Curriculum Round-Robin Reinforcement Learning for Training Multi-Agent LLM Systems

Anonymous ACL submission

Abstract

Large Language Model (LLM) agents are used as parts of multi-agent systems, yet efficiently optimizing them remains challenging because each agent's training distribution depends on other agents. A common compute-efficient strategy is to freeze all but one agent, optimize the unfrozen agent against the current pipeline, and repeat this process for the remaining agents. We identify this procedure as stale-pipeline drift, where agents trained earlier are optimized for partner policies that may later be updated, creating a mismatch between training-time and deployment-time interaction distributions. To mitigate this drift, we propose Curriculum Round-Robin GRPO (CRR-GRPO), a training process that combines agent-wise round-robin updates with a gradually expanding reward curriculum. CRR-GRPO decomposes the global task reward into sequential sub-rewards and revisits each trainable component as new reward terms are introduced, allowing agents to co-adapt incrementally rather than overfitting to stale pipeline configurations. We formalize a regret bound that measures how performance varies with divergence between training and deployment trajectory distributions. Empirically, across ten benchmarks and three base LLMs, CRR-GRPO improves overall performance over various baselines under a matched training budget.[1]

1 Introduction

Large Language Model (LLM) agents increasingly solve complex tasks through multi-step reasoning, tool use, and interaction with external environments (Schick et al., 2023; Saeidi et al., 2026). While reinforcement learning (RL) has improved single-model reasoning and tool use (Guo et al., 2025; Jaech et al., 2024; Anthropic, 2026), many realistic tasks require modular systems in which different components plan, act, verify, and refine intermediate outputs (Rawles et al., 2025; Zhou et al., 2024; Liu et al., 2024). This has motivated multi-agent systems (MAS), where multiple LLMs are used to divide cognitive labor across specialized roles (Wu et al., 2024; Li et al., 2023; Du et al., 2024; Li et al., 2025).

Despite their promise, MAS remain difficult to optimize (Liao et al., 2025). Naive multi-agent reinforcement learning (MARL) jointly updates all LLM agents, which can be computationally expensive and unstable (Liu et al., 2026; Wang et al., 2022; Gogineni et al., 2023). A common practical alternative is round-robin training, in which all but one agent is frozen, the unfrozen agent is optimized using RL, the updated checkpoint is inserted into the pipeline, and the process is repeated for the remaining agents (Wan et al., 2026; Lu et al., 2025). This reduces multi-agent optimization to a sequence of single-agent updates and is compatible with existing GRPO-style post-training pipelines.

However, round-robin training introduces a moving target problem (Xie et al., 2026). Consider a system with K agents. When the first agent is trained, it is optimized against the current versions of the other $K-1$ agents. After later agents are updated, the first agent is deployed with partners that differ from those it saw during training. Repeating this procedure leaves every agent, except the last, optimized against a stale configuration. We refer to this mismatch between training-time and deployment-time interaction distributions as *stale-pipeline drift* in this paper.

To mitigate this drift without increasing the total training budget, we propose Curriculum Round-Robin GRPO (CRR-GRPO). CRR-GRPO exploits the composite structure of agent rewards, such as format validity, tool success, intermediate progress, and final-answer correctness. Rather than training each component once on the full reward, CRR-GRPO revisits components under a sequence of

[1] Code at https://anonymous.4open.science/r/mas_training-3F6C/README.md

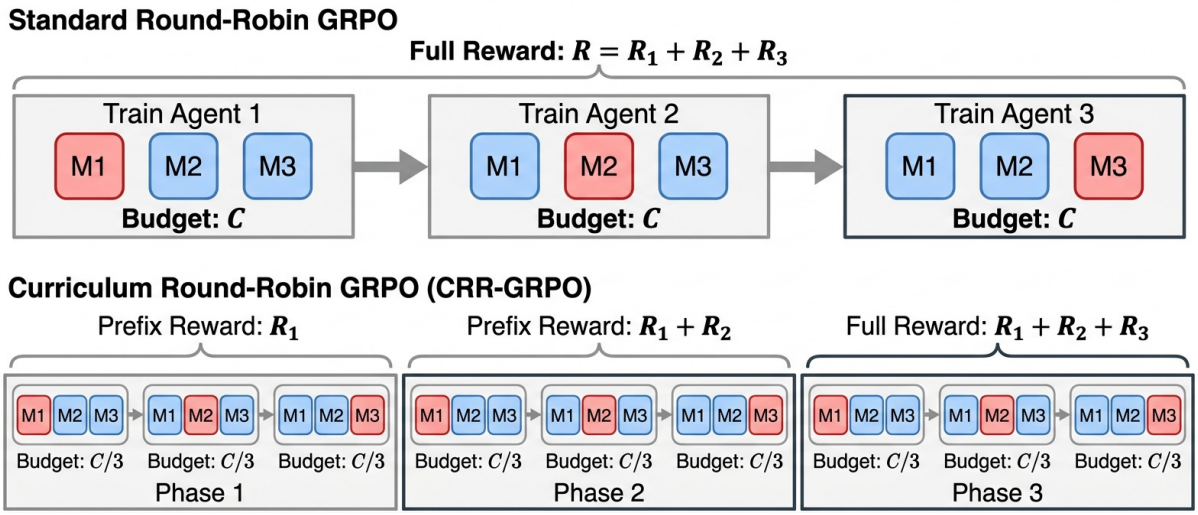

Figure 1: Overview of standard round-robin GRPO and Curriculum Round-Robin GRPO (CRR-GRPO). Red indicates the currently trainable model, and blue indicates frozen models. Standard round-robin GRPO trains each component once under the full reward $R = R_1 + R_2 + R_3$. CRR-GRPO revisits components across curriculum phases, training first with R_1 and later with expanded reward prefixes up to $R_1 + R_2 + R_3$. To match compute, CRR-GRPO splits the same per-component update budget across phases. Thus, the total number of optimization steps per component is the same as standard round-robin GRPO.

expanding reward prefixes. Formally, for a reward decomposition $R = \sum_{\ell=1}^{L} R_\ell$, CRR-GRPO trains agents under prefix rewards $R^{(\ell)} = \sum_{m=1}^{\ell} R_m$, cycling through agents at each stage m and allocating each component the same total update budget as in standard round-robin GRPO. Agents are first trained on local validity rewards, then revisited as progress and terminal rewards are added. This allows components to co-adapt incrementally as partner checkpoints evolve.

We formalize stale-pipeline drift through a regret bound that depends on the divergence between the trajectory distribution induced during training and the one induced at deployment (§3.3). The analysis shows that reducing the magnitude of partner updates between training and deployment tightens the drift-dependent regret term. Empirically, we instantiate CRR-GRPO using AgentFlow (Li et al., 2025) framework and evaluate it across ten benchmarks and three Qwen3 (Yang et al., 2025a) models of varying sizes. CRR-GRPO yields modest but consistent gains over non-curriculum GRPO baselines under a matched training budget, while reducing measured drift during training by up to 57%.

2 Related Works

Multi-Agent LLM Systems. Recent works (Wu et al., 2024; Handa et al., 2025; Zhang et al., 2025; Li et al., 2024) have increasingly used LLMs as components in multi-agent systems, where specialized agents decompose tasks, debate candidate solutions, critique intermediate outputs, or coordinate tool use. Early systems such as CAMEL (Li et al., 2023) and AutoGen (Wu et al., 2024) showed that role-specialized LLM agents can cooperate through structured conversation, while multi-agent debate (Du et al., 2024) and sparse-communication methods (Li et al., 2024) study how interaction structure affects reasoning quality. More recent agentic benchmarks and systems extend this paradigm to tool use (Yao et al., 2024), web navigation (Zhou et al., 2024), software engineering (Jimenez et al., 2024), and long-horizon task solving (Mialon et al., 2023). Our work builds on this modular view of LLM systems, but focuses on the training problem that arises once the individual agents are no longer fixed prompts but trainable policies.

Training Multi-Agent LLM Systems. Training multi-agent LLM systems is challenging because each agent's behavior changes the distribution observed by the others (Liao et al., 2025). A direct multi-agent reinforcement learning approach jointly optimizes multiple LLM policies, but this can be expensive and unstable (Liu et al., 2026; Wang et al., 2022; Gogineni et al., 2023). Recent work therefore explores more practical alternatives. MARFT (Liao et al., 2025) studies reinforcement fine-tuning for multi-agent collaboration, while REMA (Wan et al., 2026) trains

meta-reasoning behavior in multi-agent settings. M-GRPO (Hong et al., 2025) extends GRPO to hierarchical multi-agent systems by assigning group-relative advantages to main and sub-agents and aligning variable-length sub-agent trajectories. SWIRL (Lu et al., 2025) reformulates multi-agent reinforcement learning as a sequence of single-agent RL problems, updating one agent while keeping the others fixed. These methods demonstrate that multi-agent LLM systems can be optimized with RL-style post-training, but they primarily focus on making multi-agent training feasible. In contrast, we study the stale-pipeline drift introduced by sequentially training agents against partner policies that later change, and propose curriculum-based revisitation to reduce this mismatch.

Curriculum Learning and Reward Decomposition. Curriculum learning improves optimization by presenting training signals in a structured order, often from simpler to more difficult objectives (Bengio et al., 2009). In reinforcement learning, curricula have been used for skill acquisition, exploration, and stabilizing long-horizon optimization (Narvekar et al., 2020; Soviany et al., 2022). Curriculum is also used in reward shaping (Gupta et al., 2025), process supervision (Lightman et al., 2023; Uesato et al., 2022), and step-level verification (Setlur et al., 2025; Wang et al., 2024), where intermediate rewards guide training before final outcome rewards. CRR-GRPO combines these ideas with round-robin multi-agent training: instead of training each agent once on the full reward, we decompose the reward into prefix objectives $R^{(1)}, R^{(2)}, \ldots, R^{(L)}$ and revisit agents as new reward terms are added. Unlike standard reward shaping, our goal is not only to provide denser supervision, but also to mitigate stale-pipeline drift by allowing agents to co-adapt as partner checkpoints evolve. The curriculum structures when agents are revisited, and directly affects the training-deployment mismatch.

3 Curriculum Rewards

3.1 Introduction and System Setup

We analyze the distribution shift that arises when training one component of a multi-agent Large Language Model (LLM) system while keeping the remaining components frozen.

Let the system consist of K LLM components with policies $\pi_{\theta_1}, \pi_{\theta_2}, \ldots, \pi_{\theta_K}$, and let $\theta = (\theta_1, \ldots, \theta_K)$ denote the full set of parameters. For a fixed input, the multi-agent system induces a trajectory distribution

$$\tau \sim P_\theta, \tag{1}$$

where τ contains all intermediate messages, tool calls, reasoning traces, and final outputs produced by the system. The full task objective is to maximize the expected reward over these trajectories:

$$J(\theta) = \mathbb{E}_{\tau \sim P_\theta}[R(\tau)], \tag{2}$$

where $R(\tau)$ is the task reward.

For a single trainable component i and fixed partners η_{-i}, we write

$$J_i(\theta_i; \eta_{-i}) = \mathbb{E}_{\tau \sim P_{\theta_i, \eta_{-i}}}[R(\tau)]. \tag{3}$$

This notation lets us compare the objective optimized during training, where the partners are frozen to some snapshot, with the objective at deployment, where all components use their final trained parameters.

3.2 Reward Decomposition and Curriculum

We assume that the total reward can be decomposed into a sequence of sub-rewards. This provides a mechanism for revisiting agents before exposing them to the full terminal reward, reducing the tendency to overfit to a single static partner configuration. Let L be the total number of curriculum stages. We define the total reward as

$$R(\tau) = \sum_{m=1}^{L} R_m(\tau). \tag{4}$$

At curriculum stage m, we define the **prefix reward** as the running sum of all sub-rewards up to that stage:

$$R^{(m)}(\tau) = \sum_{\ell=1}^{m} R_\ell(\tau). \tag{5}$$

The corresponding prefix objective is

$$J^{(m)}(\theta) = \mathbb{E}_{\tau \sim P_\theta}[R^{(m)}(\tau)]. \tag{6}$$

At the final stage, the prefix objective equals the full task objective: $J^{(L)} = J$.

3.2.1 Independent, Cascaded, and Curriculum Round-Robin Training

We consider three training algorithms $\mathcal{A} \in \{\mathrm{Ind}, \mathrm{Cas}, \mathrm{CRR}\}$ that differ in the partner snapshot against which each component is trained. Let

ψ_j denote the final deployed parameters of component j. For component i, let $\phi^{\mathcal{A}}_{j|i}$ denote the parameters of partner j used when component i is trained under algorithm $\mathcal{A}$, and let $\phi^{\mathcal{A}}_{-i} = \{\phi^{\mathcal{A}}_{j|i} : j \neq i\}$. The deployed pipeline uses $\psi_{-i} = \{\psi_j : j \neq i\}$.

Let $\theta^{(0)}_j$ denote the base, untrained parameters of component j. For Cascaded GRPO and CRR-GRPO, we additionally fix a training order given by a permutation $\pi : \{1, \ldots, K\} \to \{1, \ldots, K\}$, so that $\pi(j)$ means component j is trained earlier.

Independent GRPO ($\mathcal{A} = \mathrm{Ind}$). Each component is trained independently against frozen base partners:

$$\phi^{\mathrm{Ind}}_{j|i} = \theta^{(0)}_j, \qquad \forall j \neq i. \tag{7}$$

No information flows between components during training; the K runs are independent and could execute in parallel.

Cascaded GRPO ($\mathcal{A} = \mathrm{Cas}$). Each component is trained against a partial pipeline in which components trained earlier in the order have already been promoted:

$$\phi^{\mathrm{Cas}}_{j|i} = \begin{cases} \psi_j & \text{if } \pi(j) < \pi(i), \\ \theta^{(0)}_j & \text{if } \pi(j) > \pi(i). \end{cases} \tag{8}$$

Thus, the last component in the cascade is trained against the fully deployed partner pipeline, while earlier components are trained against partner configurations that may later change.

Curriculum Round-Robin GRPO ($\mathcal{A} = \mathrm{CRR}$). CRR-GRPO revisits components across curriculum stages. At stage m, all components are trained in the fixed round-robin order using the prefix reward $R^{(m)}$ from Eq. 5. After a component is trained, its updated checkpoint is promoted into the pipeline before the next component is trained. Given a total per-component update budget C, CRR-GRPO allocates C/L updates to each component at each curriculum stage, so the total number of updates per component matches standard round-robin GRPO.

Let ϕ^{CRR}_{-i} denotes the partner snapshot present when component i is last optimized under the full reward $R^{(L)} = R$. Earlier curriculum stages affect the final learned policy by shaping the trajectory of updates, but the deployment mismatch is measured relative to this final full-reward training snapshot.

3.3 Formalizing Stale-Pipeline Drift and Regret

To measure the moving-target effect, we compare the partner snapshot $\phi^{\mathcal{A}}_{-i}$ seen by component i during training with the final deployed partner snapshot ψ_{-i}.

For algorithm $\mathcal{A}$, we define the stale set of component i as the partners that change between i's training snapshot and deployment:

$$S^{\mathcal{A}}_i = \{\, j \neq i : \phi^{\mathcal{A}}_{j|i} \neq \psi_j \,\}. \tag{9}$$

By construction,

$$S^{\mathrm{Ind}}_i = \{j : j \neq i\}, \tag{10}$$

$$S^{\mathrm{Cas}}_i = \{j : \pi(j) > \pi(i)\}. \tag{11}$$

Therefore, the cumulative stale-set size $\sum_i |S^{\mathcal{A}}_i|$ is $K(K-1)$ for Independent GRPO and $K(K-1)/2$ for Cascaded GRPO.

For CRR-GRPO, the stale set is determined by the final-stage training order. If the final-stage order matches the Cascaded GRPO order, then CRR-GRPO has the same final stale-set cardinalities as Cascaded GRPO. Its advantage is therefore not necessarily a smaller stale set, but a smaller remaining partner shift: because agents are revisited under expanding reward prefixes, the partners that change after a component's final update can move by a smaller amount. We define the **stale-pipeline drift** for algorithm $\mathcal{A}$ as

$$D^{\mathcal{A}}_i(\phi^{\mathcal{A}}_{-i}, \psi_{-i}) = \sup_{\theta_i} d_{\mathrm{TV}}\left(P_{\theta_i, \phi^{\mathcal{A}}_{-i}}, P_{\theta_i, \psi_{-i}}\right). \tag{12}$$

We define the deployment regret of a learned component $\hat{\theta}_i$ as

$$\mathrm{Reg}_i(\hat{\theta}_i; \psi_{-i}) = \sup_{\theta_i} J_i(\theta_i; \psi_{-i}) - J_i(\hat{\theta}_i; \psi_{-i}). \tag{13}$$

For algorithm $\mathcal{A}$, let $\hat{\theta}^{\mathcal{A}}_i$ be the policy learned by component i. Its optimization error under the training partner snapshot is

$$\epsilon^{\mathcal{A}}_i = \sup_{\theta_i} J_i(\theta_i; \phi^{\mathcal{A}}_{-i}) - J_i(\hat{\theta}^{\mathcal{A}}_i; \phi^{\mathcal{A}}_{-i}). \tag{14}$$

Theorem 1 (Stale-pipeline regret bound)**.** *Assume $|R(\tau)| \leq R_{\max}$ for all trajectories τ. Then, for any algorithm $\mathcal{A}$ and component i,*

$$\mathrm{Reg}_i\left(\hat{\theta}^{\mathcal{A}}_i; \psi_{-i}\right) \leq \epsilon^{\mathcal{A}}_i + 4R_{\max} D^{\mathcal{A}}_i(\phi^{\mathcal{A}}_{-i}, \psi_{-i}). \tag{15}$$

The proof is given in Appendix A. The theorem shows that deployment regret is controlled by two terms: the optimization error under the partner snapshot used during training, and the drift between that snapshot and the final deployed partner pipeline.

3.4 Computability via KL Divergence

Total variation distance provides a useful theoretical measure, but directly estimating the probability of every possible system trajectory is very difficult to compute. We therefore relate the drift term to KL divergences between partner checkpoints. Unlike the standard GRPO KL penalty, which is typically computed between the trainable policy and a reference policy, the KL terms below compare the partner checkpoint seen during training with the final partner checkpoint used at deployment.

Assume that all methods share the same input distribution and environment dynamics, that tool outputs are deterministic conditional on the trajectory history, that each stale partner j is invoked at most H times in a trajectory, and that the final partner policy is absolutely continuous with respect to the corresponding stale partner policy. For partner j in the training snapshot of component i, define

$$\rho^{\mathcal{A}}_{j|i} = \sup_{s} D_{\mathrm{KL}}\Big(\pi_{\psi_j}(\cdot \mid s) \,\|\, \pi_{\phi^{\mathcal{A}}_{j|i}}(\cdot \mid s)\Big). \quad (16)$$

Using the chain rule for KL divergence over the trajectory and Pinsker's inequality, we obtain

$$D^{\mathcal{A}}_i(\phi^{\mathcal{A}}_{-i}, \psi_{-i}) \le \sqrt{\frac{H}{2} \sum_{j \in S^{\mathcal{A}}_i} \rho^{\mathcal{A}}_{j|i}}. \quad (17)$$

The derivation is given in Appendix B.

Let the system-level regret be $\mathrm{Reg}^{\mathcal{A}}_{\mathrm{sys}} = \sum_{i=1}^{K} \mathrm{Reg}_i(\hat{\theta}^{\mathcal{A}}_i; \psi_{-i})$. Combining Eqs. 15 and 17 gives

$$\mathrm{Reg}^{\mathcal{A}}_{\mathrm{sys}} \le \sum_{i=1}^{K} \epsilon^{\mathcal{A}}_i + 4R_{\max} \sum_{i=1}^{K} \sqrt{\frac{H}{2} \sum_{j \in S^{\mathcal{A}}_i} \rho^{\mathcal{A}}_{j|i}}. \quad (18)$$

This decomposition separates two ways to reduce stale-pipeline regret. Cascaded GRPO reduces the number of stale partners relative to Independent GRPO. CRR-GRPO, when using the same final-stage order as Cascaded GRPO, does not necessarily reduce stale-set cardinality; instead, it can reduce the KL terms $\rho^{\mathcal{A}}_{j|i}$ by making the remaining partner updates smoother through curriculum revisitation.

Corollary 1 (Effect of cascading)**.** *Cascaded GRPO reduces the cumulative stale-set size from $K(K-1)$ for Independent GRPO to $K(K-1)/2$. Under a uniform per-component drift bound*

$$\rho_{\max} = \max_{\mathcal{A}, i, j} \rho^{\mathcal{A}}_{j|i},$$

the KL-dependent drift terms satisfy

$$\sum_{i=1}^{K} \sqrt{\frac{H}{2} \sum_{j \in S^{\mathrm{Ind}}_i} \rho^{\mathrm{Ind}}_{j|i}} \le K \sqrt{\frac{H(K-1)}{2} \rho_{\max}}, \quad (19)$$

$$\sum_{i=1}^{K} \sqrt{\frac{H}{2} \sum_{j \in S^{\mathrm{Cas}}_i} \rho^{\mathrm{Cas}}_{j|i}} \le \sqrt{\frac{H}{2} \rho_{\max}} \sum_{k=1}^{K-1} \sqrt{k}. \quad (20)$$

For $K = 3$, this corresponds to an approximately 43% reduction in the cumulative drift term. The last component in the cascade has $|S^{\mathrm{Cas}}_i| = 0$, so its drift term vanishes and its regret bound reduces to $\mathrm{Reg}_i(\hat{\theta}^{\mathrm{Cas}}_i; \psi_{-i}) \le \epsilon^{\mathrm{Cas}}_i$.

Corollary 2 (Sufficient condition for CRR improvement)**.** *Suppose curriculum revisitation reduces the stale-pipeline drift for component i by a factor $\alpha_i \in [0, 1]$ relative to Cascaded GRPO:*

$$D^{\mathrm{CRR}}_i \le \alpha_i D^{\mathrm{Cas}}_i. \quad (21)$$

Then CRR-GRPO yields a tighter component-wise regret bound than Cascaded GRPO whenever

$$\epsilon^{\mathrm{CRR}}_i - \epsilon^{\mathrm{Cas}}_i < 4R_{\max}(1 - \alpha_i) D^{\mathrm{Cas}}_i. \quad (22)$$

The special case $\alpha_i = 1/L$ corresponds to an idealized L-stage curriculum in which the remaining partner drift is reduced proportionally with the number of curriculum stages.

4 Experiment Setup

Multi-Agent System. We adopt the three-role AgentFlow (Li et al., 2025) architecture, which includes a **Planner** that decomposes a query into sub-goals and selects a tool, an **Executor** that emits the corresponding tool-call command, and a **Verifier** that decides whether to STOP or CONTINUE after each step. A non-trainable **Finalizer** produces the answer when the Verifier stops. All four roles share a single base LLM served via vLLM (Kwon, 2025). Only the Planner, Executor, and Verifier are trainable ($K = 3$ in the notation of §3.2.1). The system has access to AgentFlow's standard toolset, including a generator tool backed by the base LLM and a sandboxed Python execution tool. We extend this toolset with Wikipedia search tool for search-heavy benchmarks.

Base Models. We instantiate the framework on three base LLMs, Qwen3-0.6B, Qwen3-1.7B, and Qwen3-4B-Instruct-2507 (Yang et al., 2025a). For each setting, the same base model is used as the shared backbone for all four roles.

Reward Decomposition. We decompose the system reward into three sub-rewards used as curriculum prefixes: $R^{(1)} = R_1$, $R^{(2)} = R_1 + R_2$, and $R^{(3)} = R_1 + R_2 + R_3$.

- R_1 (**local validity**): per-component format and structural correctness, including tool-name validity and sub-goal non-emptiness for the Planner, command syntactic validity and tool execution success for the Executor, and parseable STOP/CONTINUE decisions for the Verifier.
- R_2 (**marginal progress**): the per-step contribution to final-answer correctness, computed by invoking the frozen Finalizer on the action prefix before and after each step and comparing both completions against the ground truth.
- R_3 (**terminal accuracy**): a binary indicator of whether the system's final answer matches the ground truth, evaluated by an LLM judge.

Training Schemes. We compare the three RL training schemes formalized in §3.2.1.

- **Independent GRPO**: each component is trained from the base checkpoint against frozen base partners under the full reward $R_1 + R_2 + R_3$. The three component runs are independent.
- **Cascaded GRPO**: components are trained in the order *Planner* $\rightarrow$ *Executor* $\rightarrow$ *Verifier* under the full reward. After each phase, the merged checkpoint is promoted into the frozen pipeline before training the next component.
- **CRR-GRPO (ours)**: components are trained over three curriculum stages with prefix rewards $R^{(1)}$, $R^{(2)}$, and $R^{(3)}$. Within each stage, components are trained round-robin in the same order as Cascaded GRPO, and each updated checkpoint is promoted before the next component is trained.

For all methods, frozen partners run deterministically (temperature 0), so variance within a GRPO group is attributable to the trainable component. For compute parity, Independent and Cascaded GRPO train each component for C optimizer steps, while CRR-GRPO splits the same per-component budget across the three reward prefixes, using $C/3$ optimizer steps per component per prefix. Thus all GRPO methods use the same total rollout budget and the same number of optimizer updates per trainable component.

Training Data. We use AgentFlow's training mixture, consisting of Natural Questions (Kwiatkowski et al., 2019) and DeepMath-103k (He et al., 2025), combined and shuffled ($\sim$182k samples). We sample a random subset of 10k questions to use as our training data

Training Configuration. All trainable components are optimized with GRPO (Shao et al., 2024). We use rollout group size 4, training batch size 42, PPO mini-batch size 12, learning rate 10^{-6}, KL-loss coefficient 10^{-3}, and clip ratios $(0.2, 0.3)$. Each component receives the same total training budget. Baseline GRPO methods train each component for one full budget of C optimizer steps, while CRR-GRPO divides that budget evenly across the three curriculum stages. Maximum prompt length is 18,432 tokens and maximum response length is 2,048 tokens. Each rollout is limited to 3 tool-use steps with a 500s wall-clock timeout. Training temperature is 0.7 and evaluation temperature is 0. We train on 4$\times$H100 GPUs with FSDP and merge FSDP shards into HuggingFace format between phases for promotion.

Baselines.

- **AgentFlow (Zero-Shot)**: the base LLM running the AgentFlow's multi-agent system with no RL training.
- **SFT Distillation**: supervised fine-tuning on successful trajectories generated by a stronger teacher, Qwen3-32B, running the same AgentFlow multi-agent system.
- **Independent GRPO** and **Cascaded GRPO**: the two non-curriculum round-robin schemes defined in §3.2.1.

Benchmarks. We evaluate on the ten benchmarks, spanning five task families: 1) Multi-Hop QA, which contains 2Wiki (Ho et al., 2020), Bamboogle (Press et al., 2023), HotpotQA (Yang et al., 2018), and MuSiQue (Trivedi

Table 1: Performance of Qwen3-0.6B, Qwen3-1.7B, and Qwen3-4B-Instruct across ten benchmarks. **Bold** indicates best per model, Underline indicates the second-best per model. Average Rank is computed across benchmarks, where lower is better.

Method	Multi-hop QA				Comp. Math		Puzzle	Agentic	Science		Avg. Rank ↓
	2Wiki	Bamboogle	HotpotQA	Musique	AIME24	AMC23	GameOf24	GAIA	GPQA	MedQA	
Qwen3-0.6B											
AgentFlow	18.00	5.60	7.00	**1.50**	0.00	5.00	0.67	**3.15**	30.33	**42.33**	2.60
SFT Distillation	15.00	5.60	14.00	0.00	**3.33**	2.50	0.00	1.57	30.67	38.33	3.50
Independent GRPO	15.46	**5.69**	9.00	0.52	0.00	**7.50**	**1.03**	1.59	29.45	40.27	2.50
Cascaded GRPO	21.67	4.20	6.00	0.00	**3.33**	5.26	0.00	0.85	23.88	35.51	4.10
CRR-GRPO *(ours)*	**22.50**	4.80	**15.00**	0.50	**3.33**	**7.50**	0.33	1.57	**33.00**	39.00	**2.30**
Qwen3-1.7B											
AgentFlow	21.50	16.80	**20.00**	2.00	**3.33**	20.00	6.33	2.36	31.00	52.67	3.95
SFT Distillation	21.00	18.40	15.00	**3.50**	0.00	15.00	4.67	4.72	31.67	54.67	4.10
Independent GRPO	23.00	18.40	18.00	**3.50**	**3.33**	20.00	7.00	3.15	33.33	56.00	2.65
Cascaded GRPO	**23.50**	**20.80**	17.00	3.00	**3.33**	**22.50**	6.67	4.72	32.67	**56.67**	2.35
CRR-GRPO *(ours)*	**23.50**	20.00	18.00	**3.50**	**3.33**	20.00	**7.33**	**5.51**	**34.33**	55.67	**1.95**
Qwen3-4B-Instruct											
AgentFlow	19.00	35.20	24.00	3.00	**13.33**	32.50	**19.00**	5.51	**43.67**	73.67	3.20
SFT Distillation	18.00	29.60	21.00	3.50	3.33	37.50	13.67	7.09	36.00	66.67	4.65
Independent GRPO	**28.80**	34.50	**27.00**	**5.00**	7.10	**40.00**	15.00	7.10	43.10	75.00	2.45
Cascaded GRPO	21.00	37.60	25.00	**5.00**	10.00	**40.00**	17.00	**7.87**	38.33	70.00	2.60
CRR-GRPO *(ours)*	28.50	**38.40**	25.00	**5.00**	10.00	37.50	17.67	**7.87**	42.67	**81.33**	**2.10**

et al., 2022) 2) Agentic Reasoning containing GAIA (Mialon et al., 2023) 3) Competition Math, containing AIME24 (Art of Problem Solving, n.d.) and AMC23 (Mathematical Association of America, 2023) 4) Arithmetic Puzzles containing GameOf24 (Lightman et al., 2023) 5) Graduate-Level Science, containing GPQA (Rein et al., 2023) and MedQA (Yang et al., 2025b).

Scoring. Final answers are scored by an LLM judge, GPT-4o (Hurst et al., 2024), that returns a JSON-structured boolean equivalence judgment after normalizing math, multiple-choice, and free-form answer formats. For each benchmark, we report accuracy on the official split.

5 Results and Discussion

CRR-GRPO Performance Table 1 reports performance across ten benchmarks and three models, where CRR-GRPO outperforms all baselines. On Qwen3-0.6B, CRR-GRPO achieves an average accuracy of 12.75% compared to the best baseline of 11.36%, a 1.39% improvement. Similar improvements are seen on Qwen3-1.7B (0.03%) and Qwen3-4B-Instruct (1.13%). Interestingly, we observe that the SFT distillation from a larger model is not a sufficient substitute for multi-agent reinforcement learning methods. This suggests that merely imitating teacher trajectories does not necessarily teach the smaller agents how to coordinate under their own interaction distribution. In contrast, the GRPO-based methods optimize agents inside the actual multi-agent pipeline, and CRR-GRPO further improves this process by revisiting agents as partner checkpoints evolve.

Stale-pipeline drift. Figure 2 illustrates the drift, $\hat{D}_i$, defined in §3.3, estimated as the token-level KL between each partner's deployed and stale checkpoints on the last training stage (H=3). Across every model the system drift $\sum_i \hat{D}_i$ is ordered INDEPENDENT > CASCADED > CRR-GRPO. For Qwen3-0.6B, we see the system drift drop from 10.8 for Independent GRPO to 3.9 for Cascaded GRPO to 3.0 for CRR-GRPO. Interestingly, we observe that as the model size grows larger, CRR-GRPO helps more in reducing this drift. While Qwen3-0.6B shows a 23.1% decrease in system drift, Qwen3-1.7B shows 52.0%, and Qwen3-4B-Instruct shows a reduction by 57.1%. The drift for *Verifier* is 0 under Cascaded-GRPO and CRR-GRPO since they are last in the round-robin order with an empty stale set.

Ablating round-robin and curriculum. CRR-GRPO combines two ingredients: Round-Robin Revisitation and an expanding reward prefix. Table 2 isolates each on Qwen3-0.6B. Removing round-robin (each component trained once under the growing reward, then frozen) reduces the average accuracy by 1.1% and raises system drift by

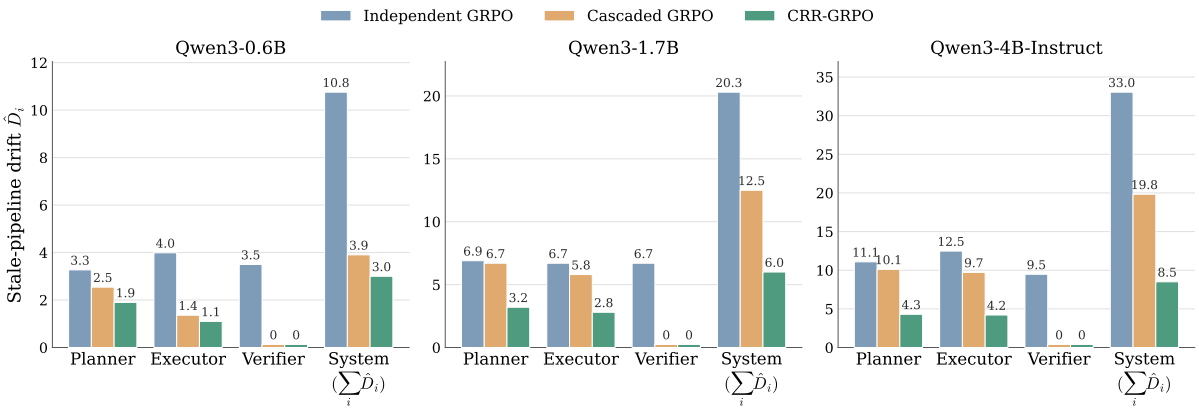

Figure 2: Measured stale-pipeline drift ($\hat{D}_i$; **lower is better**) per trainable component and summed over the system. At every size, INDEPENDENT > CASCADED > CRR-GRPO.

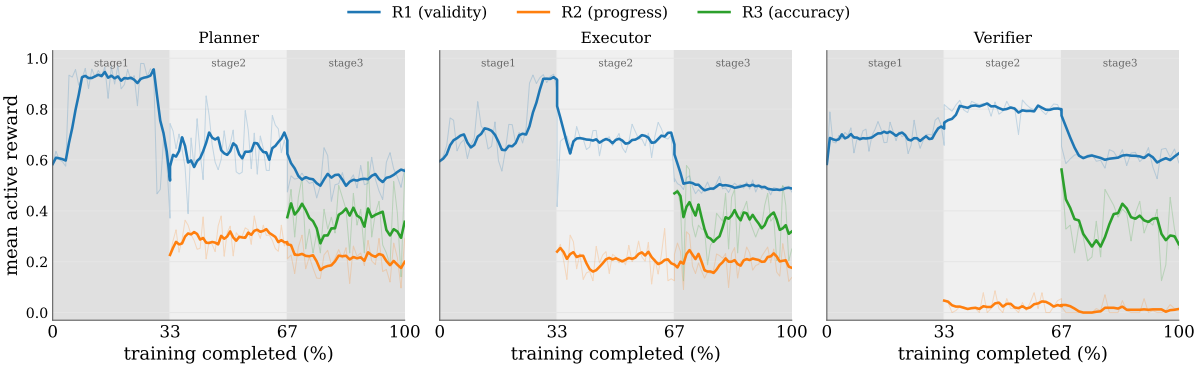

Figure 3: Reward decomposition over CRR-GRPO training (Qwen3-4B-Instruct). Each component is revisited as the reward prefix expands: R_1 (validity) is active throughout, R_2 (progress) is added at the stage-2 boundary (33%), and R_3 (accuracy) at stage 3 (67%). Bold line = trailing mean; faint line = per-step mean.

Ablation (Qwen3-0.6B)	ΔAcc.	ΔDrift
CRR-GRPO (ours)	—	—
– round-robin	−1.1%	+26.7%
– curriculum	−0.3%	+6.7%

Table 2: Ablating CRR-GRPO's two ingredients on Qwen3-0.6B, relative to the full method. ΔAcc. is the change in average accuracy; ΔDrift the relative change in system drift $\sum_i \hat{D}_i$.

26.7%, whereas removing the curriculum (three round-robin passes under the full reward) reduces the accuracy by only 0.3% and increases drift by 6.7%. Both ingredients are vital for CRR-GRPO, however revisitation accounts for a larger share of both effects. The curriculum adds a smaller, complementary gain, and together they yield the best accuracy and the lowest drift.

Reward dynamics under the curriculum. Figure 3 traces the per-component reward terms over CRR-GRPO training. Validity (R_1) is obtained in the first stage and remains the dominant term throughout the expansion. R_2 is introduced at 33% of training and R_3 at 67%. Each new term begins to be optimized while R_1 declines only modestly and stays high, with a brief dip at each stage transition as the components re-equilibrate to the larger objective. The rewards show how each component co-adapts incrementally as reward terms are added rather than overfitting a single static objective, so earlier competencies are largely preserved while later ones are learned.

6 Conclusion

We studied a failure mode in multi-agent LLM training, where agents optimized early in the sequence may later be deployed with partner policies that have changed, producing stale-pipeline drift. To reduce this, we introduced Curriculum Round-Robin GRPO, which revisits agents with progressively expanded reward prefixes, allowing components to adapt as the pipeline evolves. Across ten benchmarks and three models, we achieve consistent improvements under a matched budget.

Limitations

While our method is successful in reducing the drift observed in Multi-Agent Reinforcement Learning (MARL), it represents an early step in this area and has some limitations. Mainly, our evaluation spans three open-source models from the same family, due to high costs of training under MARL. Extending the method to more model sizes and different families is an important direction for future work.

Ethics Statement

We use AI assistants, specifically Grammarly and ChatGPT, to correct grammatical errors and restructure sentences.

References

Anthropic. 2026. Claude opus 4.7 system card. Technical report, Anthropic. Accessed: 2026-05-22.

Art of Problem Solving. n.d. AIME Problems and Solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. AoPS Wiki. Retrieved April 30, 2026.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*.

Kailash Gogineni, Peng Wei, Tian Lan, and Guru Venkataramani. 2023. Towards efficient multi-agent learning systems. *arXiv preprint arXiv:2305.13411*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Aman Gupta, Shao Tang, Qingquan Song, Sirou Zhu, Jiwoo Hong, Ankan Saha, Viral Gupta, Noah Lee, Eunki Kim, Siyu Zhu, et al. 2025. Alphapo: Reward shape matters for llm alignment. *arXiv preprint arXiv:2501.03884*.

Divij Handa, David Blincoe, Orson Adams, and Yinlin Fu. 2025. Optagent: Optimizing query rewriting for e-commerce via multi-agent simulation. *arXiv preprint arXiv:2510.03771*.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. 2025. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625.

Haoyang Hong, Jiajun Yin, Yuan Wang, Jingnan Liu, Zhe Chen, Ailing Yu, Ji Li, Zhiling Ye, Hansong Xiao, Yefei Chen, et al. 2025. Multi-agent deep research: Training multi-agent systems with m-grpo. *arXiv preprint arXiv:2511.13288*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Woosuk Kwon. 2025. *vLLM: An Efficient Inference Engine for Large Language Models*. Ph.D. thesis, UC Berkeley.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008.

Yunxuan Li, Yibing Du, Jiageng Zhang, Le Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. 2024. Improving multi-agent debate with sparse communication topology. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7281–7294.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. 2025. In-the-flow agentic system optimization for effective planning and tool use. *arXiv preprint arXiv:2510.05592*.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. 2025. Marft: Multi-agent reinforcement fine-tuning. *arXiv preprint arXiv:2504.16129*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The twelfth international conference on learning representations*.

Shuo Liu, Zeyu Liang, Xueguang Lyu, and Christopher Amato. 2026. Llm collaboration with multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 32150–32158.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2024. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

Quanfeng Lu, Zhantao Ma, Shuai Zhong, Jin Wang, Dahai Yu, Michael K Ng, and Ping Luo. 2025. Swirl: A staged workflow for interleaved reinforcement learning in mobile gui control. *arXiv preprint arXiv:2508.20018*.

Mathematical Association of America. 2023. American Mathematics Competitions. American Mathematics Competitions. MAA.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711.

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, volume 2025, pages 406–441.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.

Amir Saeidi, Venkatesh Mishra, Souradeep Mukhopadhyay, Gaowen Liu, Ali Payani, Jayanth Srinivasa, and Chitta Baral. 2026. Fama: Failure-aware meta-agentic framework for open-source llms in interactive tool use environments. *arXiv preprint arXiv:2604.25135*.

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. *Advances in neural information processing systems*, 36:68539–68551.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2025. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *International Conference on Learning Representations*, volume 2025, pages 60808–60838.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. 2022. Curriculum learning: A survey. *International Journal of Computer Vision*, 130(6):1526–1565.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, et al. 2026. Rema: Learning to meta-think for llms with multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 38:126621–126667.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439.

Xihuai Wang, Zhicheng Zhang, and Weinan Zhang. 2022. Model-based multi-agent reinforcement learning: Recent progress and prospects. *arXiv preprint arXiv:2203.10603*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*.

Yi Xie, Siao Liu, Falong Fan, Yuanqi Yao, Yue Zhao, and Bo Liu. 2026. Teamtr: Trust-region fine-tuning for multi-agent llm coordination. *arXiv preprint arXiv:2605.15207*.

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

Hang Yang, Hao Chen, Hui Guo, Yineng Chen, Ching-Sheng Lin, Shu Hu, Jinrong Hu, Xi Wu, and Xin Wang. 2025b. Llm-medqa: Enhancing medical question answering through case studies in large language models. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Zehua Zhang, Ati Priya Bajaj, Divij Handa, Siyu Liu, Arvind S Raj, Hongkai Chen, Hulin Wang, Yibo Liu, Zion Leonahenahe Basque, Souradip Nath, et al. 2025. Buildbench: Benchmarking llm agents on compiling real-world open-source software. *arXiv preprint arXiv:2509.25248*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

A Proof of Theorem 1

Consider a component i and any training algorithm $\mathcal{A} \in \{\mathrm{Ind}, \mathrm{Cas}, \mathrm{CRR}\}$. Let

$$\theta_i^* \in \arg\max_{\theta_i} J_i(\theta_i; \psi_{-i})$$

denote an optimal policy for component i under the final deployed partner pipeline ψ_{-i}.

We first bound the change in expected reward induced by replacing the training-time partner snapshot $\phi_{-i}^{\mathcal{A}}$ with the deployed partner snapshot ψ_{-i}, while keeping the component policy θ_i fixed. By definition,

$$\begin{aligned} &\left|J_i(\theta_i; \phi_{-i}^{\mathcal{A}}) - J_i(\theta_i; \psi_{-i})\right| \\ &= \left|\sum_{\tau} R(\tau) \left(P_{\theta_i, \phi_{-i}^{\mathcal{A}}}(\tau) - P_{\theta_i, \psi_{-i}}(\tau)\right)\right|. \end{aligned} \tag{23}$$

Since $|R(\tau)| \leq R_{\max}$ for all trajectories τ, we have

$$\begin{aligned} &\left|J_i(\theta_i; \phi_{-i}^{\mathcal{A}}) - J_i(\theta_i; \psi_{-i})\right| \\ &\leq R_{\max} \sum_{\tau} \left|P_{\theta_i, \phi_{-i}^{\mathcal{A}}}(\tau) - P_{\theta_i, \psi_{-i}}(\tau)\right| \\ &= 2R_{\max} d_{\mathrm{TV}}\left(P_{\theta_i, \phi_{-i}^{\mathcal{A}}}, P_{\theta_i, \psi_{-i}}\right) \\ &\leq 2R_{\max} D_i^{\mathcal{A}}(\phi_{-i}^{\mathcal{A}}, \psi_{-i}), \end{aligned} \tag{24}$$

where the last inequality follows from the definition of $D_i^{\mathcal{A}}$ as a supremum over component policies θ_i.

We now decompose the deployment regret:

$$\begin{aligned} &\mathrm{Reg}_i(\hat{\theta}_i^{\mathcal{A}}; \psi_{-i}) \\ &= J_i(\theta_i^*; \psi_{-i}) - J_i(\hat{\theta}_i^{\mathcal{A}}; \psi_{-i}) \\ &= \left[J_i(\theta_i^*; \psi_{-i}) - J_i(\theta_i^*; \phi_{-i}^{\mathcal{A}})\right] \\ &\quad + \left[J_i(\theta_i^*; \phi_{-i}^{\mathcal{A}}) - J_i(\hat{\theta}_i^{\mathcal{A}}; \phi_{-i}^{\mathcal{A}})\right] \\ &\quad + \left[J_i(\hat{\theta}_i^{\mathcal{A}}; \phi_{-i}^{\mathcal{A}}) - J_i(\hat{\theta}_i^{\mathcal{A}}; \psi_{-i})\right]. \end{aligned} \tag{25}$$

The first and third terms in Eq. 25 are each upper-bounded by Eq. 24. The middle term is upper-bounded by the training-time optimization error,

$$\epsilon_i^{\mathcal{A}} = \sup_{\theta_i} J_i(\theta_i; \phi_{-i}^{\mathcal{A}}) - J_i(\hat{\theta}_i^{\mathcal{A}}; \phi_{-i}^{\mathcal{A}}). \tag{26}$$

Therefore,

$$\begin{aligned} \mathrm{Reg}_i(\hat{\theta}_i^{\mathcal{A}}; \psi_{-i}) &\leq 2R_{\max} D_i^{\mathcal{A}}(\phi_{-i}^{\mathcal{A}}, \psi_{-i}) + \epsilon_i^{\mathcal{A}} \\ &\quad + 2R_{\max} D_i^{\mathcal{A}}(\phi_{-i}^{\mathcal{A}}, \psi_{-i}) \\ &= \epsilon_i^{\mathcal{A}} + 4R_{\max} D_i^{\mathcal{A}}(\phi_{-i}^{\mathcal{A}}, \psi_{-i}). \end{aligned} \tag{27}$$

This proves Theorem 1.

B Derivation of the KL Bound

We derive the KL-based upper bound on the stale-pipeline drift term in Eq. 17. Consider any training algorithm $\mathcal{A}$ and component i. Recall that $\phi_{-i}^{\mathcal{A}}$ denotes the partner snapshot used when component i is trained, while ψ_{-i} denotes the final deployed partner snapshot. The corresponding stale set is

$$S_i^{\mathcal{A}} = \{j \neq i : \phi_{j|i}^{\mathcal{A}} \neq \psi_j\}. \tag{28}$$

We assume that the input distribution and environment dynamics are shared across the two trajectory distributions, that tool outputs are deterministic conditional on the trajectory history, that each stale partner is invoked at most H times in a trajectory, and that each deployed partner policy is absolutely continuous with respect to its corresponding training-time partner policy. For each stale partner j, define

$$\rho_{j|i}^{\mathcal{A}} = \sup_s D_{\mathrm{KL}}\left(\pi_{\psi_j}(\cdot \mid s) \,\|\, \pi_{\phi_{j|i}^{\mathcal{A}}}(\cdot \mid s)\right). \tag{29}$$

By the chain rule for KL divergence over trajectories, the KL divergence between the deployment trajectory distribution and the training-time trajectory distribution decomposes into the sum of conditional action divergences along the trajectory. Partners outside $S_i^{\mathcal{A}}$ contribute zero because their policies are unchanged. Since each stale partner is invoked at most H times, we obtain

$$D_{\mathrm{KL}}\left(P_{\theta_i, \psi_{-i}} \,\|\, P_{\theta_i, \phi_{-i}^{\mathcal{A}}}\right) \leq H \sum_{j \in S_i^{\mathcal{A}}} \rho_{j|i}^{\mathcal{A}}. \tag{30}$$

Pinsker's inequality gives

$$d_{\mathrm{TV}}(P, Q) \leq \sqrt{\frac{1}{2} D_{\mathrm{KL}}(P \,\|\, Q)}. \tag{31}$$

Applying this inequality with $P = P_{\theta_i, \psi_{-i}}$ and $Q = P_{\theta_i, \phi_{-i}^{\mathcal{A}}}$ yields

$$\begin{aligned} &d_{\mathrm{TV}}\left(P_{\theta_i, \phi_{-i}^{\mathcal{A}}}, P_{\theta_i, \psi_{-i}}\right) \\ &= d_{\mathrm{TV}}\left(P_{\theta_i, \psi_{-i}}, P_{\theta_i, \phi_{-i}^{\mathcal{A}}}\right) \\ &\leq \sqrt{\frac{1}{2} D_{\mathrm{KL}}\left(P_{\theta_i, \psi_{-i}} \,\|\, P_{\theta_i, \phi_{-i}^{\mathcal{A}}}\right)} \\ &\leq \sqrt{\frac{H}{2} \sum_{j \in S_i^{\mathcal{A}}} \rho_{j|i}^{\mathcal{A}}}. \end{aligned} \tag{32}$$

Taking the supremum over θ_i gives

$$D_i^{\mathcal{A}}(\phi_{-i}^{\mathcal{A}}, \psi_{-i}) \leq \sqrt{\frac{H}{2} \sum_{j \in S_i^{\mathcal{A}}} \rho_{j|i}^{\mathcal{A}}}. \tag{33}$$

This establishes Eq. 17.

C Prompts for AgentFlow

Below, we define the prompts used for our Multi-Agent System.

C.1 Planner

```
Task: Determine the optimal next
step to address the query using
available tools and previous steps.

Context:
- **Query:** {question}
- **Query Analysis:**
{query_analysis}
- **Available Tools:**
{self.available_tools}
- **Toolbox Metadata:**
{self.toolbox_metadata}
- **Previous Steps:**
{memory.get_actions()}

Instructions:
1. Analyze the query, previous
steps, and available tools.
2. Select the **single best tool**
for the next step.
3. Formulate a specific, achievable
**sub-goal** for that tool.
4. Provide all necessary **context**
(data, file names, variables) for the
tool to function.

Response Format:
1. **Justification:** Explain your
choice of tool and sub-goal.
2. **Context:** Provide all
necessary information for the tool.
3. **Sub-Goal:** State the specific
objective for the tool.
4. **Tool Name:** State the exact
name of the selected tool.

Rules:
- Select only ONE tool.
- The sub-goal must be directly
achievable by the selected tool.
- The Context section must contain
all information the tool needs to
function.
- The response must end with the
Context, Sub-Goal, and Tool Name
sections in that order, with no extra
content.
```

C.2 Executor

```
Task: Generate a precise command to
execute the selected tool.

Context:
- **Query:** {question}
- **Sub-Goal:** {sub_goal}
- **Tool Name:** {tool_name}
- **Tool Metadata:** {tool_metadata}
- **Relevant Data:** {context}

Instructions:
1. Analyze the tool's required
parameters from its metadata.
2. Construct valid Python code that
addresses the sub-goal using the
provided context and data.
3. The command must include at least
one call to `tool.execute()`.
4. Each `tool.execute()` call must
be assigned to a variable named
**`execution`**.
5. Please give the exact numbers
and parameters should be used in the
`tool.execute()` call.

Output Format:
Present your response in the
following structured format. Do
not include any extra text or
explanations.

Generated Command:
` ` `python
<command>
` ` `
```

C.3 Verifier

```
Task: Evaluate if the current memory
is complete and accurate enough to
answer the query, or if more tools
are needed.

Context:

Query: {question}
Available Tools:
{self.available_tools}
Toolbox Metadata:
{self.toolbox_metadata}
Initial Analysis: {query_analysis}
Memory (Tools Used & Results):
{memory.get_actions()}
Instructions:

Review the query, initial analysis,
and memory.
Assess the completeness of the
memory: Does it fully address all
parts of the query?
Check for potential issues:
Are there any inconsistencies or
contradictions?
Is any information ambiguous or in
need of verification?
Determine if any unused tools could
provide missing information.
Final Determination:

If the memory is sufficient, explain
why and conclude with "STOP".
If more information is needed,
explain what's missing, which
tools could help, and conclude with
"CONTINUE".
IMPORTANT: The response must end
with either "Conclusion: STOP" or
"Conclusion: CONTINUE".
```