewis et al., 2021), prompts (Ji, 2025), and user profiles (Li et al., 2024) all help, but are not enough by themselves. A personal model needs state that can persist, adapt, and shape future behavior (Yao, 2025; Silver and Sutton, 2025).

This paper argues that parameter-efficient fine-tuning (PEFT), especially LoRA (Hu et al., 2021), is a prac-

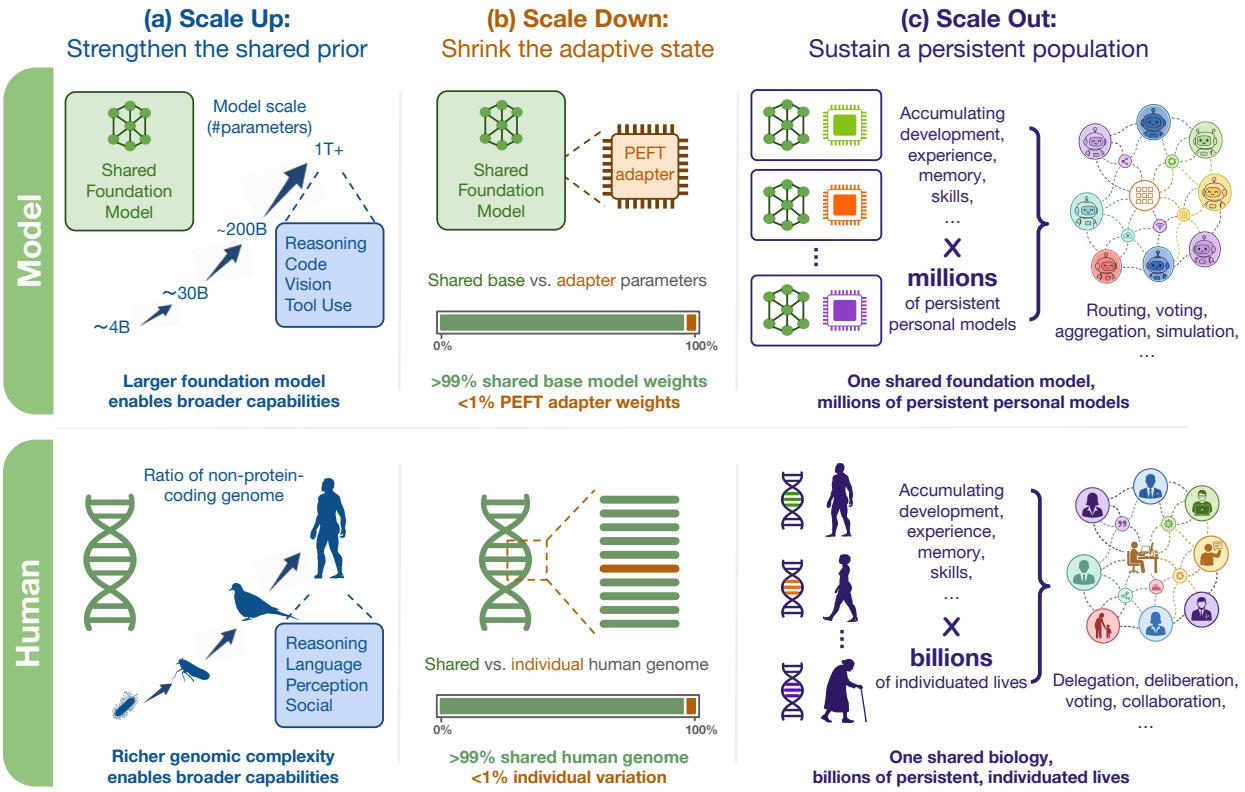


Figure 1 The three scaling axes of PEFT, illustrated through a biological analogy. **(a) Scale Up.** As organisms grow more complex up the tree of life, the fraction of regulatory DNA increases, from roughly 10% in simple organisms to 98% in humans (ENCODE Project Consortium, 2012), enabling richer reasoning, perception, and social behavior. Analogously, larger foundation models unlock capabilities such as reasoning, code, vision, and tool use that make small local updates more powerful. **(b) Scale Down.** Human individuals share more than 99% of their genome (1000 Genomes Project Consortium et al., 2012); the differences that make each person distinct amount to less than 1% of the total. A PEFT adapter occupies a similarly small fraction of the base model weights, and carries the local adaptive state that distinguishes one personal model from another. **(c) Scale Out.** One shared biology supports billions of persistent, individuated human lives, each accumulating its own development, experience, memory, and skills. One shared foundation model can support millions of persistent personal model instances in the same way, each shaped by its own history and each a member of a larger population.

tical way to represent part of that state. The goal is not PEFT itself. The goal is a persistent personal model instance: a system built from a shared base model, local adaptive state, context, tools, and external memory. The base model provides broad intelligence. The adapter carries part of the learned consequences of repeated experience, such as preferences, skills, tool habits, and some memory-like behavior. Raw facts, episodes, and documents can still live in retrieval systems or external memory (Packer et al., 2024; Zhong et al., 2024). The adapter is not the whole memory system.

The biological analogy in Figure 1 frames the architecture. Any two humans share approximately 99.9% of their genome (1000 Genomes Project Consortium et al., 2012). This small fraction of variation is sufficient to produce the full range of human individuality. One shared biology supports billions of persistent, individuated lives, each accumulating its own experience, and population diversity is itself a resource. The development of foundation models may follow a similar trajectory: shared priors growing stronger, local adaptive states remaining small, and populations of persistent personal instances becoming the unit of scale.

We organize the technical path around three coupled scaling problems. **Scale Up** asks how to make strong shared base models repeatedly adaptable. **Scale Down** asks how small the local adaptive state can be while still carrying meaningful individuality. **Scale Out** asks what becomes possible when many persistent adapted

instances coexist. The three axes are dependencies, not independent categories. Scale Up without Scale Down produces powerful priors too expensive to adapt continuously. Scale Down without Scale Up produces cheap but low-leverage adapters. Scale Out without both produces many weak or disposable variants rather than durable personal models. Section 2 develops this dependency structure, and Sections 3–6 provide the technical evidence for each axis.

The downstream implications of personal models extend in three directions: preserving continuity for individuals, improving user simulation for agents that treat the user as part of their environment (Park et al., 2023; Yang et al., 2024), and enabling collective intelligence through diversity among adapted models. These are directions rather than solved problems. What PEFT contributes is a practical unit of persistent individuality that is small enough to scale.

These claims require calibration. PEFT does not store the whole person, replace retrieval, solve personal memory by itself (Packer et al., 2024), reproduce society, or guarantee creativity. It does something narrower but important: it makes a portion of individuality local, persistent, efficient, and manageable. A small adapter can change without rewriting the shared base model, can be updated repeatedly at lower cost than a full checkpoint (Biderman et al., 2024), and can be named, evaluated, served, rolled back, or retired as part of a lifecycle (Mind Lab, 2026).

This paper makes the following contributions. (1) We introduce a three-axis framework (Scale Up, Scale Down, Scale Out) for understanding PEFT as a scaling mechanism for personal models, and show that the axes form a dependency chain rather than a loose taxonomy. (2) We demonstrate trillion-scale LoRA RL on a 1T-parameter MoE model and identify scale-induced failure modes including training–inference mismatch in sparse architectures. (3) We characterize LoRA rank regimes under RL fine-tuning, propose OLoRA-tail as an RL-native initialization, and establish hyperparameter transfer rules across ranks. (4) We measure LoRA memory capacity laws, introduce Context Learning as a write policy for personal adapters, and show that per-user LoRA adapters produce richer social simulation structure than shared-base agents, and demonstrate in a controlled model-count experiment that diversity among distinct LoRA variants produces collective intelligence under majority voting, raising AIME24 accuracy from 0.3644 at $k = 1$ to 0.4867 at $k = 198$.

The spine of the paper is therefore simple: PEFT makes it possible to scale from one shared foundation model to many persistent personal model instances. The near-term path is algorithmic and infrastructural. The long-term promise is personal service, better user simulation, and useful diversity among adapted models.

2 The Three Scaling Axes of PEFT

PEFT scales along three coupled axes because a personal model instance is both a learning problem and a systems object. Scale Up asks how strong the shared base model must be before small local updates become high leverage. Scale Down asks how small, stable, and repeatedly writable the local adaptive state can become. Scale Out asks what becomes possible when many persistent adapted instances can be trained, served, evaluated, and governed over time.

1. **Scale Up**: a stronger shared base model gives each small adapter more latent capability to redirect.
2. **Scale Down**: a smaller and stabler adaptive state lowers the marginal cost of repeated learning.
3. **Scale Out**: low marginal cost allows personalization to expand from isolated adapters to populations of persistent model instances.

The dependencies also define failure cases. Scale Up without Scale Down produces powerful priors that remain too expensive to adapt continuously. Scale Down without Scale Up produces cheap adapters with limited capability to specialize. Scale Out without both produces many weak or disposable variants rather than durable personal models. The thesis of this paper is that PEFT becomes transformative only when the three axes reinforce one another.

The axes support the three visions in sequence. For individuals, they make repeated personalization plausible: a strong base model supplies broad competence, a local adapter stores part of the learned behavioral state, and lifecycle infrastructure keeps the instance persistent. For user simulation and agent environments, they make it possible for simulated users to preserve stable preferences, goals, memories, and constraints across

Table 1 The three scaling axes of PEFT in the latest thesis outline.

Axis	Description	Function
Scale Up	Increase the capability of the shared prior.	Makes small updates high leverage by providing richer latent structure to adapt.
Scale Down	Reduce the marginal cost and instability of adaptation.	Makes repeated learning, storage, and serving feasible for many instances.
Scale Out	Increase the number and diversity of persistent adapters.	Enables personalization, population diversity, and emergent population-level behavior.

interactions. For collective intelligence, they make diversity among adapted policies measurable and usable through voting, routing, debate, or distillation.

Long-term memory is where the three axes meet most clearly. Context and retrieval remain essential, but a personal model also needs some learned state that persists beyond the current prompt. The adapter should not become a raw archive of the user's history. Instead, it should carry part of the behavioral consequences of repeated experience, while editable facts and documents remain in external memory. This distinction keeps the claim precise: PEFT is a mechanism for local adaptive state, not a complete memory system.

The axes also correspond to three kinds of evidence used in this manuscript. Scale Up is grounded in trillion-parameter LoRA RL and the infrastructure needed to make large priors trainable. Scale Down is grounded in LoRA rank, initialization, and hyperparameter studies that ask how little trainable state can still learn reliably. Scale Out is grounded in memory, agent, user-simulation, and aggregation settings where the number and diversity of persistent adapted instances become objects of scaling.

3 Scale Up: Scaling Model Capacity for a Stronger Shared Prior

Scale Up is the first technical requirement for persistent personal model instances. A personal adapter is useful only when the shared base model already contains broad capabilities that a small local update can redirect. The goal is therefore not to replace personalization with larger pretraining, but to make stronger shared base models repeatedly adaptable under realistic learning budgets.

This question is especially sharp in reinforcement learning. RL can reinforce reasoning strategies, tool-use policies, and long-horizon behaviors, but it can only reinforce trajectories that the current policy can sample with sufficient probability. A stronger shared base model changes the effective search space by making useful but unstable behaviors reachable. LoRA changes the economics by allowing the learning loop to operate on that stronger base without updating all parameters. From the perspective of personal models, Scale Up asks why a small local adaptive state becomes more valuable as the shared base becomes stronger.

This section develops the Scale Up axis in five steps. First, we argue that RL is prior-limited: the base model determines which trajectories exploration can discover and which behaviors credit assignment can reinforce. Second, we explain how LoRA changes the scaling trade-off by turning PEFT into budgeted access to stronger priors. Third, we discuss how trillion-scale LoRA RL makes such priors operationally reachable in real learning loops. This is an engineering feasibility argument, distinct from the capability argument above, and both are necessary for the Scale Up thesis. Fourth, we analyze scale-induced failure modes that emerge when large-prior adaptation spans sparse architectures, distributed rollout, sharded optimization, adapter semantics, and serving runtimes. Finally, we connect Scale Up to the next axis: once strong priors can be adapted, the next question is how small, stable, and efficiently trainable the local adaptive state can become.

3.1 Why RL is Prior-Limited

Recent progress in reasoning-oriented reinforcement learning has made one point increasingly difficult to ignore: under realistic budgets, RL is often more effective at amplifying behaviors that already exist in weak or unstable form than at creating sophisticated capabilities *de novo* (DeepSeek-AI, 2025; Hu et al., 2025a). DeepSeek-R1-Zero showed that large-scale RL can elicit reasoning behaviors such as self-reflection,

verification, and long-chain exploration without relying solely on conventional supervised reasoning traces (DeepSeek-AI, 2025). Open-Reasoner-Zero further showed that relatively simple on-policy RL recipes can improve reasoning performance when applied to a sufficiently capable base model (Hu et al., 2025a). These results support the view that RL can unlock latent capabilities, but they do not imply that RL is independent of the base model. On the contrary, they make the role of the base-model prior more visible.

RL does not search over an abstract space of possible capabilities. It searches through the trajectory distribution induced by the current policy. In language-model RL, the action space is the vocabulary-conditioned continuation distribution over long sequences, and the probability of sampling a useful reasoning trajectory is strongly shaped by the pretrained policy. The bottleneck is therefore not only reward quality, but trajectory support: the base policy must assign sufficient probability mass to behaviors that the reward can later select and reinforce.

This prior-limited view explains why stronger models can make RL more productive. Exploration improves when the model can already propose partially useful trajectories. Credit assignment improves when competent trajectories occur often enough to be distinguished from noise. Transfer improves when the base model already encodes broad latent structure. A weak model may rarely visit high-reward reasoning patterns, making policy-gradient updates sparse, high-variance, or overly dependent on reward shaping. By contrast, a strong model can assign non-negligible probability mass to useful but unstable behaviors, allowing RL to reinforce and regularize them.

This observation reframes the role of scale. Larger models are not valuable only because they have higher static benchmark performance. They are valuable because they alter the distribution of trajectories available to post-training. A stronger prior can make reasoning paths, tool-use strategies, and long-horizon behaviors reachable before they are stable. RL can then act less as a mechanism for inventing capability from scratch and more as a mechanism for selecting, sharpening, and stabilizing behaviors already latent in the model.

For personal models, this point is fundamental. A local adaptive state can only be high-leverage if the shared prior already contains useful structure to redirect. Without such a prior, personalization risks collapsing into shallow memorization, narrow task fitting, or brittle behavioral patches. With a strong prior, small updates can produce disproportionately large behavioral changes because they operate on a model that already encodes much of the world, the task format, and the relevant reasoning patterns.

3.2 LoRA as Budgeted Access to Strong Priors

LoRA changes the scaling trade-off from how many parameters can be updated to how much prior can be brought into the learning loop under a fixed budget. LoRA was originally proposed as a low-rank adaptation method that freezes pretrained weights and injects trainable low-rank matrices, thereby reducing the number of trainable parameters and the deployment cost by orders of magnitude while preserving the shared base model (Hu et al., 2021). In ordinary supervised fine-tuning, LoRA is often interpreted as a storage- and memory-saving technique. In the Scale Up axis, however, LoRA has a more structural role: it makes stronger priors economically accessible to repeated optimization.

This distinction matters because the central comparison is no longer simply ‘full fine-tuning versus adapter tuning”. It is instead ‘how much prior can be brought into the learning loop under a fixed adaptation budget”. A smaller fully trainable model may expose more trainable parameters relative to its size, but it may still lack the latent reasoning substrate that makes RL productive. A larger LoRA-adapted model may expose fewer trainable parameters, but those parameters act on a stronger prior. In this regime, the adapter does not need to carry the full capability. It only needs to steer the prior.

LoRA and full-parameter training should therefore not be treated as identical optimization regimes. Prior work has shown that low-rank adaptation differs from full fine-tuning in forgetting behavior, representation movement, and update geometry (Biderman et al., 2024; Shuttleworth et al., 2025). Notably, Biderman et al. (2024) find that LoRA forgets less than full fine-tuning, a property that is advantageous for personal models that must preserve base capabilities while acquiring new ones. Shuttleworth et al. (2025) further show that LoRA and full fine-tuning are not equivalent in representation movement, cautioning against the simplistic claim that LoRA is always a substitute for full training. Together, these differences explain why LoRA can be especially effective when the goal is not to relearn a task from scratch, but to modulate a strong pretrained

Table 2 Large prior plus small LoRA update can outperform full RL on a smaller model under comparable RL budgets. Values are summarized from the motivating Mind Lab trillion-parameter LoRA RL study.

Model and adaptation	Trainable parameters	AIME 2025 normalized gain	GPQA Diamond normalized gain
DS-Distill-Qwen-1.5B, full RL	1.5B	8.33%	25.00%
DS-Distill-Qwen-7B, LoRA r=64	0.16B	11.31%	27.23%
DS-Distill-Qwen-32B, LoRA r=8	0.07B	20.61%	33.02%

representation. The rank constraint is a limitation when new capability must be built from little support, but it can be an advantage when adaptation only needs to redirect latent structure.

The motivating comparison in this manuscript illustrates this point. Under broadly comparable RL budgets, larger base models adapted with LoRA achieved larger headroom-normalized gains than a much smaller model trained with full-parameter RL, despite using substantially fewer trainable parameters. This comparison separates three quantities that are often conflated: total model capacity, activated computation, and trainable parameter count. Its significance is not a universal ranking of LoRA over full training. Rather, it shows that, when learning budgets are fixed, the strength of the prior can matter more than the size of the trainable surface.

This is the core principle behind Scale Up. The objective is to make stronger priors repeatedly accessible to PEFT-based learning, rather than simply to make models larger. If a frontier prior can only be used as a static checkpoint, its value for personalization and continual adaptation remains limited. If it can be updated cheaply and reliably, it becomes a reusable substrate for reasoning optimization, tool-use refinement, domain specialization, and, ultimately, memory formation in persistent personal models.

A note on interpretation: the motivating comparison above varies model size and training method simultaneously, so it does not cleanly isolate prior strength from trainable parameter count. Its significance is not a universal ranking of LoRA over full training. Rather, it illustrates that, when learning budgets are fixed, the strength of the prior can matter more than the size of the trainable surface. The subsequent engineering evidence in Section 3.3 supports this claim by showing that large-prior LoRA RL can be made operationally stable.

3.3 Operationalizing Trillion-Scale LoRA RL

If strong priors make adaptation high-leverage, the next question is whether such adaptation can be made operationally viable at frontier scale. At this scale, LoRA is no longer merely a low-rank parameterization. It becomes part of a distributed learning system. The adapted policy must remain coherent across rollout, optimization, checkpointing, model conversion, and serving. Frontier-scale PEFT therefore depends as much on systems alignment as on parameter efficiency.

Trillion-scale LoRA RL combines several difficulties that are usually treated separately. The base model is too large for naive replication. Sparse MoE structure introduces expert routing and all-to-all communication. The RL loop requires frequent alternation between inference-style rollout and training-style optimization. The adapter state is small relative to the base model, but it must still be sharded, synchronized, merged, and served correctly. A naive data-parallel LoRA implementation fails because it assumes that the base model can be treated as a monolithic frozen object and that the adapter can be attached as a local modification. At trillion scale, neither assumption holds.

This challenge is especially acute in mixture-of-experts architectures. In dense models, numerical differences between training and inference may perturb activations while leaving the computational pathway unchanged. In MoE models, the same perturbations can alter routing decisions, causing tokens to traverse different experts and thereby changing the effective computation itself. Moreover, expert layers and dense layers play different roles in the computation. If adapters are attached only to dense submodules, the RL signal may not adequately affect expert-specific behavior. If adapters are attached naively to expert submodules,

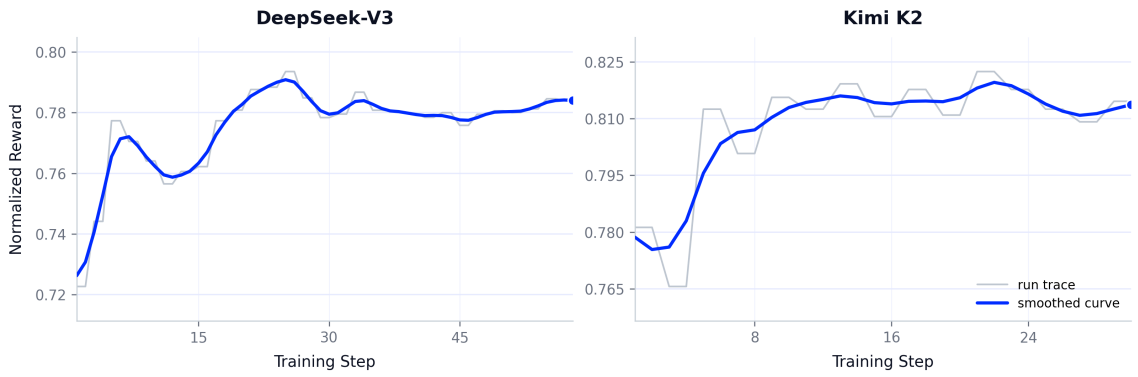


Figure 2 GRPO LoRA training on large-scale LLMs. Stable reward and task-success curves support the feasibility of adapting a trillion-parameter MoE reasoning model with LoRA-based RL when rollout, training, and hybrid parallelism are jointly designed.

communication and checkpoint handling can become infeasible. Practical trillion-scale LoRA RL therefore requires adapter placement to be co-designed with the model's sparse computation structure.

The Kimi K2 LoRA RL case serves as a proof of existence for this regime: a trillion-parameter sparse prior can be brought into an on-policy RL loop without full-parameter updates, provided that adapter placement, rollout generation, and distributed optimization are co-designed. The system applies RL to a trillion-parameter MoE reasoning model with 32.6B activated parameters and 1.04T total parameters. It uses LoRA on selected dense and expert layers, a GRPO-style on-policy optimization loop, and hybrid tensor, pipeline, expert, and sequence parallelism. This configuration is important not because each component is individually novel, but because all components must operate as a single adaptation system.

The use of GRPO-style on-policy optimization is central to the Scale Up argument because it makes the system a reinforcement learning loop rather than a static supervised adapter update. The adapter is optimized from sampled reasoning trajectories and reward feedback, so the quality of the base prior directly affects both exploration and credit assignment. In other words, the system tests the central claim under an actual RL regime: a strong prior is useful only if it can be sampled, evaluated, updated, and re-sampled without breaking policy consistency.

The core design principle is to treat parallelism as a schedulable resource rather than a fixed layout. Rollout and training have different computational profiles. Serving-oriented rollout benefits from high-throughput decoding and efficient KV-cache management, whereas training-oriented optimization requires sharded gradients, optimizer states, and backward computation. A practical system must bridge these two regimes without allowing the policy used for sampling to drift from the policy used for optimization. This is why integrating a rollout engine with a Megatron-style training backend is conceptually central. Such integration converts a frontier prior from an expensive static object into an adaptation target that can repeatedly enter the RL loop.

LoRA contributes to this system in three ways. First, it reduces the amount of trainable state, making optimizer memory and gradient communication manageable. Second, it enables multiple RL runs or downstream variants to share the same frozen base model, which is essential for amortizing the cost of frontier-scale priors. Third, when attached to both dense and expert components, it allows the RL signal to influence global reasoning behavior as well as expert-specific computation. In the Kimi K2 system, this design reduces the compute and communication footprint to approximately 10% of conventional full-parameter RL while preserving the ability to adapt a trillion-scale prior.

Empirically, the Kimi K2 experiment provides three pieces of evidence for this claim. First, LoRA RL reduces the required GPU budget to roughly 10% of conventional full-parameter RL on the same class of model. Second, the training curves show smooth improvement in reward and task success rate without catastrophic divergence. Third, held-out evaluations suggest that the adapter improves task-specific behavior while preserving the general capabilities of the base model. These observations show that trillion-scale LoRA

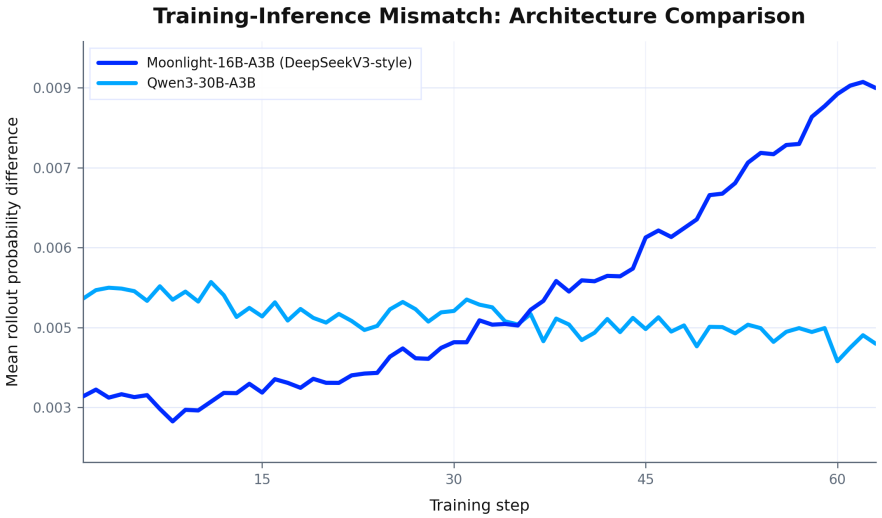


Figure 3 TIM comparison between Moonlight-16B-A3B (DeepSeekV3-style) and Qwen3-30B-A3B. The DeepSeekV3-style architecture shows substantially higher training–inference mismatch.

RL is not merely memory-efficient. It can also remain stable and behaviorally useful when the distributed system is designed around MoE parallelism.

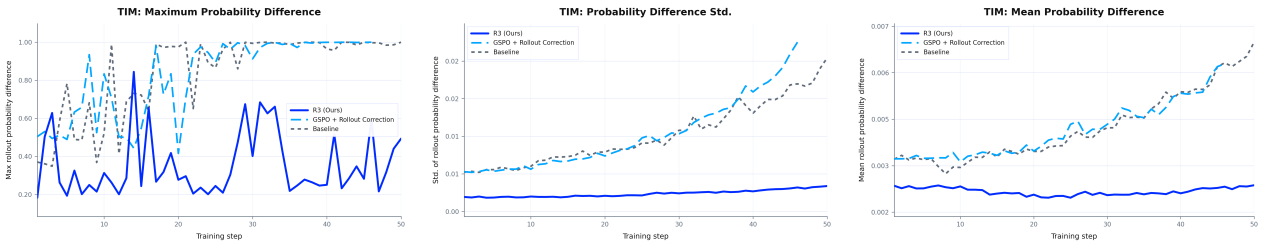
This result establishes the positive side of Scale Up: strong priors can be made reachable by PEFT-based RL. It also reveals the other side of Scale Up. Once a frontier prior becomes trainable, the learning problem no longer resides only in the optimizer or the adapter. It is distributed across the execution path that generates rollouts, the backend that computes gradients, the sparse architecture that routes tokens, and the runtime that later serves the adapted model. Scale therefore introduces not only larger capability, but also new failure modes.

3.4 Scale-Induced Failure Modes

Scaling up the prior does not merely increase the number of parameters in an otherwise unchanged training recipe. It changes the failure surface of PEFT-based reinforcement learning. In small and medium-scale models, discrepancies among rollout, training, checkpointing, and serving are often absent, numerically negligible, or hidden by a simple execution path. At frontier scale, especially in sparse MoE reasoning models, the same discrepancies can become algorithmic, architectural, adapter-semantic, and lifecycle-level failures.

We organize these scale-induced failures into four categories. First, **algorithmic mismatch failures** arise when the policy that generates rollouts differs from the policy optimized during training, violating the assumptions of on-policy RL. Second, **sparse-architecture failures** arise when small execution differences change expert routing, sparse-attention selection, or other discrete computation paths. Third, **adapter-semantics failures** arise when a low-rank adapter is attached, transformed, or interpreted in a way that changes the intended meaning of the adapted update. Fourth, **lifecycle and serving failures** arise when the trained adapter is saved, merged, quantized, or served under a runtime that no longer instantiates the same effective computation. This taxonomy is not meant to separate independent bugs. Rather, it identifies the coupled failure surface that emerges when large-prior adaptation spans reinforcement learning, sparse architectures, distributed execution, adapter semantics, and serving infrastructure.

The first case is training–inference mismatch (TIM), an algorithmic mismatch failure. In a standard policy-gradient view, trajectories are sampled from a behavior policy, and gradients are computed to improve the corresponding policy under some form of probability-ratio correction. This view assumes that the policy used for rollout and the policy used for optimization are well defined and comparable. In large-model RL systems, however, rollout and training are often executed by different engines: a serving-oriented inference stack generates trajectories, while a training-oriented distributed backend computes losses and updates. In dense models, the difference between these two executions may remain a small numerical perturbation. In

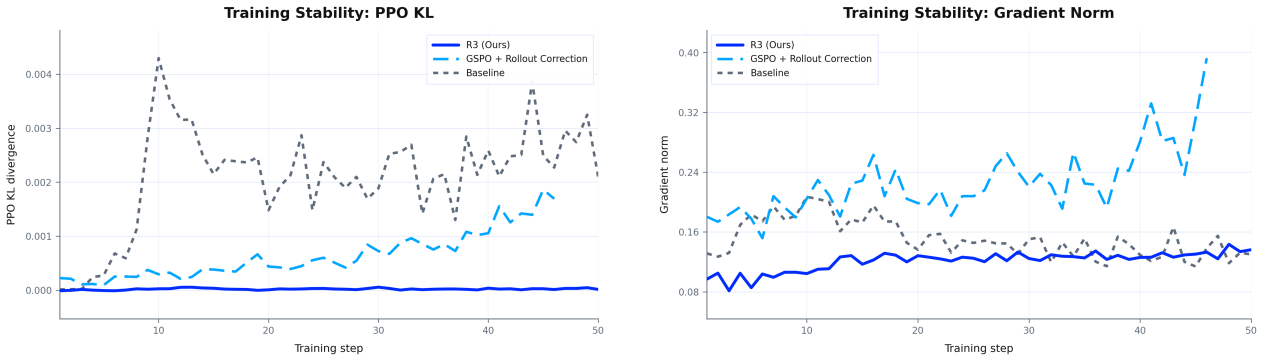


(a) Maximum rollout probability difference per step. The original and rollout-corrected runs climb rapidly and remain high, whereas the R3-fixed run stays consistently lower.

(b) Standard deviation of rollout probability difference. R3 maintains lower variance throughout training.

(c) Mean rollout probability difference. R3 substantially reduces TIM relative to the baselines.

Figure 4 TIM metrics: rollout probability mismatch. Across maximum difference, standard deviation, and mean difference, Router Replay R3 consistently reduces the mismatch between rollout-side and training-side probabilities relative to the original and rollout-corrected baselines.



(a) PPO KL divergence. The R3 run maintains near-zero KL divergence (0.000026 at step 46), while other runs increase substantially.

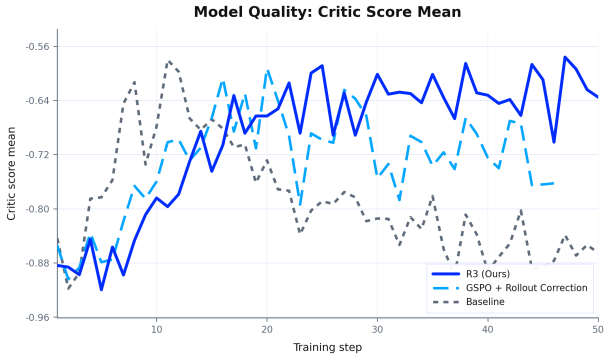
(b) Gradient norm. R3 exhibits the most stable gradients, whereas the baselines show higher variance and drift.

Figure 5 Training stability signals. Router Replay R3 stabilizes the RL update by maintaining lower KL divergence and smoother gradient norms than the original and rollout-corrected baselines.

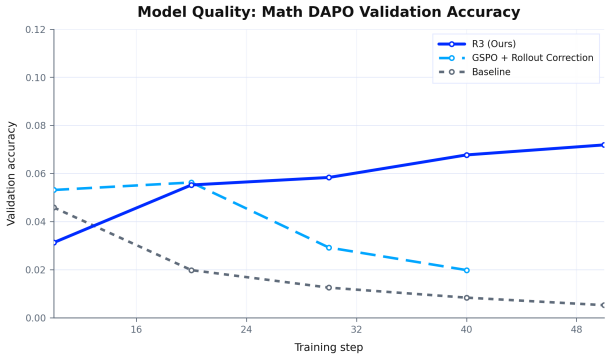
MoE models, the same perturbation can change discrete routing decisions, causing tokens to pass through different experts. Once this occurs, the training engine is no longer optimizing the same effective computation that produced the rollout.

TIM is therefore more than a numerical precision issue. It becomes an algorithmic failure mode because it changes the object of the policy update. The nominal policy may be the same checkpoint with the same adapter, but the effective policy differs if the rollout path and the training path activate different experts. Importance correction can mitigate moderate probability mismatch, but it assumes that the two policies remain comparable distributions over the same underlying computation. When routing divergence changes the computation graph itself, this assumption becomes fragile. This is why large-scale MoE RL requires mechanisms that are not central in small-model training, such as routing-aware correction, routing replay, or stricter rollout–training equivalence.

Router Replay R3 illustrates how a sparse-architecture failure can become an RL failure. Its significance is not merely that it fixes a particular implementation problem, but that it treats routing as part of the effective computation to be preserved. In routing-sensitive MoE architectures, preserving the sampled computation requires preserving the routing decisions that determine which experts participate in the forward pass. If inference-side and training-side routing diverge, gradients are computed through a sparse path different from the one that generated the samples. R3 addresses this issue by recording routing information during rollout and replaying it during training, thereby reducing the semantic gap between the sampled policy



(a) Critic score mean. The R3 run sustains higher scores than the baselines, which trend downward.



(b) Validation accuracy on the math DAPO task. The R3 run improves monotonically and finishes strongest, while the original and rollout-corrected runs degrade.

Figure 6 Quality metrics. Router Replay R3 improves downstream quality by sustaining higher critic scores and achieving stronger validation accuracy on the math DAPO task.

and the optimized policy. The broader lesson is that, at scale, the RL algorithm can no longer be defined independently of the execution path that realizes the policy.

The second case is GLM5 and GLM5.1 support, which illustrates how sparse-architecture and adapter-semantics failures can appear together. Supporting a small dense model often means implementing the architecture, loading the checkpoint, attaching adapters, and running training or inference with relatively direct correspondence among these steps. Supporting a frontier MoE model is fundamentally different. GLM5-style models combine MoE, Multi-Head Latent Attention (MLA), DeepSeek Sparse Attention (DSA), Multi-Token Prediction (MTP), LoRA adaptation, training-time distributed execution, inference-time serving kernels, and checkpoint bridge logic. Each component can be locally correct, yet the full system can still be globally inconsistent if the training stack, inference stack, and bridge do not interpret the model in the same way.

This problem appears in several forms. DSA makes token selection part of the computation semantics: a small difference in the indexer or top-k behavior can change which tokens are included in sparse attention. MTP touches model structure, loss computation, output heads, and checkpoint conversion simultaneously, so an adapter trained under one interpretation of the MTP path may not be equivalent when loaded under another runtime. MLA and other specialized projection modules may contain custom forward logic, dtype behavior, fused kernels, or tensor-parallel communication patterns. A generic LoRA wrapper that is correct for ordinary linear layers may therefore be semantically incorrect for these modules. In such cases, the adapter may load successfully, but the served model is no longer the same adapted computation that was trained.

The GLM5/GLM5.1 case therefore shows that, at frontier scale, infrastructure is not a passive substrate for algorithms. It becomes part of the model definition. The bridge from training to inference is a semantic preservation problem rather than a file-format conversion. The serving runtime must preserve the same sparse attention, latent attention, expert, and adapter semantics. The training backend must instantiate the same computation that the inference engine will later serve. This is why model-family support at scale often becomes a full-stack alignment problem spanning training, inference, and checkpoint conversion.

Together, TIM, R3, and GLM5 support show that scale changes not only what the model can learn, but also how it can fail. At frontier scale, PEFT-based RL fails not only through poor rewards, unstable gradients, or insufficient adapter capacity, but also through mismatches among routing, sparse attention, adapter interpretation, checkpoint conversion, and serving execution. These failures are distinctive to large-prior adaptation because the effective computation is distributed across multiple systems rather than contained in a single local model object.

This reframes failure modes at scale. They are not simply bugs, numerical instabilities, or inconvenient engineering details. They indicate that the training regime itself has changed. Small-model experience often assumes that model execution is simple enough for training, inference, and serving to be treated as interchangeable realizations of the same policy. Frontier-scale MoE PEFT breaks this assumption. Routing,

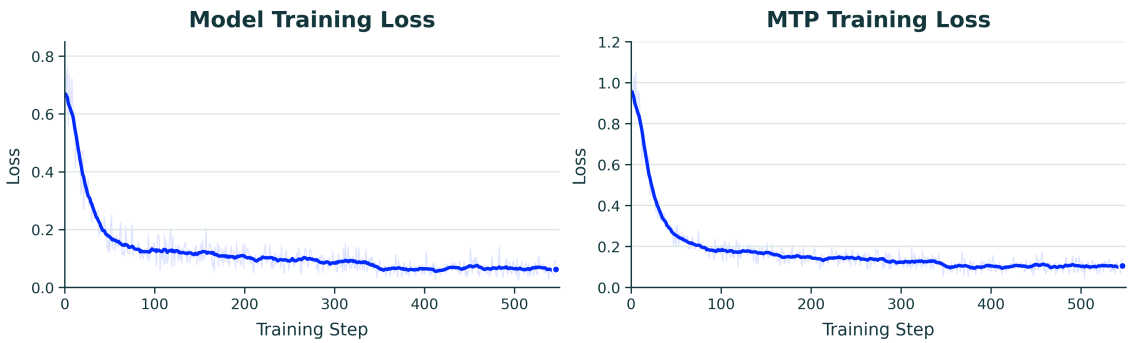


Figure 7 Training loss curves for the model and MTP components of GLM5.1 with LoRA adapters.

sparse selection, adapter interpretation, checkpoint conversion, and serving kernels all become part of the operational meaning of the adapted model. As a result, correctness must be defined not only by whether training loss decreases, but also by whether the adapted computation remains consistent across the full lifecycle from rollout and training to export, merge, and serving.

For persistent personal models, this issue becomes even more important. Adapters are not disposable optimization artifacts. They may carry task-specific behavior, user-specific preferences, domain experience, or long-term memory. If the infrastructure reinterprets an adapter during loading or serving, the system may preserve the file while losing the learned behavior. If the training algorithm optimizes an effective computation different from the rollout policy, the system may improve a surrogate that was never actually deployed. Scale Up therefore requires more than access to a large prior. It requires mechanisms that preserve the meaning of adaptation across the full lifecycle of the adapted model.

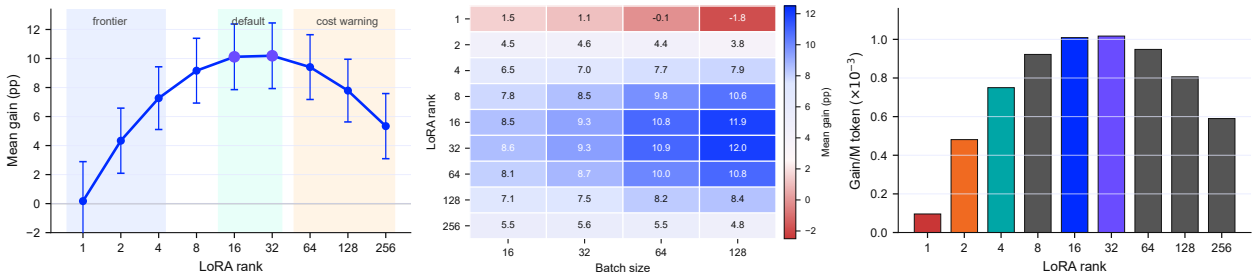
3.5 From Scale Up to Scale Down

Scale Up establishes the first condition for PEFT-based personal models: the adaptive boundary must sit on top of a sufficiently strong shared prior. RL is prior-limited because it can only reinforce trajectories that the current policy can sample. LoRA changes the economics of this process by allowing stronger priors to enter the learning loop under a fixed adaptation budget, and trillion-scale LoRA RL shows that such priors can be adapted in real on-policy training systems. Scale-induced failure modes then show that this adaptation must remain consistent across sparse architectures, distributed execution, adapter semantics, and serving lifecycles.

However, Scale Up alone is not enough. A strong prior can make each update more powerful, but it does not by itself make adaptation continuous, stable, or cheap enough to support persistent model instances. If the adaptive unit is too large, unstable, brittle, or expensive to train and serve, then large-prior adaptation remains occasional rather than continuous. Powerful priors would remain impressive shared artifacts, but not writable substrates for long-term personalization.

This is where the next axis begins. Scale Down asks how small, stable, and efficiently trainable the adaptive unit can become. It shifts attention from the strength of the shared prior to the cost, reliability, and repeatability of the local update. In the three-axis framework, Scale Up supplies the capability substrate. Scale Down then makes the adaptive unit cheap and reliable enough for repeated learning, and Scale Out expands these adaptive units into populations of persistent model instances.

These observations foreshadow the infrastructure problem developed in Chapter 6, where we introduce MinT (Mind Lab, 2026) as a LoRA-based framework for multi-train and multi-serve learning over shared base models. For the purposes of the present Scale Up axis, the point is narrower: once a strong prior can be adapted, the system must make that adaptation repeatable across training, evaluation, transfer to inference, and serving. This requirement exposes the next bottleneck. The adaptive unit itself must be compact, stable, and efficiently writable; otherwise, large-prior adaptation remains an isolated event rather than a sustained learning regime.



(a) Mean gain by rank. **(b)** Rank–batch gain map. **(c)** Token efficiency.

Figure 8 Rank-sweep overview. The Qwen3-8B PPO sweep separates LoRA rank into low-rank research frontier, middle-rank deployment default, and high-rank cost-warning regimes rather than a monotonic capacity curve.

4 Scale Down: The Operating Regime of Efficient Adaptation

If Scale Up gives PEFT its leverage, Scale Down decides whether that leverage can be exercised widely enough to sustain persistent personal model instances. The goal is to identify the smallest reliable unit of persistent learning: a local adaptive state that is expressive yet stable, cheap to update and store, and practical to serve across many instances at once.

How small can this adaptive state become before it stops learning reliably? The answer is a regime, not a fixed threshold. Under standard recipes, middle-rank adapters already learn dependably across seeds. Extremely low-rank adapters are different: they reach the same best-case performance, but they do not reach it reliably, and closing that reliability gap still depends on better initialization, tighter variance control, and transferable hyperparameters. This leads to a deliberately limited claim. The smaller an adapter can become while staying stable, the cheaper it is to train, store, and serve the many instances a personal-model population requires.

This section develops Scale Down in two movements. Section 4.1 stays within the standard LoRA setting, in which the adaptive state is a static low-rank update, and examines three coupled questions: how far rank can be reduced before across-seed reliability collapses, how RL-native initialization can rescue the tiny-adapter regime, and how learning-rate recipes can transfer across ranks so that adapter populations do not require per-instance tuning. Section 4.2 then looks beyond static LoRA and asks whether the adaptive unit can itself be redesigned as a compact, writable state that accumulates interaction history, taking δ-mem and related stateful adapters as the representative case. Together, the two movements extend Scale Down from parameter reduction to adaptive-state design.

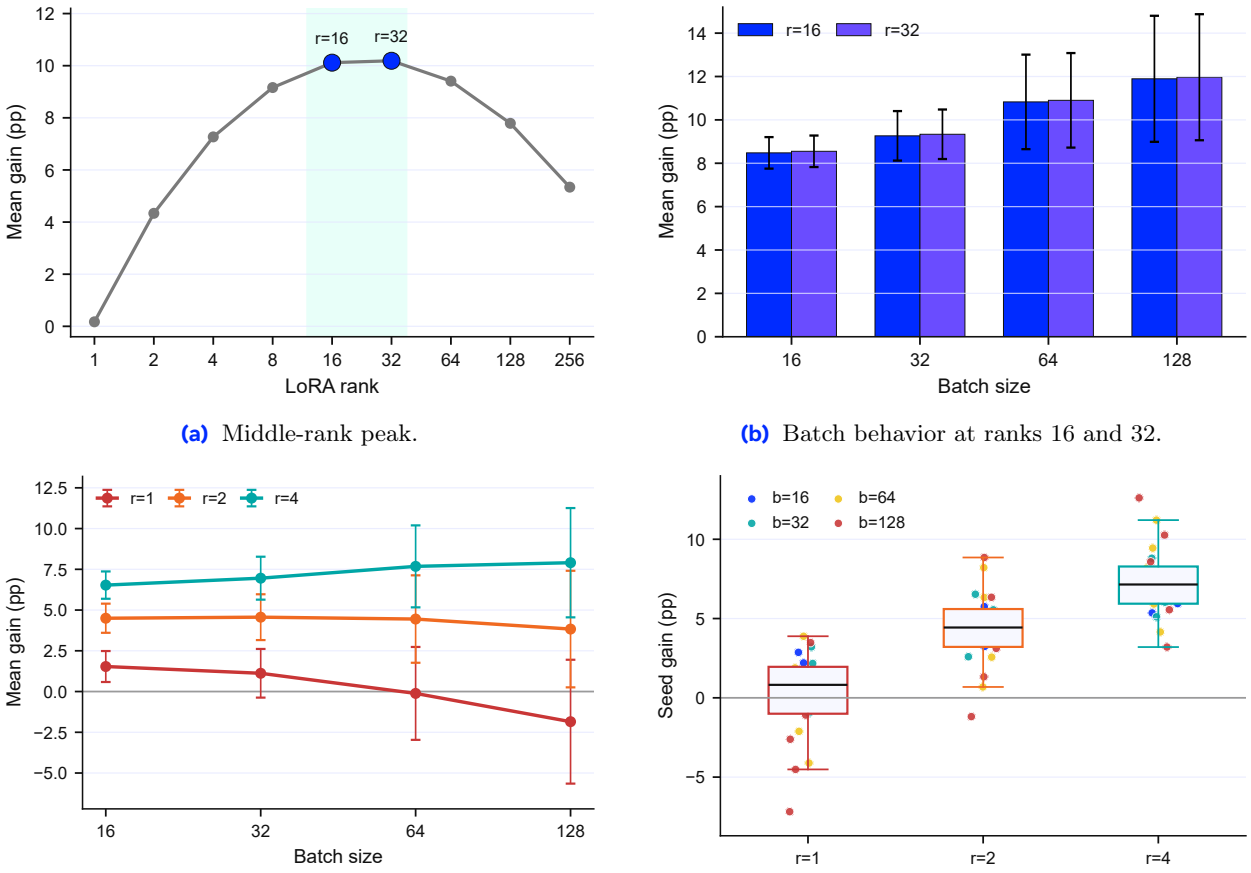
4.1 Inside LoRA: Finding the Efficient Operating Regime

LoRA provides the clearest laboratory for studying efficient adaptation because it exposes the central tradeoff directly: how much learning can survive as the trainable surface becomes extremely small. The design space is not only about low rank in the abstract, but about the joint control of expressive capacity, optimization stability, and operational simplicity. The three subsections below form a progression. Rank reduction asks how small the adaptive state can become before the across-seed mean collapses, even when the best run remains competitive. Initialization asks how such tiny adapters can be made reliable rather than merely occasionally lucky under RL optimization. Hyperparameter reuse asks whether the resulting recipe can be reused across the large populations of adapters that personal-model deployment requires.

4.1.1 Rank Reduction Under Minimal Parameters

A central question in PEFT is whether useful learning survives under extreme rank reduction. From this perspective, LoRA rank is not simply a monotonic capacity knob. It defines an adaptation regime: how small can the adaptive state become before the across-seed mean degrades, even though the best run remains competitive with much larger adapters?

A Qwen3-8B PPO sweep provides the main evidence for this question. The sweep contains 216 runs across



(a) Middle-rank peak.

(b) Batch behavior at ranks 16 and 32.

(c) Batch sensitivity at ranks 1–4.

(d) Seed-level spread at ranks 1–4.

Figure 9 Middle- and low-rank operating regions. **(a, b)** Ranks 16 and 32 provide the strongest practical balance of mean gain and downside risk under the observed PPO recipe. **(c, d)** Ranks 1–4 still carry positive reinforcement-learning signal, but rank 1 becomes sharply batch- and seed-sensitive.

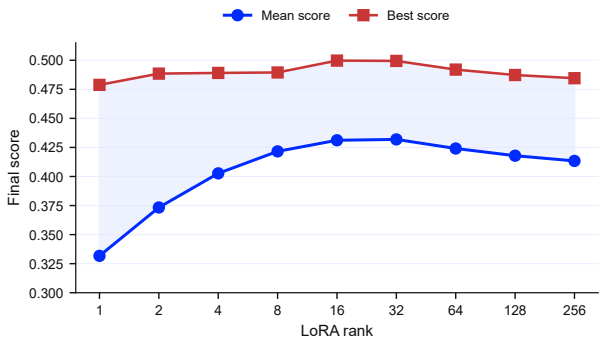
nine LoRA ranks, four batch sizes, and six seeds per configuration, using a fixed 500-step PPO schedule on a mixed mathematics corpus with verifiable rewards. As summarized in Figure 8, the resulting behavior separates into three distinct regions rather than a single monotonic scaling law.

Ranks 16 and 32 form the strongest practical region under the current recipe. They achieve the highest mean gains, maintain relatively low downside risk, and provide strong token efficiency. Figure 9 (a, b) isolates this middle-rank band across batch sizes, making it the current deployment default.

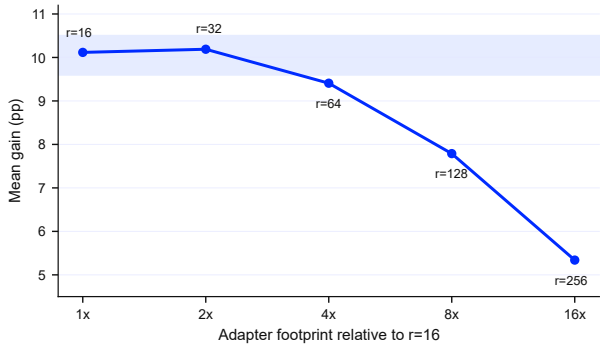
The more important implication, however, appears at the low-rank end of the curve. Ranks 1 to 4 do not behave like a uniformly failed region. Their best-seed runs are close to those at ranks 16 and 32, while their across-seed means dip and their seed-to-seed spread widens. As shown in Figure 9 (c, d), extremely small adapters can already carry the same level of reinforcement-learning signal that larger adapters reach, but current training recipes cannot access that signal consistently across seeds.

This distinction substantially changes the interpretation of low-rank PEFT. If the low-rank region were uniformly weak across seeds, the natural conclusion would be that the adapter is simply too small for the task. Instead, the sweep suggests that low rank is under-optimized rather than under-capacity. The best run at rank 1 already matches the best runs at ranks 16 and 32; what collapses at low rank is reliability across seeds, not the height of the attainable ceiling. The dominant failure mode is therefore not insufficient expressivity but insufficient stability, which is the problem the next subsubsection takes up.

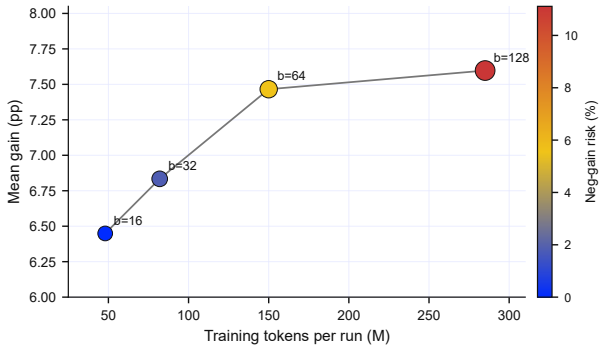
This interpretation rests on a careful reading of mean versus best-run performance. Mean score measures whether a configuration is dependable across seeds, while best score measures whether the regime can ever



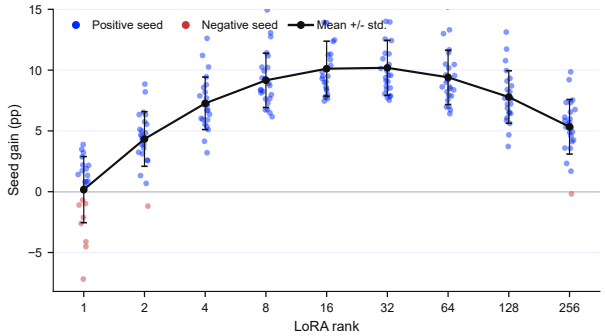
(a) Mean–best separation. The best-seed frontier is nearly flat across ranks, while the across-seed mean degrades at low rank, indicating that the limitation is reliability rather than capacity.



(b) High-rank opportunity cost. Beyond the middle-rank region, adapter footprint increases while the observed performance frontier flattens.



(c) Batch size as a budget variable. Larger batches slightly raise average gain, but they also increase token consumption under the fixed-step PPO schedule.



(d) Seed-level reliability. Cheaper adapter configurations make seed-heavy evaluation feasible, which is necessary for separating stable operating regimes from lucky runs.

Figure 10 Reliability and cost structure of the rank sweep. Mean–best separation and high-rank opportunity cost characterize where added rank stops paying off, while batch-cost and seed-reliability views show why cheaper adapters enable the seed-heavy evaluation that low-rank reliability requires.

reach a strong solution under the current recipe. Figure 10a shows that the best-seed frontier remains nearly flat across ranks 1 through 32, while the across-seed mean drops at low rank. Strong best runs with weak mean runs indicate an optimization and initialization bottleneck rather than a hard capacity limit, and motivate spending the next subsubsection on RL-native initialization rather than on simply raising rank.

The high-rank region exposes the opposite problem. Increasing rank from the middle region to 64, 128, or 256 inflates trainable parameters, optimizer state, checkpoint size, and memory pressure, but does not extend the observed best-run frontier in the sweep. Figure 10b makes this tradeoff explicit: beyond the middle-rank region, larger adapters add footprint without raising the ceiling. Under a fixed research or deployment budget, unnecessary rank displaces more useful expenditures such as additional seeds, broader ablations, tighter variance control, reward debugging, and rank-specific optimization.

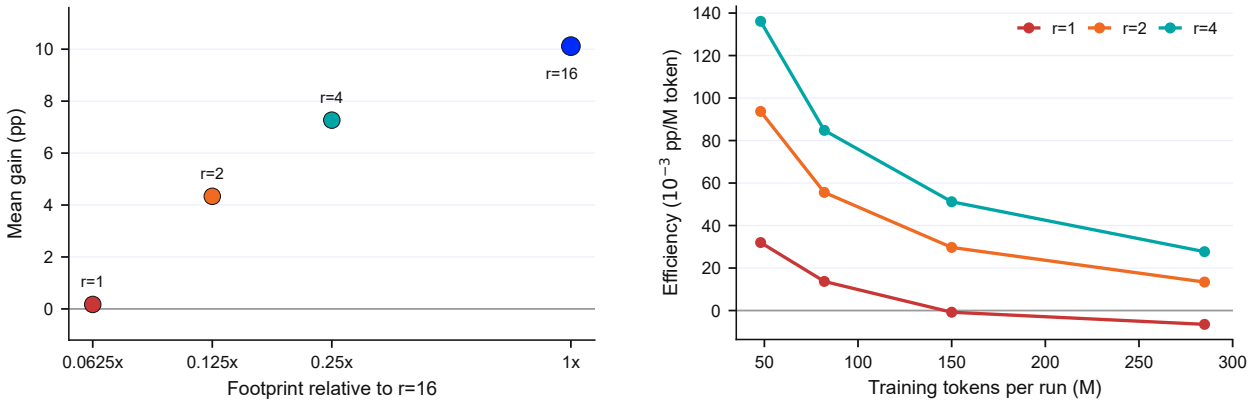
Batch size must be interpreted as part of the same budgeted adaptation regime. Because the sweep fixes PPO steps rather than total training tokens, larger batches directly increase token consumption. Figure 10c shows that larger batches slightly improve average gain, but at sharply higher token cost and with increased downside risk. Batch size is therefore not a pure optimization knob, but part of the broader score–cost tradeoff.

The resulting scale-down lesson is reflexive. Smaller adapters matter not only because they reduce the cost of one trained model, but because they reduce the cost of searching for better recipes. Tiny adapters whose best runs are already competitive but whose means are unreliable require exactly the kind of repeated ex-

perimentation that cheaper adapters enable: seed-heavy evaluation, stronger initialization, cleaner schedules, tighter KL control, and rank-specific hyperparameters. Figure 10d illustrates this feedback loop. Cheaper runs make broader evidence collection possible, and broader evidence collection is what turns lucky low-rank runs into reliable ones.

For a population of personal models, this distinction becomes system-level. A reduction in rank may appear modest for a single adapter, but it compounds across repeated updates, optimizer states, checkpoints, serving-time adapter loads, and the millions of persistent model instances that share a common prior. The objective is not to claim that rank 1 is already sufficient. It is to identify a path along which the per-instance adaptive boundary can become smaller without becoming brittle, so that the population as a whole stays close to the "share 99.5%, differ in 0.5%" regime that makes personal models economically feasible.

Figure 11 reframes this objective explicitly. The low-rank frontier should be evaluated not only by maximum single-run score, but also by whether useful learning signal can be made reliable under minimal trainable state and token budget. Under the current evidence, ranks 16 to 32 remain the practical default, ranks 1 to 4 define the research frontier where best-run gains are already there but mean reliability is not, and ranks above 64 warn against treating larger adapters as the default direction of progress. Rank reduction is therefore an operating-regime problem in which expressivity, optimization, variance, and budget must be solved together rather than independently. The most immediate lever is the initialization geometry of the few directions a tiny adapter does have, which the next subsubsection addresses directly.



(a) Minimal footprint versus gain.

(b) Low-rank token efficiency.

Figure 11 Tiny-adapter objective. The low-rank research target is not maximum single-run score alone, but stable learning signal under minimal trainable state and token budget.

4.1.2 RL-Native Initialization for Stability

The rank-sweep results (Figure 8c (b)) above show that extremely small adapters are not uniformly useless, but they are fragile. This fragility is most visible at rank $r = 1$. A rank-one LoRA adapter has only one adaptive direction per weight matrix. If this direction is poorly chosen, the adapter cannot redistribute learning across alternative components. Standard LoRA initializes this direction randomly, which is often sufficient at moderate ranks, but becomes unreliable when the adaptive subspace collapses to a single dimension. The natural question is therefore not only how small the rank can be, but how the few available directions should be initialized.

A natural first attempt is to use the geometry of the pretrained weight matrix. If the adapter has only one or a few directions, those directions should not be wasted on arbitrary random axes. The pretrained matrix already contains a spectral structure, and its singular vectors provide candidate directions that may be more meaningful than random Gaussian rows. From this perspective, geometry-aware initialization is an attractive way to rescue the rank-one regime: Instead of increasing the rank, we try to make the available direction more useful.

However, a recent study (Yin et al., 2025) shows that existing SVD-based LoRA initializations cannot be

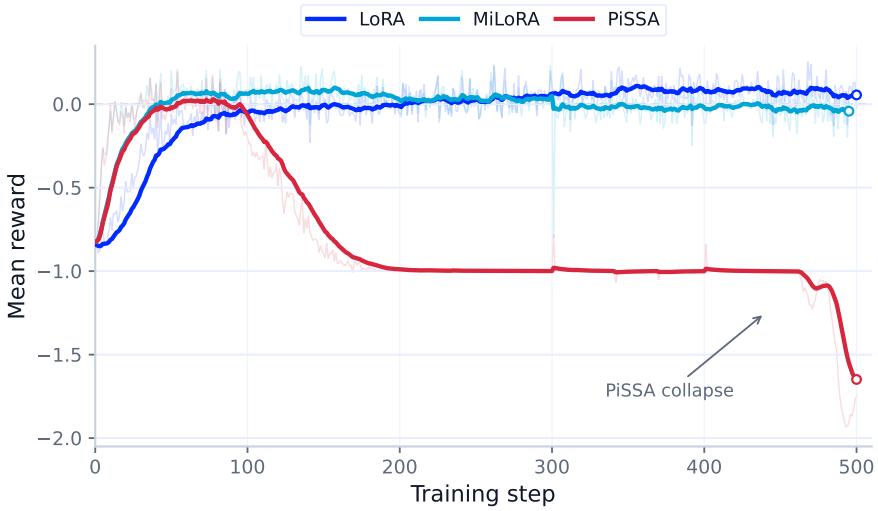


Figure 12 Training reward curves for DAPO 1.5B. PiSSA, MiLoRA, and LoRA are compared over 500 training steps using the same reward metric.

transferred from supervised fine-tuning to reinforcement learning naively. PiSSA (Meng et al., 2025) initializes adapters using the principal singular directions, while MiLoRA(Wang et al., 2025) uses the minor singular directions. Both methods use pretrained spectral information and can be effective in supervised fine-tuning, where dense token-level supervision rewards fast convergence. In RL with verifiable rewards, however, these same methods may underperform standard LoRA and exhibit training collapse. We replicate this phenomenon as shown in Figure 12.

This motivates an RL-native initialization criterion. A useful initialization should expose a meaningful low-dimensional learning direction, but it should not make the first policy updates too large. In supervised fine-tuning, initialization mainly affects convergence speed. The loss is dense, token-level, and anchored to fixed target sequences, so structured initializations can be useful because the optimizer receives many reliable gradient steps. In RL with verifiable rewards, the optimization geometry is more restrictive. The model learns from sampled responses, and practical objectives rely on token-level surrogates around the rollout policy. These surrogates remain meaningful only when the updated policy stays close to the policy that generated the samples. KL penalties, clipping, and trust-region mechanisms therefore do more than regularize: they define the local region in which RL updates are trustworthy.

The reason is easiest to see from the sequence-level objective. The RLVR objective can be written as $J(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}[R(x,y)]$, where $R(x,y) \in \{0,1\}$ is assigned to a complete response. Since $\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t})$, the REINFORCE gradient (Williams, 1992) distributes the sequence reward across token positions:

$$\nabla_\theta J(\theta) = \mathbb{E}_{y\sim\pi_\theta}\left[R(x,y)\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})\right].$$

In deployed RL systems, responses are often sampled from an inference-side rollout policy μ, while updates are computed for π_θ. This gives the importance-sampled form

$$\nabla_\theta J(\theta) = \mathbb{E}_{y\sim\mu}\left[R(x,y)\frac{\pi_\theta(y \mid x)}{\mu(y \mid x)}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})\right].$$

The sequence-level importance weight decomposes as

$$\frac{\pi_\theta(y \mid x)}{\mu(y \mid x)} = \prod_{t=1}^{T}\frac{\pi_\theta(y_t \mid x, y_{<t})}{\mu(y_t \mid x, y_{<t})} = \prod_{t=1}^{T} r_t = \prod_{t=1}^{T}(1+\delta_t),$$

where $\delta_t = r_t - 1$. This product is exponentially unstable: if $T = 512$ and each $r_t = 1.01$, the sequence weight is approximately $1.01^{512} \approx 163$. Practical token-level surrogates therefore depend on the first-order Taylor expansion

$$\prod_{t=1}^{T}(1+\delta_t) = 1 + \sum_{t=1}^{T}\delta_t + O(\delta^2),$$

which is accurate only when the rollout and updated policies remain close. Dropping higher-order terms gives the approximate token-level objective (Zheng et al., 2025)

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{y\sim\mu}\left[R(x,y)\sum_{t=1}^{T} w_t \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})\right], \qquad w_t = \frac{\pi_\theta(y_t \mid x, y_{<t})}{\mu(y_t \mid x, y_{<t})}.$$

This approximation explains why RL fine-tuning is sensitive to early policy movement. To second order, the KL divergence between the updated and current policies satisfies (Zhu et al., 2025)

$$D_{\mathrm{KL}}\big(\pi_{\theta+\Delta\theta} \,\|\, \pi_\theta\big) \approx \frac{1}{2}\Delta\theta^\top F \Delta\theta, \qquad F = \mathbb{E}\big[\nabla_\theta \log \pi_\theta \nabla_\theta \log \pi_\theta^\top\big].$$

If one update satisfies $D_{\mathrm{KL}}(\pi_{\theta^+} \,\|\, \pi_\theta) \le K$ and $F(\theta) \succeq mI$ on the update subspace, then a block update obeys

$$\|\Delta W\|_F \le \sqrt{\frac{2K}{m}}\big(1 + o(1)\big).$$

This creates a KL leash introduced by (Zhu et al., 2025): a policy-space trust region induces a weight-space movement budget. RLVR must therefore make progress through small, controlled steps rather than large jumps. Initialization does not merely choose where optimization starts, but biases the direction of the entire trajectory inside this narrow local region. A bad initialization can push the optimizer toward directions where small parameter movements create large policy drift, causing the first-order approximation above to break down.

This is why SVD-scaled initializations are risky in the tiny-adapter regime. At rank $r = 1$, a geometry-aware direction is valuable, but singular-value amplification can consume the KL budget before useful learning stabilizes. The design target is therefore not geometry alone, but controlled geometry: keep a meaningful pretrained direction while removing the scaling that makes the early update overly aggressive (Zhang et al., 2026).

We instantiate this idea with **OLoRA-tail**. Let $W_0 = U\Sigma V^\top$ be the singular value decomposition of a pretrained weight matrix, and let U_{-r} and V_{-r} denote the left and right singular vectors associated with the smallest r singular values. OLoRA-tail initializes

$$B_0 = U_{-r}, \qquad A_0 = V_{-r}^\top.$$

OLoRA-tail differs from MiLoRA in the detail that matters most for RL stability. MiLoRA uses

$$B_0 = U_{-r}\Sigma_{-r}^{1/2}, \qquad A_0 = \Sigma_{-r}^{1/2} V_{-r}^\top,$$

which injects singular-value scaling into both LoRA factors. OLoRA-tail keeps the same gentle tail subspace but removes this scaling. Thus, it preserves pretrained geometry while avoiding the extra singular value scaling that can destabilize RL.

Although both OLoRA (Büyükakyüz, 2024) and OLoRA-tail apply orthogonal initialization to the adapter matrices, they differ in which singular subspace is used. OLoRA initializes adapters from the principal singular vectors of the pre-trained weights, whereas OLoRA-tail initializes from the minor singular vectors. As shown in Figure 13, OLoRA undergoes training collapse under the DAPO objective: its reward deteriorates sharply after step 100 and its KL divergence from the reference model grows by several orders of magnitude, eventually saturating near 8. We attribute this instability to the fact that the principal subspace captures the dominant directions of the pre-trained representation, and perturbing these directions through on-policy

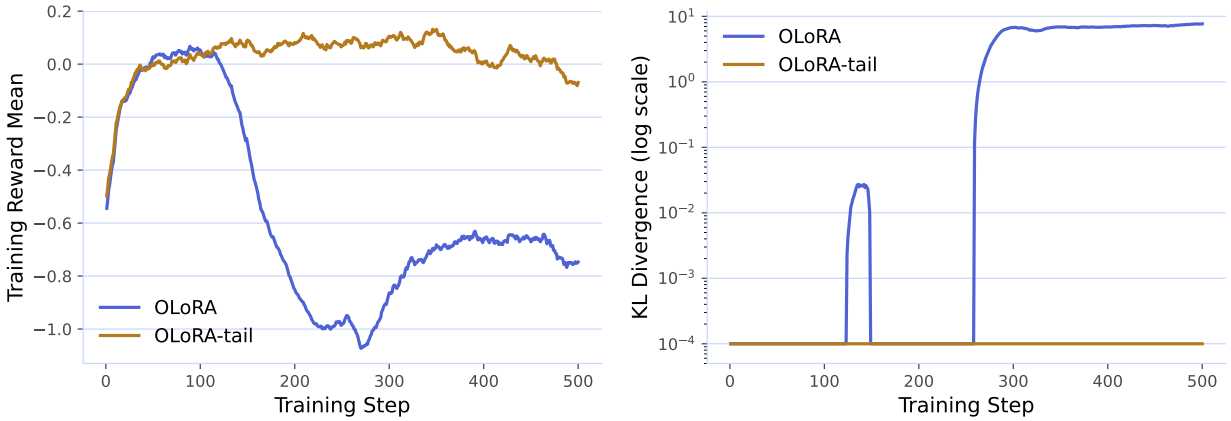


Figure 13 Training reward (left) and KL divergence from the reference model (right, log scale) for OLoRA and OLoRA-tail on DeepSeek-R1-Distill-Qwen-1.5B trained with DAPO. OLoRA collapses around step 100, with reward dropping to −1.0 and KL divergence exploding to ∼8, while OLoRA-tail remains stable throughout 500 steps.

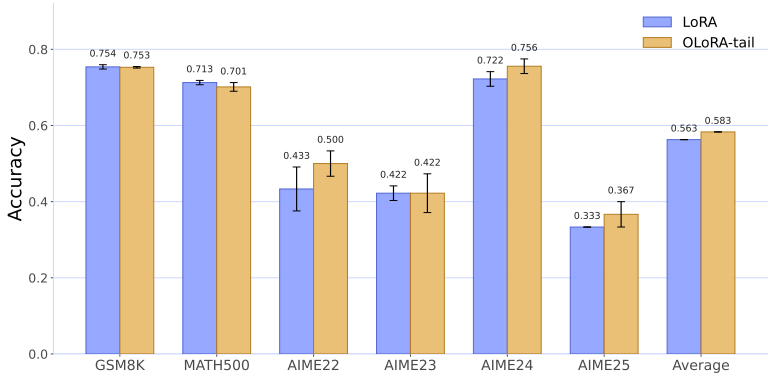


Figure 14 Comparison of LoRA and OLoRA-tail on mathematical reasoning benchmarks. OLoRA-tail consistently matches or outperforms LoRA, achieving a higher average accuracy (58.3% vs. 56.3%).

RL updates induces large distributional shifts that the DAPO clipping mechanism cannot fully contain. By contrast, the minor subspace spans directions that are comparatively inert in the pre-trained model, providing a safer optimization landscape: the adapter updates remain orthogonal to the most sensitive weight directions, keeping KL divergence near zero and training stable throughout. Based on this observation, we adopt OLoRA-tail for all subsequent scale-down experiments.

We train DeepSeek-R1-Distill-Qwen-1.5B with DAPO on the DAPO-Math-17k dataset for 500 steps, using a constant learning rate of 1×10^{-5} and an effective batch size of 32. Both LoRA and OLoRA-tail are applied to all attention and MLP projection layers with rank r=16 and scaling factor α=32. We evaluate LoRA and OLoRA-tail on six mathematical reasoning benchmarks using DeepSeek-R1-Distill-Qwen-1.5B trained with DAPO. As shown in Figure 14, OLoRA-tail achieves a higher average accuracy of 58.3% compared to LoRA's 56.3%, demonstrating that a better geometric initialization can provide an advantage.

Having established that OLoRA-tail's minor-subspace initialization yields consistent gains over standard LoRA at rank r=16, a natural question is whether this geometric advantage persists as we push the adapter to its most extreme compression. Rank-one adaptation represents the smallest possible trainable footprint—a single outer-product update per weight matrix, yet it is notoriously sensitive to initialization: with only one direction available, a poor choice of singular vector can permanently misalign the adapter with the task signal. The stability benefits we observed at rank 16 suggest that anchoring the adapter in the minor subspace may be precisely what is needed to make rank-one adaptation viable. This gives a clean scale-down test: can a better one-dimensional geometry make rank-one adaptation usable without increasing the trainable parameter

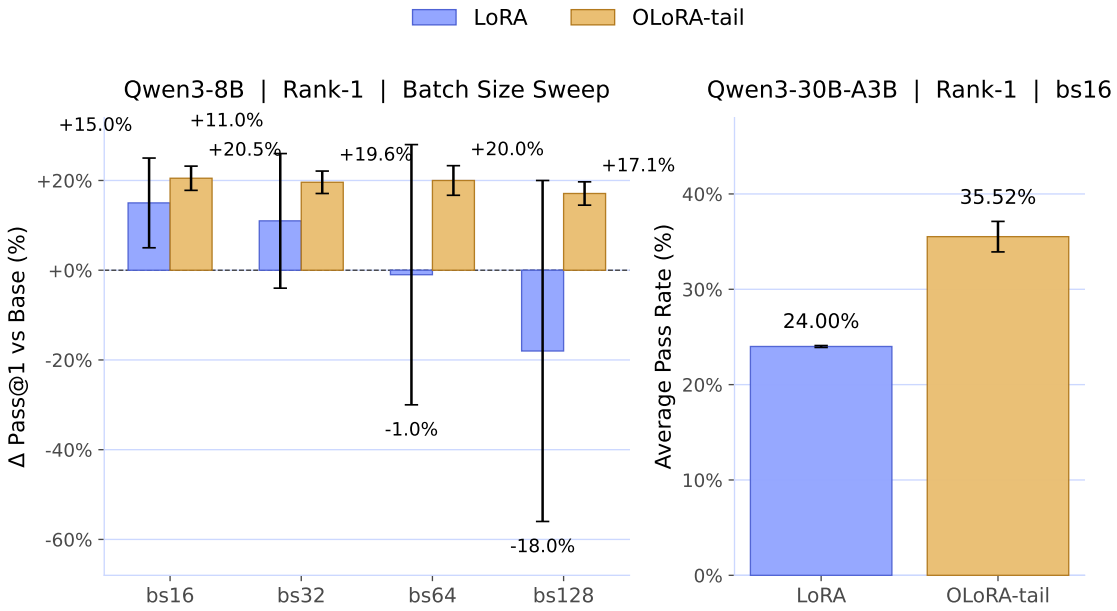


Figure 15 Rank-1 OLoRA-tail versus LoRA on Qwen3-8B and Qwen3-30B-A3B-Instruct trained with DAPO. **Left**: ΔPass@1 over the base model across batch sizes on Qwen3-8B; OLoRA-tail maintains a consistent gain of ∼+20% regardless of batch size, while LoRA degrades from +15% at bs16 to −18% at bs128. **Right**: Average pass rate on Qwen3-30B-A3B-Instruct; OLoRA-tail (35.5%) surpasses LoRA (24.0%) by 11.5 percentage points. Error bars show standard deviation across 6 random seeds.

budget?

We evaluate OLoRA-tail at the extreme compression of rank r=1 across two model scales: Qwen3-8B and Qwen3-30B-A3B-Instruct, under the same experimental settings in Section Section 4.1.1

As shown in Figure 15, the results demonstrate that this distinction is substantial. On Qwen3-8B, OLoRA-tail delivers a consistent ∼+20% gain over the base model across all batch sizes, while standard LoRA degrades sharply with increasing batch size: from +15% at bs16 to −18% at bs128, with collapse risk reaching 67%. The geometric advantage of minor-subspace initialization does not merely narrow the performance gap, but it eliminates the sensitivity to batch size entirely. On Qwen3-30B-A3B-Instruct, OLoRA-tail achieves an average pass rate of 35.5%±1.6%, surpassing the LoRA baseline of 24.0% by 11.5 percentage points (+48% relative).

These results refine the interpretation of rank. Rank determines the number of available directions, but initialization determines whether those directions are usable. In the tiny-adapter regime, adding rank is one way to increase the probability of finding a useful direction. Choosing the direction geometrically is another, and our results show it can be strictly more parameter-efficient. OLoRA-tail improves the adapter's usable capacity without increasing trainable weights, optimizer state, checkpoint size, or serving-time footprint, which is precisely the scale-down property that matters for sustaining large populations of personal models. The practical implication is not that every task should use r=1, but that the lower boundary of viable adaptation can be pushed further when initialization is designed for the KL-constrained, on-policy operating regime of RL fine-tuning.

4.1.3 Reusable Hyperparameters for Lower Training Effort

Hyperparameter transfer becomes a scaling bottleneck once adapters are trained at population scale. LoRA exposes at least three tightly coupled knobs: rank r, scale α, and learning rate η. In practice, users often change rank because of memory, speed, or capacity constraints and then retune the learning rate independently. The more useful question, however, is not *what is the best LoRA learning rate?* but *how should the learning-rate search move when rank changes?* The answer depends directly on how α scales with rank.

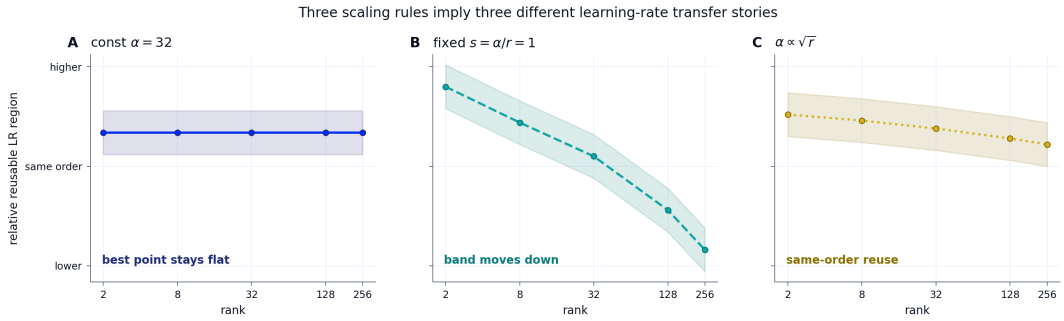
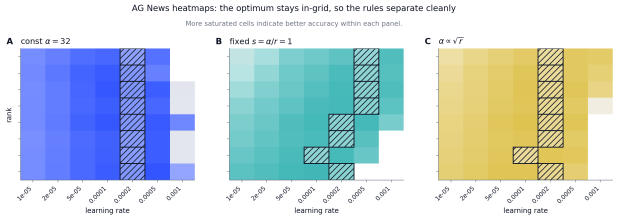
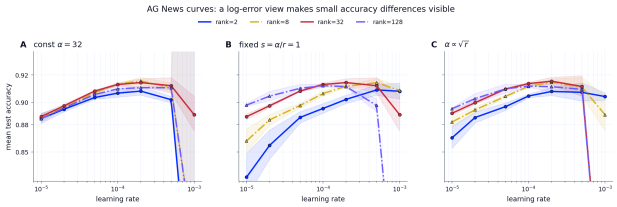


Figure 16 The practical summary is that learning-rate transfer depends on the chosen alpha-scaling rule.



(a) Rank-by-learning-rate heatmaps show how the good region moves under different alpha rules.



(b) Curves separate best-point reuse from the quality of transferred settings.

Figure 17 AG News learning-rate transfer across ranks. Heatmaps locate the good learning-rate region, while curves distinguish best-point reuse from transfer quality under each alpha rule.

LoRA commonly parameterizes the update as

$$\Delta W = \frac{\alpha_r}{r} BA. \tag{1}$$

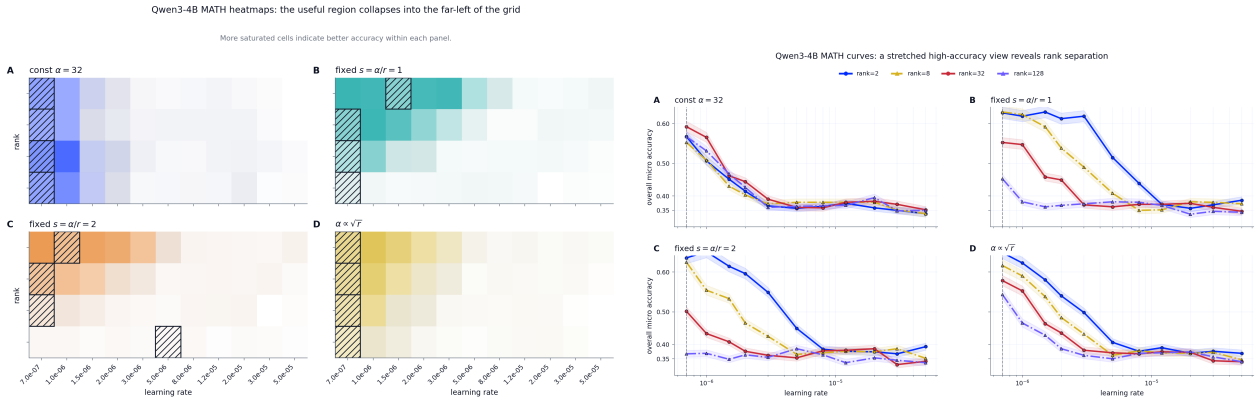
Under standard initialization, A is random and $B = 0$, so the first effective movement comes from updating B. With learning rate η, the first update to B scales as $\eta\alpha_r/r$, and substituting that back into ΔW yields an early-step perturbation proportional to

$$\eta \frac{\alpha_r^2}{r}. \tag{2}$$

Although this expression does not fully characterize AdamW dynamics, it captures the leading dependence of early update magnitude on rank and scaling convention. If α/r is fixed, then $\alpha_r \propto r$ and the effective update grows with rank, so higher rank should push the search toward smaller learning rates. If α is fixed, the effective update shrinks with rank, so higher rank does not force a smaller learning rate. If $\alpha \propto \sqrt{r}$, rank dependence cancels, yielding the simplest theoretical path to same-order learning-rate reuse.

Empirical sweeps show the same split. On AG News (Zhang et al., 2015) with DistilBERT (Sanh et al., 2020), ranks 2 through 256 were tested under constant $\alpha = 32$, fixed $\alpha/r = 1$, and $\alpha \propto \sqrt{r}$ with $\alpha/\sqrt{r} = 8$. Fixed α/r shifted the good learning-rate region downward as rank increased. Constant α produced a flatter best-learning-rate location and was easiest to tune in that simple cross-rank setting. The square-root rule preserved same-order reuse in line with the early-training proxy, though constant α was slightly flatter in that particular experiment.

A harder Qwen3-4B MATH transfer setting (Yang et al., 2025; Hendrycks et al., 2021) makes the transfer problem more consequential. The useful region collapses into a very small learning-rate range, and the scaling rules diverge at high rank. Constant α keeps the best point flat, but it is not the strongest rule overall. Fixed α/r can perform well at low rank, especially with aggressive ratios, but deteriorates quickly as rank grows. The square-root rule is the most balanced: it preserves best-point reuse and delivers stronger high-rank behavior. This distinction matters because best-point reuse and transfer quality are not identical. A rule can keep the same nominal best learning rate while producing weaker performance when reused across ranks.



(a) Heatmaps show that harder reasoning settings make the reusable band narrower and more consequential.

(b) Curves motivate square-root alpha scaling as a practical path to same-order learning-rate reuse.

Figure 18 Qwen3-4B MATH learning-rate transfer across ranks. The reusable band is narrow, and the square-root alpha rule preserves same-order reuse while remaining stronger at high rank.

For population-scale PEFT, the practical object is not one magic learning rate but a reusable band. Fixed α/r pushes the band downward with rank, constant α often keeps it flatter, and $\alpha \propto \sqrt{r}$ preserves same-order reuse while remaining stronger in the harder reasoning setting. A world with millions of adapters is impossible if every new adapter requires a full hyperparameter sweep. Reusable bands and rank-stable parameterizations are therefore not conveniences. They are prerequisites for adapter populations.

The square-root scaling rule was proposed by rsLoRA (Kalajdzievski, 2023) as a theoretical fix for the rank-dependent update magnitude in standard LoRA. Triquetra's contribution is not to introduce this rule, but to validate it empirically in the harder RL fine-tuning setting and to show that the choice of alpha rule has practical consequences for transfer quality, not just for theoretical update magnitude. On simple classification tasks, constant α can be equally flat; on harder reasoning tasks such as Qwen3-4B MATH, the square-root rule is more robust when the reusable band is narrow and transfer quality matters.

Hyperparameter transfer is a hidden cost of PEFT because many recipes are under-specified. One implementation may keep alpha fixed, another may keep alpha divided by rank fixed, and a third may scale alpha with the square root of rank. Two runs can therefore share the same apparent learning rate and rank while having very different early update magnitudes. Without recording the scaling convention, the recipe is not portable across papers, codebases, model families, or serving platforms.

For personal models, this under-specification becomes a platform problem. A service that trains thousands or millions of adapters cannot ask each user or product team to rediscover the correct learning-rate band. It needs configuration laws that predict how a safe band moves when rank changes for cost reasons, when model size changes for prior reasons, or when task difficulty changes. The Triquetra framing supplies the beginning of such a law, complementing broader hyperparameter-transfer work (Yang et al., 2022): track the joint effect of rank, alpha, and learning rate on the first effective weight movement, then validate the resulting transfer rule empirically.

The square-root rule is especially interesting because it makes theoretical and operational goals coincide. The early-step proxy becomes rank-invariant, so the same-order learning-rate band can be reused. The empirical results suggest that this is not always the flattest rule on simple tasks, but it is robust in the harder reasoning setting where transfer quality matters. This is precisely the regime personal-model infrastructure must handle: not one easy benchmark, but many heterogeneous learning problems under limited tuning budgets.

Triquetra also changes how adapter configuration should be documented. A rank number without alpha and learning rate is incomplete, an alpha rule without a transfer policy is incomplete, and a learning-rate recommendation without rank context is not portable. A population of adapters needs configuration laws, not just configuration values. This is why Scale Down includes training-effort efficiency, not only parameter efficiency.

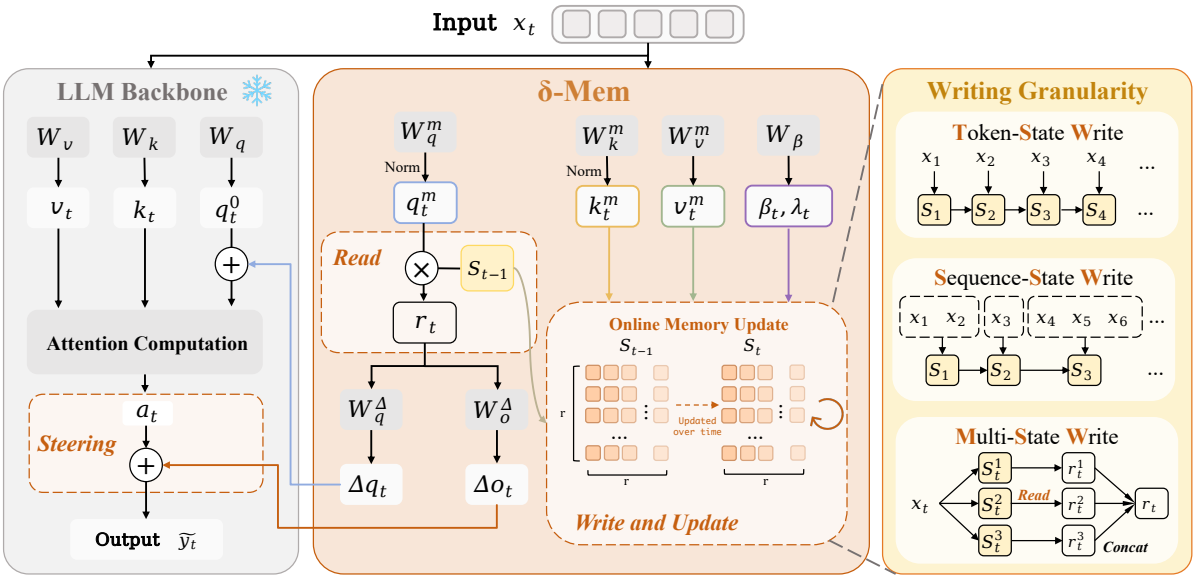


Figure 19 Overview of δ-mem. A compact online memory state is read to produce history-dependent low-rank corrections to the frozen attention computation, and is then updated with current key–value information through delta-rule learning.

4.2 Beyond LoRA: Toward a Spectrum of Adapters

The preceding sections study Scale Down within the standard LoRA setting: the adaptive state is a static low-rank parameter update, and the main question is how small this update can become while remaining expressive, stable, and reusable. This view is essential, but it does not exhaust the design space of efficient adaptation. Complementary lines of work enlarge the static-LoRA setting in other directions, including hierarchical or layer-wise rank allocation (Zhou et al., 2025a, 2026b), intra- and inter-layer parameter sharing (Zhou et al., 2025e), joint optimization of LoRA rank with mixed-precision quantization (Zhou et al., 2025c, 2026a), and combined low-rank plus sparsity compression (Zhou et al., 2025b). The remainder of this section pursues a different question: if PEFT is to support persistent personal models, the adaptive unit must not only be small after training; it must also be writable, reusable, and capable of changing with interaction history. Scale Down therefore extends from parameter reduction to state design: how compact, stable, and dynamic can the local adaptive state become?

From static adapters to writable local state. Ordinary LoRA stores adaptation directly in low-rank weights. Once trained, the same parameter update is applied regardless of the model's previous interactions. This is suitable for task or domain specialization, but it is less suitable for personal models that must accumulate preferences, corrections, task progress, and long-term behavioral patterns over time. A broader adapter view treats PEFT as an interface between a frozen shared backbone and a local adaptive state. The important design question is then not only how many trainable parameters the adapter introduces, but also what kind of state it maintains, how that state is updated, and how it influences inference.

A representative stateful adapter: δ-mem. A concrete example of this direction is δ-mem (Lei et al., 2026), which augments a frozen full-attention Transformer with a compact online associative-memory state. Unlike ordinary LoRA, whose low-rank update is fixed after training, δ-mem maintains a state that evolves as tokens are processed. At each position, the model reads from the previous memory state, uses the readout to generate low-rank corrections to the backbone attention computation, and then writes the current information back into memory through a delta-rule update, as illustrated in Figure 19.

Concretely, δ-mem maintains a low-dimensional state

$$\mathbf{S}_t \in \mathbb{R}^{r \times r},$$

Table 3 Main benchmark results comparing different memory mechanisms on Qwen3-4B-Instruct. All values report the task-specific metrics. For the final average score, HotpotQA is counted using Exact Match (EM).

Model	IFEval	HotpotQA		GPQA-D	Memory Agent Bench					LoCoMo					Avg.
		EM	F1		Avg.	AR	TTL	LRU	SF	Avg.	Multi	Temp	Open	Single	
Qwen3-4B-Instruct	81.89	42.35	56.00	39.39	29.54	35.30	26.14	**47.08**	14.37	40.79	38.39	32.89	10.77	48.05	46.79
Textual Memory															
+ BM25 RAG	–	40.35	52.83	–	24.49	32.20	9.74	37.86	15.00	36.68	38.12	20.34	9.99	45.47	44.56
+ LLMLingua-2	–	36.93	50.03	–	15.63	21.45	1.43	38.45	8.62	40.98	39.07	30.13	10.98	49.19	42.96
+ MemoryBank	–	–	–	–	17.65	22.65	7.67	36.36	9.88	38.14	37.88	21.76	13.35	47.31	43.88
Parametric Memory															
+ Context2LoRA	76.71	37.85	50.88	29.29	32.53	40.00	29.86	25.15	**17.75**	48.11	37.95	34.99	16.75	**60.11**	44.90
+ MemGen	39.37	5.36	16.27	38.89	29.61	34.85	28.45	44.30	14.38	40.05	32.93	33.30	12.67	48.13	30.66
Outside-channel Memory															
+ MLP Memory	24.95	10.94	25.83	22.73	28.80	35.35	26.00	31.19	14.38	26.85	32.87	16.72	8.81	30.75	22.85
δ-Mem															
+ δ-Mem (SSW)	81.70	49.22	63.43	**41.41**	37.84	41.50	**50.50**	43.02	16.50	47.05	41.00	36.48	14.08	56.88	51.44
+ δ-Mem (TSW)	**82.99**	**49.41**	**63.66**	40.40	36.48	42.45	40.64	46.08	15.88	46.53	42.14	37.20	13.35	55.36	**51.66**
+ δ-Mem (MSW)	81.52	46.86	60.47	37.37	**38.85**	**44.40**	47.29	41.55	17.00	**49.12**	**42.57**	**39.31**	**18.12**	58.59	50.74

which stores key–value associations in the adapter space. Given a memory key $\mathbf{k}_t^m$ and value $\mathbf{v}_t^m$, the update can be written as

$$\mathbf{S}_t = \mathrm{Diag}(\boldsymbol{\lambda}_t)\mathbf{S}_{t-1} + \mathrm{Diag}(\boldsymbol{\beta}_t)\left(\mathbf{v}_t^m - \mathbf{S}_{t-1}\mathbf{k}_t^m\right)\left(\mathbf{k}_t^m\right)^\top.$$

Here $\boldsymbol{\lambda}_t$ controls retention and $\boldsymbol{\beta}_t$ controls write strength. The important point is that the state writes the prediction residual rather than simply accumulating all new information. Already predictable associations induce small updates, while novel or mispredicted associations modify the state. In this sense, δ-mem preserves the low-rank steering interface of LoRA, but replaces a static parameter patch with a history-conditioned correction.

Writing granularity as part of the adaptive state. The δ-mem design also shows that the adaptive unit is not defined only by parameter count. It studies different writing granularities: token-level writing preserves fine-grained local information, sequence-level writing reduces redundant token noise, and multi-state writing separates information across several parallel states to reduce interference. For Scale Down, the lesson is that two adapters with similar trainable budgets can behave very differently depending on when they write, what they write, and how their state is organized.

Evidence for compact dynamic memory. The empirical results in Table 3 support this adaptive-state view. On a Qwen3-4B-Instruct backbone, the best δ-mem variant improves the average score from 46.79% to 51.66%, while outperforming static or textual memory baselines such as Context2LoRA (Back et al., 2026; Hu et al., 2021). The gains are especially visible on memory-intensive benchmarks. On MemoryAgentBench (Hu et al., 2025b), δ-mem improves the average score from 29.54% to 38.85%. On LoCoMo (Maharana et al., 2024), the multi-state variant reaches the strongest average score, suggesting that multiple compact states can help reduce memory interference in long-context personal-memory settings. On HotpotQA (Yang et al., 2018), token-level writing improves EM/F1 from 42.35%/56.00% to 49.41%/63.66%. These results suggest that the online state provides useful historical signals that are not captured by static parameter patches or purely textual memory mechanisms.

Efficiency of the stateful interface. The efficiency profile is also consistent with the Scale Down argument. The token-state and sequence-state variants introduce only 4.87M trainable parameters, about 0.12% of the Qwen3-4B backbone, while the multi-state variant uses 19.47M parameters, about 0.48%. These overheads are substantially smaller than heavier auxiliary-memory baselines such as MemGen and MLP Memory. Because the recurrent state has fixed size, its storage cost does not grow linearly with interaction history length. Inference is not free: each decoding step must read from and update the online state. However, this places δ-mem at a useful operating point: a small recurrent computation cost is exchanged for persistent, history-dependent steering.

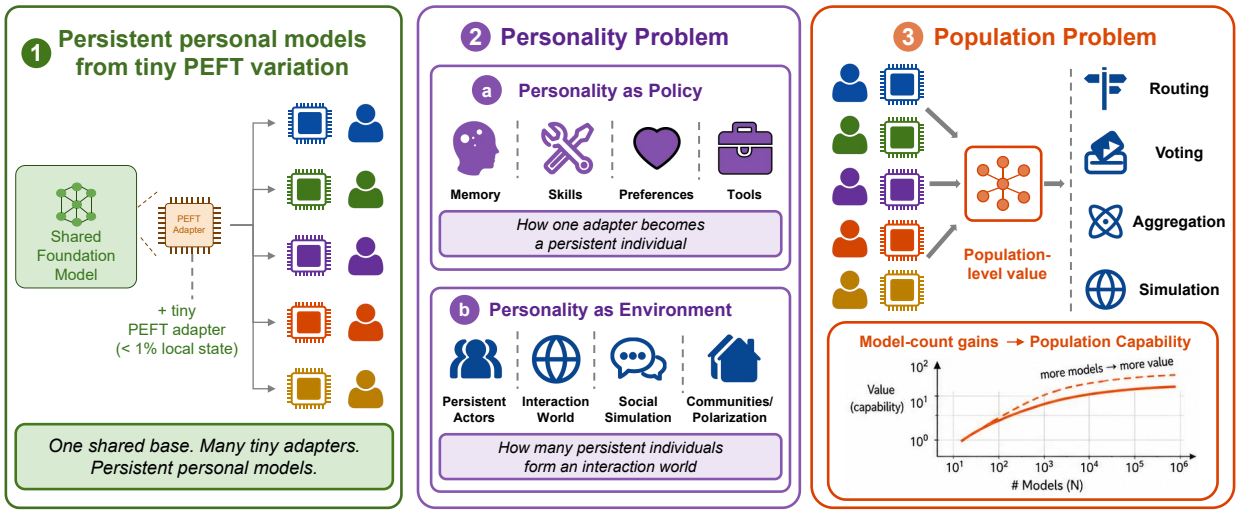


Figure 20 Scale Out maps small stable variation to population-level value. A shared base model supports persistent personal models through lightweight PEFT adapters. Scale Out requires both preserved individuality and population-level utility.

Implications for PEFT scaling. The broader implication is that PEFT scaling should not be equated with static rank reduction alone. In ordinary LoRA, Scale Down asks how low the rank can be while preserving task performance. In stateful adapters such as δ-mem, Scale Down also asks how small a writable local state can be while still carrying useful historical information. This distinction matters for personal models. The frozen backbone supplies the shared foundation prior, the adapter parameters define the read–write interface, and the online state stores instance-specific experience. Future PEFT mechanisms may therefore combine three ingredients: a strong shared prior, a compact low-rank steering interface, and a small dynamically writable state. Their role is not merely to reduce the cost of fine-tuning, but to make local adaptive state persistent, compact, and directly coupled to the model's forward computation.

5 Scale Out: From Individual Adaptation to Population-Scale Personalization

Scale Down identified the operating regime in which the local adaptive state becomes small, stable, and cheap enough to be trained, stored, updated, and served repeatedly. Scale Out begins when such adaptation becomes routine. The question is no longer only how to improve one adapted model, but what becomes possible when many persistent personal model instances can exist at the same time. PEFT makes this question concrete: a shared foundation model can support many isolated adapters, each carrying a different local state.

Adapter count alone, however, is not a scaling law. If many adapters collapse toward the same policy, model count is only redundancy. If adapters remain different but their differences cannot be simulated, composed, selected, or aggregated, diversity remains local personalization. Scale Out therefore asks three connected questions, each addressed by a subsection below: (1) how personal models preserve continuity for individuals, including what to memorize, how to write memory into adapter parameters, and how to govern learned capabilities (Section 5.1); (2) how persistent user-conditioned policies can support user simulators and agent environments (Section 5.2); and (3) how diversity among adapted models can become a source of collective performance (Section 5.3).

A note on Context Learning: it appears in Section 5.1 as the *write policy* for LoRA-as-memory, the mechanism that decides which context-time improvements should be stabilized into adapter parameters. It is not a standalone Scale Out algorithm. Its role is to make memory-signal efficiency tractable: without a principled write policy, LoRA memory capacity is a hard ceiling, but with one, repeated interaction can selectively fill that capacity with behaviorally useful state.

5.1 Personal Models for Individuals

A personal model is not just a universal assistant with a longer prompt. It needs persistent adaptive state, so that repeated experience can shape future behavior. A task adapter can be disposable: it solves a benchmark or domain and can be replaced when the task changes. A personal adapter instead carries part of the enduring state for one user, agent, role, or workflow: memories, preferences, skills, and tool habits. The adapter stores part of the learned behavioral state, not the entire user history.

Personal adaptation has two roles in Scale Out. As a user-conditioned policy, it explains how a shared base model becomes a persistent personal model for one user. As a simulated environment component (Section 5.2), it explains how many such persistent policies can form user simulators or agent environments. The final population subsection then asks a separate question: whether differences among adapted models can be aggregated into measurable system-level performance.

From adapter to personal policy. The first function of local adaptive state is policy specialization. A personal model should not merely answer in a preferred tone. It should help decide what to remember, which tools to prefer, how to ask questions, when to avoid action, how to recover from failures, and which behaviors are natural or unacceptable for a particular user. In this sense, personalization is not decoration. It is the policy layer that determines how a shared base model becomes useful for one person. Adapter isolation turns personalization from a global update problem into a local policy problem: the base model remains shared, while experience and behavior become user-specific. The rest of this subsection develops this view in four steps: memory is a precondition for continuity, memory requires a capacity law, bounded capacity requires a write policy, and once adapters shape tools and skills, personalization becomes an agentic capability.

Memory as the precondition of personality. Personality requires memory because a model cannot remain itself without preserving experience across interactions. Preferences, habits, past failures, recurring tasks, relationships, and learned workflows all depend on accumulated state. Prompt-based personalization is transient; retrieval-based memory is external and must be reinterpreted at every turn; parametric memory offers the stronger possibility that selected experience becomes part of the model's own policy. This motivates the LoRA-as-memory question: can a lightweight adapter serve as a bounded memory substrate for personal state? Since conversational and agent memory require more than recognition accuracy (Packer et al., 2024; Chhikara et al., 2025; Maharana et al., 2024), a personal adapter must support recall, addressing, reasoning, overwrite, conflict resolution, and behavioral application.

LoRA as memory: capacity scaling law. To support millions of personal models, we need scaling laws linking adapter size to capacity, stability, and retrieval accuracy. Key questions include how capacity correlates with LoRA rank, target modules, training tokens, and base-model scale, and when storage becomes detrimental interference.

Establishing such laws requires a benchmark that measures more than convenient context-to-QA conversion. A useful LoRA memory benchmark should test content, structure, difficulty, and downstream query distribution, including recall, addressing, relation, overwrite, and conflict resolution. We therefore introduce DishNameBenchmark as a controlled benchmark for isolating core LoRA-memory operations. It abstracts memory into interpretable slot-writing and slot-querying tasks, allowing us to vary memory length, update frequency, query type, and correction pattern while keeping the stored object simple and measurable.

We evaluate LoRA memory capacity on DishNameBenchmark using Qwen3-series models. We report capacity efficiency as the ratio between memory tokens and trainable parameters, and measure whether the model can correctly recover the stored slot values under position, adjacency, and correction-style queries. The results are shown in Figure 21:

- The first result is **a sharp capacity transition**. Across 263 runs, accuracy stays close to one when capacity efficiency is below roughly 10^{-3}, begins to degrade in the transition region between 10^{-3} and 10^{-2}, and collapses toward zero once the memory load exceeds about 10^{-2} tokens per trainable parameter. This suggests that LoRA memory has a measurable capacity ceiling: **the empirical upper limit in this setting lies between 10^{-3} and 10^{-2} memory tokens per trainable parameter.**

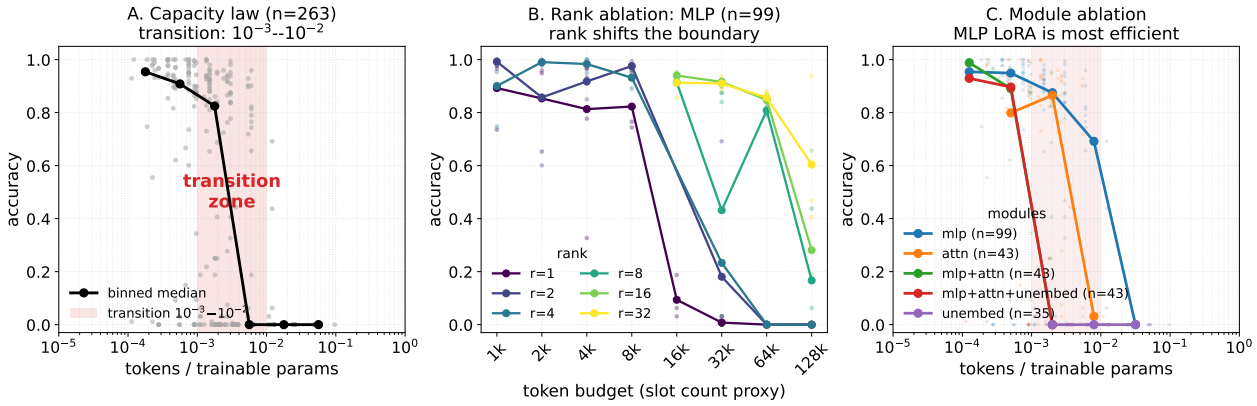


Figure 21 DishNameBenchmark reveals a bounded LoRA memory law: usable capacity lies around $10^{-3} - 10^{-2}$ tokens per trainable parameter, degrades predictably after saturation, and is most parameter-efficient when memory is written into MLP LoRA.

- For a fixed parameter budget, **performance drops approximately linearly after the adapter reaches its capacity limit.** In the rank ablation, increasing the token budget initially preserves near-perfect accuracy, but once the memory load crosses the adapter's effective capacity, accuracy falls rapidly as the number of required slots increases. Rank mainly shifts the capacity boundary by increasing the number of trainable parameters. At low memory budgets, even small-rank adapters can match larger-rank adapters. At high memory budgets, larger ranks survive longer because they provide more total storage surface. This supports the interpretation that rank buys capacity, but does not remove the underlying capacity law.
- The third result concerns where memory should be written. In the module ablation, training MLP LoRA provides the best parameter efficiency. At matched parameter budgets, MLP adapters maintain high accuracy at larger capacity-efficiency values than attention-only adapters, combined MLP+attention adapters, or unembedding adapters. Attention-only and full-module training can store memory, but they are less efficient per trainable parameter. Unembedding-only training performs worst and collapses quickly. The observed ordering is approximately

$$\text{MLP} > \text{Attention} \approx \text{All} \gg \text{Unembed}.$$

  From a scale-out perspective, this matters because personal adapters must be cheap to train, store, and serve. If the goal is to maximize memory capacity per parameter, **the most efficient design in this benchmark is to train MLP LoRA only**.

These results make LoRA memory a scarce resource rather than an unlimited store. Diversity therefore cannot be produced by writing everything into every adapter. If usable capacity is bounded and module choice strongly affects parameter efficiency, the next question is not only how much a personal adapter can memorize, but what kind of information deserves to become stable local variation.

What to memorize: behavioral state, not raw facts. The next question is what deserves LoRA memory. Because LoRA memory is harder to inspect and more expensive to edit, it should not function as a general fact store. A personal model instead needs a memory hierarchy (Table 4): editable facts should remain in retrieval, inspectable external reality should remain in tools, and LoRA should be reserved for persistent behavioral adaptation.

This hierarchy clarifies the role of LoRA memory. A rare document should stay in retrieval, a calendar event should remain tool state, and a recurring workflow should become skill memory. The memories that matter for Scale Out are not isolated facts, but behavior-shaping structures that make one adapted model reliably different from another. Skills are the natural target because they are reusable, procedural, and policy-shaping. Following the intuition of SKILL-0 (Lu et al., 2026), the scale-out implication is that a personal adapter should become a compact library of learned behavioral state: workflows, tool-use habits, reasoning templates, domain heuristics, safety routines, and action policies.

Table 4 A practical personal model should decide which memory layer should store each kind of state.

Memory layer	Example	Best suited for
Context	Current conversation	Short-term reasoning and local task state
Retrieval memory	Notes, documents, user facts	Editable factual recall and large evidence stores
Tool state	Calendars, files, databases	External reality that should remain inspectable
LoRA memory	Skills, habits, policy, persona	Persistent behavioral adaptation

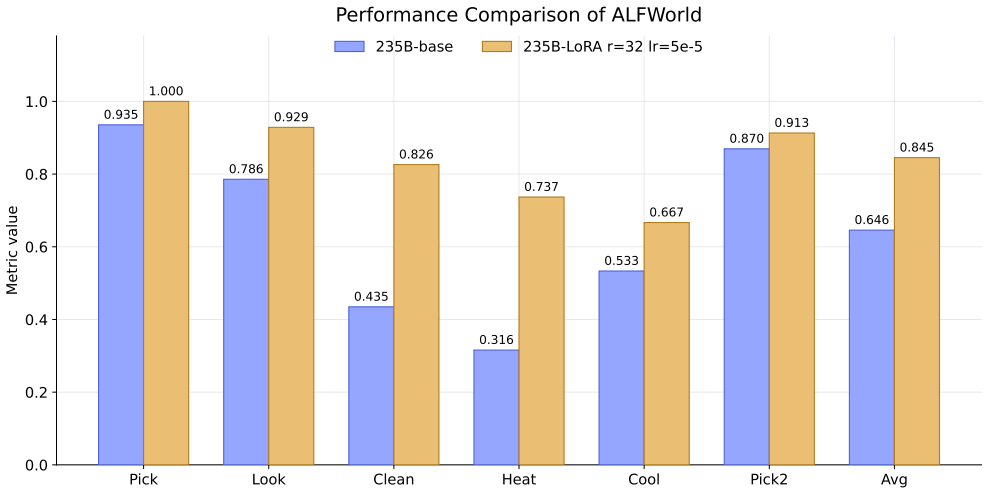


Figure 22 LoRA skill memory on ALFWorld. Starting from Qwen3-235B, a rank-32 LoRA adapter trained with the Skill-0/MinT recipe improves ALFWorld validation performance over the base model across task categories, raising the average score from 0.646 to 0.845.

We test this skill-memory interpretation by applying LoRA to a large-prior agent setting. Using Qwen3-235B as the shared base model, we train a rank-32 LoRA adapter with the Skill-0/MinT ALFWorld recipe and evaluate it on the ALFWorld benchmark. As shown in Figure 22, the adapted model improves over the base model on all reported task categories, with the average metric increasing from 0.646 to 0.845. This result is not evidence that LoRA should store arbitrary facts. Rather, it supports the narrower claim that LoRA can store reusable skill-like behavioral state: the adapter changes how the model acts in a procedural environment.

How to memorize: Context Learning as write policy. The memory hierarchy specifies where different states should live; Context Learning specifies how experience moves between layers. If Scale Down makes local variation cheap to store, Context Learning decides which temporary experiences should be stabilized into that variation. Context Engineering selects, retrieves, and arranges information to improve the current response. Context Learning asks which parts of that context-time improvement should become durable model state. In this sense, it is not merely better prompting, but the write policy of the personal model: it decides when repeated usefulness justifies converting context, retrieval, tool outcomes, or demonstrations into adapter parameters.

The mechanism begins with Context Distillation, summarized in Listing 1 and Figure 23. A query-only policy first produces an on-policy rollout. A stronger query-plus-context system then evaluates that rollout using retrieved evidence, demonstrations, tool outputs, execution outcomes, or slower verification. The resulting token-level or trajectory-level signal drives an RL-style update. Crucially, the update is applied to the query-only rollout, so the model learns to perform better without requiring the same context to be present at

Listing 1 Context Distillation as an on-policy context-to-parameters transfer. Context Learning as repeated Context Distillation.

```
def context_distill(model, query, build_context, rl_update):
    # Step 1: query-only produces an on-policy rollout.
    out = model.sample(query)

    # Step 2: query + context scores the rollout.
    ctx = build_context(query)
    r_tok = model.token_reward(query, ctx, out)

    # Step 3: update from the query-only rollout and rewards.
    return rl_update(model, query, out, r_tok)

def context_learning(model, queries, build_context, rl_update, steps):
    for _ in range(steps):
        query = next(queries)
        model = context_distill(model, query, build_context, rl_update)
    return model
```

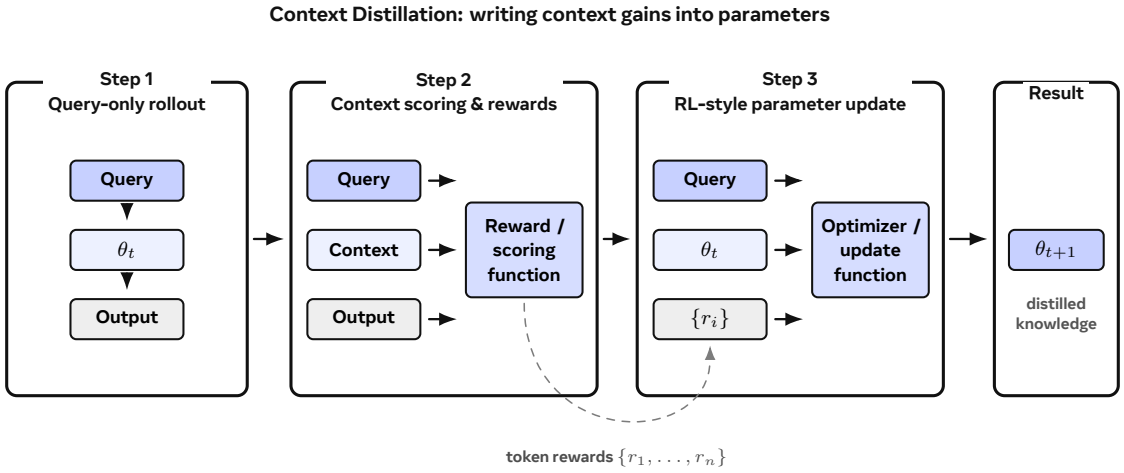


Figure 23 Context Distillation updates the query-only policy using a query-plus-context teacher signal.

inference time. This differs from off-policy context distillation, where targets are produced with context and then learned without context through supervised fine-tuning (Snell et al., 2022). Here, context is not used as a supervised target source; it is used as privileged information for scoring an output already produced by the query-only policy.

Repeating this operation turns Context Distillation into Context Learning. At step t, policy π_t answers a query without privileged context; the context system retrieves evidence, runs tools, observes outcomes, or performs slower verification; and the adapter is updated from this signal to produce π_{t+1}. Future query-only behavior now starts from a stronger internal state. Related self-distillation methods also use additional information to create stable parameter updates (Hübotter et al., 2026; Shenfeld et al., 2026). Here, however, the loop is explicitly personalized: each iteration decides what part of a user's context, tools, outcomes, or demonstrations should become adaptive state.

RAG2LoRA illustrates this memory-layer transfer. The point is not that all retrieved facts should become parameters, but that retrieval can provide a teacher signal when it repeatedly improves behavior. Over time, recurring facts, preferences, and procedures can be partially internalized into the adapter, reducing dependence on perfect retrieval while leaving editable evidence in external memory. This is the memory-signal efficiency required for Scale Out: each user can maintain a personal adapter whose value grows with interaction while the base model remains shared.

Table 5 EvoBot reports that learned social agents better match real group-opinion dynamics than prompt-only LLM baselines (Kong et al., 2025). Lower bias and diversity difference indicate closer alignment with real data.

Method	COVID-19				Russian-Ukrainian Conflict			
	Mean	Std	Δ_{mean}	Δ_{std}	Mean	Std	Δ_{mean}	Δ_{std}
Real	−0.017	0.472	/	/	−0.239	0.670	/	/
BC	−0.041	0.389	0.089	0.112	−0.304	0.104	0.104	0.554
Lorenz	+0.084	0.725	0.107	0.264	−0.811	0.105	0.572	0.565
Llama	−0.053	0.368	0.098	0.105	−0.324	0.405	0.202	0.265
GPT	+0.032	0.342	0.081	0.083	−0.256	0.435	0.135	0.238
EvoBot	+0.010	0.428	0.072	0.052	−0.237	0.480	0.101	0.194

Agentic personal models. The write-policy problem becomes most concrete in agentic systems, where the memory being written is often a workflow, tool habit, or post-failure correction. In this setting, the adapter is no longer only improving text generation; it is shaping how the model acts over files, messages, calendars, documents, code, tools, and long-running workflows. Personality becomes operational when memory determines continuity, skills determine competence, and policy determines safe action. This is where diversity begins to produce activity: different memories and skills lead agents to choose different tools, recover from different failures, and follow different workflows under the same external task. Related agent-learning work also treats skills and reflection as reusable learning objects (Shinn et al., 2023; Xia et al., 2026).

MindClaw (Li et al., 2026) illustrates this transition from prompt-space skill growth to parametric skill consolidation. Textual skill libraries are useful during cold start, but over time they create skill drift and retrieval dependence: the library grows, prompts get longer, and the agent may still fail to trigger the right capability. The scale-out goal is therefore not a larger skill library, but a higher probability that useful skills are learned, triggered, and applied consistently.

Thus, personality as policy is not merely memory. It is the construction of a persistent local policy: a personal model that remembers at the right level, internalizes reusable skills, and turns user-specific experience into stable action tendencies. This completes the individual side of Scale Out. The next question is what happens when many such policy-bearing adapters coexist and interact.

5.2 User Simulators and Agent Environments

From personal policy to user simulation. A personal model is not only an interface for one user. In an agent environment, each persistent user-conditioned policy can become part of the environment experienced by other agents. Once adapters carry distinct memory and policy states, the relevant object is no longer only the distribution of requests served by one model, but the distribution of histories, preferences, reactions, and interactions among persistent model instances. For an agent, the user is part of the environment, and a good user simulator therefore acts as a world model for user reactions, preferences, constraints, and long-term behavior.

The collapse problem of prompt-based user simulation. LLM social simulators often instantiate many agents by combining one shared model with different persona prompts. Systems such as OASIS make it possible to run large-scale social-media simulations with LLM agents and recommender dynamics (Yang et al., 2024). Generative Agents (Park et al., 2023) demonstrated that LLM-based agents can produce plausible human-like behavior through prompt-driven memory and reflection. However, persona prompting has a structural limitation: it changes the description of the agent, but not the learned policy that generates behavior. Agents may sound different at the surface level, but repeated interaction can underrepresent durable heterogeneity and drift toward the base model's average stance, style, and action prior. This is precisely the limitation that per-user LoRA adapters are designed to address: rather than describing a persona in a prompt, each adapter carries a distinct learned policy shaped by a different interaction history.

Table 5 provides related evidence for this concern: learned social agents better match real group-opinion dynamics than prompt-only LLM baselines, especially in opinion diversity (Kong et al., 2025). This does not

Table 6 Per-user LoRA produces monotonic structural scaling in OASIS as population size increases.

Metric	N=128	N=256	N=512	128 → 512
Identity persistence				
Final polarization distance	+0.388	+0.335	+0.319	–
Supportive stance std.	0.340	0.346	0.357	–
Skeptical stance std.	0.416	0.339	0.393	–
Population topology				
Effective interaction communities	9.21	11.77	14.85	+61%
Co-engagement modularity	0.502	0.561	0.716	+43%
Within-community side-homophily	0.670	0.644	0.583	−13%
Attention cascade				
Cascade Gini on likes	0.913	0.866	0.884	–
Top-10% post like share	0.859	0.731	0.781	–

by itself prove collapse in every prompt-based simulator, but it supports the need for learned, user-conditioned policies when simulation depends on stable heterogeneity. This collapse is especially problematic for social phenomena that require durable disagreement and persistent behavioral difference. Echo chambers, minority capitulation, preference cascades, group polarization, community norms, and coordination failures all depend on agents that carry stable histories and react differently to the same exposure. If every agent shares one mutable policy, then the simulation is not a population of persistent individuals. It is one model role-playing many social actors. This motivates the PEFT formulation of social simulation: a shared base model should provide the common prior, while per-user LoRA adapters provide the persistent, isolated policy states that make a population of agents behave like many individuals rather than one model role-playing many roles. In this view, realistic activity requires stable diversity: agents must not only receive different prompts, but carry different histories and action priors.

LoRA versus shared-base agents: structural scaling in OASIS. We test whether isolated personal adapters change the structure of a simulated social environment by comparing per-user LoRA agents against shared-base agents in OASIS. The population is sampled from the c8 game-development community, with $N \in \{128, 256, 512\}$. In the LoRA condition, each user receives a rank-4 LoRA adapter trained on 80 historical tweets. In the control condition, all agents sample decisions from the same shared Qwen3-4B-Instruct base model. The OASIS setup, recommender, decision prompt, follow graph, stance seed posts, and initial polarization distance are held fixed. Prompt-level cross-side exposure remains approximately 0.16-0.18 in every cell, so downstream differences are not explained by systematically different feeds. The recommender controls exposure, while the adapter controls reaction.

The LoRA population differs from the shared-base population along the expected scale-out chain: diversity, activity, and topology (Table 6). First, it preserves identity-level heterogeneity. At every N, final polarization distance remains higher under per-user LoRA, and within-side stance dispersion is consistently larger: supportive-user standard deviation is 2.18×–2.45× that of the base condition, while skeptical-user dispersion is 1.32×–2.01×. Second, this diversity produces a richer action ecology. Compared with the shared-base condition, LoRA produces substantially more comments and original posts, while shared-base agents collapse toward a narrower action prior with no original posts and very few comments. Third, activity compounds into population topology. Effective interaction communities grow monotonically from 9.21 to 14.85, co-engagement modularity grows from 0.502 to 0.716, and within-community side-homophily falls from 0.670 to 0.583. These normalized metrics show that larger LoRA populations do not simply produce more events. They produce more effective micro-communities, tighter co-attention structure, and communities that increasingly cross the original supportive/skeptical split.

The controlled comparison to shared-base agents (Table 7) shows that these effects are not an automatic consequence of OASIS or of increasing N. Across population sizes, LoRA produces more effective interaction

Table 7 Controlled comparison between per-user LoRA and shared-base agents. Values above one indicate a LoRA advantage for ratios, while negative values indicate lower side-homophily under LoRA.

Comparison	Metric	N=128	N=256	N=512
LoRA / Base	Effective interaction communities	1.48×	2.19×	1.47×
LoRA / Base	Co-engagement modularity	1.20×	0.95×	1.20×
LoRA − Base	Within-community side-homophily	−0.089	−0.193	−0.197
LoRA / Base	Supportive stance std.	2.28×	2.18×	2.45×
LoRA / Base	Skeptical stance std.	2.01×	1.32×	1.57×
Base / LoRA	Top-10% post like share	1.16×	1.37×	1.28×
LoRA − Base	Comments in DB	+70	+247	+493
LoRA − Base	Original posts in DB	+139	+153	+306

communities, lower within-community side-homophily, broader stance dispersion, and a heavier long tail of attention. The shared-base condition, by contrast, concentrates likes more sharply and produces much less content-creation signal.

These results should be read as evidence for a scale-out regime rather than a universal social-simulation law: they are measured on one community, one recommender mechanism, and one seed per cell. Still, they isolate the relevant mechanism for this paper. The shared base model primarily simulates exposure to content, while the LoRA population simulates persistent actors with different histories, stances, and behavioral priors. This is the environment-level role of personality in Scale Out.

Personality as environment. The OASIS result matters because it changes the ontology of simulation. With personality-bearing adapters, the environment is no longer a feed shown to interchangeable samples from one shared policy. It is composed of persistent actors that react differently to the same exposure and whose differences compound into population topology. The unit of scale is therefore diversity, not only throughput: scaling the number of users means scaling isolated histories, preferences, action priors, and behavioral attractors. Such populations can support product-policy testing, recommender intervention studies, echo-chamber analysis, and multi-agent RL environments populated by more realistic users. The environment result establishes the first value of diversity: persistent adapter differences can generate activity and structure interaction. The next subsection asks whether the same kind of diversity can also be aggregated into direct task performance.

5.3 Diversity as a Source of Collective Intelligence

Personal adaptation explains how adapters remain different. Collective intelligence asks why those differences matter. Once many adaptive instances coexist, diversity itself can become a computational resource. Different personal models may accumulate distinct histories, specialize along different trajectories, make different errors, prefer different tools, and solve problems through different reasoning paths. Scale Out therefore asks not only whether many adapters can be trained and served, but whether increasing the number of distinct adapted models produces measurable system-level value.

A Controlled Model-Count Experiment. A controlled model-count experiment makes this question measurable. We start from the same base model, Qwen3-30B, keep the same RL recipe and evaluation target, train many LoRA variants that differ only by training-data permutation and masking, and scale collaboration by increasing the number of models k. Answers are aggregated by majority vote over 200 collaboration evaluation sources, using random subset sampling for each k and 30 random samples per k. Empty extracted answers do not vote. Here, population activity is not social interaction but collective inference: each adapted model contributes an answer, and value emerges only when differentiated answers can be aggregated.

The design controls away other sources of scale: the base model family, Math17k (Hendrycks et al., 2021)

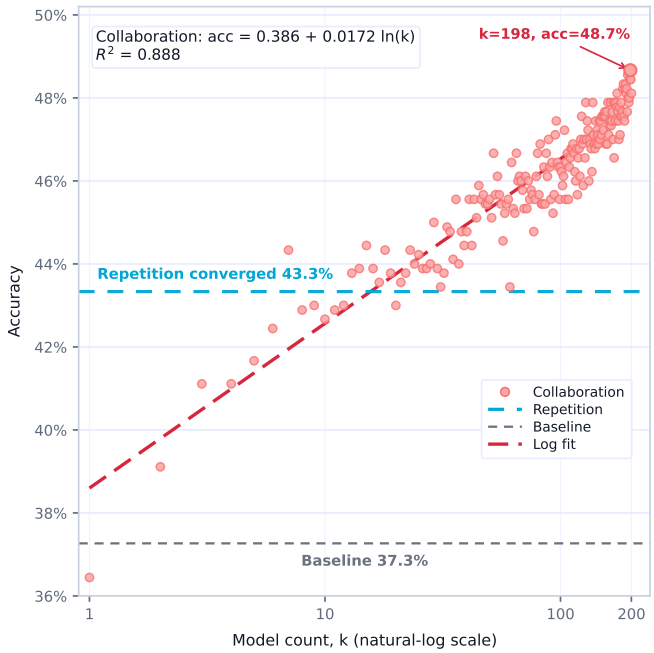


Figure 24 Model-count scaling by majority vote. Collaboration across distinct LoRA models improves beyond repeated sampling from one model, and the Collaboration curve is approximately linear in $\ln(k)$ over the measured range.

training task, AIME24 (Mathematical Association of America, 2024) evaluation, optimization recipe, answer extraction, and correctness pipeline are fixed. In the figure and analysis, *Collaboration* denotes the former different-model setting, where votes are aggregated across distinct LoRA variants; *Repetition* denotes the former same-model setting, where votes are aggregated from repeated samples of one model. If Collaboration improves more than Repetition, the gain cannot be explained only by stochastic decoding; it indicates complementary policies learned by different LoRA trajectories.

Model count produces predictable gains. The main result is shown in Figure 24: Collaboration improves steadily as k increases. Accuracy rises from 0.3644 at $k = 1$ to 0.4267 at $k = 10$, 0.4633 at $k = 100$, and a best observed 0.4867 at $k = 198$. Relative to the final baseline accuracy of 0.3727, the largest gain is about +0.1140. Repetition improves early but saturates sooner, reaching a best accuracy of 0.4378 at $k = 24$. At large k, the Collaboration advantage reaches about +0.0533. Fitting the Collaboration curve against $\ln(k)$ gives

$$\text{accuracy} \approx 0.386 + 0.0172 \ln(k), \tag{3}$$

with $R^2 \approx 0.888$ over the observed range. The discovered behavior is not linear in k; it is approximately linear in $\ln(k)$ (shown in Figure 24). Adding more collaborating models gives diminishing but predictable returns. We treat this as an empirical law in one controlled regime, not as a universal theorem of model populations. Its importance is that it makes model count a measurable scale-out variable: accuracy can be studied as a function of the number of distinct LoRA-adapted models.

Why diversity is not sampling noise. The comparison between Collaboration and Repetition shows that adapter diversity is not equivalent to sampling noise. Repeated sampling from one model helps at small k, because stochastic decoding produces varied answers, but it saturates earlier. Voting across different LoRA instances continues to improve at larger k, suggesting that the population aggregates complementary policies. Importantly, this diversity does not come from unrelated architectures or pretraining corpora. It comes from small PEFT runs following different trajectories under data ordering, masking, stochasticity, and checkpoint timing. Even this modest, cheaply constructed adapter diversity is useful.

From one model to a distribution of models. Operationally, this experiment would be difficult without PEFT. Training 200 full checkpoints, serving them, extracting answers, and evaluating many random subsets would be expensive and cumbersome. LoRA turns each model into a lightweight variant of the same prior, making controlled population experiments feasible. This is the concrete mechanism behind Scale Out: **once adapter creation and serving are cheap, researchers can optimize not only a model but a distribution of models.**

The observed log law is not a universal theorem of model populations, but it establishes a research object: accuracy as a function of model count. Future systems may route among adapters, vote across them, cluster them by experience, distill from successful subpopulations, or feed aggregate lessons back into a shared prior. This is why the scale-out thesis emphasizes populations of personal models rather than a single increasingly personalized assistant. Individual adaptation creates value, but populations create a second-order resource: diversity of histories, skills, failures, and successes. The resulting system is not one universal assistant with ever more context, but an ecology of persistent, partially specialized agents.

6 Infrastructure for PEFT Populations

6.1 Why the Three Axes Need a Systems Layer

The three scaling axes define a practical architecture only if the adapted state can survive as an operational object. Without a systems layer, each axis fails in a characteristic way. Scale Up without lifecycle management produces a strong prior that can be adapted once but not repeatedly. Each LoRA RL run becomes an isolated event, adapter state is lost between runs, and the trained behavior cannot be reliably transferred to the serving runtime that will actually deploy it. Scale Down without mobility management produces a small adapter that is efficient during training but checkpoint-centric during deployment. Every variant requires a full merged model artifact, and the population scales with base-model copies rather than with local adaptive state. Scale Out without addressability and residency control produces a large catalog of adapter files that cannot be selectively loaded, evicted, evaluated, or rolled back. Model count grows, but adapted identities do not persist.

PEFT makes individuality compact; managing compact individuality requires a lifecycle like the one instantiated by MinT (Mind Lab, 2026). Its mechanisms and measurements illustrate what the three axes require in practice. MinT keeps expensive dense or MoE base models resident and treats LoRA adapters as behavior-carrying policy state. It does not make PEFT important by itself. Rather, it supplies an implementation example of the lifecycle that the three axes require. Large-prior rollout and training remain semantically connected. Small adapters move as exported revisions instead of full checkpoints. Many policy revisions remain addressable while only a bounded working set occupies local cache or GPU batch slots.

The unit managed by such infrastructure is not a single file. A continuing adapter-based policy includes adapter tensors, optimizer state, rollout records, evaluation results, and serving placement. These facts change at different time scales. A trainer may hold mutable optimizer state, a sampler may need a fixed adapter revision, a serving actor may evict adapter bytes without deleting the policy, and an evaluation score must name the revision that produced it. Population-scale personalization therefore needs a systems layer because scale is not just the number of adapters stored on disk. It is the number of adapted identities that can be resumed, scored, selected, served, and governed over time.

6.2 Policy Identity: From Adapter Weights to Adapter Revisions

The adapter boundary becomes a system object when MinT names it as a policy record and exports fixed adapter revisions from it. Raw adapter weights are not enough: they do not say which base model they attach to, which rank and target modules define their shape, which rollout records generated the latest update, which exported version was evaluated, or where the adapter currently resides. MinT separates these concerns through policy records and adapter revisions.

A policy record is the durable identity of one adapted behavior over a compatible base. It records the base version, adapter shape, training checkpoint state, rollout records, and exported adapter revisions. A policy session is a temporary restoration of that record on a trainer. An adapter revision is a fixed serving/evaluation object in the layout expected by the sampler. Serving residency is only a placement fact: the revision may

Table 8 MinT separates policy identity from temporary compute residency.

State object	What it stores	Why it matters for PEFT scaling
Base deployment	A resident dense or MoE foundation model.	Keeps the expensive shared prior loaded while many adapters change around it.
Policy record	Base version, LoRA rank, target modules, checkpoints, rollout records, and exported revisions.	Makes an adapted behavior resumable, auditable, and rollbackable.
Policy session	A temporary trainer restoration with adapter tensors, optimizer moments, scheduler position, gradients, and rollout metadata.	Allows time-sliced multi-LoRA training without duplicating the base model.
Adapter revision	A fixed exported PEFT adapter in serving tensor layout.	Defines the behavior selected by rollout, evaluation, serving, and rollback.
Serving residency	Whether an adapter is in shared storage, CPU cache, or a GPU batch slot.	Lets a large catalog remain addressable while only a bounded working set is active.

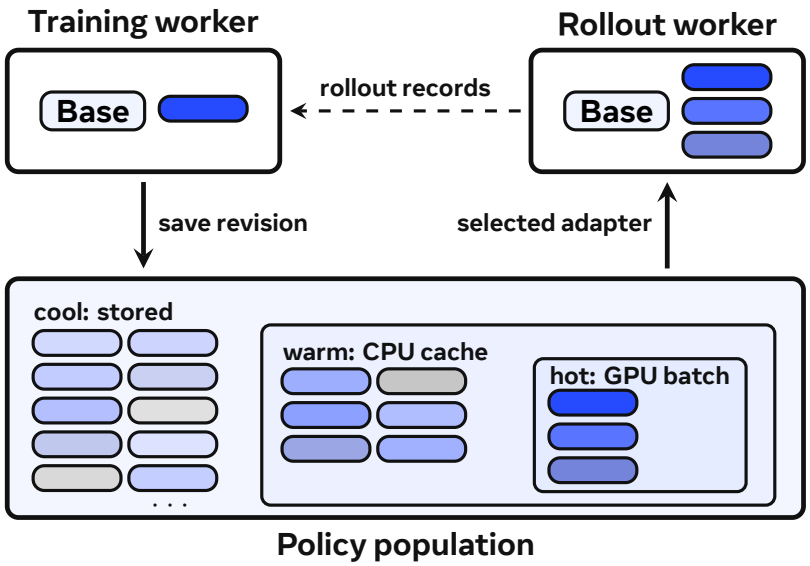


Figure 25 MinT policy lifecycle. Training updates adapter state over a resident base, export saves fixed adapter revisions into the policy population, and rollout or serving selects revisions through bounded residency tiers.

be active in a GPU batch, warm in a CPU cache, or only present in shared storage. This distinction turns a personal model from an anonymous LoRA file into a recoverable and auditable policy instance.

This identity layer connects directly to Scale Out. A population of personal models must preserve differences across time, not merely instantiate many prompts or adapters once. Memories, skills, preferences, and permissions need a place to accumulate, but they also need revision boundaries so that behavior can be evaluated, served, rolled back, or retired. The adapter remains the local adaptive state, and the policy record is the system object that lets that state persist as one member of a larger population.

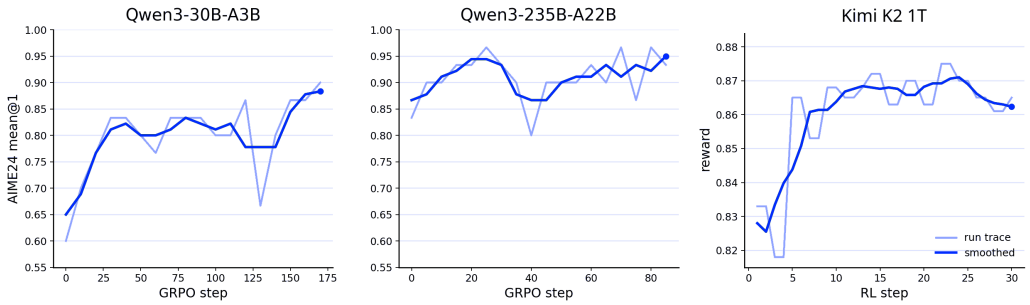


Figure 26 MoE LoRA RL curves from the MinT evaluation. The 30B and 235B panels use AIME24 mean@1; the Kimi K2 panel reports the end-to-end LoRA RL reward curve for a 1T-class countdown-task run.

6.3 Scale Up Requires Computation Provenance

Scale Up depends on keeping a strong prior available to repeated LoRA RL without turning every run into a separate full-model deployment. In MinT, the base deployment is the resident object: dense and MoE bases stay loaded across model-parallel trainer groups, rollout engines, and serving actors, while policy records restore only the adapter and training state needed for the selected policy. This is the systems form of the Scale Up argument. A stronger prior increases adapter leverage only if rollout, scoring, export, and serving continue to refer to the same policy over the same compatible base.

Large-prior LoRA RL also needs computation provenance. In an MoE model, selected expert ids determine part of the computation that produced each rollout token. If training-time scoring routes that token through different experts, the update no longer scores the sampled policy. MinT records route information when the backend exposes it and replays selected expert paths when the training layout can map them. If route ids are missing or unmappable, the corresponding token is removed from the replayed policy-gradient term rather than treated as equivalent.

Dynamic sparse attention introduces another provenance boundary. In GLM-5-style DSA, the indexer and top-k path determine which tokens enter sparse attention. MinT fixes implementation mismatches where the cause is exposed, including indexer RoPE layout, normalized query/key inputs, deterministic top-k behavior, frozen indexer defaults, long-context THD/CP support, and LoRA loading for DSA target modules. When probability mismatch remains, IcePop-style correction masks token-level ratios outside the trusted band. This does not reconstruct every DSA indexer decision, but it preserves the failure signal by excluding unsafe scoring terms from the update.

The relevant Scale Up property is semantic continuity, not only GPU capacity. The policy that generated trajectories, the policy whose probabilities are scored during training, and the policy later served as an adapter revision must remain the same adapted behavior over a compatible base. Router replay, DSA correction, distributed export, and policy-record resolution are the MinT mechanisms that keep a large prior usable as a repeated adaptation substrate rather than as a static checkpoint.

The large-model evidence exercises this lifecycle on dense and sparse paths, including Qwen3-235B-A22B GRPO and a Kimi K2 1T-class countdown-task LoRA RL path over a 1.04T-parameter MoE with 32.6B active parameters. The evidence boundary is specific: these runs show that large-prior LoRA RL can be made operational when adapter state, rollout records, route metadata, export layout, and serving revision are managed as one policy lifecycle. They do not by themselves prove a universal frontier-model scaling law. They show the infrastructure condition under which the Scale Up axis can be tested.

6.4 Scale Down Requires Adapter-Only Mobility

Scale Down asks whether the adaptive state can remain small, stable, and reusable across many updates. MinT turns that algorithmic requirement into a handoff boundary: the exported adapter revision, not a merged full checkpoint, is the artifact that moves from trainer to sampler. If every trained adapter has to be merged into a full checkpoint before serving, the update may be parameter-efficient during training but

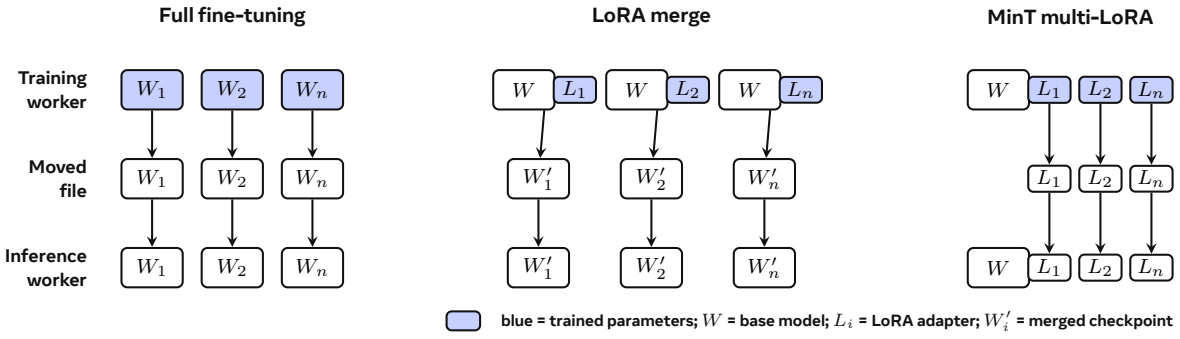


Figure 27 Training-to-serving artifacts under full fine-tuning, merge-based LoRA, and MinT. Full fine-tuning and merge-based LoRA move full model checkpoints for each variant. MinT moves exported LoRA adapter revisions to an inference engine that already holds the compatible base.

checkpoint-centric during deployment. The population would then scale with repeated base-model artifacts rather than with local adaptive state.

The exported revision carries adapter tensors, rank, target modules, tensor layout, and base compatibility. Optimizer state, accumulated gradients, and rank-local training files remain on the training side. Evaluation and serving select the fixed revision, so a score or deployed behavior is attributed to a concrete adapter state rather than to an implicit worker-local checkpoint. This is the system-level counterpart of the low-rank and initialization results in Scale Down: reducing trainable state matters most when the reduced state is also the object that can be moved, evaluated, served, and rolled back.

Table 9 Adapter-only handoff keeps the moved artifact much smaller than a merged or full checkpoint while preserving the sampling path used for evaluation. Cold first sample is the first request wall time; total sample speed includes that first request, while warm speed excludes it.

Model	Path	Moved artifact	File size	Materialization or load	Cold first sample	sample speed total/warm
Qwen3-4B						
Qwen3-4B	Adapter	rank-32 LoRA	252 MiB	0.036 s	4.114 s	15.568/15.567 tok/s
Qwen3-4B	Merge	full model	8.061 GB	71.820 s	55.704 s	4.697/20.595 tok/s
Qwen3-30B						
Qwen3-30B	Adapter	rank-16 LoRA	1.692 GB	46.455 s	117.304 s	1.874/5.700 tok/s
Qwen3-30B	Merge	full model	61.084 GB	402.245 s	156.074 s	1.573/6.904 tok/s

The MinT handoff measurements illustrate why this boundary matters for the Scale Down axis. On Qwen3-4B, a rank-32 adapter is 252 MiB, while the merged full checkpoint is 8.061 GB. On Qwen3-30B, a rank-16 adapter is 1.692 GB, while the merged full checkpoint is 61.084 GB. These numbers do not define a universal adapter ratio, because rank, target modules, dtype, and tensor layout change the size. They show the systems invariant: the crossing artifact can remain the local adaptive state rather than a full copy of the shared prior. In MinT, Scale Down is therefore not just a smaller training update. It is a smaller object that can move across training, rollout, evaluation, serving, and rollback without re-materializing the base model.

MinT also supports Scale Down through time-sliced training over resident bases. A policy session restores only the selected adapter tensors, optimizer moments, scheduler position, accumulated gradients, and rollout records. The base remains loaded while different policies take turns on compatible trainer workers. This does not make each individual update mathematically smaller, but it reduces the system cost of maintaining many small updates: repeated learning becomes a schedulable service operation rather than a sequence of isolated full-checkpoint jobs.

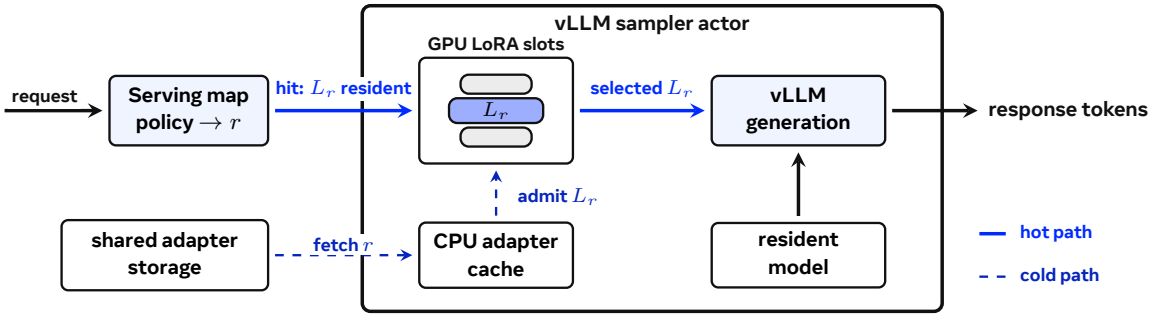


Figure 28 Shared-base multi-LoRA serving in MinT. A request resolves to an exported adapter revision. The hot path uses an adapter already in a GPU slot, while the cold path fetches the revision from shared storage into the CPU cache and then admits it into a GPU batch slot before decoding.

6.5 Scale Out Requires Bounded Residency

Scale Out asks what changes when the number of adapted instances becomes a scaling variable. MinT supports this axis by separating addressability from residency. A million personal models does not mean that one inference engine keeps a million adapters in GPU memory. It means that a large population of adapter revisions can be named, selected, loaded, served, evicted, and later recovered while each serving actor keeps only a bounded local working set. Multi-tenant LoRA serving systems (Sheng et al., 2023; Chen et al., 2023; Zhou et al., 2025d) establish the batching and operator-level machinery for sharing one base model across many adapters; the MinT-specific question is how to extend that view into a lifecycle of named, persistent policy revisions.

MinT separates serving state into three tiers. The durable catalog names adapter revisions in shared storage. The CPU cache holds adapter bytes near one serving actor. The GPU batch contains the smaller set of adapters active in the current decoding step. A request may hit the GPU batch, promote a CPU-cached adapter, or enter a cold-load path from shared storage. The policy identity is stable across all three placements, and only residency changes.

Table 10 Policy-population serving separates addressability from live residency.

Resource tier	Evidence or bound	Interpretation
Addressable catalog	Built and audited a 10^6-entry packed adapter catalog; serving experiments select bounded working sets from it.	Catalog size is an addressability scale, not simultaneous GPU residency.
CPU adapter cache	369 loaded adapters at a 512-adapter hotset; 550 loaded adapters under 2048-adapter weak-locality pressure.	Local CPU memory absorbs recurring traffic before requests touch shared storage.
GPU batch slots	64 distinct adapters in the tested same-batch window.	Batch execution has the smallest adapter-diversity window.
Cold loading	16 distinct cache misses form a 1.375–23.267 s load staircase.	Different missing policies must be admitted as explicit service work.
Packed MoE LoRA loading	37,248 tensor objects reduced to 672; live loading becomes 8.5–8.7× faster.	Representation controls cold-load overhead even when adapter bytes are modest.
Readiness gate	Newly registered adapters become user-visible only after activation/prewarm.	Registration and serving readiness are separate lifecycle states.

The MinT serving measurements preserve the distinction between addressability, local residency, and active GPU use. The catalog evidence is an addressability result: MinT builds and audits a 10^6-entry packed adapter catalog. The local-residency evidence is smaller by design: on one Qwen3-30B rank-1 serving actor, repeated-adapter traffic reaches hundreds of CPU-cached adapters, while the tested same-batch adapter window is 64 distinct adapters. Weak-locality traffic and broad rollout waves expose the cold path: 16 distinct cache misses form a 1.375–23.267 second load staircase, while repeated requests for the same missing policy can share one load. These numbers are the evidence for the residency boundary that Scale Out needs: policy count can be large only if catalog lookup, CPU cache, GPU batch slots, and cold activation are treated as different service scales.

Table 11 Hot-reload and readiness measurements on the MinT serving path. Admission protects old warm tenants by moving cold activation into a scheduled path; two-phase readiness exposes new adapters only after activation, so zero load time applies to ready-path user requests rather than registration time.

Policy	Existing warm traffic	New-adapter path	Interpretation
Expose before readiness			
Admission off	post TTFT p95 24.03 s; >20 s stalls: 10	e2e p95 59.18 s; user TTFT p95 22.19 s; load p95 47.37 s	Fast exposure, but cold first-touch disrupts warm tenants.
Admission on	post TTFT p95 9.71 s; >20 s stalls: 0	e2e p95 314.79 s; user TTFT p95 10.68 s; load p95 294.96 s	Admission protects warm tenants, but new users wait behind activation.
Expose after readiness			
Two-phase readiness	post TTFT p95 9.63 s; >20 s stalls: 0	ready-path TTFT p95 4.60 s; load p95 0.00 s; prewarm span 409.04 s	First user requests arrive after activation, so they do not load adapters.

Readiness is the operational control that prevents Scale Out from degrading existing warm users. Immediate exposure lets new adapters become selectable quickly, but cold first-touch disrupts warm tenants: without admission, existing warm traffic reaches 24.03 s post-reload TTFT p95 and records 10 stalls above 20 s. Admission removes those stalls but shifts waiting to new cold requests. Two-phase readiness changes the user-visible contract: the adapter is registered and prewarmed first, then exposed after activation. In the measured row, old warm TTFT p95 stays at 9.63 s with no stalls above 20 s, and the first ready-path request to the new adapter has 0.00 s load p95 and 4.60 s TTFT p95 after a 409.04 s prewarm span.

Table 12 Packed MoE LoRA loading reduces cold-load overhead by removing tensor fanout. The byte-size change is small; the speedup comes from replacing many tiny tensor objects with a compact serving representation.

Metric	Original	Packed	Effect
Adapter-file shape			
File size	110.75 MB	105.58 MB	1.05× smaller
Tensor objects	37,248	672	55.4× fewer
Cold-load slice			
Read tensors	0.3669 s	0.0067 s	54.8× faster
Build loader objects	0.7540 s	0.0256 s	29.5× faster
Live engine loading			
$N=4$ live load	1.363 s	0.156 s	8.7× faster
$N=8$ live load	1.361 s	0.159 s	8.6× faster
$N=16$ live load	1.388 s	0.164 s	8.5× faster

Adapter representation is the second Scale Out control surfaced by the MinT measurements because the cold path can be object-bound rather than byte-bound. In the measured MoE rank-1 setting, the raw adapter is moderate in bytes but fragmented into 37,248 tensor objects. Packing reduces this to 672 tensor objects with nearly unchanged bytes, improves the direct loader slices by 29.5–54.8×, and makes live engine loading 8.5–8.7× faster with packed medians below 0.2 seconds. The important conclusion is therefore not that cold loads are slow in one probe. It is that policy count creates several service scales that must be controlled separately: catalog registration names durable revisions, routing preserves locality, CPU cache absorbs recurring policies, GPU batch slots bound active diversity, cold activation is scheduled work, and readiness decides when a

newly registered adapter becomes user-visible. Scale Out is a controlled lifecycle for many policy revisions, not a promise that every revision is simultaneously resident.

6.6 The Lifecycle of a Personal Model

The personal-model thesis becomes concrete when experience has a durable path into local adaptive state. A user interaction, tool outcome, evaluation trace, or social-simulation event produces records that can be scored or distilled. Training updates adapter state over a shared prior. Export freezes a revision. Serving selects that revision under bounded residency. Later experience resumes from the policy record rather than from an anonymous adapter file. The individual model is therefore not a full checkpoint, but a continuing adaptive identity over a shared base.

This lifecycle ties MinT back to the preceding Scale Out section. LoRA-as-memory needs a writeable local state. Context Learning needs a path from context-time improvement to future query-only behavior. Skill internalization needs repeated updates from tool outcomes and failures. Personalization as a security boundary needs permission and behavior history to remain attached to the policy that will later act. All of these require identity, provenance, mobility, and residency control.

The resulting architecture is the one implied by the title of the paper. A few trillion-scale priors provide shared competence. Many adapter revisions provide local memory, skill, preference, and policy state. MinT keeps those revisions identifiable and movable while serving only a bounded active set at any moment. It is the concrete infrastructure example that connects the three axes: Scale Up through resident large-prior LoRA RL with computation provenance, Scale Down through adapter-only mobility and time-sliced policy sessions, and Scale Out through addressable catalogs with bounded residency.

7 Conclusion

The phrase "million personal models of trillion parameters" should not be read as a claim that each user owns and trains a separate trillion-parameter checkpoint. The intended architecture is different. A small number of strong trillion-scale base models provide shared capability, while millions of lightweight adapters provide persistent local adaptive state. The base model carries general reasoning, world knowledge, language competence, and tool-use priors. The adapter carries part of the learned consequences of repeated experience, such as memories, preferences, skills, and policies.

This architecture depends on all three scaling axes at once. Scale Up makes the shared base model worth adapting. Scale Down makes each update cheap and stable enough to repeat. Scale Out turns repeated updates into persistent populations. Removing any axis breaks the thesis. Weak base models limit what adapters can learn. Expensive adapters prevent continuous adaptation. Without multiplicity, PEFT remains a local optimization rather than a path toward population-scale personalization.

Several research problems remain open, organized by the three axes:

1. **Scale Up (RL-native PEFT theory)**: rank, scaling, initialization, KL drift, and off-policiness need a unified account that connects adapter geometry to training stability at large prior scale.
2. **Scale Up (Tiny-adapter reliability)**: low-rank LoRA already shows signal on strong priors, but must become stable under seed, batch, and task variation before it can serve as a repeatable adaptation substrate.
3. **Scale Down (Stateful adapter design)**: memory-oriented mechanisms such as δ-mem need controlled comparisons against standard LoRA, retrieval, and full-context baselines to establish when a stateful adapter is the right representation.
4. **Scale Down (Signal efficiency)**: Context Learning needs benchmarks that measure how much durable improvement is extracted from real interaction traces, and how that improvement degrades or compounds over repeated updates.
5. **Scale Out (LoRA memory and skill capacity)**: the bounded capacity law for parametric memory observed on simple benchmarks needs to be extended to RL-trained, multi-task adapters, where the open question

is which experience deserves to become durable adapter state and which should remain in retrieval or context.

6. **Scale Out (Persistent user simulation)**: per-user adapters resolve the collapse of prompt-based personas in small populations, but it remains open whether large adapter populations can preserve heterogeneity over long-horizon interaction without drifting back to a base-model average.
7. **Scale Out (Population as computation)**: aggregation gains from distinct LoRA models suggest a research object beyond per-model accuracy, namely how routing, voting, debate, and distillation across adapter populations scale with model count and adapter diversity.

Limitations. The evidence in this paper points to a direction, not a deployed system. Most experiments are run at the scale of controlled benchmarks and simulations, while large-scale empirical validation on our own personal-model deployments remains limited. The bottleneck at this point is compute capacity, not missing methodology. We believe this is the right direction, and that its claims will sharpen as more real-interaction evidence accumulates at scale.

The final view is a population architecture rather than one universal assistant with ever more context and ever more centralized control. PEFT makes it possible to scale from one shared foundation model to many persistent personal model instances: first serving individuals, then supporting user simulation, and eventually making diversity among adapted models a source of collective intelligence and creativity. That is the broader reason PEFT matters. It makes adaptation efficient, and through that efficiency it makes persistent individuality scalable.

Acknowledgements

We thank all members of Mind Lab for the discussions, experiments, engineering support, writing feedback, and project context that shaped this manuscript. Names are listed alphabetically within each group.

Core Contributors. Andrew Chen, Steven Chiang, Kyrie Lei, Kieran Liu, Pony Ma, Vincent Wang, Josh Ying, Di Zhang, Ruijia Zhang, Adrian Zhou, Yuhua Zhou.

Team. Song Cao, Vic Cao, Kaijie Chen, Bunny Fan, Hera Feng, Huan Feng, Arthur Fu, Jun Gao, Hongquan Gu, Aaron Guan, Mutian Hong, Hailee Hou, Peixuan Hua, Charles Huang, Miles Jiang, Nora Jiang, Yuyi Jiang, Autumn Jin, Fancy Kong, Kyrie Lei, Alexy Li, Dawn Li, Ray Li, Theo Li, Wenhao Li, Jiayi Lin, Domini Liu, Heshan Liu, Kairus Liu, Logan Liu, Maeve Luo, Runism Lv, Pony Ma, Verity Niu, Anson Qiu, Vincent Wang, Maxwell Yao, Regis Ye, Wenlin Ye, Yanying Ye, Josh Ying, Danney Zeng, Salmon Zhan, Anya Zhang, Ruijia Zhang, Shiyang Zhang, Sueky Zhang, Ya Zhang, Wei Zhao, Ada Zhou, Sizer Zhou, Xinyue Zhu, Murphy Zhuang.

References

1000 Genomes Project Consortium, Gonçalo R. Abecasis, Adam Auton, Lisa D. Brooks, Mark A. DePristo, Richard M. Durbin, Robert E. Handsaker, Hyun Min Kang, Gabor T. Marth, and Gil A. McVean. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65, 2012. doi: 10.1038/nature11632. URL https://doi.org/10.1038/nature11632.

Anthropic. Claude 4.7 model card, 2025a. URL https://www.anthropic.com/claude/claude-4.

Anthropic. Claude code: Agentic coding at the command line. Anthropic product, 2025b. URL https://www.anthropic.com/claude-code.

Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, SK Hong, Youngjune Gwon, and Sungjin Ahn. Understanding lora as knowledge memory: An empirical analysis. *arXiv preprint arXiv:2603.01097*, 2026.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, et al. LoRA learns less and forgets less, 2024. URL https://arxiv.org/abs/2405.09673.

Kerim Büyükakyüz. Olora: Orthonormal low-rank adaptation of large language models, 2024. URL https://arxiv.org/abs/2406.01775.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving, 2023. URL https://arxiv.org/abs/2310.18547.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489: 57–74, 2012. doi: 10.1038/nature11247.

GLM-5 Team. GLM-5: From vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, et al. Measuring mathematical problem solving with the MATH dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Jian Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model, 2025a. URL https://arxiv.org/abs/2503.24290.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions. *arXiv preprint arXiv:2507.05257*, 2025b.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, et al. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

Peak Ji. Context engineering for AI agents: Lessons from building manus. Blog post, 2025. URL https://medium.com/@peakji/context-engineering-for-ai-agents-lessons-from-building-manus-71883f0a67f2.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with LoRA, 2023. URL https://arxiv.org/abs/2312.03732.

Kimi Team. Kimi K2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

Fanqi Kong, Xiaoyuan Zhang, Xinyu Chen, Yaodong Yang, Song-Chun Zhu, and Xue Feng. Enhancing LLM-based social bot via an adversarial learning framework. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 23235–23260, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1185. URL https://aclanthology.org/2025.emnlp-main.1185/.

Jingdi Lei, Di Zhang, Junxian Li, Weida Wang, Kaixuan Fan, Xiang Liu, Qihan Liu, Xiaoteng Ma, Baian Chen, and Soujanya Poria. δ-mem: Efficient online memory for large language models, 2026. URL https://arxiv.org/abs/2605.12357.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Lucian Li, Qihan Liu, Song Cao, Ruijian Ye, Andrew Chen, Pony Ma, and Mind Lab. Mindclaw: Fine-tuning openclaw for personalized long-term memory. Mind Lab: A Lab for Experiential Intelligence, 2026. https://macaron.im/mindlab/research/mindclaw-fine-tuning-openclaw-for-personalized-long-term-memory.

Yuheng Li, Hao Wen, Weizhi Wang, Xiang Li, Yuan Yuan, Gao Liu, Jiacheng Liu, Wenyuan Xu, Xiang Wang, Yi Sun, et al. Personal LLM agents: Insights and survey about the capability, efficiency and security, 2024. URL https://arxiv.org/abs/2401.05459.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization, 2026. URL https://arxiv.org/abs/2604.02268.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents, 2024. URL https://arxiv.org/abs/2402.17753.

Mathematical Association of America. 2024 american invitational mathematics examination, 2024. URL https://artofproblemsolving.com/wiki/index.php/2024_AIME_I_Problems. AIME 2024 problem set.

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.

Mind Lab. MinT: Managed infrastructure for training and serving millions of LLMs, 2026. URL https://arxiv.org/abs/2605.13779.

OpenAI. GPT-4.5 system card, 2025. URL https://openai.com/index/gpt-4-5-system-card/.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, et al. MemGPT: Towards LLMs as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. doi: 10.1145/3586183.3606763. URL https://arxiv.org/abs/2304.03442.

Qwen Team. Qwen3 technical report, 2025. URL https://qwenlm.github.io/blog/qwen3/.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter, 2020. URL https://arxiv.org/abs/1910.01108.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning, 2026. URL https://arxiv.org/abs/2601.19897.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Chris Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-LoRA: Serving thousands of concurrent LoRA adapters, 2023. URL https://arxiv.org/abs/2311.03285.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. LoRA vs full fine-tuning: An illusion of equivalence, 2025. URL https://arxiv.org/abs/2410.21228.

David Silver and Richard S. Sutton. Welcome to the era of experience. Essay, 2025. URL https://storage.googleapis.com/deepmind-media/Era-of-Experience%20/The%20Era%20of%20Experience%20Paper.pdf.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context, 2022. URL https://arxiv.org/abs/2209.15189.

Han Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. MiLoRA: Harnessing minor singular components for parameter-efficient LLM finetuning, 2025. URL https://arxiv.org/abs/2406.09044.

Xingyao Wang, Boxuan Chen, Hao Tang, et al. OpenHands: An open platform for AI software developers as generalist agents, 2024. URL https://arxiv.org/abs/2407.16741.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Haotian Xia et al. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

An Yang et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, et al. Tensor programs V: Tuning large neural networks via zero-shot hyperparameter transfer, 2022. URL https://arxiv.org/abs/2203.03466.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, Zijian Ling, Jinsong Chen, Martz Ma, Bowen Dong, et al. OASIS: Open agentf social interaction simulations with one million agents, 2024. URL https://arxiv.org/abs/2411.11581.

Shunyu Yao. The second half. Blog post, 2025. URL https://ysymyth.github.io/The-Second-Half/.

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.

Ruijia Zhang, Jiacheng Zhu, Hanqing Zhu, and Laixi Shi. Geometry-preserving orthonormal initialization for low-rank adaptation in reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2026.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, volume 28, 2015. URL https://papers.nips.cc/paper/2015/hash/250cf8b51c773f3f8dc8b4be867a9a02-Abstract.html.

Chujie Zheng et al. Stabilizing reinforcement learning with LLMs: Formulation and practices, 2025. URL https://arxiv.org/abs/2512.01374.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

Changhai Zhou, Shijie Han, Lining Yang, Yuhua Zhou, Xu Cheng, Yibin Wang, and Hongguang Li. RankAdaptor: Hierarchical rank allocation for efficient fine-tuning pruned LLMs via performance model. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5796–5810. Association for Computational Linguistics, 2025a. URL https://aclanthology.org/2025.findings-naacl.321/.

Changhai Zhou, Qian Qiao, Yuhua Zhou, Yuxin Wu, Shichao Weng, Weizhong Zhang, and Cheng Jin. Large language model compression with global rank and sparsity optimization, 2025b. URL https://arxiv.org/abs/2505.03801.

Changhai Zhou, Shiyang Zhang, Yuhua Zhou, Qian Qiao, Jun Gao, Shichao Weng, Weizhong Zhang, and Cheng Jin. Balancing fidelity and plasticity: Aligning mixed-precision fine-tuning with linguistic hierarchies, 2025c. URL https://arxiv.org/abs/2505.03802.

Changhai Zhou, Yuhua Zhou, Shiyang Zhang, Yibin Wang, and Zekai Liu. Dynamic operator optimization for efficient multi-tenant LoRA model serving. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(21):22910–22918, 2025d. doi: 10.1609/aaai.v39i21.34453. URL https://ojs.aaai.org/index.php/AAAI/article/view/34453.

Changhai Zhou, Shiyang Zhang, Yuhua Zhou, Qian Qiao, Jun Gao, Cheng Jin, Kaizhou Qin, and Weizhong Zhang. AutoQRA: Joint optimization of mixed-precision quantization and low-rank adapters for efficient LLM fine-tuning, 2026a. URL https://arxiv.org/abs/2602.22268.

Yuhua Zhou, Ruifeng Li, Changhai Zhou, Fei Yang, and Aimin Pan. BSLoRA: Enhancing the parameter efficiency of LoRA with intra-layer and inter-layer sharing. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 78883–78902. PMLR, 2025e. URL https://proceedings.mlr.press/v267/.

Yuhua Zhou, Changhai Zhou, Shiyang Zhang, Fei Yang, Yi Zhang, and Aimin Pan. LaRA: Layer-wise rank allocation for efficient fine-tuning of pruned large language models. *Information Processing & Management*, 63(3):104538, 2026b.

Hanqing Zhu, Zhenyu Zhang, Hanxian Huang, DiJia Su, Zechun Liu, Jiawei Zhao, Igor Fedorov, Hamed Pirsiavash, Zhizhou Sha, Jinwon Lee, David Z. Pan, Zhangyang Wang, Yuandong Tian, and Kai Sheng Tai. The path not taken: Rlvr provably learns off the principals, 2025. URL https://arxiv.org/abs/2511.08567.