ASPECT: AGENT-SPECIFIC PARAMETER-EFFICIENT CORE TUNING FOR MULTI-AGENT LLM WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model multi-agent workflows increasingly use reinforcement learning to optimize specialized agents toward a shared system-level objective. This raises a fundamental question of what should be shared across agents and what should remain agent-specific. Existing parameter-sharing strategies either force multiple agents to update the same adaptation parameters or separate their adaptations entirely, creating a tension between parameter reuse and individual adaptation. To this end, we propose **A**gent-**S**pecific **P**arameter-**E**fficient **C**ore **T**uning (**ASPECT**), which separates shared task-aligned adaptation structure from agent-specific parameter updates. ASPECT constructs a shared adaptation space from task calibration data and assigns each agent a compact private core within this space. This design allows agents to reuse common task structure while retaining distinct parameter adaptations and workflow-level coordination. Extensive experiments on Math and Code tasks demonstrate that ASPECT achieves a stronger performance–parameter efficiency trade-off than representative parameter-sharing baselines, with over **90% fewer** trainable parameters.

1 INTRODUCTION

Large language models (LLMs) are increasingly organized into multi-agent systems, where different agents take on distinct roles within a shared workflow (Tran et al., 2025; Chen et al., 2024b; Qian et al., 2024). Their effectiveness often comes from task decomposition and role specialization, rather than simply increasing the number of agents (Hong et al., 2024; Chan et al., 2024; Smit et al., 2024). In most existing systems, such specialization is induced through prompts and interaction protocols while the underlying model parameters remain fixed (Du et al., 2024; Chen et al., 2024a). Recent work has begun to optimize multi-agent workflows end to end through reinforcement learning (RL) with system-level rewards (Zhang et al., 2026; Zhao et al., 2026; Chen et al., 2026c). Once agents can adapt through training, a new question arises: *what should be shared across agents, and what should remain agent-specific?*

Despite the shared objective, agents in the same workflow receive different local contexts and serve different functions (Chen et al., 2026a). As a result, the same system-level reward can induce different learning signals for different agents. When multiple agents update the same parameters, these signals can conflict or become unbalanced (Yu et al., 2020; Chen et al., 2018). A direct solution is to maintain a separately trainable LLM for each agent, which eliminates shared parameter updates but incurs substantial training and storage costs. Parameter-efficient fine-tuning offers a cheaper alternative by freezing the backbone and introducing lightweight Low-Rank Adaptation (LoRA) modules for policy optimization (Hu et al., 2022; Liao et al., 2025). One common strategy is **workflow-level sharing**, where all agents update a single adaptation module. Such global sharing maximizes parameter reuse, but couples updates from agents with different roles and contexts. A more flexible alternative is **role-level sharing**, which assigns a separate adaptation module to each role. While this separates updates across roles, different instances of the same role still share adaptation parameters. Recent work has linked these two forms of sharing to *role capture* and *same-role gradient amplification*, respectively (Zeng et al., 2026). At the finest granularity, **agent-level separation** assigns an independent adaptation module to each agent. Each agent requires a complete set of low-rank factors, while common task-specific structure remains unshared. These designs expose a tension: stronger sharing improves parameter reuse but restricts independent adaptation, whereas full separation provides greater freedom at the cost of sharing less adaptation structure.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

What is missing is a way to reuse common task structure without forcing different agents to share the same adaptation parameters. Although agents may need different updates, their shared task distribution and system-level objective suggest that these updates may exhibit common task-specific structure. Prior work suggests that language-model adaptation often concentrates in a low-dimensional space (Aghajanyan et al., 2021). This makes it possible to share a compact set of task-relevant update directions while allowing each agent to adapt differently within them. The remaining challenge is to identify a low-rank space aligned with the target task. Task gradients provide a practical way to construct such *task-aligned spaces* (Wang et al., 2025; 2024). Recent studies further show that the resulting bases can remain fixed while only a compact set of parameters is optimized (Ponkshe et al., 2024). Together, these findings suggest that agents can share a task-aligned adaptation space while learning distinct parameter updates within it.

To realize this idea, we propose **A**gent-**S**pecific **P**arameter-**E**fficient **C**ore **T**uning (**ASPECT**). ASPECT first estimates a task-specific update from a small calibration set. It then applies truncated singular value decomposition (SVD) (Golub & Reinsch, 1971) to extract low-rank bases that define a shared, task-aligned adaptation space. The LLM backbone and the shared bases remain frozen during reinforcement learning. Each agent optimizes only *a compact private core*, which specifies its own parameter update within the task-aligned space. We therefore interpret each core as *an agent-specific coordinate matrix* over the shared bases. This design reuses common task-level structure without forcing different agents to update the same parameters. Figure 1 contrasts ASPECT with workflow-level sharing, role-level sharing, and agent-level separation. Meanwhile, agents continue to coordinate through workflow interactions and shared outcome rewards. Our contributions are as follows:

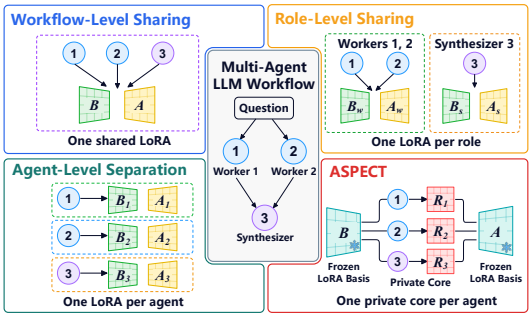


Figure 1: Comparison of workflow-level sharing, role-level sharing, agent-level separation, and ASPECT in a multi-agent LLM workflow.

- We propose **ASPECT**, which separates a shared task-aligned adaptation space from agent-specific parameter updates.
- By optimizing only compact agent-specific core matrices while keeping the LLM backbone and shared bases frozen, ASPECT achieves agent-specific adaptation with over **90% fewer** trainable parameters than representative parameter-sharing baselines.
- Extensive experiments across Math and Code tasks, multiple model scales, and diverse workflows demonstrate a stronger overall performance–parameter efficiency trade-off than representative alternatives.

2 Related Work

RL-Trained Multi-Agent LLM Workflows. Recent frameworks have made multi-agent LLM workflows trainable through RL. MARTI integrates multi-agent interaction with distributed policy optimization (Zhang et al., 2026), while UnityMAS-O supports configurable workflows and parameter-sharing strategies (Chen et al., 2026c). Joint training introduces challenges in both stability and efficiency. Dr. MAS improves optimization stability through agent-specific advantage normalization (Feng et al., 2026). To reduce computational demands, MARFT employs LoRA-based policy adaptation (Liao et al., 2025). However, how these adapters are shared across agents also matters, as different sharing strategies can introduce distinct optimization difficulties (Zeng et al., 2026). ASPECT addresses this issue by sharing frozen task-aligned adaptation structure while keeping each agent's trainable core private.

Parameter Sharing in Multi-Agent Systems. Parameter sharing improves learning and parameter efficiency in traditional multi-agent reinforcement learning (MARL), but complete sharing can limit specialization across roles or observations (Terry et al., 2020). Selective Parameter Sharing relaxes this constraint by grouping agents and sharing parameters only within groups (Christianos et al., 2021). At a finer granularity, SNP-PS constructs partially overlapping agent networks through

structured pruning (Kim & Sung, 2023). GradPS adapts the sharing structure by identifying and separating neurons affected by conflicting gradients (Qin et al., 2025). HyperMARL instead generates agent-conditioned policy parameters through a shared hypernetwork (Tessera et al., 2025). Developed primarily for conventional MARL policies, these mechanisms are nontrivial to transfer to LLMs given their scale and architecture.

Low-Rank and Task-Aligned Adaptation. LoRA freezes pretrained weights and represents task updates with two trainable low-rank factors (Hu et al., 2022). To further reduce trainable parameters, VeRA shares frozen random projections and learns compact scaling parameters (Kopiczko et al., 2024). LoRA-XS instead derives frozen bases from pretrained weights and optimizes a small core matrix (Bałazy et al., 2025). Such parameter savings, however, do not ensure task alignment. LoRA-SB addresses this limitation by constructing the adaptation space from an approximate task-specific update (Ponkshe et al., 2024). Whereas these methods focus on single-model adaptation, ASPECT shares task-aligned structure across agents while preserving their distinct parameter updates.

3 PRELIMINARIES

3.1 LLM-BASED MULTI-AGENT WORKFLOW FORMULATION

Given a task distribution $\mathcal{D}$, a task instance is denoted by $x \sim \mathcal{D}$. The agent set is defined as $\mathcal{A} = \{1, \ldots, N\}$, and the role set as $\mathcal{C} = \{1, \ldots, C\}$. A role mapping $\rho : \mathcal{A} \to \mathcal{C}$ assigns each agent i to a role $\rho(i)$. Multiple agent instances may share the same role. A workflow is represented by a directed graph $\mathcal{W} = (\mathcal{A}, \mathcal{E})$, where $\mathcal{E}$ specifies the permitted information flow and execution dependencies among agents. The graph can represent sequential execution, parallel branches, and iterative feedback.

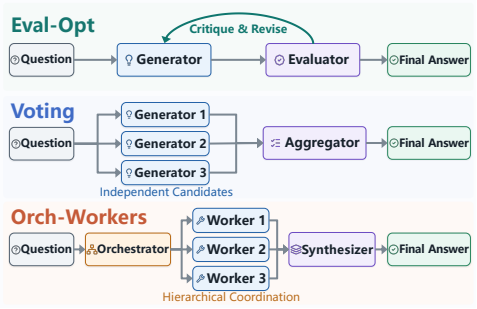


Figure 2: Three representative multi-agent LLM workflows used in this study.

In this work, we focus on parameter-sharing workflows in which all agents are instantiated from the same pretrained LLM backbone and differ in their trainable adaptation parameters. The shared backbone parameters are denoted by $\mathbf{W}_0$ and remain frozen during training. The trainable adaptation parameters are organized into a set of blocks $\Theta = \{\theta_m\}_{m=1}^{M}$. A parameter mapping $\phi : \mathcal{A} \to \{1, \ldots, M\}$ associates each agent i with the policy $\pi_{\theta_{\phi(i)}}$. Different choices of ϕ correspond to different parameter-sharing strategies across agents. The dependence on the shared backbone $\mathbf{W}_0$ is omitted from the policy notation for clarity.

Executing the workflow on task x produces a structured joint trajectory

$$\tau = \left(x, \{i_t, h_t, y_t\}_{t=1}^{T}\right), \tag{1}$$

where t indexes an LLM invocation, i_t denotes the invoked agent, h_t is its local context, and y_t is its generated response. A workflow-level outcome reward $\mathcal{R}(\tau)$ is computed from the final answer. This reward provides the learning signal for the trainable agent responses along the trajectory. As illustrated in Figure 2, we consider three representative workflows following prior work (Zeng et al., 2026). **(i) Eval-Opt** follows an iterative refinement process, where a generator produces an answer and an evaluator provides feedback for subsequent revision. **(ii) Voting** captures parallel exploration by allowing three same-role generators to independently produce candidate solutions, which are then reviewed by an aggregator to determine the final answer. **(iii) Orch-Workers** represents hierarchical coordination, where an orchestrator assigns subtasks to three same-role workers and a synthesizer integrates their responses into the final answer. Together, these workflows cover representative coordination patterns commonly used in existing multi-agent systems.

3.2 GROUP RELATIVE POLICY OPTIMIZATION

In this paper, the multi-agent workflow is optimized using Group Relative Policy Optimization (GRPO) (Shao et al., 2024). For each task x, the old policies sample a group of G complete workflow trajectories $\{\tau_g\}_{g=1}^{G}$. Each trajectory τ_g receives a workflow-level outcome reward $\mathcal{R}_g = \mathcal{R}(\tau_g)$.

Then the corresponding group-normalized advantage is

$$\widehat{A}_g = \frac{\mathcal{R}_g - \mu_{\mathcal{R}}}{\sigma_{\mathcal{R}} + \delta}, \tag{2}$$

where $\mu_{\mathcal{R}}$ and $\sigma_{\mathcal{R}}$ denote the mean and standard deviation of the rewards within the group, respectively. The constant $\delta > 0$ ensures numerical stability. All trainable responses in the same trajectory share the corresponding advantage $\widehat{A}_g$.

For a token a generated by agent i under context h, the probability ratio between the current and old policies is

$$\omega_\Theta(a \mid h, i) = \frac{\pi_{\theta_{\phi(i)}}(a \mid h)}{\pi_{\theta^{\text{old}}_{\phi(i)}}(a \mid h)}. \tag{3}$$

The ratio measures the change in token probability under the updated adaptation parameters. The GRPO objective is

$$\mathcal{J}(\Theta) = \mathbb{E}_{\text{old}}\left[\min\Big(\omega_\Theta \widehat{A},\ \text{clip}(\omega_\Theta, 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})\widehat{A}\Big)\right]. \tag{4}$$

The expectation $\mathbb{E}_{\text{old}}$ is taken over task samples and workflow trajectory groups generated by the old policies. The variable $\widehat{A}$ denotes the trajectory-level advantage associated with the current token. Clipping with ϵ_{low} and ϵ_{high} constrains excessively large policy updates. To isolate the effect of parameter-sharing strategies and avoid introducing an additional regularization term, the reference-policy KL penalty is omitted in this study.

3.3 Parameter-Efficient Fine-Tuning

Parameter-efficient fine-tuning (PEFT) adapts pretrained models by optimizing a small subset of parameters. For a linear layer ℓ with frozen weights $\mathbf{W}_0^{(\ell)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, where d_{in} and d_{out} are its input and output dimensions, LoRA (Hu et al., 2022) parameterizes the adapted weight as

$$\mathbf{W}^{(\ell)} = \mathbf{W}_0^{(\ell)} + \gamma \mathbf{B}\mathbf{A}, \tag{5}$$

where $\mathbf{A} \in \mathbb{R}^{r \times d_{\text{in}}}$ and $\mathbf{B} \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable factors, r is the adaptation rank, and γ is a scaling coefficient. The per-layer trainable parameter count is $r(d_{\text{in}} + d_{\text{out}})$.

LoRA-XS (Bałazy et al., 2025) further reduces this cost using the rank-r truncated singular value decomposition (SVD) (Golub & Reinsch, 1971) of the pretrained weights:

$$\mathbf{W}_0^{(\ell)} \approx \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^\top. \tag{6}$$

The matrices $\mathbf{U}_r$ and $\mathbf{V}_r$ contain the leading left and right singular vectors, and $\mathbf{\Sigma}_r$ contains their singular values, yielding a best rank-r approximation in the Frobenius norm (Eckart & Young, 1936). LoRA-XS freezes $\mathbf{B} = \mathbf{U}_r \mathbf{\Sigma}_r$ and $\mathbf{A} = \mathbf{V}_r^\top$ and uses

$$\mathbf{W}^{(\ell)} = \mathbf{W}_0^{(\ell)} + \gamma \mathbf{B}\mathbf{R}\mathbf{A}. \tag{7}$$

Only $\mathbf{R} \in \mathbb{R}^{r \times r}$ is trainable, requiring r^2 parameters per layer and restricting updates to the space defined by the frozen factors.

However, the pretrained weight directions may not align with downstream task updates. LoRA-SB (Ponkshe et al., 2024) instead estimates an initial full fine-tuning update from a small training subset and applies truncated SVD to construct task-aligned bases and initialize the core. Building on this construction, ASPECT shares the task-aligned adaptation space across agents while retaining agent-specific parameter updates.

4 Method

This section describes the design of ASPECT and its components. ASPECT separates shared task-aligned adaptation structure from agent-specific parameter updates. As illustrated in Figure 3, it estimates an initial task-specific update from calibration data and applies truncated SVD to obtain low-rank bases. These bases define a shared adaptation space and remain frozen across agents. Each agent optimizes a private core matrix that specifies its parameter update within this space. During RL, only the agent-specific cores are updated, while the LLM backbone and shared bases remain fixed. Implementation details are provided in Appendix A.

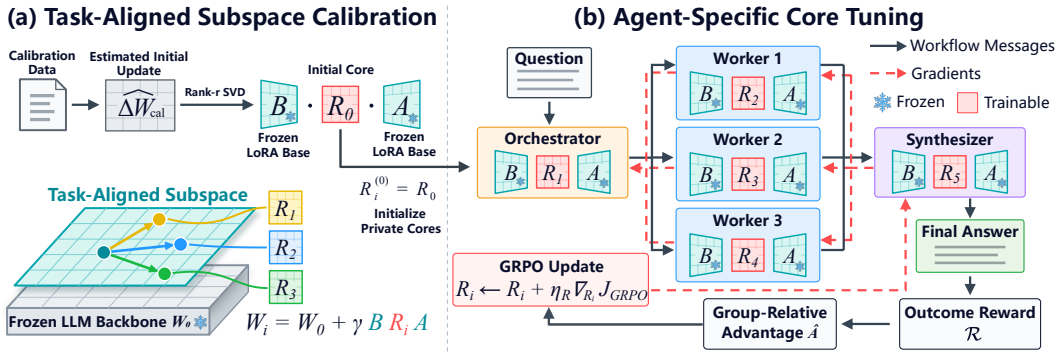


Figure 3: Overview of ASPECT. **(a)** Task-aligned subspace calibration constructs shared bases and initializes private cores. **(b)** Agents in an Orch-Workers workflow collaborate through messages and optimize their private cores using shared rewards.

4.1 Task-Aligned Subspace Calibration

First, ASPECT constructs a shared task-aligned adaptation space that captures update directions relevant to the common task objective. To this end, a small calibration subset $\mathcal{D}_{\text{cal}} \subset \mathcal{D}_{\text{train}}$ is sampled from the training data, where each example contains a question x and its reference answer y. The calibration data are then used to estimate a task-specific update that is shared as the reference structure for all agents. The answer-level cross-entropy loss is defined as:

$$\mathcal{L}_{\text{cal}}(x, y; \mathbf{W}_0) = -\frac{1}{|\mathcal{T}_y|} \sum_{k \in \mathcal{T}_y} \log p_{\mathbf{W}_0}(y_k \mid x, y_{<k}), \tag{8}$$

where $\mathcal{T}_y$ indexes the answer tokens included in the loss. For clarity, the following derivation considers an arbitrary adapted linear layer and omits the layer index ℓ. Its pretrained weight matrix is denoted by $\mathbf{W}_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Calibration gradients are accumulated without modifying the pretrained backbone, and an approximation to the initial AdamW update (Loshchilov & Hutter, 2017) is constructed as:

$$\widehat{\mathbf{G}}_{\text{cal}} = \sum_{(x,y) \in \mathcal{D}_{\text{cal}}} \nabla_{\mathbf{W}_0} \mathcal{L}_{\text{cal}}(x, y), \qquad \widehat{\Delta \mathbf{W}}_{\text{cal}} = -\eta \, \text{sign}\left(\widehat{\mathbf{G}}_{\text{cal}}\right), \tag{9}$$

where η controls the calibration step size and $\text{sign}(\cdot)$ is applied element-wise. The step size η determines the overall magnitude of the estimated update, while $\text{sign}(\cdot)$ retains only the direction of each gradient entry to form a scale-normalized update proxy. This update serves only as a task-specific calibration signal and does not modify the pretrained weights.

To extract a compact set of directions that captures the estimated task-specific update, a low-rank decomposition is performed. A rank-r truncated SVD is then applied to the estimated update:

$$\widehat{\Delta \mathbf{W}}_{\text{cal}} \approx \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^\top. \tag{10}$$

Based on this decomposition, the shared bases and initial core are defined as:

$$\mathbf{B} = \mathbf{U}_r, \qquad \mathbf{A} = \mathbf{V}_r^\top, \qquad \mathbf{R}_0 = \mathbf{\Sigma}_r. \tag{11}$$

Specifically, $\mathbf{B}$ and $\mathbf{A}$ serve as the shared bases that define the task-aligned adaptation space, while $\mathbf{R}_0$ serves as the initial core and provides a common coordinate initialization for all agents. Consequently, $\mathbf{B}\mathbf{R}_0\mathbf{A}$ reconstructs the rank-r approximation of the estimated task-specific update. The calibration procedure is applied independently to each adapted layer in the shared LLM backbone, yielding layer-specific shared bases and core initialization. Calibration is performed only once before RL, after which the bases are shared across agents and remain fixed throughout training.

4.2 Agent-Specific Core Tuning

With the shared task-aligned bases fixed after calibration, ASPECT introduces agent-specific adaptation through private core matrices. For each agent $i \in \mathcal{A}$, a private core $\mathbf{R}_i \in \mathbb{R}^{r \times r}$ is assigned to

every adapted layer. All private cores share the same initialization, $\mathbf{R}_i^{(0)} = \mathbf{R}_0 = \mathbf{\Sigma}_r$, but remain independent trainable parameters. The effective weight for agent i at an adapted layer is

$$\mathbf{W}_i = \mathbf{W}_0 + \gamma \mathbf{B}\mathbf{R}_i\mathbf{A}. \tag{12}$$

The scaling coefficient is set to $\gamma = 1$, ensuring that the initial adaptation reproduces the rank-r approximation of the estimated task-specific update. The backbone and shared bases remain frozen throughout RL. Private cores remain separate even for same-role agents, such as the *generators* in *Voting* and the *workers* in *Orch-Workers*. Accordingly, the parameter mapping ϕ assigns each agent to its own adaptation parameters, with $\phi(i) = i$ and θ_i containing the private cores of agent i.

During RL, the private cores are optimized using the GRPO objective in Eq. (4). For a fixed sampled trajectory τ, $\mathcal{J}_i$ denotes the objective terms contributed by responses from agent i. With sampled contexts and advantages fixed, the gradients with respect to the private cores satisfy

$$\nabla_{\mathbf{R}_i}\mathcal{J}_i = \gamma \mathbf{B}^\top \left(\nabla_{\mathbf{W}_i}\mathcal{J}_i\right)\mathbf{A}^\top, \qquad \frac{\partial \mathcal{J}_i}{\partial \mathbf{R}_j} = 0, \quad j \neq i. \tag{13}$$

Each agent therefore updates only its own private cores, so gradients from different agents are not accumulated into the same trainable parameters. This property also holds for sequential workflows because gradients are not propagated through sampled messages between agents. Agents nevertheless continue to coordinate through workflow interactions and shared outcome rewards.

Schematically, GRPO updates each private core by gradient ascent:

$$\mathbf{R}_i \leftarrow \mathbf{R}_i + \eta_R \nabla_{\mathbf{R}_i}\mathcal{J}_i, \tag{14}$$

where η_R denotes the learning rate for private core optimization. Each core acts as an agent-specific coordinate matrix, enabling distinct parameter updates within the shared task-aligned space. This compact parameterization substantially reduces the cost of agent-specific adaptation, which is analyzed in the following subsection. The formal properties are established in Appendix E.

4.3 Parameter Efficiency

This section analyzes the trainable parameter cost under parameter-sharing strategies and quantifies the parameter efficiency of ASPECT. For an adapted linear layer with rank r, different parameter-sharing strategies incur different trainable parameter costs. *Workflow-level sharing* requires a single LoRA adapter, with $r(d_{\text{in}} + d_{\text{out}})$ trainable parameters. *Role-level sharing* maintains one adapter per role, multiplying the parameter count by the number of roles. *Agent-level separation* assigns a complete adapter to each agent. In contrast, ASPECT assigns each agent a private $r \times r$ core matrix while sharing the frozen task-aligned bases. The per-layer parameter counts P are

$$P_{\text{workflow}} = r(d_{\text{in}}{+}d_{\text{out}}),\ P_{\text{role}} = |\mathcal{C}|\, r(d_{\text{in}}{+}d_{\text{out}}),\ P_{\text{agent}} = |\mathcal{A}|\, r(d_{\text{in}}{+}d_{\text{out}}),\ P_{\text{ASPECT}} = |\mathcal{A}|\, r^2. \tag{15}$$

The quantities $|\mathcal{A}|$ and $|\mathcal{C}|$ denote the numbers of agents and roles, respectively. Full-model parameter counts are obtained by summing the corresponding costs over all adapted layers. This comparison highlights that ASPECT retains agent-specific adaptation while reducing its per-agent trainable cost from $r(d_{\text{in}} + d_{\text{out}})$ for a complete LoRA adapter to only r^2 per adapted layer.

5 Experiments

5.1 Experimental Setup

Tasks and Datasets. We evaluate mathematical reasoning (*Math*) on DAPO-Math (Yu et al., 2025) with 12,704 training and 1,412 validation problems, and code generation (*Code*) on DeepCoder (Luo et al., 2025) with 13,995 training and 1,000 validation problems. These tasks provide complementary settings for evaluating multi-agent reasoning and collaboration. All methods use identical dataset splits to ensure a consistent comparison.

Models and Workflows. We use *Qwen3-0.6B*, *Qwen3-1.7B*, and *Qwen3-4B* (Yang et al., 2025) to examine performance across model scales. Each model is evaluated with the Eval-Opt, Voting, and Orch-Workers workflows defined in Section 3.1. All agents within a workflow use the same pretrained backbone, which remains frozen across all methods.

Table 1: Validation accuracy (%) across tasks, models, and workflows; methods report peak accuracy among checkpoints evaluated within the 200 training steps. Base is evaluated without training. Bold marks the best result per row. Average gives the per-method mean value across configurations.

Task	Model	Workflow	Base	WS	RS	AS	ASPECT (ours)
Math	Qwen3-0.6B	Eval-Opt	10.69	21.25	**38.67**	37.25	29.04
		Voting	11.19	24.08	26.20	25.57	**27.05**
		Orch-Workers	12.68	32.86	29.82	30.10	**33.14**
	Qwen3-1.7B	Eval-Opt	26.70	42.42	50.07	**56.94**	**56.94**
		Voting	30.67	43.13	49.01	43.48	**52.27**
		Orch-Workers	32.15	41.01	51.63	49.72	**62.46**
	Qwen3-4B	Eval-Opt	48.37	65.37	73.73	**76.13**	69.69
		Voting	52.76	56.16	65.37	**68.56**	58.71
		Orch-Workers	54.18	70.75	71.46	70.33	**74.22**
Code	Qwen3-0.6B	Eval-Opt	7.90	11.70	16.30	**18.10**	15.30
		Voting	7.30	**12.50**	12.40	**12.50**	12.00
		Orch-Workers	7.30	13.00	**15.00**	12.10	13.70
	Qwen3-1.7B	Eval-Opt	16.30	19.60	25.80	24.10	**29.40**
		Voting	17.40	20.90	20.40	20.40	**22.00**
		Orch-Workers	15.80	19.50	19.00	18.60	**27.10**
	Qwen3-4B	Eval-Opt	26.70	32.20	39.70	38.80	**45.00**
		Voting	27.70	32.70	36.50	31.90	**41.70**
		Orch-Workers	28.30	33.70	38.00	40.10	**46.60**
Average			24.12	32.94	37.73	37.48	**39.80**

Table 2: Trainable parameters in millions (M) for each model scale, workflow, and method. All methods use rank $r = 64$. Frozen backbone weights are excluded.

Model	Workflow	WS	RS	AS	ASPECT
Qwen3-0.6B	Eval-Opt	40.37	80.74	80.74	**1.61**
	Voting	40.37	80.74	161.48	**3.21**
	Orch-Workers	40.37	121.11	201.85	**4.01**
Qwen3-1.7B	Eval-Opt	69.73	139.46	139.46	**1.61**
	Voting	69.73	139.46	278.92	**3.21**
	Orch-Workers	69.73	209.19	348.65	**4.01**
Qwen3-4B	Eval-Opt	132.12	264.24	264.24	**2.06**
	Voting	132.12	264.24	528.48	**4.13**
	Orch-Workers	132.12	396.36	660.60	**5.16**

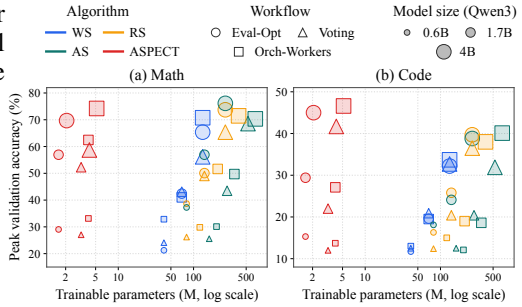


Figure 4: ASPECT versus baselines: peak validation accuracy against trainable parameters across model scales and workflows on Math (left) and Code (right).

Baselines. We compare ASPECT with three baselines: *Workflow-Level Sharing (WS)*, *Role-Level Sharing (RS)*, and *Agent-Level Separation (AS)*. WS trains one LoRA adapter shared by all agents, whereas RS trains a separate adapter for each role. Under RS, multiple instances of the same role still share their adapter. AS assigns an independent LoRA adapter to each agent instance, including those with the same role. In contrast, ASPECT shares frozen task-aligned LoRA bases while training a compact private core for each agent instance.

Evaluation Metrics. We report *training success rate* and *validation accuracy*. Training success rate is the fraction of sampled workflow trajectories whose final outputs are correct in each training batch. Checkpoints are saved every 10 steps and evaluated on the complete validation set, with one workflow execution per problem. Validation accuracy measures the fraction of correct final outputs: Math answers are checked against reference answers using the task verifier, while Code solutions must pass all retained hidden tests. Each experiment runs for 200 training steps. In the training success rate plots, faint lines show raw values, while solid lines show a 10-step average. Detailed training configurations are provided in Appendix C.

5.2 Main Results

Table 1 reports peak validation accuracy within the 200 training steps across 72 experimental configurations covering two tasks, three model scales, three workflows, and four methods. ASPECT achieves the best performance in most task–model–workflow combinations and the highest average performance across all combinations. Its advantages extend across both tasks and multiple model scales and workflows, rather than being confined to a single setting. The overall results support sharing task-aligned adaptation structure while retaining private agent-specific updates. Complete learning curves and additional experimental analyses are provided in Appendix B.

5.3 Parameter Efficiency

Table 2 reports the trainable parameter counts of WS, RS, AS, and ASPECT across model scales and workflows. ASPECT consistently uses substantially fewer trainable parameters than all three baselines, retaining independent adaptation through compact private cores rather than complete LoRA adapters. As shown in Figure 4, it achieves higher average peak validation accuracy despite this smaller parameter budget, demonstrating a favorable performance–parameter efficiency trade-off.

5.4 Ablation Study

We evaluate the effect of task-aligned subspace calibration (TSC) on Qwen3-1.7B Math across all three workflows. The variant **ASPECT (w/o TSC)** constructs *frozen LoRA bases from pretrained backbone weights and randomly initializes its private cores*. In contrast, ASPECT derives both the bases and core initialization from an estimated task-specific update. ASPECT (w/o TSC) retains the same core-only trainable structure.

As shown in Figure 5, ASPECT (w/o TSC) achieves competitive performance with substantially fewer trainable parameters than the three baselines, supporting the effectiveness of agent-specific core tuning. The complete ASPECT shows larger early-stage gains in training success rate than ASPECT (w/o TSC). These improvements demonstrate the effectiveness of task-aligned subspace calibration, particularly in accelerating early learning.

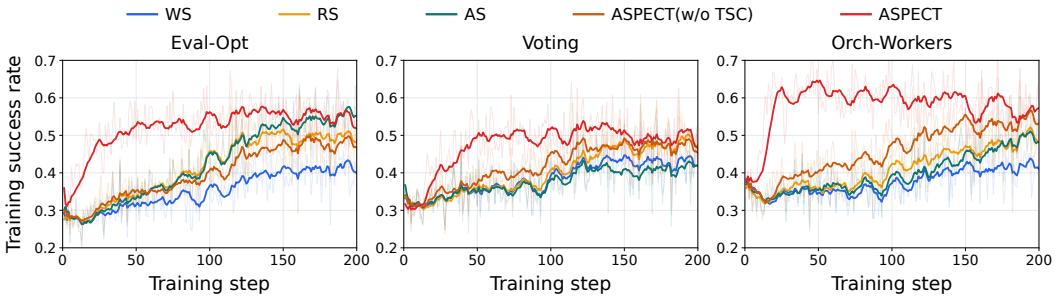


Figure 5: Task-aligned subspace calibration ablation on Qwen3-1.7B Math across three workflows.

5.5 Training Dynamics and Stability

Figure 6 compares training dynamics on Qwen3-4B Math under Orch-Workers. Token-level χ^2 is computed by averaging the squared ratios of training-side recomputed token probabilities to rollout-recorded probabilities over valid response tokens and subtracting one, measuring training–rollout policy mismatch. Training perplexity (PPL) is computed by exponentiating each response's mean negative token log-probability under the training policy and averaging across responses. RS exhibits progressively increasing policy mismatch and perplexity, accompanied by declining training success rate in later stages, while WS shows more moderate but persistent increases in both diagnostics. Their worker-level metrics closely track or exceed the global values. AS maintains low and stable diagnostics, consistent with its isolation of trainable parameters across agent instances. After an initial transient, ASPECT achieves similarly stable diagnostics and a higher late-stage training success rate while using less than 1% of AS's trainable parameters. These results highlight the benefits of agent-specific core tuning for both training stability and parameter efficiency.

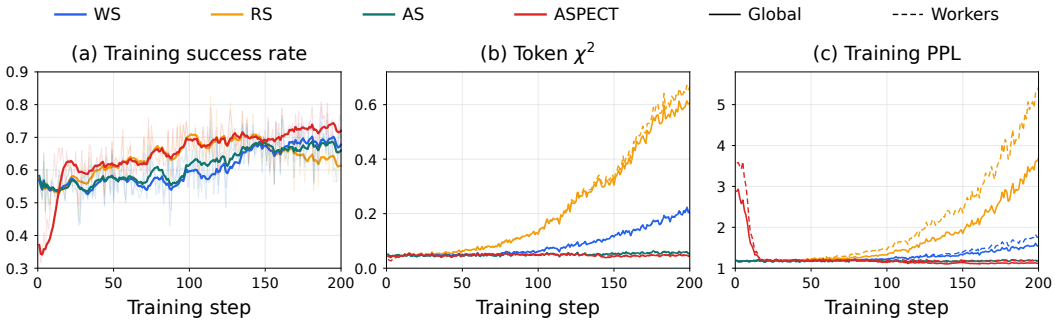


Figure 6: Training stability of all methods on Qwen3-4B Math under Orch-Workers. **(a)** Training success rate. **(b)** Training-rollout policy mismatch measured by token-level χ^2. **(c)** Perplexity on sampled trajectories. Solid and dashed diagnostic curves indicate global and worker-level metrics.

5.6 Agent-Specific Core Evolution

Figure 7 examines three same-role workers at a fixed query-projection layer of Qwen3-1.7B on Math under Orch-Workers. Despite identical initialization and shared frozen bases, the workers develop distinct core matrices after 200 training steps. Meanwhile, the mean leading singular value increases from 0.0127 to 0.1013, while the mean energy share of the leading component rises from 28.0% to 46.2%. This spectral concentration is qualitatively consistent with rank-1 dominance in RL-induced LLM weight updates (Cai et al., 2026) and policy learning along a few major parameter directions (Tang et al., 2024). Together, these visualizations show that workers learn different core matrices within the shared subspace while their core spectra develop stronger dominant components.

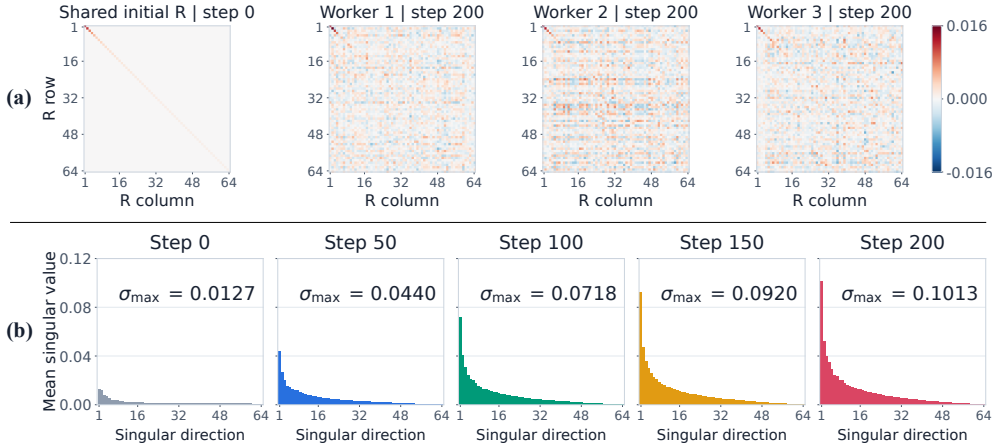


Figure 7: Agent-specific core evolution in Qwen3-1.7B on Math under Orch-Workers, using cores from the same layer. **(a)** The shared initialization and three worker cores at step 200, with a common color scale. **(b)** Evolution of the singular values of $\mathbf{R}_i$ averaged across workers at five checkpoints.

6 Conclusion

We presented ASPECT for parameter-efficient reinforcement learning in multi-agent LLM workflows. It shares frozen task-aligned low-rank bases while training compact private cores, preserving agent-specific updates and workflow collaboration. Experiments on Math and Code across model scales and workflows demonstrate a favorable performance–parameter efficiency trade-off, with over 90% fewer trainable parameters than the three baselines: workflow-level sharing, role-level sharing and agent-level separation. Ablations support the benefits of both agent-specific core tuning and task-aligned subspace calibration. Training diagnostics indicate more stable learning dynamics, while core visualizations show distinct agent-specific adaptations with stronger dominant spectral components. Future work will explore adaptive subspace calibration and extend ASPECT to larger, dynamically structured multi-agent workflows.

AI USE STATEMENT

We used generative AI tools to polish the language and improve the readability of the manuscript. The authors reviewed all AI-assisted revisions and take full responsibility for the final content.

REPRODUCIBILITY STATEMENT

The method is described in Section 4, with the training procedure and GRPO implementation detailed in Appendix A. Section 5.1 specifies the datasets, pretrained models, workflows, and evaluation protocol. Hyperparameters and initialization settings are provided in Appendix C, while software dependencies and computational resources are documented in Appendix D. The assumptions and proofs underlying the theoretical properties are presented in Appendix E. The supplementary material includes source code, workflow prompts, and instructions for data preparation, calibration, training, and evaluation.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pp. 7319–7328. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.568. URL https://aclanthology.org/2021.acl-long.568/.

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. LoRA-XS: Low-rank adaptation with extremely small number of parameters. In *Proceedings of the 28th European Conference on Artificial Intelligence*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pp. 3194–3201. IOS Press, 2025. doi: 10.3233/FAIA251185. URL https://doi.org/10.3233/FAIA251185.

Yuchen Cai, Ding Cao, Xin Xu, Zijun Yao, Yuqing Huang, Zhenyu Tan, Benyi Zhang, Guiquan Liu, and Junfeng Fang. On predictability of reinforcement learning dynamics for large language models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=SdHmA6BYVJ.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. ChatEval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/25cc3adf8c85f7c70989cb8a97a691a7-Abstract-Conference.html.

Guanzhong Chen, Shaoxiong Yang, Chao Li, Wei Liu, Jian Luan, and Zenglin Xu. End-to-end optimization of LLM-driven multi-agent search systems via heterogeneous-group-based reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 30319–30338. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.acl-long.1399. URL https://aclanthology.org/2026.acl-long.1399/.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. ReConcile: Round-table conference improves reasoning via consensus among diverse LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 7066–7085. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.acl-long.381. URL https://aclanthology.org/2024.acl-long.381/.

Nan Chen, Soledad Villar, and Soufiane Hayou. Learning rate scaling across LoRA ranks and transfer to full finetuning, 2026b. URL https://arxiv.org/abs/2602.06204.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/578e65cdee35d00c708d4c64bce32971-Abstract-Conference.html.

Yiqun Chen, Wei Yang, Erhan Zhang, Shijie Wang, Qi Liu, Zechun Niu, Bin Zhang, Haitao Li, Rui Li, Lingyong Yan, Jinyuan Feng, Biqing Qi, Xiaochi Wei, Yan Gao, Yi Wu, Yao Hu, and Jiaxin Mao. UnityMAS-O: A general RL optimization framework for LLM-based multi-agent systems. *arXiv preprint arXiv:2605.26646*, 2026c. URL https://arxiv.org/abs/2605.26646.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 794–803. PMLR, 2018. URL https://proceedings.mlr.press/v80/chen18a.html.

Filippos Christianos, Georgios Papoudakis, Muhammad A. Rahman, and Stefano V. Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1989–1998. PMLR, 2021. URL https://proceedings.mlr.press/v139/christianos21a.html.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936. doi: 10.1007/BF02288367.

Lang Feng, Longtao Zheng, Shuo He, Fuxiang Zhang, and Bo An. Dr. MAS: Stable reinforcement learning for multi-agent LLM systems. *arXiv preprint arXiv:2602.08847*, 2026. URL https://arxiv.org/abs/2602.08847.

Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Linear algebra*, pp. 134–151. Springer, 1971.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6507b115562bb0a305f1958ccc87355a-Abstract-Conference.html.

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

Woojun Kim and Youngchul Sung. Parameter sharing with network pruning for scalable multi-agent deep reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1942–1950, 2023. URL https://arxiv.org/abs/2303.00912.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. MARFT: Multi-agent reinforcement fine-tuning. *arXiv preprint arXiv:2504.16129*, 2025. URL https://arxiv.org/abs/2504.16129.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. DeepCoder: A fully open-source 14b coder at o3-mini level. DeepCoder Technical Blog, 2025.

Kaustubh Ponkshe, Raghav Singhal, Eduard Gorbunov, Alexey Tumanov, Samuel Horváth, and Praneeth Vepakomma. Initialization using update approximation is a silver bullet for extremely efficient low-rank fine-tuning. *arXiv preprint arXiv:2411.19557*, 2024. URL https://arxiv.org/abs/2411.19557.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 15174–15186. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Haoyuan Qin, Zhengzhu Liu, Chenxing Lin, Chennan Ma, Songzhu Mei, Siqi Shen, and Cheng Wang. GradPS: Resolving futile neurons in parameter sharing network for multi-agent reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 50246–50268. PMLR, 2025. URL https://proceedings.mlr.press/v267/qin25c.html.

John Schulman and Thinking Machines Lab. LoRA without regret. *Thinking Machines Lab: Connectionism*, September 2025. doi: 10.64434/tml.20250929. URL https://thinkingmachines.ai/blog/lora/.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. doi: 10.48550/arXiv.2402.03300. URL https://arxiv.org/abs/2402.03300.

Andries Petrus Smit, Nathan Grinsztajn, Paul Duckworth, Thomas D. Barrett, and Arnu Pretorius. Should we be going MAD? a look at multi-agent debate strategies for LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 45883–45905. PMLR, 2024. URL https://proceedings.mlr.press/v235/smit24a.html.

Hongyao Tang, Min Zhang, Chen Chen, and Jianye Hao. The ladder in chaos: Improving policy learning by harnessing the parameter evolving path in a low-dimensional space. In *Advances in Neural Information Processing Systems*, volume 37, pp. 71004–71033, 2024. doi: 10.52202/079017-2269.

Jordan K. Terry, Nathaniel Grammel, Sanghyun Son, Benjamin Black, and Aakriti Agrawal. Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*, 2020. URL https://arxiv.org/abs/2005.13625.

Kale-ab Abebe Tessera, Arrasy Rahman, Amos Storkey, and Stefano V. Albrecht. HyperMARL: Adaptive hypernetworks for multi-agent RL. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://openreview.net/forum?id=56CgYnf9Dr.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of LLMs. *arXiv preprint arXiv:2501.06322*, 2025. URL https://arxiv.org/abs/2501.06322.

Shaowen Wang, Linxi Yu, and Jian Li. LoRA-GA: Low-rank adaptation with gradient approximation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1741. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/62c4718cc334f6a0a62fb81c4a2095a1-Abstract-Conference.html.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. LoRA-Pro: Are low-rank adapters properly optimized? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/ea184f920a0f0f8d8030aa1bd7ac9fd4-Abstract-Conference.html.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5824–5836, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/3fe78a8acf5fda99de95303940a2420c-Abstract.html.

Yifan Zeng, Yiran Wu, Yaolun Zhang, Wentian Zhao, Kun Wan, Qingyun Wu, and Huazheng Wang. When does multi-agent RL improve LLM workflows? workflow, scale, and policy-sharing tradeoffs. *arXiv preprint arXiv:2605.24202*, 2026. URL https://arxiv.org/abs/2605.24202.

Kaivan Zhang, Kai Tian, Runze Liu, Sihang Zeng, Xuekai Zhu, Guoli Jia, Yuchen Fan, Xingtai Lv, Yuxin Zuo, Che Jiang, Yuru Wang, Jianyu Wang, Ermo Hua, Xinwei Long, Junqi Gao, Youbang Sun, Zhiyuan Ma, Ganqu Cui, Ning Ding, Biqing Qi, and Bowen Zhou. MARTI: A framework for multi-agent LLM systems reinforced training and inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=E7jZqo0A50.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-MAS: Multi-agent reinforcement learning for collaborative LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IdF6JqXWzx.

APPENDIX

A METHOD DETAILS

A.1 TABLE OF NOTATION

Table 3 summarizes the notation used in the main text.

Table 3: Notation used in the main text.

Symbol	Explanation
$\mathcal{D}$	Task distribution.
$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{cal}}$	Training dataset and calibration subset.
x, y	Task question and reference answer.
$\mathcal{A}, N$	Agent set and count; i, j index agents.
$\mathcal{C}, C$	Role set and number of roles.
$\rho(i)$	Role assigned to agent i.
$\mathcal{W}, \mathcal{E}$	Workflow graph and directed dependency edges.
Θ, θ_m, M	Trainable parameter collection, block m, and block count.
$\phi(i)$	Parameter-block index assigned to agent i.
$\pi_{\theta_{\phi(i)}}$	Current policy of agent i.
$\pi_{\theta^{\text{old}}_{\phi(i)}}$	Old policy used to sample responses from agent i.
τ, τ_g	Workflow trajectory and the gth sampled trajectory.
t, T	Invocation index and number of invocations in a trajectory.
i_t, h_t, y_t	Invoked agent, local context, and generated response.
G, g	Number of workflow rollouts per problem and rollout index.
$\mathcal{R}(\tau), \mathcal{R}_g$	Workflow outcome reward and reward of trajectory g.
a, h	Generated token and token-level conditioning context.
$\widehat{A}_g, \widehat{A}$	Normalized trajectory advantage and token-surrogate advantage.
$\mu_{\mathcal{R}}, \sigma_{\mathcal{R}}$	Group reward mean and standard deviation.
δ	Numerical stabilizer for advantage normalization.
$\omega_\Theta(a \mid h, i)$	Current-to-old token probability ratio.
$\mathcal{J}(\Theta), \mathcal{J}_i$	GRPO objective and agent i's fixed-trajectory terms.
$\mathbb{E}_{\text{old}}$	Expectation over tasks and old-policy workflow rollouts.
$\epsilon_{\text{low}}, \epsilon_{\text{high}}$	Lower and upper policy-ratio clipping offsets.
$\ell, d_{\text{in}}, d_{\text{out}}$	Layer index, input dimension, and output dimension.
$\mathbf{W}_0, \mathbf{W}_0^{(\ell)}$	Frozen pretrained parameters and layer ℓ's weight matrix.
$\mathbf{W}^{(\ell)}, \mathbf{W}_i$	Adapted layer weight and agent i's effective layer weight.
r, γ	Adaptation rank and adapter scaling coefficient.
$\mathbf{A}, \mathbf{B}$	Input/output low-rank factors, frozen and shared in ASPECT.
$\mathbf{U}_r, \mathbf{V}_r, \mathbf{\Sigma}_r$	Leading SVD left/right singular vectors and singular-value matrix.
$\mathbf{R}, \mathbf{R}_i$	Trainable core matrix and agent i's private core.
$\mathbf{R}_0, \mathbf{R}_i^{(0)}$	Common layer-specific initial core and agent i's initial core.
$\mathcal{L}_{\text{cal}}$	Calibration cross-entropy loss on reference-answer tokens.
$\mathcal{T}_y, k$	Scored answer-token positions and token index.
$y_k, y_{<k}$	Reference-answer token k and its preceding tokens.
$p_{\mathbf{W}_0}$	Conditional token probability under the pretrained model.
$\widehat{\mathbf{G}}_{\text{cal}}$	Accumulated calibration gradient.
$\widehat{\Delta\mathbf{W}}_{\text{cal}}$	Estimated task-specific calibration update.
η, η_R	Calibration step size and core optimization learning rate.
P	Trainable parameters per adapted layer.
$P_{\text{workflow}}, P_{\text{role}}$	Trainable parameters per layer for WS and RS.
$P_{\text{agent}}, P_{\text{ASPECT}}$	Trainable parameters per layer for AS and ASPECT.
χ^2, PPL	Policy-mismatch diagnostic and mean response perplexity.
sign, clip	Element-wise sign and clipping operators.
$\nabla, \partial, (\cdot)^\top, \lvert\cdot\rvert$	Gradient, partial derivative, transpose, and set cardinality.

A.2 ASPECT TRAINING PROCEDURE

Algorithm 1 summarizes task-aligned subspace calibration and agent-specific core tuning. We use $\mathbf{W}_0$ for the complete pretrained backbone and $\mathbf{W}_0^{(\ell)}$ for the weight matrix of an adapted layer ℓ. Calibration constructs layer-specific bases shared by all agents and a common core initialization. During reinforcement learning, the backbone and bases remain frozen, while each agent updates only its private cores.

Algorithm 1 Training ASPECT

Require: Pretrained backbone $\mathbf{W}_0$; workflow $\mathcal{W}$ with agents $\mathcal{A}$; training split $\mathcal{D}_{\text{train}}$; rank r; calibration step size η

Select the first 50 examples of the fixed training split as $\mathcal{D}_{\text{cal}}$
for all adapted linear layers ℓ **do**
  $\widehat{\mathbf{G}}_{\text{cal}}^{(\ell)} \leftarrow \sum_{(x,y)\in\mathcal{D}_{\text{cal}}} \nabla_{\mathbf{W}_0^{(\ell)}} \mathcal{L}_{\text{cal}}(x, y; \mathbf{W}_0)$
  $\widehat{\Delta\mathbf{W}}_{\text{cal}}^{(\ell)} \leftarrow -\eta \, \text{sign}(\widehat{\mathbf{G}}_{\text{cal}}^{(\ell)})$
  $(\mathbf{U}_r, \mathbf{\Sigma}_r, \mathbf{V}_r) \leftarrow \text{SVD}_r(\widehat{\Delta\mathbf{W}}_{\text{cal}}^{(\ell)})$
  $\mathbf{B}^{(\ell)} \leftarrow \mathbf{U}_r$; $\mathbf{A}^{(\ell)} \leftarrow \mathbf{V}_r^\top$
  $\mathbf{R}_0^{(\ell)} \leftarrow \mathbf{\Sigma}_r$; $\mathbf{R}_i^{(\ell)} \leftarrow \text{copy}(\mathbf{R}_0^{(\ell)})$ for every $i \in \mathcal{A}$
end for
Freeze $\mathbf{W}_0$ and all shared bases $\{\mathbf{A}^{(\ell)}, \mathbf{B}^{(\ell)}\}$
for $step = 1, \ldots, 200$ **do**
  Synchronize the current agent-specific adaptation parameters with the rollout engines
  Sample 64 problems and $G = 8$ workflow trajectories per problem, using $\phi(i) = i$
  Assign each trajectory's outcome reward $\mathcal{R}(\tau)$ to its trainable responses
  Compute advantages within aligned response groups
  for all agents i with collected responses **do**
    Form agent i's response batch and activate its private cores $\{\mathbf{R}_i^{(\ell)}\}$
    Optimize only these cores with AdamW for one epoch over the sampled responses using the masked, dual-clipped GRPO loss.
  end for
  Update the learning rate
  if $step \bmod 10 = 0$ **then**
    Save a checkpoint
  end if
end for
return Private cores $\{\mathbf{R}_i^{(\ell)}\}$ and shared frozen bases $\{\mathbf{A}^{(\ell)}, \mathbf{B}^{(\ell)}\}$

Calibration uses the answer-token cross-entropy in Eq. (8), with sequences truncated to at most 512 tokens. Gradients are accumulated across calibration examples without updating the pretrained backbone. The sign-based update approximates the first AdamW update, omitting the optimizer's numerical stabilizer and decoupled weight decay.

A.3 GRPO IMPLEMENTATION DETAILS

Under the trajectory-level formulation in Eq. (2), each sampled workflow trajectory receives an outcome reward $\mathcal{R}(\tau)$. For each problem, advantages are computed by normalizing these rewards within the group of sampled workflow trajectories, using the group mean and sample standard deviation with a numerical stabilizer $\delta = 10^{-6}$. All responses used for training within the same trajectory share the resulting normalized advantage. Eq. (4) presents the basic clipped surrogate. Let $f_{\text{clip}} = \min(\omega_\Theta \widehat{A}, \text{clip}(\omega_\Theta, 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})\widehat{A})$ denote its token-level term, where ω_Θ is the probability ratio in Eq. (3). The implementation applies dual clipping to negative-advantage tokens:

$$f_{\text{dual}} = \begin{cases} \max(f_{\text{clip}}, c\widehat{A}), & \widehat{A} < 0, \\ f_{\text{clip}}, & \widehat{A} \geq 0, \end{cases} \qquad c = 3. \tag{16}$$

The training loss is the negative surrogate averaged over valid generated tokens within each response and then over responses in the update batch. Padding tokens are masked out. For numerical stability, the log probability ratio is limited to $[-20, 20]$ before exponentiation. No explicit reference-policy KL penalty or entropy bonus is added.

B Additional Experimental Results

B.1 Complete Training and Validation Curves

Figures 8 and 9 present the training success rate and validation accuracy curves of WS, RS, AS, and ASPECT on Math and Code, respectively, across all three model scales and workflows. These curves cover 200 training steps and complement the peak validation results in Table 1 by showing the learning dynamics of each method. Overall, ASPECT achieves higher validation accuracy and exhibits faster empirical convergence in most configurations.

B.2 Hyperparameter Analysis

Figure 10 compares ranks $r \in \{32, 64, 128\}$ on Qwen3-0.6B Code and learning rates $lr \in \{2 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}\}$ on Qwen3-1.7B Math, both under Eval-Opt. Among the evaluated ranks, $r = 64$ achieves the best training performance and is therefore selected for the main experiments, balancing performance and parameter efficiency. RS and WS train most stably at 2×10^{-5}, whereas ASPECT improves faster during early training at higher learning rates. In the main experiments, we use 2×10^{-5} for WS, RS, and AS, while ASPECT uses 1×10^{-4} or 2×10^{-4}, with detailed configurations in Appendix C. Notably, LoRA-XS and LoRA-SB also use larger learning rates than standard LoRA (Bałazy et al., 2025; Ponkshe et al., 2024), motivating method-specific tuning to account for differences in optimization variables, gradient scales, and effective weight updates. We select method-specific hyperparameters based on the observed training performance and stability.

B.3 Performance Across Workflows

Figure 11 compares workflow-level performance by averaging validation accuracy across Qwen3-0.6B, Qwen3-1.7B, and Qwen3-4B. Base represents the untrained backbone, while trained methods report peak validation accuracy. Within each task, Eval-Opt and Orch-Workers generally achieve higher performance than Voting. All training methods substantially improve upon Base across every task–workflow combination, demonstrating the benefits of post-training for multi-agent workflows.

B.4 Comparison with Full Fine-Tuning

We compare ASPECT with Full Fine-Tuning (Full-FT) on Qwen3-1.7B Math under Eval-Opt and Orch-Workers. WS Full-FT trains one model shared across agents, RS Full-FT trains one model per role, and AS Full-FT trains an independent model per agent. Table 4 reports trainable parameter counts, while Table 5 summarizes peak validation accuracy within 200 training steps. At the LoRA baseline learning rate of 2×10^{-5}, Full-FT exhibits substantial early performance degradation (Figure 12, left). Prior work reports that optimal LoRA learning rates can be approximately an order of magnitude larger than those for Full-FT (Schulman & Thinking Machines Lab, 2025). We therefore adopt 10^{-6}, consistent with the default learning rate in the UnityMAS-O implementation (Chen et al., 2026c). The right panels show improved stability, but Full-FT does not consistently outperform its LoRA counterparts over the observed training interval. These results highlight the importance of method-specific learning-rate selection, while Full-FT incurs substantially greater parameter and memory costs (Chen et al., 2026b). Prior work suggests that task-specific adaptation can have low intrinsic rank, allowing compact low-rank updates to amplify task-relevant features already present in pretrained weights (Hu et al., 2022). This provides a plausible explanation for why parameter-efficient adaptation can match Full-FT on particular downstream tasks without updating the entire backbone. In comparison, ASPECT achieves faster improvements and stronger performance with substantially fewer trainable parameters. These findings suggest that, for specialized tasks, RL can achieve better performance with compact trainable parameterizations rather than the larger parameter sets used by Full-FT or standard LoRA.

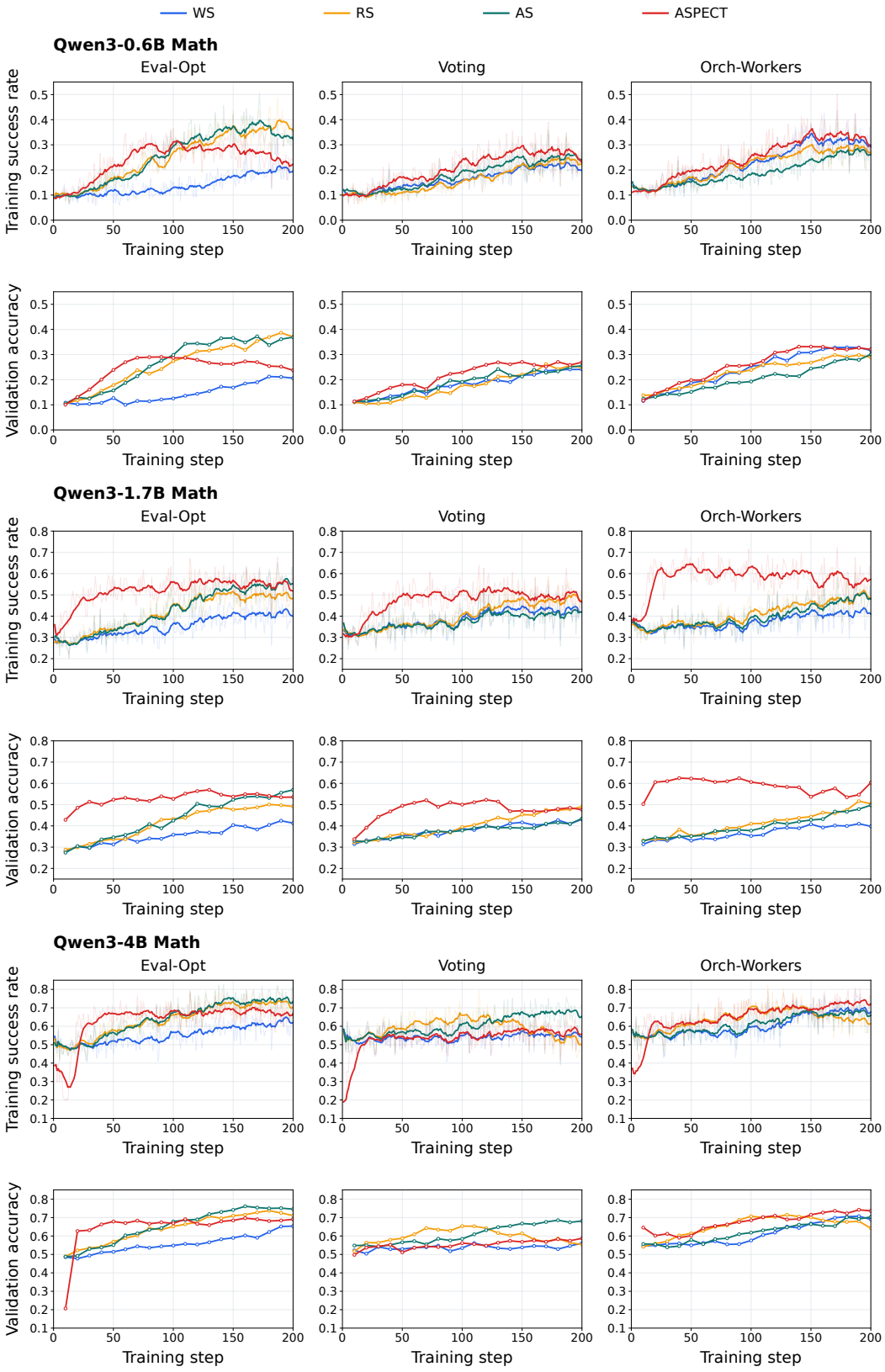


Figure 8: Training and validation curves on Math across model scales and workflows. Columns correspond to workflows. For each model, the upper row shows training success rate and the lower row shows validation accuracy.

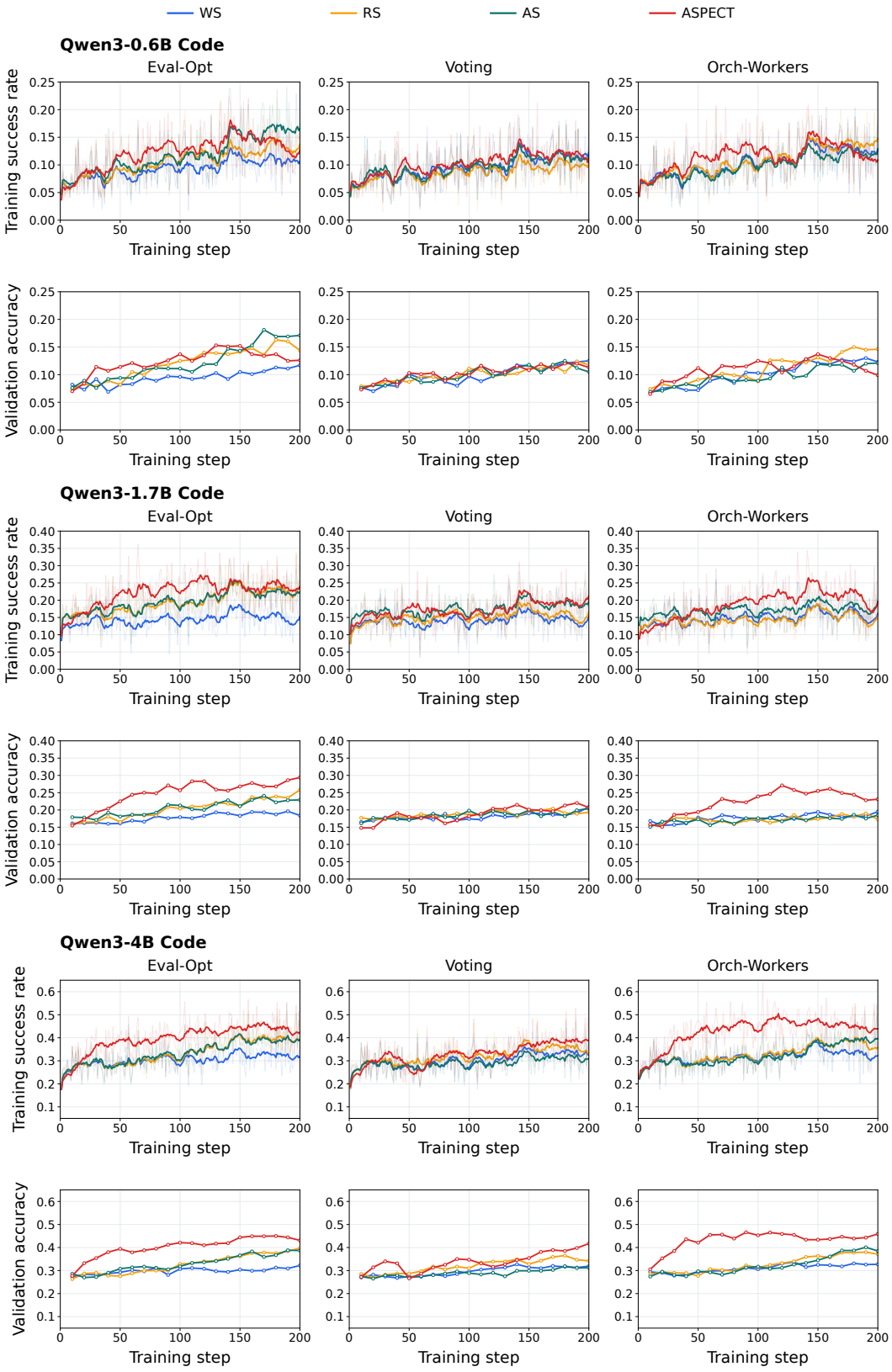


Figure 9: Training and validation curves on Code across model scales and workflows. Columns correspond to workflows. For each model, the upper row shows training success rate and the lower row shows validation accuracy.

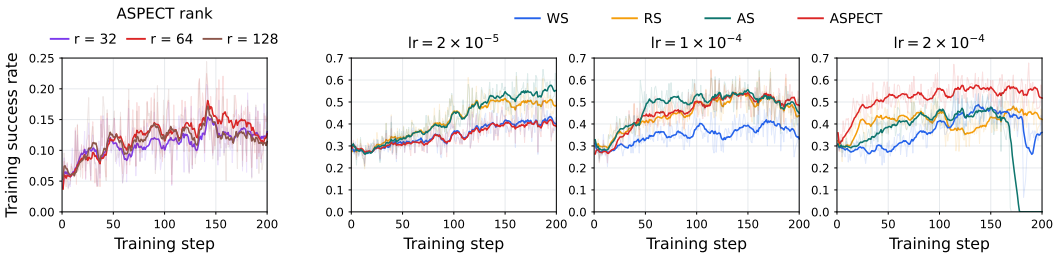


Figure 10: Hyperparameter analysis. **Left:** ASPECT with different ranks on Qwen3-0.6B Code under Eval-Opt. **Right:** Learning-rate comparisons on Qwen3-1.7B Math under Eval-Opt.

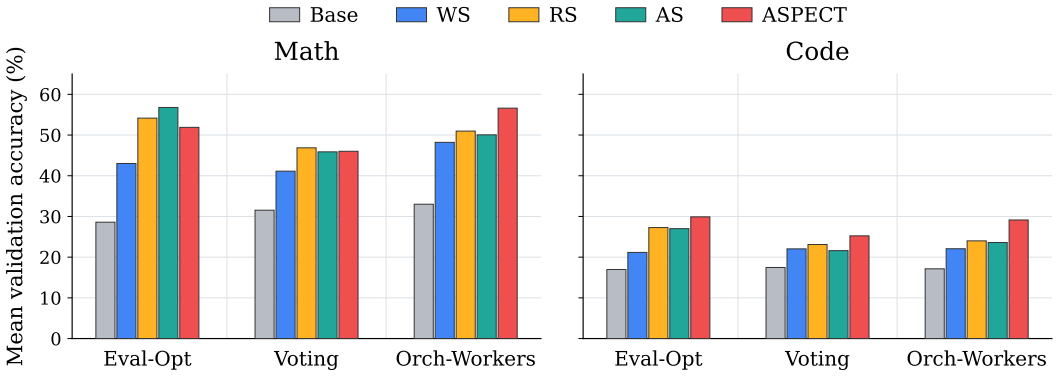


Figure 11: Workflow-level validation accuracy on Math and Code averaged across all model scales.

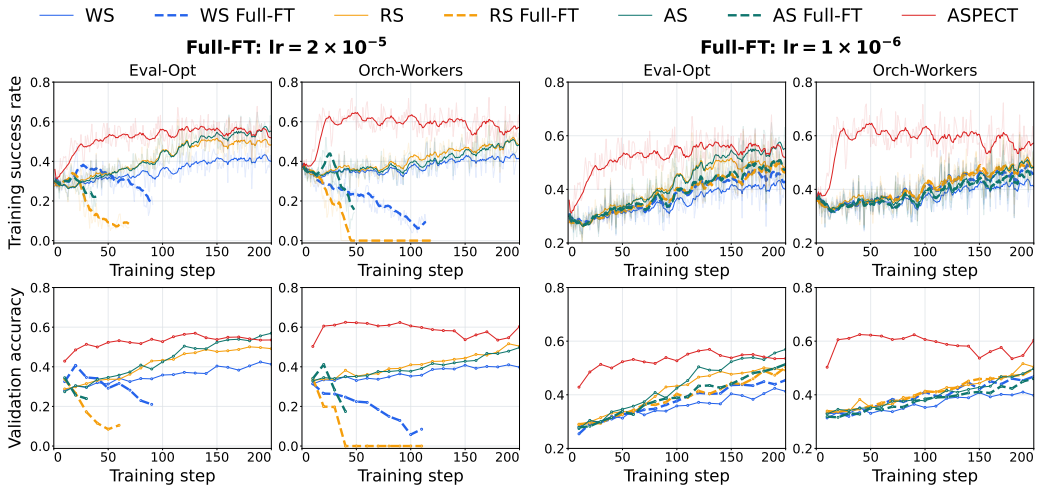


Figure 12: Full fine-tuning comparisons on Qwen3-1.7B Math under Eval-Opt and Orch-Workers. **Left:** Full-FT with learning rate 2×10^{-5}. **Right:** Full-FT with learning rate 10^{-6}. Rows show training success rate and validation accuracy.

Table 4: Trainable parameters (M) for Qwen3-1.7B on Math. Full-FT counts all trainable parameters of each independent model, including the entire backbone.

Model	**Workflow**	**LoRA-FT**			**Full-FT**			**ASPECT**
		WS	RS	AS	WS	RS	AS	
Qwen3-1.7B	Eval-Opt	69.73	139.46	139.46	1720.57	3441.15	3441.15	**1.61**
	Orch-Workers	69.73	209.19	348.65	1720.57	5161.72	8602.87	**4.01**

Table 5: Peak validation accuracy (%) within the first 200 training steps on Qwen3-1.7B Math. Full-FT uses a learning rate of 1×10^{-6}.

Task	Model	Workflow	Base	LoRA-FT			Full-FT			ASPECT
				WS	RS	AS	WS	RS	AS	
Math	Qwen3-1.7B	Eval-Opt	26.70	42.42	50.07	**56.94**	45.47	50.28	51.42	**56.94**
		Orch-Workers	32.15	41.01	51.63	49.72	46.81	49.79	46.03	**62.46**

C Experimental Hyperparameter Configuration

We compare ASPECT with three baselines: workflow-level sharing (WS), role-level sharing (RS), and agent-level separation (AS). WS uses one LoRA adapter for the entire workflow, RS assigns one adapter to each role, and AS assigns an independent adapter to every agent instance. Their parameter mappings are $\phi_{\mathrm{WS}}(i) = 1$, $\phi_{\mathrm{RS}}(i) = \rho(i)$, and $\phi_{\mathrm{AS}}(i) = i$, respectively. Although AS and ASPECT share the same instance-level mapping, AS trains complete LoRA factors, whereas ASPECT trains only private cores within shared frozen bases. All methods freeze the pretrained backbone and adapt the attention and MLP linear projections without trainable biases. Tables 6 and 7 summarize the main-experiment settings.

Table 6: Common main-experiment hyperparameters. Training steps refer to workflow sampling and update iterations.

Hyperparameter	Value
Training steps	200
Problems per step / G	64 / 8
PPO minibatch size / epochs	64 / 1
Dynamic microbatching	Enabled
Nominal microbatch size per GPU	4
Token budget per actor microbatch	40,960 per GPU
Optimizer / (β_1, β_2)	AdamW / $(0.9, 0.999)$
Weight decay / gradient-norm clipping	0.01 / 1.0
LR schedule / warmup	Cosine / 15
$(\epsilon_{\mathrm{low}}, \epsilon_{\mathrm{high}})$	$(0.2, 0.28)$
Dual-clipping coefficient c	3.0
GRPO stabilizer δ	10^{-6}
Loss reduction	Token mean, then response mean
Reference-policy KL penalty	Disabled
Entropy coefficient	0
Sampling temperature	0.7
LoRA dropout	0
Thinking mode	Disabled
Split / shuffle seed	42 / 42
Checkpoint / validation interval	10 steps

Table 7: Method-specific main-experiment settings for Math and Code. Learning rates are the configured peak values.

Hyperparameter	WS	RS	AS	ASPECT
Trainable tensors	$\mathbf{A}, \mathbf{B}$	$\mathbf{A}_{\rho(i)}, \mathbf{B}_{\rho(i)}$	$\mathbf{A}_i, \mathbf{B}_i$	$\mathbf{R}_i$
Rank r	64	64	64	64
Learning rate: Qwen3-0.6B	2×10^{-5}	2×10^{-5}	2×10^{-5}	1×10^{-4}
Learning rate: Qwen3-1.7B / 4B	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-4}

For the main experiments, the calibration step size is $\eta = 10^{-4}/15 \approx 6.67 \times 10^{-6}$ for Qwen3-0.6B and $\eta = 2 \times 10^{-4}/15 \approx 1.33 \times 10^{-5}$ for Qwen3-1.7B and Qwen3-4B, distinct from the core optimization learning rate η_R. Calibration uses 50 training examples with a maximum sequence length of 512 tokens, followed by randomized rank-r SVD with 10 subspace iterations and seed 42.

Dataset splitting and training-data shuffling also use seed 42. The ASPECT (w/o TSC) variant constructs frozen bases from pretrained weights and initializes core entries from a zero-mean Gaussian distribution with standard deviation 10^{-5}, using layer-specific seeds derived from seed 42 and module names. Within each adapted layer, the same initial core is copied to all agents as independent trainable parameters.

The main experiments use rank $r = 64$ for all methods. Each training step samples 64 problems and eight workflow trajectories per problem. We use cosine learning-rate scheduling with warmup and method-specific learning rates listed in Table 7. We use the three workflows defined in Section 3.1. Eval-Opt permits at most three refinement iterations for Math and two for Code. Voting uses three same-role generators and one aggregator. Orch-Workers uses an orchestrator, three available same-role worker instances, and a synthesizer; three workers are invoked according to the generated plan. Math input/response token limits are 30,720/5,120 for Eval-Opt and 20,480/5,120 for Voting and Orch-Workers. Code limits are 10,240/2,048 for Qwen3-0.6B and Qwen3-1.7B, and 20,480/5,120 for Qwen3-4B. DAPO-Math contains 12,704 training and 1,412 validation problems, while DeepCoder contains 13,995 training and 1,000 validation problems. All methods use the same splits. Math outputs are checked against reference answers using the task verifier; Code solutions must pass all retained hidden tests.

D Implementation and Computational Resources

Experimental Coverage. The main experiments comprise 72 configurations spanning three model scales (Qwen3-0.6B, Qwen3-1.7B, and Qwen3-4B), two tasks (Math and Code), three workflows (Eval-Opt, Voting, and Orch-Workers), and four methods (WS, RS, AS, and ASPECT). Each configuration runs for 200 training steps with validation on the complete split every 10 steps, yielding 20 evaluation points per configuration and 1,440 in total.

Hardware. The experiments were primarily executed on computing nodes equipped with NVIDIA L20X GPUs (140.4 GiB VRAM per GPU) and Intel(R) Xeon(R) Platinum 8558 processors, with four GPUs allocated to each experimental configuration.

Software. We use Python 3.10.16, PyTorch 2.7.1 with CUDA 12.6, Transformers 4.57.0, and PEFT 0.19.1, with vLLM 0.10.0 for rollout generation and Ray 2.55.1 for distributed execution.

Computational Cost. The main experimental campaign consumed approximately 12,846 GPU-hours, measured as aggregate GPU time rather than elapsed wall-clock time. It is estimated that one configuration requires approximately 44 hours to complete 200 training steps. Rollout generation, workflow execution, and weight synchronization account for approximately 51% of the runtime, parameter updates for 35%, old-policy log-probability computation for 12%, and advantage computation, checkpoint saving, and other overhead for the remaining 2%.

Task-Aligned Subspace Calibration. ASPECT performs a one-time calibration stage using 50 examples with a maximum sequence length of 512 tokens. The recorded calibration computation time averaged 5.08 seconds per initialization (range: 4.00–7.09 seconds) using eight GPUs. This measurement includes gradient estimation, aggregation, and rank-64 randomized SVD, but excludes model loading, initial buffer allocation, and final artifact publication. The resulting subspace basis is cached and reused across compatible configurations.

RS versus AS in Eval-Opt. RS and AS induce the same parameter partition in Eval-Opt, with one parameter set for the generator and one for the evaluator. At the code implementation level, their LoRA implementations differ in rollout dispatch and batching. These differences, together with asynchronous execution and stochastic sampling across separate runs, can lead to different training trajectories and validation accuracies. This provides an explanation for the differences between RS and AS on Eval-Opt in Table 1, despite their identical parameter allocation.

E Theoretical Properties of ASPECT

We establish three properties of ASPECT, extending the fixed-basis analysis of LoRA-SB (Ponkshe et al., 2024) to multi-agent optimization. These results characterize the parameterization rather than guarantee convergence or superior performance.

Setup. Consider one adapted layer per agent, holding all other layers fixed. The results extend blockwise to multiple adapted layers. Let $N = |\mathcal{A}|$, $m = d_{\text{out}}$, $n = d_{\text{in}}$, and $\gamma > 0$. The frozen bases satisfy $\mathbf{B}^\top\mathbf{B} = \mathbf{A}\mathbf{A}^\top = \mathbf{I}_r$, with $\mathbf{B} \in \mathbb{R}^{m\times r}$ and $\mathbf{A} \in \mathbb{R}^{r\times n}$. Define

$$\Phi_i(\mathbf{R}_i) = \mathbf{W}_0 + \gamma\mathbf{B}\mathbf{R}_i\mathbf{A}, \qquad \mathcal{S} = \{\mathbf{B}\mathbf{Z}\mathbf{A} : \mathbf{Z} \in \mathbb{R}^{r\times r}\}. \tag{17}$$

Write $\boldsymbol{R} = (\mathbf{R}_i)_{i\in\mathcal{A}}$, $\boldsymbol{W} = (\mathbf{W}_i)_{i\in\mathcal{A}}$, and $\Phi(\boldsymbol{R}) = (\Phi_i(\mathbf{R}_i))_{i\in\mathcal{A}}$. Given its context, each agent's policy depends only on its own effective weights. The workflow and reward function are otherwise unchanged.

Proposition E.1 (Objective preservation within the shared subspace)**.** *Let $F(\boldsymbol{W}) = \mathbb{E}_{x\sim\mathcal{D},\,\tau\sim p_{\boldsymbol{W}}(\cdot|x)}[\mathcal{R}(\tau)]$ be the expected workflow reward, assumed finite. Under the ASPECT parameterization, the expected workflow reward is $F_R = F \circ \Phi$, and*

$$\sup_{\boldsymbol{R}} F_R(\boldsymbol{R}) = \sup_{\boldsymbol{W}\in(\mathbf{W}_0+\mathcal{S})^N} F(\boldsymbol{W}) \le \sup_{\boldsymbol{W}\in(\mathbb{R}^{m\times n})^N} F(\boldsymbol{W}). \tag{18}$$

The same composition identity holds for a fixed-batch GRPO surrogate when sampled contexts, old probabilities, advantages, and reduction weights are kept fixed.

Proof. For each agent and each fixed context, the core parameterization and its corresponding effective weights define identical conditional output probabilities. Hence the induced workflow distribution and expected reward are identical. Each Φ_i maps bijectively onto $\mathbf{W}_0 + \mathcal{S}$, with inverse $\gamma^{-1}\mathbf{B}^\top(\mathbf{W}_i - \mathbf{W}_0)\mathbf{A}^\top$. This proves the equality of restricted suprema; set inclusion gives the inequality. For a fixed sampled batch, identical token probabilities also preserve every probability ratio and clipped surrogate term, proving the corresponding GRPO identity. □

Thus, ASPECT changes the feasible parameter space, not the reward or surrogate definition. This does not imply equality with the unrestricted optimal return, nor does it equate the GRPO surrogate with the expected reward.

Proposition E.2 (Gradient isolation for a fixed sampled batch)**.** *Fix the sampled contexts, old probabilities, advantages, and reduction weights. Let $\widehat{\mathcal{J}}_{i,\mathcal{B}}$ collect the GRPO terms generated by agent i in batch $\mathcal{B}$, with no additional parameter-dependent cross-agent regularizer. Then*

$$\widehat{\mathcal{J}}_{R,\mathcal{B}}(\boldsymbol{R}) = \sum_{i\in\mathcal{A}} \widehat{\mathcal{J}}_{i,\mathcal{B}}(\mathbf{R}_i), \qquad \frac{\partial\widehat{\mathcal{J}}_{i,\mathcal{B}}}{\partial\mathbf{R}_j} = 0 \quad (j \ne i) \tag{19}$$

where the derivatives exist.

Proof. For a response from agent i, its recomputed probabilities depend only on $\Phi_i(\mathbf{R}_i)$ because the input context is fixed. The remaining quantities in its surrogate terms are constants. Neither ordinary clipping nor dual clipping introduces another agent's parameters. Grouping terms by their generating agent yields the decomposition and the zero cross-agent derivatives. At clipping boundaries, ordinary differentiability is not guaranteed, but the parameter dependence remains separated. □

This is a conditional property of the sampled surrogate. Agents remain statistically coupled through the workflow distribution and shared rewards; the expected reward need not decompose across agents. The statement does not preclude optimizer-level coupling through global gradient clipping.

Proposition E.3 (Core gradients and orthogonal projection)**.** *For any differentiable scalar objective $H(\boldsymbol{W})$, let $\mathbf{G}_i = \nabla_{\mathbf{W}_i} H$ at $\boldsymbol{W} = \Phi(\boldsymbol{R})$. Then*

$$\nabla_{\mathbf{R}_i}(H \circ \Phi) = \gamma\mathbf{B}^\top\mathbf{G}_i\mathbf{A}^\top, \qquad \mathcal{P}_{\mathcal{S}}(\mathbf{G}_i) = \mathbf{B}\mathbf{B}^\top\mathbf{G}_i\mathbf{A}^\top\mathbf{A}, \tag{20}$$

where $\mathcal{P}_{\mathcal{S}}(\mathbf{G}_i)$ uniquely minimizes $\|\mathbf{G}_i - \mathbf{D}\|_F^2$ over $\mathbf{D} \in \mathcal{S}$. A plain core gradient-ascent step, $\Delta\mathbf{R}_i = \eta_R\nabla_{\mathbf{R}_i}(H \circ \Phi)$, induces

$$\Delta\mathbf{W}_i = \eta_R\gamma^2\mathcal{P}_{\mathcal{S}}(\mathbf{G}_i). \tag{21}$$

Proof. Using the Frobenius inner product and $\mathrm{d}\mathbf{W}_i = \gamma\mathbf{B}(\mathrm{d}\mathbf{R}_i)\mathbf{A}$ gives

$$\mathrm{d}H = \sum_i \langle\mathbf{G}_i, \gamma\mathbf{B}(\mathrm{d}\mathbf{R}_i)\mathbf{A}\rangle_F = \sum_i \langle\gamma\mathbf{B}^\top\mathbf{G}_i\mathbf{A}^\top, \mathrm{d}\mathbf{R}_i\rangle_F. \tag{22}$$

This proves the chain-rule identity. Set $\mathbf{P}_i = \mathbf{B}\mathbf{B}^\top \mathbf{G}_i \mathbf{A}^\top \mathbf{A} \in \mathcal{S}$. Orthonormality implies $\mathbf{B}^\top(\mathbf{G}_i - \mathbf{P}_i)\mathbf{A}^\top = 0$, so the residual is orthogonal to every matrix in $\mathcal{S}$. Consequently, for every $\mathbf{D} \in \mathcal{S}$,

$$\|\mathbf{G}_i - \mathbf{D}\|_F^2 = \|\mathbf{G}_i - \mathbf{P}_i\|_F^2 + \|\mathbf{P}_i - \mathbf{D}\|_F^2, \tag{23}$$

which shows that $\mathbf{P}_i$ is the unique minimizer. Finally, substituting $\Delta\mathbf{R}_i = \eta_R \nabla_{\mathbf{R}_i}(H \circ \Phi)$ into $\Delta\mathbf{W}_i = \gamma \mathbf{B}\Delta\mathbf{R}_i\mathbf{A}$ proves Eq. (21). □

The approximation is optimal within the fixed subspace $\mathcal{S}$, not over all matrices of rank at most r. Eq. (21) describes an unpreconditioned gradient step, not the full AdamW update used in experiments. For any optimizer, however, $\Delta\mathbf{W}_i = \gamma\mathbf{B}\Delta\mathbf{R}_i\mathbf{A} \in \mathcal{S}$ remains exact. These identities explain how agents can share adaptation directions while updating disjoint coordinates, without asserting monotonic reward improvement or equivalence to unrestricted Full-FT.