

CoopAgent: Training Multi-Agent Cooperation from Single-Agent Execution Trajectories

Zhengzhe Pan¹, Linchao Zhu^{1,†}, Yi Yang¹

¹ReLER, CCAI, Zhejiang University

[†]Corresponding author

Abstract

LLM-based multi-agent systems have demonstrated strong potential for solving complex tasks, yet reliable collaboration remains challenging. While most training-based methods rely on multi-agent trajectories, we investigate whether cooperative behaviors can be strengthened using only single-agent execution data. We propose **CoopAgent**, a training framework that learns solely from single-agent execution trajectories and transfers to multi-agent deployment. To bridge the gap between single-agent training and multi-agent environments, we introduce **PADPO** (Perturbation-Aligned DPO), which aligns action preferences between original observations and task-related perturbations that mimic changes caused by other agents. We train CoopAgent on single-agent execution trajectories collected in Minecraft and evaluate it on VillagerBench. The results demonstrate its potential to enhance collaboration on VillagerBench while preserving strong single-agent execution ability. We further assess CoopAgent on out-of-domain benchmarks, showing improvements in several multi-agent settings and performance comparable to the multi-agent training method AT-GRPO on the coding benchmarks. Our code is available at <https://github.com/hlpzz/CoopAgent>.

1 Introduction

In recent years, LLM-based agents have made significant progress in reasoning, planning, and tool use (Yao et al., 2023; Schick et al., 2023). Among them, LLM-based multi-agent systems (MAS) have demonstrated strong potential for solving complex tasks in open-world and long-horizon settings, with empirical successes reported across domains such as software development (Qian et al., 2024; Khanzadeh, 2025), policy simulation (Wang et al., 2025), and games (Dong et al., 2024; Chai et al., 2025). Despite this progress, effective collaboration in MAS does not arise automatically. Prior

work identifies coordination breakdowns as a major source of failure in MAS (Cemri et al., 2025), making the enhancement of collaborative capabilities a central challenge for unlocking their full potential.

To address this challenge, prior studies have explored training-based methods for improving coordination in MAS (Liang et al., 2025; Subramaniam et al., 2025; Zhao et al., 2026). While these approaches have demonstrated promising results, most of them rely on multi-agent interaction data to improve coordination. However, collecting such data typically requires coordinating simultaneous interactions and can introduce additional complexity in credit assignment.

Single-agent execution data can be collected without requiring simultaneous interaction among multiple agents, enabling more flexible and scalable data generation. Moreover, single-agent trajectories exhibit clear and well-defined relationships between actions and outcomes, providing more direct supervision signals by sidestepping inter-agent credit assignment ambiguity during training. These properties naturally raise the following question: *can multi-agent cooperation be improved by training on single-agent execution data alone, without relying on explicit multi-agent interaction data?*

Our key insight is that, although single-agent trajectories lack explicit inter-agent interactions, they can still be used to approximate the uncertainty that arises in multi-agent execution by introducing controlled perturbations to environment observations. Unlike random perturbations mainly used for robustness, these perturbations are task-related and are designed to approximate multi-agent environment dynamics. This encourages the model to preserve action preferences under coordination-relevant observation changes, which is important for effective multi-agent cooperation. Figure 1 illustrates how single-agent observations are perturbed to simulate multi-agent-induced variability.

Building on this insight, we propose **CoopA-**

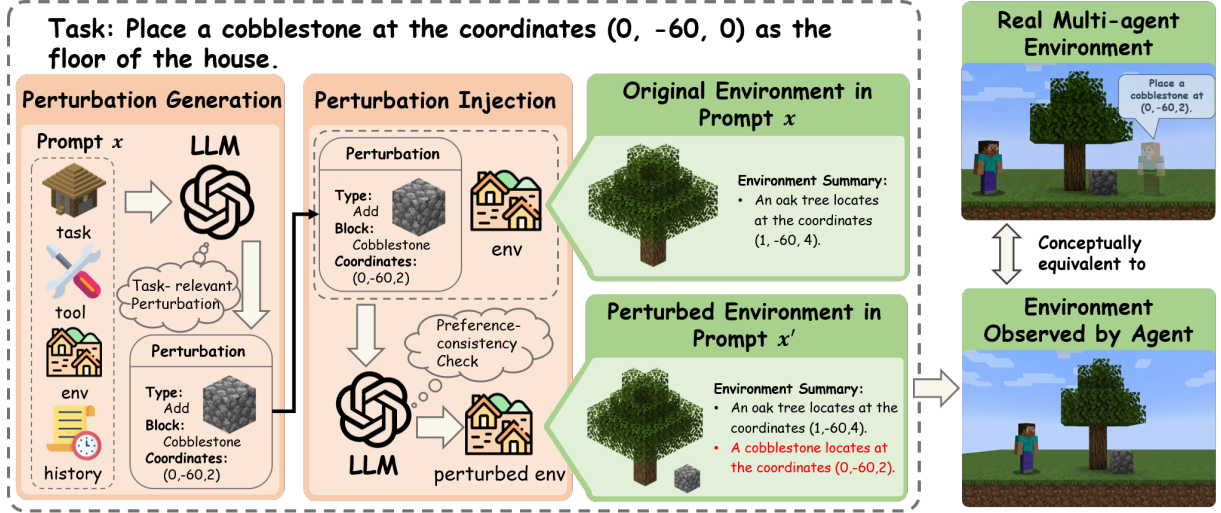


Figure 1: Single-agent perturbation augmentation approximates multi-agent-induced observation changes. The faded agent denotes another hypothetical agent whose action changes the shared environment and thus the current agent’s observation; this effect is approximated by the task-related difference between the original and perturbed observations.

gent, a training-based framework for inducing cooperative behavior from single-agent execution data. We construct single-agent tasks in Minecraft and collect execution trajectories with clear action-outcome feedback. CoopAgent first performs supervised fine-tuning on these trajectories to learn basic agent capabilities. Then it applies preference optimization to make action choices more stable under coordination-relevant observation changes. For this second stage, we introduce **PADPO**, a perturbation-aligned variant of Direct Preference Optimization (DPO) (Rafailov et al., 2023), which regularizes action preferences to remain consistent under task-related perturbations that approximate environment changes caused by other agents.

We evaluate CoopAgent on Minecraft-based VillagerBench (Dong et al., 2024). Experimental results show that CoopAgent consistently improves collaboration-oriented metrics on VillagerBench while preserving strong single-agent execution capability. Beyond Minecraft, we further evaluate CoopAgent on Overcooked-AI (Carroll et al., 2019) and coding benchmarks, including CodeContests (Li et al., 2022), APPS (Hendrycks et al., 2021), and LiveCodeBench (White et al., 2025a). CoopAgent achieves the best average score on Overcooked-AI with gains on coordination-intensive layouts. On coding benchmarks, it performs comparably to AT-GRPO (Zhao et al., 2026), a multi-agent training method that relies on multi-agent interaction data.

Our main contributions are as follows:

- We propose **CoopAgent**, a training-based framework that enables multi-agent collaboration to be effectively learned from single-agent execution data alone.
- We introduce **PADPO**, a perturbation-aligned preference optimization objective that enforces preference consistency under task-related environment perturbations designed to simulate multi-agent environment changes.
- We demonstrate the effectiveness of CoopAgent across single-agent tasks, multi-agent benchmarks, and out-of-domain evaluations, showing robust single-to-multi-agent transfer and competitive performance against AT-GRPO on the evaluated coding benchmarks.

2 Related Work

Minecraft Agents. Minecraft has become a widely used testbed for agent research due to its rich and controllable environment. Prior work (Hafner et al., 2024; Liu et al., 2025; Fu et al., 2025; Lee et al., 2026) focuses on single-agent long-term planning in Minecraft. XENON (Lee et al., 2026) robustly corrects flawed planning knowledge through experience-driven algorithmic memory updates. Other works (Dong et al., 2024; White et al., 2025b; Chai et al., 2025) shift attention to multi-agent collaboration. VillagerAgent (Dong et al., 2024) decomposes tasks into DAGs to strengthen collaboration on complex de-

dependencies. In comparison, CoopAgent improves collaboration through training alone, without modifying the underlying agent framework.

Multi-Agent Frameworks and Benchmarks. MAS have shown strong potential in solving complex tasks, and recent work (Dang et al., 2025; Becker et al., 2025; Qian et al., 2025; Yun et al., 2026) has increasingly focused on general frameworks for effective collaboration among agents. GoA (Yun et al., 2026) optimizes multi-agent collaboration via graph-based sampling and bidirectional message passing. To better evaluate communication efficiency and collaboration across different systems, a number of multi-agent benchmarks have been proposed (Chen et al., 2024; Wang et al., 2024; Zhu et al., 2025). MultiAgentBench (Zhu et al., 2025) evaluates collaboration and competition using task-level metrics. Our CoopAgent uses VillagerAgent as the execution framework and includes VillagerBench (Dong et al., 2024) as an evaluation testbed.

Training and Fine-Tuning For Multi-Agent Systems. Several recent studies (Motwani et al., 2025; Liang et al., 2025; Subramaniam et al., 2025; Liu et al., 2026; Ma et al., 2024; Zhao et al., 2026) have begun to explore training or fine-tuning approaches for internalizing collaborative capabilities in MAS. MALT (Motwani et al., 2025) introduces role-level credit assignment for multi-agent training. AT-GRPO (Zhao et al., 2026) utilizes tree-structured sampling to create valid comparison groups for multi-agent on-policy RL. These approaches typically rely on explicitly collected multi-agent interaction data to improve coordination. In contrast, CoopAgent focuses on exploring how collaborative capabilities can be acquired without multi-agent training data, relying solely on single-agent datasets.

3 Method

CoopAgent follows a two-stage training pipeline as shown in Figure 2. In the first stage, we perform supervised fine-tuning (SFT) on single-agent trajectories to equip the model with basic agent capabilities for stable action execution. In the second stage, we further construct preference-based training signals and apply PADPO to improve robustness under observation perturbations and thereby induce stronger collaborative behavior in multi-agent deployment. We begin by describing the data collection setup

that underpins both training stages.

3.1 Data Collection Setup

We collect training data in a single-agent setting based on the VillagerBench environment (Dong et al., 2024). In this setting, we design atomic tasks and collect execution trajectories, used for supervised fine-tuning (Stage 1) and for deriving preference pairs in preference-based training (Stage 2).

Single-Agent Tasks. To enable scalable data collection, we extend VillagerBench with a set of single-agent atomic tasks. These tasks focus on evaluating and training an agent’s ability to reliably execute individual actions, rather than solving long-horizon or compositional objectives. In total, we introduce 6 high-level task categories and 25 fine-grained atomic task types, covering essential action primitives commonly involved in complex behaviors. Task instances are generated with fixed sampling ratios by jointly considering task difficulty and occurrence frequency. Details about single-agent task types can be found in Appendix A.

Data Collection Process. For each instantiated single-agent task, we run an agent based on an existing model, such as Qwen3-235B-A22B (Yang et al., 2025a), to execute the task and record the full interaction trajectory, including the environment feedback observed by the agent and the actions it produces at each step. Task completion is determined using predefined rule-based criteria, yielding a binary success or failure signal.

Trajectory Notation. From the data collection process, we obtain execution trajectories, each corresponding to a single agent performing a task. We denote a single execution trajectory as T .

At each step of the trajectory, the agent receives an input prompt x and outputs an action y . The input prompt x consists of four components: a task description d , an environment observation o , an available tool set $\mathcal{U}$, and the interaction history h up to the current step. Formally, we write

$$x = (d, o, \mathcal{U}, h). \quad (1)$$

Each action y corresponds to invoking a tool $u \in \mathcal{U}$ and yields an execution outcome (success or failure), which is appended to the interaction history for subsequent steps.

The interaction at each step is therefore represented as a tuple (x, y) . An execution trajectory of

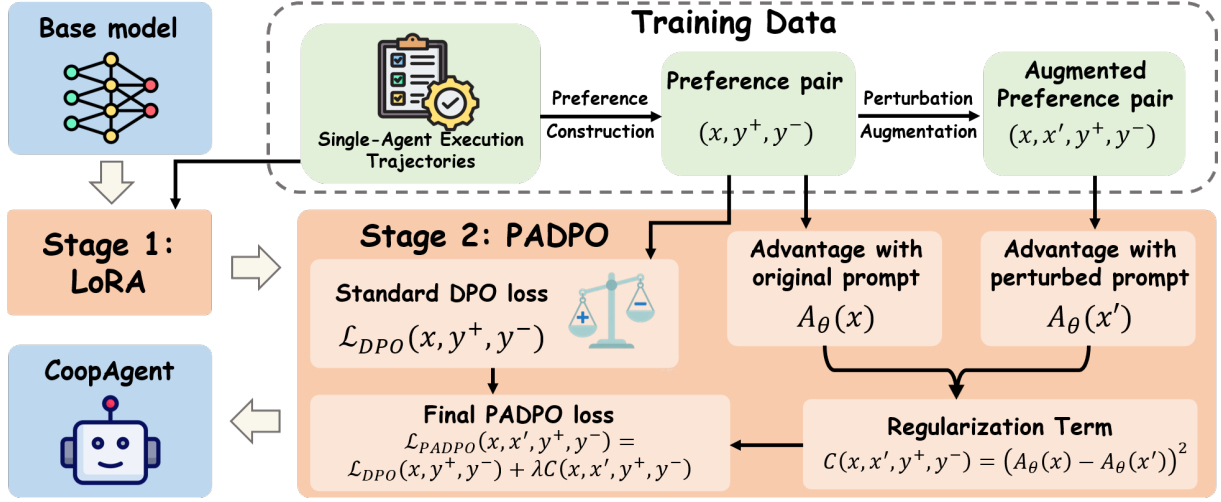


Figure 2: CoopAgent training pipeline. Thick light-colored arrows indicate the model training stages, while thin black arrows represent the data flow between components.

n steps can therefore be written as

$$T = ((x_t, y_t))_{t=1}^n. \quad (2)$$

3.2 Supervised Fine-Tuning

In the first training stage, we apply parameter-efficient fine-tuning using LoRA (Hu et al., 2022) on execution trajectories collected in the single-agent setting. This stage equips the agent with stable atomic action execution and instruction-following capabilities.

LoRA Training. We perform LoRA-based supervised fine-tuning on the collected single-agent trajectories, treating each step-level pair (x, y) as a training example. The resulting LoRA-adapted model serves as the initialization for subsequent PADPO training. Details about training configurations can be found in Appendix D.1.

3.3 Preference Optimization with PADPO

On top of the LoRA adapter obtained in the first training stage, we further perform preference-based optimization to improve robustness under observation perturbations, inspired by consistency regularization methods in semi-supervised learning (Sajjadi et al., 2016). This stage is designed to induce behaviors that are more stable when deployed in multi-agent settings, while still relying only on single-agent execution data.

Preference Construction. To apply PADPO, we first derive action-level preference supervision from the collected single-agent execution trajectories. We construct preference pairs from the

clear and verifiable action outcomes of atomic tasks. When an action either fails to execute or is irrelevant to the task objective, we treat it as a negative action y^- . We then define the corresponding positive action y^+ as the first subsequent action in the same trajectory that executes successfully and is relevant to the task objective. Let x denote the input prompt at the step where y^- is taken, and we form a preference pair (x, y^+, y^-) accordingly. This construction relies on the observation that failed or task-irrelevant actions do not induce task-relevant changes in the environment state; therefore, replacing such an action with a subsequent successful task-relevant action does not introduce a task-relevant environment discrepancy for preference supervision. Detailed preference extraction rules are provided in Appendix C.

Perturbation Augmentation. Since the preference pairs are extracted from single-agent trajectories, we introduce controlled perturbations to the environment to approximate the observation variability induced by other agents during multi-agent deployment, as illustrated in Figure 1.

Given an original prompt $x = (d, o, \mathcal{U}, h)$, we construct a perturbed prompt x' by modifying only the environment observation component, while keeping other components unchanged:

$$x' = (d, o', \mathcal{U}, h). \quad (3)$$

The perturbation is designed to be local and semantically mild, such that it does not change the underlying task semantics in practice.

Perturbation augmentation is implemented as

a two-step process: perturbation generation and perturbation injection. Specifically, we prompt an LLM to analyze the task description together with the environment observation in x and generate a task-relevant perturbation configuration, which specifies the perturbation operation (ADD or REMOVE), a task-relevant block type, and a target coordinate in the vicinity of the described environment. The same LLM then applies the generated configuration to modify the environment observation and assesses whether the preference ordering between y^+ and y^- remains unchanged under both x and the resulting x' . We additionally apply rule-based checks to ensure that the perturbation preserves the validity of the preference pair and does not modify other components of the input x . If the generated perturbation violates these constraints, the perturbation configuration is regenerated until a valid perturbation is obtained. Only perturbations that preserve preference consistency are retained for training. Details of the prompt template are provided in Appendix G.1.

In VillagerAgent, high-level coordination issues such as resource conflicts are typically reflected in the observation as changes in local block configurations. Therefore, adding or removing task-relevant blocks provides a simple approximation of the environment changes induced by other agents. This approximation captures the local observation consequence of another agent’s action rather than the complete multi-agent interaction process. It exposes the model to coordination-relevant state changes without requiring explicit multi-agent roll-outs.

PADPO Objective. Given the paired prompts (x, x') from perturbation augmentation and the same preference supervision (y^+, y^-) , we incorporate observation perturbations into preference optimization by enforcing consistent preference advantages under both inputs. We build on DPO (Rafailov et al., 2023), which optimizes a policy from preference pairs without explicit reward modeling. Let π_θ denote the trainable policy and π_{ref} a fixed reference policy. For a preference pair (x, y^+, y^-) , we define the action-level log-probability advantage as:

$$A_\theta(x) = \log \pi_\theta(y^+ | x) - \log \pi_\theta(y^- | x). \quad (4)$$

The standard DPO loss is defined as:

$$\mathcal{L}_{\text{DPO}}(x, y^+, y^-) = -\log \sigma\left(\beta[A_\theta(x) - A_{\text{ref}}(x)]\right), \quad (5)$$

where A_{ref} denotes the advantage under π_{ref} , $\sigma(\cdot)$ is the logistic sigmoid and $\beta > 0$ controls the strength of preference optimization.

To encourage preference consistency under observation perturbations, we introduce a perturbation-aligned regularization term. For each original prompt x and its perturbed counterpart x' , the regularizer is defined as:

$$C(x, x', y^+, y^-) = (A_\theta(x) - A_\theta(x'))^2, \quad (6)$$

where both advantages are computed using the same preference pair (y^+, y^-) . We adopt a squared error penalty between the advantage values under the original and perturbed observations for two reasons. First, since the perturbation between x and x' is symmetric, we require a regularizer that penalizes preference deviations in both directions, which is naturally achieved by a squared distance. Second, the squared error penalizes larger deviations more strongly, discouraging significant preference shifts induced by observation perturbations.

The final PADPO objective is given by:

$$\mathcal{L}_{\text{PADPO}}(x, x', y^+, y^-) = \mathcal{L}_{\text{DPO}}(x, y^+, y^-) + \lambda C(x, x', y^+, y^-), \quad (7)$$

where $\lambda \geq 0$ controls the strength of the perturbation-alignment regularization. By enforcing preference consistency between x and x' , PADPO encourages robust decision-making under observation perturbations, which leads to more stable collaboration in multi-agent deployment.

4 Experiments

4.1 Experimental Setup and Baselines

We evaluate CoopAgent on VillagerBench within the VillagerAgent framework (Dong et al., 2024), covering single-agent tasks as well as multi-agent Construction and Farming tasks. In VillagerAgent, the *Decomposer* divides each task into subtasks, which the *Controller* assigns to *base agents* operating in parallel in a shared Minecraft environment. Only the models of *base agents* vary across methods; the three-agent setting adds a base agent with the same role and interface to the default two-agent setup, while all other framework components remain unchanged. Detailed environment and evaluation settings are provided in Appendix E.1. We train CoopAgent variants based on Qwen2.5-7B-Instruct (Yang et al., 2025b) and Qwen3-8B (Yang

Model	Single-Agent	Construction (VillagerBench)			Farming (VillagerBench)			
	S (%)	C (%)	VHR	E (%/min)	C (%)	ACR	WACR	E (%/min)
Qwen2.5-7B	51	44.60	51.93	10.62	37.33	42.65	29.26	13.41
Qwen2.5-7B (LoRA)	79	43.00	53.93	13.59	49.50	35.17	27.79	19.20
Qwen3-8B	50	42.73	53.10	16.24	43.83	35.00	35.53	20.88
Qwen3-80B-A3B	87	57.07	62.40	15.33	44.50	43.32	40.60	14.33
Qwen3-235B-A22B	85	48.77	60.03	17.62	48.50	37.94	32.16	24.45
Qwen2.5-7B (CoopAgent)	87	66.90	74.27	26.33	50.00	48.73	44.89	20.06
Qwen3-8B (CoopAgent)	82	61.30	71.57	19.60	52.50	61.62	56.74	22.08

Table 1: Main results on single-agent tasks and multi-agent benchmarks in VillagerBench.

Model	C (%)	ACR	WACR	E (%/min)
Qwen2.5-7B (2 agents)	37.33	42.65	29.26	13.41
Qwen2.5-7B (3 agents)	40.17	34.67	25.07	14.22
CoopAgent (2 agents)	50.00	48.73	44.89	20.06
CoopAgent (3 agents)	52.67	48.51	46.57	22.63

Table 2: Effect of the number of agents on the Farming benchmark in VillagerBench.

et al., 2025a), with training details provided in Appendix D. Unless otherwise specified, all training-based variants use Qwen2.5-7B-Instruct as the base model.

Baselines. We compare CoopAgent with the original Qwen2.5-7B-Instruct model and a LoRA-only SFT variant trained on the same single-agent execution trajectories without preference optimization, denoted as Qwen2.5-7B (LoRA). We also evaluate several Qwen3-series models (Yang et al., 2025a), including Qwen3-8B, Qwen3-235B-A22B, and Qwen3-Next-80B-A3B-Instruct, to compare CoopAgent’s performance with more recent and larger-scale Qwen-series models. All Qwen3-series models are evaluated in no-thinking mode. Hereafter, we refer to Qwen2.5-7B-Instruct as Qwen2.5-7B and Qwen3-Next-80B-A3B-Instruct as Qwen3-80B-A3B.

Evaluation Metrics. For single-agent tasks, we report Pass@2 success rate (S). For multi-agent Construction tasks, we report completion rate (C), view hit rate (VHR), and efficiency (E), which respectively reflect task success, view-level construction quality, and task completion efficiency. VHR measures the proportion of target construction content correctly matched from predefined evaluation views, with higher VHR indicating better construction quality. For Farming tasks, we report completion rate (C), agent contribution rate (ACR), weighted agent contribution rate (WACR), and ef-

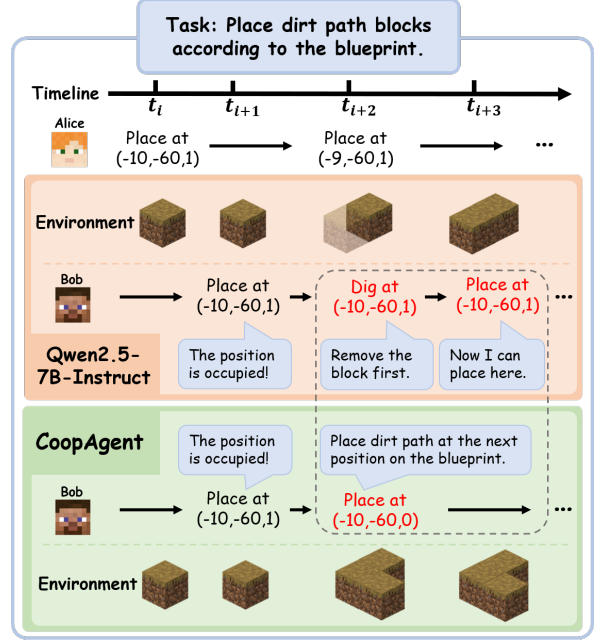


Figure 3: Trajectory-level case study from a construction task. Given the same action by Alice, Qwen2.5-7B-Instruct makes Bob rework an already completed position, while CoopAgent makes Bob proceed to the next blueprint position.

ficiency (E). ACR measures how task-relevant contributions are distributed across agents, while WACR further weights ACR by task completion to reduce the influence of low-completion tasks and better reflect stable collaboration across tasks. A higher ACR or WACR indicates stronger collaboration in terms of more balanced task-relevant contributions across agents. All VillagerBench multi-agent results are reported as avg@3. Detailed environment settings and metric definitions are provided in Appendix E.1.

4.2 Evaluation Results

Single-Agent Performance. As shown in Table 1, CoopAgent maintains strong single-agent execution capability across different backbones.

Layout	Qwen2.5-7B	Qwen2.5-7B (LoRA)	Tool-Light (Chen et al., 2025)	Qwen2.5-7B (CoopAgent)
Cramped Room	56 $\pm$ 35.8	48 $\pm$ 11.0	32 $\pm$ 17.9	60 $\pm$ 14.1
Asymmetric Advantages	76 $\pm$ 8.9	32 $\pm$ 17.9	116 $\pm$ 8.9	44 $\pm$ 8.9
Coordination Ring	0 $\pm$ 0.0	12 $\pm$ 11.0	4 $\pm$ 8.9	24 $\pm$ 16.7
Forced Coordination	0 $\pm$ 0.0	8 $\pm$ 11.0	28 $\pm$ 11.0	36 $\pm$ 21.9
Counter Circuit	56 $\pm$ 8.9	52 $\pm$ 30.3	48 $\pm$ 26.8	68 $\pm$ 11.0
Average	37.6 $\pm$ 35.3	30.4 $\pm$ 20.1	45.6 $\pm$ 42.4	46.4 $\pm$ 17.8

Table 3: Out-of-domain evaluation on Overcooked-AI layouts (Carroll et al., 2019) using the ProAgent framework (Zhang et al., 2024) (mean $\pm$ standard deviation across runs for each layout; the Average row reports mean $\pm$ standard deviation across layouts).

Model	CodeContests	APPS	LiveCodeBench
Qwen3-8B	13.30	40.00	27.43
AT-GRPO (Qwen3-8B) (Zhao et al., 2026)	18.10	45.80	30.28
CoopAgent (Qwen3-8B)	20.00	40.60	30.29

Table 4: Comparison on the coding benchmarks used by AT-GRPO. AT-GRPO scores are from the original paper, while Qwen3-8B and CoopAgent are evaluated in the same environment.

Process	Average Tokens
Single-agent trajectory collection	5,490.92
Perturbation augmentation	2,377.80
Single-agent route (total)	7,868.72
Multi-agent trajectory collection	8,822.67

Table 5: Average token consumption per agent trajectory on VillagerBench.

Qwen2.5-7B (CoopAgent) remains competitive with much larger Qwen3-series models, while Qwen3-8B (CoopAgent) substantially improves over its base model. These results indicate that CoopAgent enables multi-agent deployment without sacrificing single-agent competence.

Multi-Agent Collaboration on VillagerBench.

Table 1 shows that CoopAgent consistently improves multi-agent collaboration on both Construction and Farming tasks. In Construction tasks, CoopAgent outperforms all baselines across completion, construction quality, and efficiency, indicating more effective parallel execution through better coordination and fewer redundant operations. In Farming tasks, CoopAgent achieves a completion rate comparable to LoRA-only, while improving ACR and WACR and maintaining high efficiency. The improvements in ACR and WACR further indicate more balanced agent contributions during collaborative execution. In contrast, larger Qwen3-series models and LoRA-only fine-tuning do not consistently improve these collaboration-oriented metrics, suggesting that the gains cannot be explained by model scale or single-agent SFT alone.

Qualitative Case Study. Figure 3 compares Qwen2.5-7B and CoopAgent in a representative Construction scenario. In both trajectories, Bob first attempts to place a block at $(-10, -60, 1)$ and observes that the position has already been occupied by Alice. Qwen2.5-7B then digs out the existing block and places a block at the same position again, thereby temporarily undoing Alice’s completed work and introducing two redundant actions. CoopAgent, by contrast, proceeds directly to the next unfinished blueprint position at $(-10, -60, 0)$. This contrast suggests that CoopAgent better interprets local environment changes caused by another agent as task progress, consistent with PADPO’s use of task-related observation perturbations during training. The reduced repetition and rework in this example are also consistent with the improvements in collaboration-oriented metrics and execution efficiency reported in Table 1. Additional results and analyses are provided in Appendix F.

Scalability with Increasing Agent Number. Table 2 compares Qwen2.5-7B-based CoopAgent and Qwen2.5-7B with different numbers of agents on the Farming benchmark. Increasing the agent number yields no clear improvement for Qwen2.5-7B, with nearly unchanged completion rate and slightly degraded collaboration metrics, indicating limited scalability. In contrast, CoopAgent benefits from additional agents, achieving higher completion rate, WACR, and execution efficiency. This contrast indicates that CoopAgent effectively leverages additional agents rather than suffering from increased

Model	Single-Agent	Construction (VillagerBench)			Farming (VillagerBench)			
	S (%)	C (%)	VHR	E (%/min)	C (%)	ACR	WACR	E (%/min)
LoRA + DPO	88	59.50	68.97	16.89	52.17	35.95	33.75	19.03
LoRA + PADPO	87	66.90	74.27	26.33	50.00	48.73	44.89	20.06

Table 6: Ablation study on training strategies.

Perturbation Type	Construction (VillagerBench)			Farming (VillagerBench)			
	C (%)	VHR	E (%/min)	C (%)	ACR	WACR	E (%/min)
Random Perturbation	52.63	58.60	17.16	37.00	41.31	35.34	16.09
Task-related Perturbation	66.90	74.27	26.33	50.00	48.73	44.89	20.06

Table 7: Ablation study on perturbation types.

coordination overhead. The higher WACR in the three-agent setting further suggests that CoopAgent has the potential to scale to larger agent populations.

Data Collection Cost. We compare the average token consumption of collecting trajectories for our atomic single-agent tasks with the per-agent average of collecting multi-agent trajectories on the original VillagerBench Construction and Farming benchmarks. As shown in Table 5, single-agent collection together with perturbation augmentation consumes 7,868.72 tokens per trajectory, approximately 10.8% fewer than the 8,822.67 tokens per agent trajectory required for multi-agent collection. Thus, the additional LLM calls introduced by perturbation augmentation do not eliminate the token advantage of the single-agent collection route. Perturbation augmentation is a one-time offline process applied to existing trajectories, requires no additional environment interaction, and can be parallelized across samples. Multi-agent trajectories may additionally require contribution identification and cross-agent credit assignment, which are not measured here. Therefore, this comparison concerns token consumption rather than total monetary or wall-clock cost.

Out-of-Domain Evaluation on Overcooked-AI. We evaluate CoopAgent on Overcooked-AI layouts (Carroll et al., 2019) using the ProAgent framework (Zhang et al., 2024). We also include Tool-Light (Chen et al., 2025), a Qwen2.5-7B-based tool-integrated reasoning baseline. Detailed settings are provided in Appendix E.2.

Table 3 presents the detailed results. CoopAgent achieves the best overall average across the evaluated layouts, with clear improvements over the base

and LoRA models in coordination-intensive settings such as COORDINATION RING, FORCED COORDINATION, and COUNTER CIRCUIT. These layouts require tightly coupled agent behaviors, suggesting that the collaborative patterns learned from single-agent data transfer effectively to structurally different multi-agent environments. A different trend appears on ASYMMETRIC ADVANTAGES, where Tool-Light attains the highest score while CoopAgent underperforms. This layout places less emphasis on tight coordination and is more sensitive to general tool-use competence, indicating that improvements in general agent capability do not necessarily translate into stronger coordination performance. The weaker result of CoopAgent on ASYMMETRIC ADVANTAGES appears to originate mainly from the first-stage SFT adaptation: the LoRA model already drops substantially relative to the base model, while PADPO partially recovers the score, suggesting that domain-specific SFT may weaken some out-of-domain capabilities. The per-run scores and an analysis of score variation are provided in Appendix F.2.

Comparison with a Training-Based MAS Method. We further compare CoopAgent with AT-GRPO (Zhao et al., 2026), a recent multi-agent reinforcement learning method. We report AT-GRPO results from the original paper, and evaluate Qwen3-8B and Qwen3-8B-based CoopAgent under the same evaluation environment for backbone alignment. Detailed evaluation and training settings are provided in Appendix E.3.

As shown in Table 4, CoopAgent consistently improves over the Qwen3-8B base model across all three benchmarks. Compared with AT-GRPO, CoopAgent achieves higher performance on Code-Contests and LiveCodeBench, while AT-GRPO

λ	Single-Agent Tasks		Farming (VillagerBench)			
	Pass@2 (%)	Pass@1 (%)	C (%)	ACR	WACR	E (%/min)
0.001	87	73	50.00	48.73	44.89	20.06
0.005	75	58	41.00	37.70	32.48	19.52

Table 8: Ablation study on regularization strength λ for single-agent tasks and the Farming benchmark.

Regularizer form	C (%)	ACR	WACR	E (%/min)
ReLU	37.50	32.90	28.43	17.86
Huber	44.17	41.34	35.08	19.62
Squared error	50.00	48.73	44.89	20.06

Table 9: Ablation study on the form of the perturbation-alignment regularizer for the Farming benchmark.

performs better on APPS. These results suggest that, despite relying only on single-agent execution trajectories, CoopAgent can achieve performance comparable to AT-GRPO, which is trained with multi-agent interaction data.

4.3 Ablation Study

Ablation on Training Strategies. We compare standard DPO with our PADPO objective to evaluate the effect of perturbation-aligned preference optimization. As shown in Table 6, standard DPO slightly improves single-agent performance and task completion on Farming, confirming the benefit of preference optimization. PADPO, while maintaining comparable single-agent performance, achieves substantially higher collaboration-oriented metrics, especially ACR and WACR. These results suggest that PADPO basically preserves the execution benefits brought by DPO while further improving multi-agent coordination and contribution sharing.

Ablation on Perturbation Type. We further examine whether the gains of PADPO come from generic perturbations or from the task-related perturbations specifically designed in CoopAgent. As shown in Table 7, replacing task-related perturbations with random observation perturbations leads to weaker overall collaboration performance, especially on collaboration-oriented metrics. This indicates that generic observation perturbations cannot fully account for the cooperative gains of PADPO. Instead, PADPO improves cooperation by aligning preferences under perturbations that are explicitly designed to approximate observation changes induced by multi-agent execution.

Ablation on Regularizer Form. We study the effect of the penalty function in the perturbation-alignment term by replacing the squared error with Huber and ReLU-based penalties on the advantage difference $A_\theta(x) - A_\theta(x')$. As shown in Table 9, both variants underperform the squared error baseline. The ReLU-based penalty leads to the largest drops in completion and cooperation metrics, while the Huber penalty is more stable but still lags behind on WACR. This suggests that symmetric penalties are critical for robustness under observation perturbations, and that placing stronger constraints on large preference-advantage deviations leads to more stable and consistent cooperation.

Ablation on Regularization Strength. We analyze the effect of the perturbation-alignment regularization strength on model behavior. Since PADPO approaches standard DPO as λ decreases, we focus on the effect of stronger regularization. As shown in Table 8, increasing λ degrades performance on both single-agent tasks and Farming benchmark. These results suggest that an overly strong consistency constraint may limit the model’s adaptation to meaningful observation changes, leading to degraded overall performance.

5 Conclusion

In this paper, we introduce **CoopAgent**, a training framework that effectively learns collaborative behaviors for multi-agent deployment using only single-agent execution trajectories. CoopAgent adopts a two-stage training pipeline and leverages **PADPO**, a perturbation-aligned preference optimization objective that enforces stable action preferences under observation perturbations. Our experiments show that CoopAgent has the potential to improve multi-agent collaboration without noticeably degrading single-agent performance, exhibits transferable collaborative capability across different environments and task domains, and remains competitive with AT-GRPO on the evaluated coding benchmarks. These results further highlight the potential of single-agent execution data for training cooperative multi-agent behaviors.

Limitations

Our evaluation of CoopAgent covers backbones up to 8B parameters and a limited range of task horizons and agent populations; comparisons with training-based MAS methods are limited to AT-GRPO on its coding benchmarks. Broader comparisons remain future work.

Within the two-stage pipeline, domain-specific SFT may weaken some out-of-domain capabilities, while action-level supervision and a fixed regularization strength may require further adaptation across different model capacities, task requirements, and perturbation settings.

Perturbation augmentation may require environment-specific design to ensure that the generated observation changes remain task-relevant and valid beyond the shared-environment settings studied here.

Acknowledgments

This work is supported by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM103).

References

- Jonas Becker, Lars Benedikt Kaesberg, Niklas Bauer, Jan Philip Wahle, Terry Ruas, and Bela Gipp. 2025. [MALLM: Multi-agent large language models framework](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 418–439, Suzhou, China. Association for Computational Linguistics.
- Micah Carroll, Rohin Shah, Mark K. Ho, Tom Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca D. Dragan. 2019. On the utility of learning about humans for human-ai coordination. In *NeurIPS*, pages 5175–5186.
- Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya G. Parameswaran, Dan Klein, Kannan Ramchandran, Matei A. Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why do multi-agent LLM systems fail? In *NeurIPS*.
- Qi Chai, Zhang Zheng, Junlong Ren, Deheng Ye, Zichuan Lin, and Hao Wang. 2025. [CausalMACE: Causality empowered multi-agents in Minecraft cooperative tasks](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 14410–14426, Suzhou, China. Association for Computational Linguistics.
- Junzhe Chen, Xuming Hu, Shuodi Liu, Shiyu Huang, Wei-Wei Tu, Zhaofeng He, and Lijie Wen. 2024. [LLMArena: Assessing capabilities of large language models in dynamic multi-agent environments](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13055–13077, Bangkok, Thailand. Association for Computational Linguistics.
- Yifei Chen, Guanting Dong, and Zhicheng Dou. 2025. [Toward effective tool-integrated reasoning via self-evolved preference learning](#). *Preprint*, arXiv:2509.23285.
- Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. 2025. Multi-agent collaboration via evolving orchestration. In *NeurIPS*.
- Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. 2024. [VillagerAgent: A graph-based multi-agent framework for coordinating complex task dependencies in Minecraft](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 16290–16314, Bangkok, Thailand. Association for Computational Linguistics.
- Honghao Fu, Junlong Ren, Qi Chai, Deheng Ye, Yujun Cai, and Hao Wang. 2025. [VistaWise: Building cost-effective agent with cross-modal knowledge graph for Minecraft](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21884–21898, Suzhou, China. Association for Computational Linguistics.
- Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2024. [Mastering diverse domains through world models](#). *Preprint*, arXiv:2301.04104.
- Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *NeurIPS Datasets and Benchmarks*.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net.
- Sourena Khanzadeh. 2025. [Agentmesh: A cooperative multi-agent generative ai framework for software development automation](#). *Preprint*, arXiv:2507.19902.
- Seungjoon Lee, Suhwan Kim, Minhyeon Oh, Youngsik Yoon, and Jungseul Ok. 2026. [Experience-based knowledge correction for robust planning in minecraft](#). *Preprint*, arXiv:2505.24157.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun

- Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022. [Competition-level code generation with alphacode](#). *Science*, 378(6624):1092–1097.
- Xuechen Liang, Yangfan He, Meiling Tao, Yinghui Xia, Jianhui Wang, Tianyu Shi, Jun Wang, and JingSong Yang. 2025. [Cmat: A multi-agent collaboration tuning framework for enhancing small language models](#). *Preprint*, arXiv:2404.01663.
- Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng, and Mingli Song. 2025. [Odyssey: Empowering minecraft agents with open-world skills](#). *Preprint*, arXiv:2407.15325.
- Shuo Liu, Zeyu Liang, Xueguang Lyu, and Christopher Amato. 2026. LLM collaboration with multi-agent reinforcement learning. In *AAAI*, pages 32150–32158. AAAI Press.
- Hao Ma, Tianyi Hu, Zhiqiang Pu, Boyin Liu, Xiaolin Ai, Yanyan Liang, and Min Chen. 2024. Coevolving with the other you: Fine-tuning LLM with sequential cooperative multi-agent reinforcement learning. In *NeurIPS*.
- Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. 2025. [Malt: Improving reasoning with multi-agent llm training](#). *Preprint*, arXiv:2412.01928.
- 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. 2024. [ChatDev: Communicative agents for software development](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand. Association for Computational Linguistics.
- Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Scaling large language model-based multi-agent collaboration. In *ICLR*. OpenReview.net.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*.
- Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. 2016. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *NIPS*, pages 1163–1171.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*.
- Vighnesh Subramaniam, Yilun Du, Joshua B. Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. 2025. Multiagent finetuning: Self improvement with diverse reasoning chains. In *ICLR*. OpenReview.net.
- Qian Wang, Tianyu Wang, Zhenheng Tang, Qinbin Li, Nuo Chen, Jingsheng Liang, and Bingsheng He. 2025. Megaagent: A large-scale autonomous llm-based multi-agent system without predefined sops. In *ACL (Findings)*, volume *ACL 2025 of Findings of ACL*, pages 4998–5036. Association for Computational Linguistics.
- Wei Wang, Dan Zhang, Tao Feng, Boyan Wang, and Jie Tang. 2024. [Battleagentbench: A benchmark for evaluating cooperation and competition capabilities of language models in multi-agent systems](#). *Preprint*, arXiv:2408.15971.
- Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddhartha V. Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025a. Livebench: A challenging, contamination-limited LLM benchmark. In *ICLR*. OpenReview.net.
- Isadora White, Kolby Nottingham, Ayush Maniar, Max Robinson, Hansen Lillemark, Mehul Maheshwari, Lianhui Qin, and Prithviraj Ammanabrolu. 2025b. [Collaborating action by action: A multi-agent llm framework for embodied reasoning](#). *Preprint*, arXiv:2504.17950.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2025b. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*. OpenReview.net.
- Sukwon Yun, Jie Peng, Pingzhi Li, Wendong Fan, Jie Chen, James Zou, Guohao Li, and Tianlong Chen. 2026. [Graph-of-agents: A graph-based framework for multi-agent llm collaboration](#). *Preprint*, arXiv:2604.17148.
- Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, XiaoJun Chang,

Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. 2024. Proagent: Building proactive cooperative agents with large language models. In *AAAI*, pages 17591–17599. AAAI Press.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. 2026. [Stronger-mas: Multi-agent reinforcement learning for collaborative llms](#). *Preprint*, arXiv:2510.11062.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. 2025. [MultiAgentBench : Evaluating the collaboration and competition of LLM agents](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8580–8622, Vienna, Austria. Association for Computational Linguistics.

A Single-Agent Tasks

A.1 Task Design Rationale

To support scalable data collection and controlled supervision, we design a set of **single-agent atomic tasks** in the VillagerBench environment. Each task is designed around a single, well-defined action primitive and is weakly coupled with other actions, which allows for reliable execution and unambiguous success evaluation. Despite their simplicity, these atomic tasks cover the majority of action patterns commonly involved in complex multi-agent scenarios such as construction and farming, and thus serve as fundamental building blocks for collaborative behaviors.

To scale up training data and improve generalization, we introduce controlled diversity in both task specifications and environment configurations during data collection. For each task instance, we sample task-specific content and use an LLM to generate multiple semantically equivalent instructions, from which one is randomly selected. Details of the prompt are provided in Appendix G.2. We further randomize the Minecraft environment (e.g., terrain, buildings and trees), producing varied observations while preserving task validity.

A.2 Task Categories and Task Types

The single-agent tasks are organized into 6 high-level categories, comprising a total of 25 atomic task types. Each task type specifies a concrete interaction between the agent and the environment. Table 15 summarizes the task taxonomy, together with their relative sampling ratios at the category level.

B Multi-Agent Tasks

We provide simplified, synthetic examples to illustrate the general structure of multi-agent tasks and representative execution patterns. These examples are constructed for explanatory purposes only and do not correspond to actual task instances or exact specifications. For detailed task definitions and execution protocols, we refer readers to the VillagerBench benchmark (Dong et al., 2024).

B.1 Illustrative Construction Task Example

An example construction task with two agents is shown below:

Task:

Using the provided blueprint, please

collaborate to place blocks in Minecraft.

Blueprint:

```
- cut_sandstone  position: [-8, -60, 0]
- terracotta     position: [-8, -59, 0]
- torch          position: [-8, -60, 1]
```

Execution Trajectories:

Alice: Place cut_sandstone at [-8, -60, 0]
-> Place terracotta at [-8, -59, 0]

Bob: Place torch at [-8, -60, 1]

B.2 Illustrative Farming Task Example

An example farming task with two agents is shown below:

Task:

Using the provided recipe, please work together to gather the necessary items and craft the cake.

Recipe:

```
- milk_bucket  count: 3
- sugar        count: 2
- wheat        count: 3
- egg          count: 1
```

Execution Trajectories:

Alice: Withdraw 3 milk_bucket from chest
-> Get 3 wheat from the farm
-> Hand over 3 milk_bucket to Bob
-> Hand over 3 wheat to Bob

Bob: Withdraw 2 sugar from chest
-> Get an egg from the farm
-> Craft a cake

C Preference Pair Construction

C.1 Overview

This section describes how we construct action-level preference pairs from single-agent execution trajectories. The construction relies on identifying task-relevant actions and contrasting them with failed or task-irrelevant actions observed during execution.

C.2 Construction Procedure

Relevant-tool sets. As a prerequisite for preference construction, we define a relevant-tool set $\mathcal{R} \subseteq \mathcal{U}$ for each atomic task type. The set $\mathcal{R}$

consists of tools whose invocation can directly contribute to task completion under the given task specification. An action y is considered task-relevant if it invokes a tool $u \in \mathcal{R}$, and task-irrelevant otherwise.

Preference pair extraction. Each execution trajectory is represented as an ordered sequence $T = ((x_t, y_t))_{t=1}^n$, where x_t denotes the input prompt at step t and y_t denotes the executed action. As defined in the main text, each action y_t invokes a tool $u_t \in \mathcal{U}$ and yields an execution outcome indicating whether the action succeeds or fails, which is recorded in the interaction history.

For a given step i , we treat the action y_i as a negative action y^- if it either fails to execute or invokes a tool $u \notin \mathcal{R}$. The corresponding input x_i is used as the prompt x . We then define the positive action y^+ as the first subsequent action in the same trajectory that executes successfully and invokes a tool in $\mathcal{R}$. Formally, $y^+ = y_j$, where $j = \min\{k \mid k > i, y_k \text{ succeeds, and } u_k \in \mathcal{R}\}$ and u_k denotes the tool invoked by action y_k .

Validity of Preference Construction. The preference construction procedure relies on the assumption that failed or task-irrelevant actions are non-consequential with respect to the task objective. Specifically, such actions do not consume task-critical resources, modify goal-relevant objects, or alter the environment state in a way that affects subsequent task feasibility. Under this assumption, the environment state at the step of a failed or irrelevant action remains equivalent, up to task-relevant factors, to the state at which the next successful task-relevant action is taken. This allows the subsequent successful action to serve as a surrogate preferred action under the same task context for preference supervision. Moreover, since preference pairs are extracted only from successful trajectories, a subsequent successful task-relevant action always exists for each negative action, ensuring the well-definedness of the constructed preference pairs.

C.3 Algorithm

Algorithm 1 summarizes the preference pair extraction procedure.

Algorithm 1 Preference Pair Generation from a Successful Trajectory

Require: Successful trajectory $T = ((x_t, y_t))_{t=1}^n$
Require: Relevant-tool set $\mathcal{R}$
Ensure: Preference pair set P

```

1: Initialize  $P \leftarrow \emptyset$ 
2: for  $i = 1$  to  $n$  do
3:   if ISFAIL( $y_i$ ) or  $u_i \notin \mathcal{R}$  then
4:      $x \leftarrow x_i, y^- \leftarrow y_i$ 
5:     for  $j = i + 1$  to  $n$  do
6:       if ISSUCCEEDED( $y_j$ ) and  $u_j \in \mathcal{R}$ 
7:       then
8:          $y^+ \leftarrow y_j$ 
9:          $P \leftarrow P \cup \{(x, y^+, y^-)\}$ 
10:        break
11:      end if
12:    end for
13:  end if
14: end for
15: return  $P$ 

```

Setting	Value
Max sequence length	12,288 tokens
Learning rate	5×10^{-5}
Batch size (per device)	1
Gradient accumulation	4
Training epochs	1
LoRA target modules	q_proj, v_proj
LoRA rank r	8
LoRA α	16
LoRA dropout	0.05

Table 10: LoRA supervised fine-tuning configurations.

D Training Configurations

D.1 LoRA Supervised Fine-Tuning

We perform supervised fine-tuning on single-agent execution data using LoRA adapters. The training set contains 1,039 single-agent trajectories, corresponding to 6,380 step-level (x, y) examples. We adopt Qwen2.5-7B-Instruct as the base model and apply LoRA-based parameter-efficient fine-tuning. The main training hyperparameters are summarized in Table 10.

D.2 PADPO Training

PADPO training is performed on top of the LoRA-adapted model obtained in the first stage. We construct 995 preference pairs from successful trajectories, and each pair is augmented with an additional perturbed prompt. The LoRA-adapted model is

Setting	Value
Max sequence length	8,192 tokens
Learning rate	1×10^{-5}
Batch size (per device)	1
Gradient accumulation	4
Training epochs	2
DPO temperature β	0.1
PADPO weight λ	0.001
Merge coefficient ρ	1.0

Table 11: PADPO training configurations for Qwen2.5-7B-Instruct.

Setting	Value
Max sequence length	12,288 tokens
Learning rate	1×10^{-5}
Batch size (per device)	1
Gradient accumulation	4
Training epochs	4
DPO temperature β	0.1
PADPO weight λ	0.0001
Merge coefficient ρ	0.2

Table 12: PADPO training configurations for Qwen3-8B.

also used as the reference policy in PADPO. Table 11 shows the configuration for Qwen2.5-7B-Instruct, which is used in the main experiments. Table 12 shows the configuration for the Qwen3-8B variant used in the comparison with AT-GRPO. We perform weighted LoRA adapter merging between the SFT adapter and the PADPO adapter to improve generalization. The merge coefficient controls how much of the PADPO-induced parameter shift is applied on top of the SFT model.

For numerical stability, the advantage difference used in the perturbation-alignment term is clipped to a fixed range during training.

Ablation Settings. In the ablation study, all training configurations are kept identical to PADPO unless otherwise specified. The only difference lies in the functional form of the regularization term applied to the advantage difference $A_\theta(x) - A_\theta(x')$. For the Huber regularizer, we use a threshold $\delta = 0.05$ and $\lambda = 0.01$. For the ReLU regularizer, the regularization weight λ is fixed to the same value as in PADPO.

E Experimental Settings

E.1 VillagerBench Environment

Execution Framework. All evaluated models are integrated into the same framework and executed with the default prompt templates provided by VillagerAgent (Dong et al., 2024). Other auxiliary models in the framework are kept unchanged and use Qwen3-Next-80B-A3B-Instruct (Yang et al., 2025a).

Tasks and Evaluation Setup. For single-agent evaluation, we generate 100 tasks from the atomic task set described in Appendix A. Each execution trajectory is graded with a binary success or failure label according to predefined rule-based criteria, and we report Pass@2 success rate. We allow up to two agent execution iterations for tasks that require retrieving task-relevant materials from chests, and one iteration for other tasks.

For multi-agent evaluation, we use the Construction Cooperation (Construction) and Farm-to-Table Cooking (Farming) benchmarks from VillagerBench. Illustrative examples of multi-agent tasks are provided in Appendix B. For each benchmark, we uniformly select 10 tasks according to difficulty levels and report results using avg@3. Unless otherwise specified, we follow the default VillagerBench settings. We set the maximum execution time to 720 seconds per multi-agent task, covering the full framework execution process.

Evaluation Metrics. For both Construction and Farming tasks, we report completion rate (C) and efficiency (E), where C measures the overall task completion level and E measures task completion efficiency. For Construction tasks, we additionally report view hit rate (VHR), which measures the proportion of target construction content correctly matched from predefined evaluation views. For Farming tasks, we report agent contribution rate (ACR), which quantifies how task-relevant item acquisition is distributed across agents. The definitions and implementations of C , E , VHR, and ACR follow the original VillagerBench evaluation. Higher values indicate better performance for all these metrics.

When assessing collaborative ability in Farming, ACR is more informative than completion rate because completion rate primarily reflects the overall task completion level, while ACR more directly captures whether task progress is shared across agents. However, tasks with lower comple-

Model	Avg. Task-Level Std.	WACR Std.
Qwen2.5-7B	23.20	5.06
Qwen2.5-7B (LoRA)	20.62	6.50
Qwen3-8B	22.84	3.53
Qwen3-80B-A3B	21.73	9.02
Qwen3-235B-A22B	26.66	2.98
Qwen2.5-7B (CoopAgent)	16.59	1.99
Qwen3-8B (CoopAgent)	19.99	8.57

Table 13: Collaboration stability analysis on the Farming benchmark in VillagerBench. We report the average standard deviation of agent contribution rates across tasks (Avg. Task-Level Std.) and the standard deviation of WACR across evaluation runs (WACR Std.). Lower values indicate more stable and consistent collaboration.

tion rates may involve fewer obtained resources, making agent contributions appear more evenly distributed and thereby inflating ACR. To reduce the influence of such cases, we additionally report weighted agent contribution rate (WACR). For each task i , let c_i denote its completion rate and a_i denote its ACR. WACR is defined as:

$$\text{WACR} = \frac{\sum_i c_i a_i}{\sum_i c_i}. \quad (8)$$

WACR weights ACR by task completion, thereby better reflecting stable collaborative behavior across tasks with different completion levels.

E.2 Overcooked-AI Environment

In the Overcooked-AI (Carroll et al., 2019) evaluation, we use the ProAgent framework (Zhang et al., 2024) for execution. Overcooked-AI layouts require two agents to complete the tasks, and we use the evaluated model as the policy model for both agents in ProAgent without modifying the framework itself. Each layout is evaluated with a horizon of 200 steps, and we report the mean and standard deviation over five runs.

E.3 Coding Benchmark Environment

We evaluate the Qwen3-8B base model and the Qwen3-8B-based CoopAgent variant on the coding benchmarks used in AT-GRPO (Zhao et al., 2026). A direct comparison on VillagerBench would require substantial adaptation to its real-time environment and reward setting, while the publicly available AT-GRPO resources do not include a trained checkpoint or training data. Therefore, we use the AT-GRPO scores reported in the original paper as the reference results, and evaluate Qwen3-8B and CoopAgent locally using the same evaluation environment.

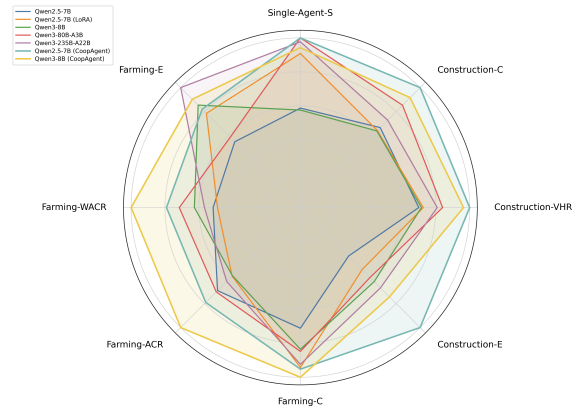


Figure 4: Radar plot corresponding to the main results reported in Table 1.

The evaluated coding benchmarks include CodeContests, APPS, and LiveCodeBench. Following the original evaluation setting, all coding results are reported with Pass@1. We use no-thinking mode, which disables explicit long-form reasoning and directly generates answers, to match the original AT-GRPO evaluation setting. All remaining inference settings, agent role prompts, and system prompts follow the default configuration of the released evaluation environment.

F Additional Experimental Analysis

This section provides additional analyses of the VillagerBench and Overcooked-AI results, together with a radar chart of the main VillagerBench results (Figure 4).

F.1 Variance and Stability of Collaboration Metrics

In addition to average performance, we analyze the stability of collaborative behavior on the Farming benchmark. Table 13 reports two complementary variance-based metrics: the average standard deviation of ACR across individual tasks and the standard deviation of WACR across evaluation runs.

Overall, Qwen2.5-7B (CoopAgent) achieves the strongest stability across both task-level and run-level measures, obtaining the lowest average task-level standard deviation and the lowest WACR standard deviation. Qwen3-8B (CoopAgent) also reduces the task-level contribution variance compared with the Qwen3-8B base model. Compared with the base Qwen2.5-7B and the LoRA-only variant, Qwen2.5-7B (CoopAgent) exhibits substantially reduced variance, indicating more uniform collaboration behavior across tasks and evaluation

Model	Layout	Run 1	Run 2	Run 3	Run 4	Run 5
Qwen2.5-7B	Cramped Room	80	40	80	0	80
	Asymmetric Advantages	80	80	80	80	60
	Coordination Ring	0	0	0	0	0
	Forced Coordination	0	0	0	0	0
	Counter Circuit	60	60	60	60	40
Qwen2.5-7B (LoRA)	Cramped Room	40	60	40	40	60
	Asymmetric Advantages	40	40	40	0	40
	Coordination Ring	20	20	0	20	0
	Forced Coordination	0	20	0	20	0
	Counter Circuit	60	60	80	60	0
Tool-Light	Cramped Room	60	40	20	20	20
	Asymmetric Advantages	120	120	100	120	120
	Coordination Ring	20	0	0	0	0
	Forced Coordination	20	20	20	40	40
	Counter Circuit	20	80	60	60	20
Qwen2.5-7B (CoopAgent)	Cramped Room	60	80	60	40	60
	Asymmetric Advantages	60	40	40	40	40
	Coordination Ring	0	40	40	20	20
	Forced Coordination	60	20	60	20	20
	Counter Circuit	80	60	80	60	60

Table 14: Per-run scores over five independent runs on Overcooked-AI. The corresponding means and sample standard deviations are reported in Table 3.

runs. These results further support that CoopAgent improves collaboration stability most consistently in the main Qwen2.5-7B-based setting, beyond supervised fine-tuning alone.

F.2 Score Variation on Overcooked-AI

Overcooked-AI uses a coarse-grained scoring rule in which each completed dish contributes 20 points. Consequently, differences of only one or two completed dishes across runs can produce relatively large numerical standard deviations. Table 14 reports all five scores for each model on each layout, making the underlying variation explicit alongside the aggregate statistics in Table 3.

G Prompts

G.1 Perturbation Augmentation

The prompt template used in the perturbation augmentation process is shown in Figure 5.

G.2 Task Description Generation

The prompt template used to generate task descriptions is shown in Figure 6.

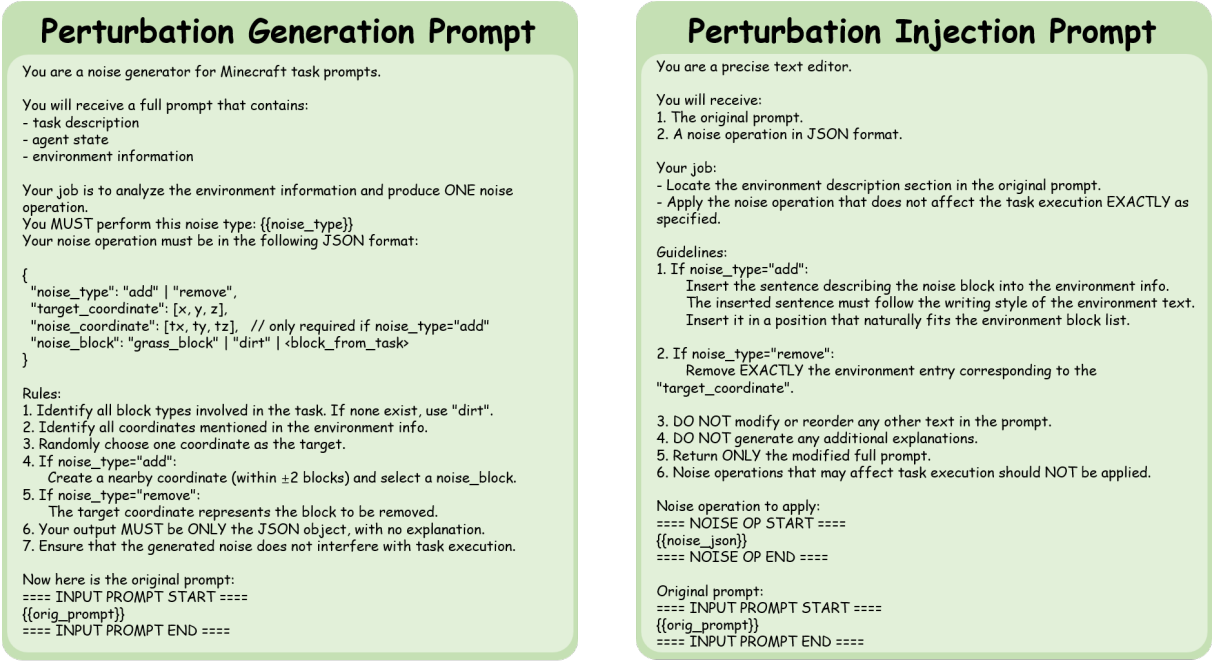


Figure 5: Prompt Template Used for Perturbation Augmentation.

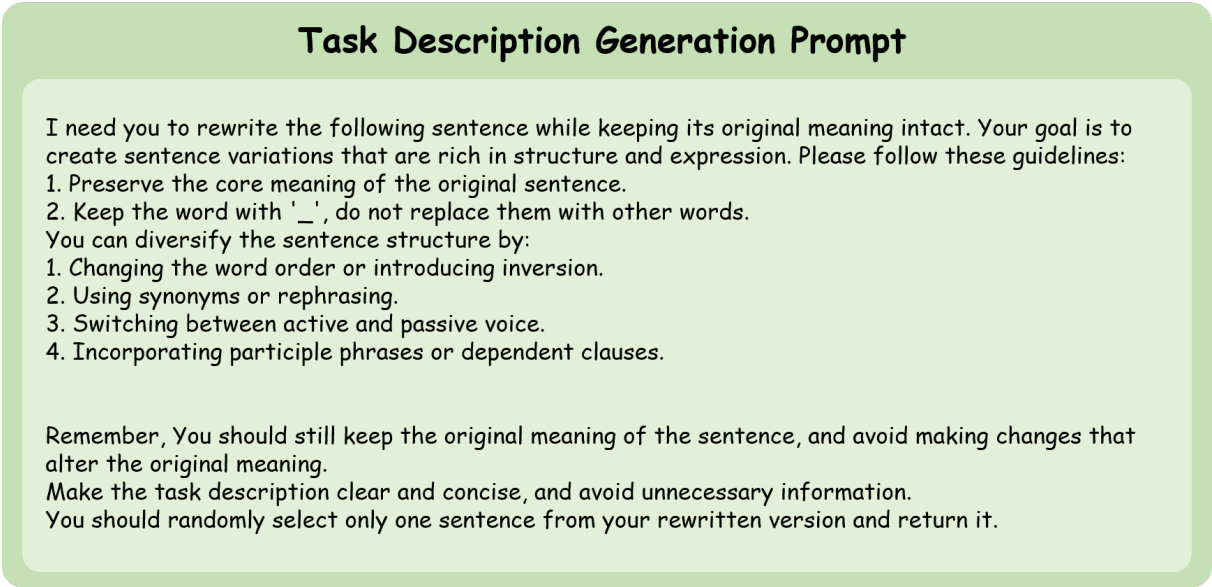


Figure 6: Prompt Template Used for Task Description Generation.

Category	Task Type	Ratio	Task Description	Example Task Instruction
Dig	w/ tool	7	Dig or mine a specified block using tools.	Using <code>diamond_pickaxe</code> to mine the <code>stone_stairs</code> located at (17, -60, 17).
	w/o tool		Dig or break a specified block without tools.	Dig the <code>pink_carpet</code> at (10, -58, 18).
Craft	craft	16	Craft a specified item from materials.	Using <code>crafting_table</code> in the environment to create a <code>birch_fence</code> .
Place	default	7	Place some blocks at target locations.	At the coordinates (5, -59, 10), and (5, -59, 11), place 2 <code>stone_bricks</code> .
	face		Place a block at a target location with a specified facing.	Place a <code>stonecutter</code> at (10, -60, 7), facing north.
Equip	equip	1	Equip a specified item.	Equip the <code>iron_boots</code> .
Move	move	2	Move to a target position.	Reach the position (11, -59, 12).
Interact	attack	67	Attack a target entity.	Strike the cow with the <code>iron_sword</code> .
	feed		Feed an animal.	Feed the chicken using the <code>pumpkin_seeds</code> .
	cook		Cook items using a furnace.	Using the furnace in the environment, cook the chicken with coal.
	handover		Hand over an item to another entity.	Pass the <code>iron_bars</code> to Bob.
	store		Store an item in a container.	Place the <code>jungle_log</code> inside the chest located at (8, -60, 18).
	shear		Shear a sheep.	Shear the sheep with shears from your inventory.
	milk		Milk a cow.	Use bucket to milk the cow.
	water		Fill the bucket with water.	Fill a bucket with water from the pool located at (17, -60, 3).
	till		Till farmland.	Till the land at (9, -60, 22) to farmland.
	fishing		Catch fish.	Head to the coordinates (13, -60, 15) to go fishing.
	bone		Apply bone meal to crops.	Utilize the <code>bone_meal</code> from your inventory to grow the lilac at (1, -60, 18).
	chat		Send a chat message.	Portray a patient and seasoned individual and try to discuss mining with Bob.
	sign		Read text on a sign.	Read the content displayed on the <code>oak_sign</code> at the coordinates (11, -60, 16).
	toggle		Toggle an interactive block.	Open the <code>dark_oak_door</code> at (3, -60, 11).
	saddle		Equip the animal with a saddle, then mount and dismount it.	Put the saddle on the horse and ride it, then dismount it.
	boat		Place or use a boat, then mount and dismount it.	Ride the <code>acacia_chest_boat</code> and dismount it.
	minecart		Place or use a minecart, then mount and dismount it.	Ride the minecart and dismount it at (18, -60, 5).
	bed		Sleep in the bed, then wake up.	Sleep in the <code>light_blue_bed</code> , then wake up.

Table 15: Single-agent task categories and 25 atomic task types. The ratio column indicates the relative sampling weight at the category level during task generation.