Improving Multi-Agent Coordination with a Drift-Aware RL Objective

Sangeun Park*, **Guhyeon Kang***, **Minhae Kwon**†
Department of Electrical and Computer Engineering
Sungkyunkwan University, Republic of Korea
{sangeun.park, guhyeon.kang, minhae.kwon}@skku.edu

Abstract

Large language model (LLM)-based multi-agent systems have shown promise for collaborative problem solving, yet maintaining coordination remains challenging. In interactive environments, agents only observe part of the task state, and locally reasonable decisions can gradually lead to misalignment over time, which we refer to as *coordination drift*. To address this problem, we propose **DriCo**, a **dri**ft-aware **co**ordination framework for LLM-based multi-agent systems. DriCo introduces a coordinator that constructs a shared context from agent-level information and guides team-level decision-making. Each agent follows an LLM-based hierarchical policy composed of a planner for sub-goal generation and an actor for low-level Q-guided execution. We formulate coordination as a planning-step sequential process and optimize the coordinator with a drift-derived preference-based objective that favors shared contexts, reducing coordination drift and supporting stable long-horizon execution. We further introduce **LLM-Overcooked**, an LLM-oriented extension of Overcooked-AI with separate training and evaluation environments, diverse layouts and recipes, and held-out configurations for evaluating in-layout coordination generalization to held-out recipe compositions. Experiments show that DriCo improves coordination stability and reduces coordination drift.[1]

1 Introduction

Recent advances in LLMs have enabled a new paradigm of multi-agent systems, in which multiple agents collaborate through natural language to solve complex tasks [2, 26, 46, 47]. Their language capabilities enable natural language as a flexible interface for sharing observations, planning, and coordinating across agents [7, 14, 30]. This opens new opportunities for tackling collaborative tasks that require both distributed decision-making and adaptive interaction among agents.

Despite this progress, sustaining multi-agent cooperation over long horizons remains challenging [34, 38]. This is because coordination unfolds as a sequence of interdependent decisions, where early decisions influence subsequent ones and errors can propagate over time rather than being resolved in a single exchange [33, 36, 44]. As agents operate under partial observability, they cannot fully observe the global task state. Evolving task dependencies can make earlier plans or role assignments outdated, leading to conflicts, redundant work, or looping behaviors when coordination breaks down [4, 17, 19]. Thus, short-term collaboration does not necessarily imply stable long-term coordination.

Figure 1 illustrates common coordination failures, including conflicting actions, redundant sub-goals, and looped dependencies caused by uncoordinated decisions. DriCo (proposed) detects such drift and performs targeted intervention to recover a feasible team-level plan. Since these failures emerge

*Equal contribution. †Corresponding author: Minhae Kwon.
[1]Project page: https://anonymous-projectpage.github.io/DriCo/

ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning.

across context construction, sub-goal planning, and subsequent execution, coordination should be modeled as a planning-step sequential process rather than treated as a one-shot prompting problem.

We refer to the gradual degradation of team-level alignment as ***coordination drift***. Unlike isolated execution errors, coordination drift arises from the accumulation of locally reasonable decisions that increase the risk of conflicts, redundant sub-goals, and recovery failures [15, 25, 35]. This perspective highlights coordination drift as a key factor underlying long-horizon failures, and motivates treating it as a measurable signal for evaluating and improving coordination quality during learning [28, 11].

Existing multi-agent systems induce collaboration through repeated communication, predefined workflows, or agent routing, but rarely optimize coordination stability over time [5, 6, 9, 18, 44]. Under partial observability, individual agents cannot reliably infer team-level intent and progress from local observations alone [29]. Stable coordination, therefore, requires an explicit team-level mechanism that aggregates dispersed local information and maintains shared context, as the environment evolves [27]. This raises a central question:

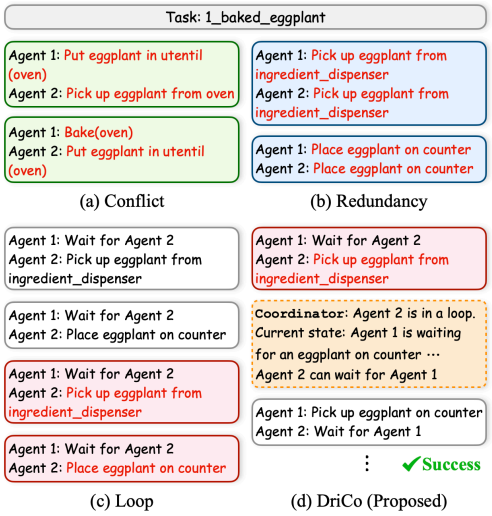


Figure 1: Illustration of coordination drift in multi-agent systems. (a) Conflict, (b) Redundancy, (c) Loop, and (d) DriCo (Proposed)

How can LLM-based multi-agent systems maintain efficient and stable coordination over long horizons in interactive environments?

Motivated by this view, we propose **DriCo**, a drift-aware coordination framework with a learned coordinator for long-horizon multi-agent systems. DriCo uses a coordinator to construct **shared context** and decide when to **intervene**. The resulting shared context guides an LLM-based planner for agent-specific sub-goal generation, and an actor for primitive action execution. The actor further performs value-aware execution using learned Q-value estimates, enabling more stable low-level decision-making under long-horizon environmental dynamics. We optimize the coordinator using **coordination drift** as a preference signal, while improving task performance. We further introduce **LLM-Overcooked**, a new LLM-oriented extension of Overcooked-AI [3]. Unlike prior benchmarks [34], LLM-Overcooked provides separate training and evaluation environments, diverse layouts and recipes, and reference trajectory datasets for learning coordination policies. This design enables controlled evaluation of within-benchmark generalization to unseen task composition.

In summary, our contributions are as follows: (1) We formally define **coordination drift** as a measurable abstraction of long-horizon team-level misalignment, providing a unified metric to capture conflicts, redundancy, and loop over long horizons. (2) We propose **DriCo**, a coordinator-driven framework that constructs a shared context to synchronize hierarchical agents, effectively bridging high-level sub-goal planning with Q-guided low-level action execution. (3) We model drift-aware coordination as a sequential process and train the coordinator with a preference-based objective that favors shared-context updates associated with lower coordination drift. (4) We provide theoretical foundations demonstrating that shared context reduces predictive ambiguity in sub-goal inference and analyze the communication cost of coordinator-mediated updates under sparse invocation. (5) We introduce **LLM-Overcooked**, a new benchmark designed for the systematic study of long-horizon coordination and failure recovery, featuring disjoint training and evaluation recipe sets for controlled evaluation of within-benchmark coordination generalization.

2 Related Works

Recent studies explore LLM-based multi-agent systems for collaborative problem solving [2, 26, 46, 47], utilizing natural language for observation exchange and role specialization [7, 14, 30]. However, repeated dialogue often incurs high communication costs [34, 38] and fails to ensure persistent alignment under partial observability and long horizons [33, 36, 44]. While multi-agent decision-making requires maintaining coordination stability [4, 17, 19], especially under non-stationarity [15, 25, 35], prior works primarily address misalignment through fixed workflows or communication

topologies [5, 6, 9, 18]. Although reinforcement learning (RL) is well-suited for long-horizon coordination [12, 13, 22, 31, 32, 40] and has been integrated to improve individual agent execution [8, 21], coordination itself is rarely optimized as a learnable policy [1, 29], leaving systems vulnerable to long-term misalignment [16, 20, 39, 37, 42]. In contrast, **DriCo** explicitly targets coordination drift by optimizing the coordination policy through drift-aware preference-based objective and sparse shared-context updates. More details are provided in Appendix A.

3 Problem Formulation

3.1 Cooperative Sequential Decision-Making in Multi-Agent Systems

We consider a cooperative multi-agent setting with N environment-interacting agents and an additional coordinator. The agents act in a shared environment to accomplish a shared team objective, while the coordinator does not execute environment actions but constructs shared context to support coordinated decision-making. We formalize this sequential decision-making problem as a decentralized partially observable Markov decision process (Dec-POMDP), defined as $M = \langle \mathcal{I}, \mathcal{S}, \{\mathcal{O}_i\}_{i\in\mathcal{I}}, \{\mathcal{A}_i\}_{i\in\mathcal{I}}, \mathcal{T}, \{\Omega_i\}_{i\in\mathcal{I}}, \mathcal{R}, \gamma\rangle$, where $\mathcal{I} = \{1, 2, \ldots, N\}$ denotes the set of agents, $\mathbf{s}_t \in \mathcal{S}$ denotes the global state at timestep t, $o_{t,i} \in \mathcal{O}_i$ denotes the observation of agent i, and $a_{t,i} \in \mathcal{A}_i$ denotes its action. Each agent has its own action space $\mathcal{A}_i$, which is not necessarily shared across agents. The joint observation is denoted by $\mathbf{o}_t = (o_{t,1}, \ldots, o_{t,N})$ and $\mathbf{a}_t = (a_{t,1}, \ldots, a_{t,N})$, respectively, with the joint action space defined as $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$. The state transition function is given by $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, the observation function for agent i is given by $\Omega_i : \mathcal{S} \to \Delta(\mathcal{O}_i)$, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ denotes the reward, and $\gamma \in [0, 1)$ is the discount factor.

3.2 Model Structures

We introduce an LLM-based multi-agent system consisting of `Coordinator` for team-level coordination and a hierarchical decision-making module composed of agent-level `Planner` and `Actor`.

Team-Level Coordination. We introduce `Coordinator` $\pi_{\theta_{\text{crd}}}$ that supports team-level coordination without directly executing environment actions. At each planning step ℓ, agents send local observations and sub-goals to a communication buffer $\mathcal{C}$, from which `Coordinator` constructs a shared context $\mathbf{c}_\ell$ summarizing the team state and assigns each agent a role ρ_i. The pair $(\mathbf{c}_\ell, \rho_i)$ is then provided to agent i's `Planner`. We model coordination as a planning-step sequential process over planning steps. For `Coordinator` training, we use the drift-derived local score $s(c_\ell) = -D_\ell^{\text{trig}}(\mathbf{c}_\ell)$, where D_ℓ^{trig} is the coordination drift defined in Section 3.3.

Hierarchical Decision-Making. Each agent follows a hierarchical policy consisting of a high-level `Planner` π_{ψ_i} and a low-level `Actor` π_{θ_i}. At planning step ℓ, `Planner` generates an agent-specific sub-goal $g_{\ell,i} \sim \pi_{\psi_i}(\cdot \mid \rho_i, o_{\tau_\ell,i}, \mathbf{c}_\ell)$. During the execution segment following planning step ℓ, `Actor` executes primitive actions $a_{t,i} \sim \pi_{\theta_i}(\cdot \mid o_{t,i}, g_{\ell,i})$. This factorization separates team-aware sub-goal planning from low-level execution. `Actor` is trained with an agent-specific reward $r_{t,i}^{\text{agent}}$ to optimize sub-goal execution $\max_{\theta_i} \mathcal{J}_i^{\text{act}} = \mathbb{E}\big[\sum_t \gamma^t r_{t,i}^{\text{agent}}\big]$.

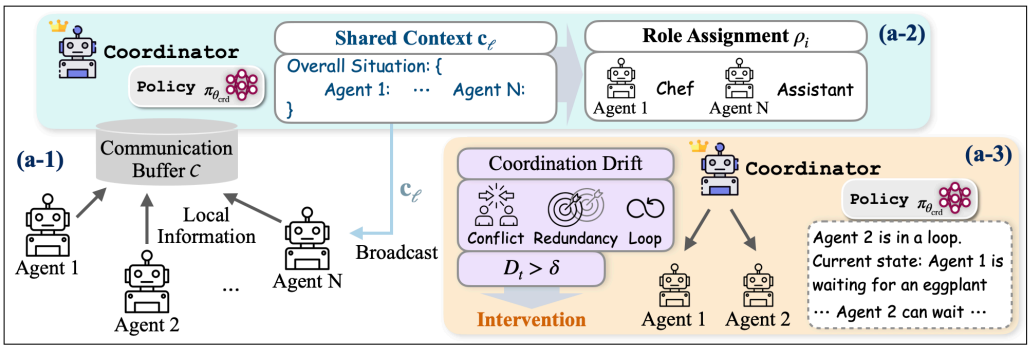
3.3 Coordination Objective with Drift

We introduce a ***coordination drift*** that measures the extent to which the multi-agent system deviates from coordinated behavior over time. In long-horizon collaborative environments, agents may pursue mutually incompatible intentions, duplicate each other's work, or enter cyclic waiting and recovery patterns. We refer to these recurring failure modes as conflict, redundancy, and loop, respectively.

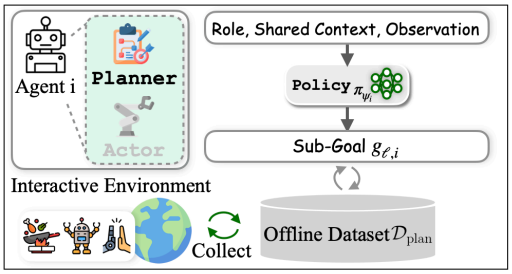
Definition 3.1 (Coordination Drift)**.** Let $\pi_{\theta_{\text{crd}}}$ denote the `Coordinator` policy, and let $\{\pi_{\psi_i}\}_{i=1}^N$ and $\{\pi_{\theta_i}\}_{i=1}^N$ denote the decentralized `Planner` and `Actor` policies, respectively. At planning step ℓ, let τ_ℓ denote the environment timestep at which the ℓ-th planning decision is made. Coordination drift is computed from the candidate sub-goals $\mathbf{g}_\ell = (g_{\ell,1}, \ldots, g_{\ell,N})$, the shared context $\mathbf{c}_\ell$, and the previously executed action history up to τ_ℓ. We define the coordination drift components as $\mathbf{d}_\ell = (d_\ell^{\text{conf}}, d_\ell^{\text{red}}, d_\ell^{\text{loop}})$, where

$$d_\ell^{\text{conf}} = \sum_{i<j} \phi_g(g_{\ell,i}, g_{\ell,j}, \mathbf{c}_\ell),\ d_\ell^{\text{red}} = \sum_{i<j} \mathbb{1}[\kappa(g_{\ell,i}) = \kappa(g_{\ell,j})]\,,\ d_\ell^{\text{loop}} = \sum_{i=1}^{N} \mathbb{1}[\text{Loop}(a_{\tau_\ell - k:\tau_\ell - 1, i})]\,.$$

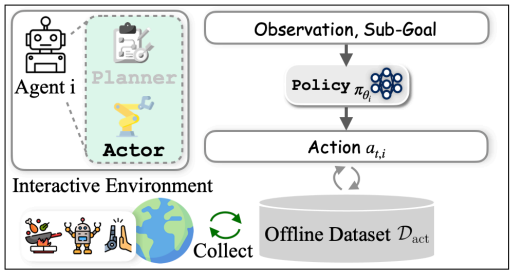
The scalar trigger drift used to invoke intervention is $D_\ell^{\text{trig}} = \lambda_c d_\ell^{\text{conf}} + \lambda_r d_\ell^{\text{red}} + \lambda_l d_\ell^{\text{loop}}$, where $\lambda_c, \lambda_r, \lambda_l \geq 0$. For within-step revision, we define the planning drift as $D_\ell^{\text{plan}} = \lambda_c d_\ell^{\text{conf}} + \lambda_r d_\ell^{\text{red}}$.



(a) DriCo Framework: `Coordinator`-Driven Cooperative Multi-Agent Systems



(b) `Planner`: Context-Aware Planning



(c) `Actor`: Q-Guided Action Execution

Figure 2: Overview of the proposed **DriCo** framework. **(a) DriCo Framework:** `Coordinator`-**Driven Cooperative Multi-Agent Systems. (a-1)** Agents send local information to a communication buffer $\mathcal{C}$. **(a-2)** `Coordinator` aggregates this information to construct a shared context $\mathbf{c}_\ell$ and assigns each agent's role. **(a-3)** When high drift is detected, `Coordinator` intervenes by requesting affected agents to revise sub-goals with updated shared context. **(b)** `Planner`**: Context-Aware Planning.** Conditioned on the role, shared context, and local observation, `Planner` generates an agent-specific sub-goal $g_{\ell,i}$. **(c)** `Actor`**: Q-Guided Action Execution.** Conditioned on the observation and sub-goal, `Actor` policy outputs an action $a_{t,i}$ for low-level value-aware execution.

Here, d_ℓ^{conf} measures pairwise incompatibility between candidate sub-goals under the shared context, d_ℓ^{red} captures redundant sub-goal allocation after mapping natural-language sub-goals to normalized task-level forms via $\kappa(\cdot)$, and d_ℓ^{loop} detects repeated action patterns in the recently executed action history. The function ϕ_g is a task-level incompatibility rule constructed from the training split of reference trajectories and applied unchanged during evaluation. The two scalar scores have distinct roles: D_ℓ^{trig} triggers intervention, while D_ℓ^{plan} evaluates revised sub-goals within the same intervention cycle. D_ℓ^{plan} excludes d_ℓ^{loop} because loop drift depends on past actions and cannot be reduced by resampling current sub-goals. Instead, loop drift serves as a recovery trigger and is re-evaluated after subsequent actions. Operational details are provided in Appendix B.1.

4 DriCo: Drift-Aware Coordination in Cooperative Multi-Agent Systems

To address coordination instability in multi-agent systems, we propose **DriCo**, a drift-aware coordinator-based framework, as illustrated in Figure 2(a). At each planning step, agents send their local information to a communication buffer, from which `Coordinator` constructs a shared context and assigns agent-specific roles. Conditioned on the shared context, role, and local observation, `Planner` generates agent-specific sub-goals. `Actor` then executes low-level actions conditioned on the local observation and the generated sub-goal.

4.1 `Coordinator`: Drift-Aware Coordination

We introduce `Coordinator` as a meta-level control module that constructs shared context from agent-level information, performs drift-aware intervention when coordination quality degrades, and optimizes its coordination policy through preference-based optimization.

Shared Context Construction. At planning step ℓ, `Coordinator` maintains a shared context $\mathbf{c}_\ell$ from agent-level information. The current shared context and role are provided to `Planner` for candidate sub-goal generation. After candidate sub-goals are generated, agents send their local observations and sub-goals to $\mathcal{C}$, forming the coordination state $\mathbf{x}_\ell = (\{o_{\tau_\ell,i}\}_{i=1}^N, \{g_{\ell,i}\}_{i=1}^N)$. This state is used to revise the shared context when intervention is required.

Intervention. Given candidate sub-goals generated by `Planner`, `Coordinator` evaluates the trigger drift D_ℓ^{trig} defined in Definition 3.1. If $D_\ell^{\text{trig}} \leq \delta$, where δ is the intervention threshold, the candidate sub-goals $\{g_{\ell,i}\}_{i=1}^N$ are accepted. Otherwise, `Coordinator` identifies the affected agents $\mathcal{I}_\ell^{\text{aff}}$ involved in the active **drift components**. For revision, `Coordinator` uses the planning drift D_ℓ^{plan}, which includes the components directly reducible by updating the current shared context and sub-goals. It samples a shared context $\mathbf{c}_\ell \sim \pi_{\theta_{\text{crd}}}(\cdot \mid \mathbf{x}_\ell)$ and asks only agents in $\mathcal{I}_\ell^{\text{aff}}$ to regenerate their sub-goals, while the remaining agents keep their current candidates. The revision process follows NEEDREVISION. It returns true if $D_\ell^{\text{plan}} > \delta_{\text{plan}}$, where δ_{plan} denotes the revision threshold for planning drift. If loop drift is active, NEEDREVISION permits at most one loop-triggered recovery update, since the loop component cannot be reduced within the same planning step. Loop drift is treated as a recovery trigger rather than a within-cycle revision objective, because it is computed from previously executed actions and cannot be reduced by resampling the current sub-goals.

Algorithm 1 DriCo: Drift-Aware Coordination via Targeted Intervention

Require: Buffer $\mathcal{C}$, Coordinator $\pi_{\theta_{\text{crd}}}$, Planners $\{\pi_{\psi_i}\}_{i=1}^N$, Actors $\{\pi_{\theta_i}\}_{i=1}^N$, planning steps T_{plan}, thresholds $\delta, \delta_{\text{plan}}$, max revisions $K_{\max}$
Agents send $\{o_{0,i}\}_{i=1}^N$ to $\mathcal{C}$
Sample $\mathbf{c}_0 \sim \pi_{\theta_{\text{crd}}}(\cdot \mid \{o_{0,i}\}_{i=1}^N)$ and $\{\rho_i\}_{i=1}^N$
Broadcast $\mathbf{c}_0$ and roles $\{\rho_i\}_{i=1}^N$
for planning step $\ell \in T_{\text{plan}}$ **do**
  Let τ_ℓ be the current environment timestep
  Sample $g_{\ell,i} \sim \pi_{\psi_i}(\cdot \mid o_{\tau_\ell,i}, \mathbf{c}_\ell, \rho_i)$ for all i
  Agents send $\{g_{\ell,i}\}_{i=1}^N$ to $\mathcal{C}$
  Form $\mathbf{x}_\ell \leftarrow (\{o_{\tau_\ell,i}\}_{i=1}^N, \{g_{\ell,i}\}_{i=1}^N)$
  Compute D_ℓ^{trig} and D_ℓ^{plan} using Definition 3.1
  if $D_\ell^{\text{trig}} > \delta$ **then**
    Identify affected agents $\mathcal{I}_\ell^{\text{aff}}$ and active drift types $\mathcal{M}_\ell$
    $j \leftarrow 0$
    while NEEDREVISION$(D_\ell^{\text{plan}}, \mathcal{M}_\ell)$ **and** $j < K_{\max}$ **do**
      Sample and broadcast revised context $\mathbf{c}_\ell \sim \pi_{\theta_{\text{crd}}}(\cdot \mid \mathbf{x}_\ell)$
      Regenerate $g_{\ell,i} \sim \pi_{\psi_i}(\cdot \mid o_{\tau_\ell,i}, \mathbf{c}_\ell, \rho_i)$ for $i \in \mathcal{I}_\ell^{\text{aff}}$
      Update $\mathbf{x}_\ell$ and recompute D_ℓ^{plan}
      $\mathcal{M}_\ell \leftarrow \mathcal{M}_\ell \setminus \{\text{loop}\}$, $j \leftarrow j + 1$
    end while
  end if
  Accept $\{g_{\ell,i}\}_{i=1}^N$ and execute them until termination or replanning
end for

Preference-Based Policy Optimization. We train `Coordinator` using preference-based optimization to select shared contexts that improve downstream coordination. Given a set of candidate contexts, we define the coordination score $s(\mathbf{c}_\ell) = -D_\ell^{\text{trig}}(\mathbf{c}_\ell)$, where $D_\ell^{\text{trig}}(\mathbf{c}_\ell)$ denotes the trigger drift from Definition 3.1 evaluated under the candidate context $\mathbf{c}_\ell$, using the sub-goals generated from that context. The loop component is shared across candidate contexts at the same planning step because it is computed from previously executed actions. Each candidate context is scored by computing the trigger drift induced by the sub-goals generated under that context. Contexts with higher scores are preferred, yielding pairwise preferences $(\mathbf{c}_\ell^w, \mathbf{c}_\ell^l)$ with $s(\mathbf{c}_\ell^w) > s(\mathbf{c}_\ell^l)$. We then optimize `Coordinator` as follows.

$$\mathcal{L}_{\text{crd}}(\theta_{\text{crd}}) = -\mathbb{E}_{(\mathbf{x}_\ell, \mathbf{c}_\ell^w, \mathbf{c}_\ell^l) \sim \mathcal{D}_{\text{crd}}} \left[\log \sigma \left(\beta \log \frac{\pi_{\theta_{\text{crd}}}(\mathbf{c}_\ell^w \mid \mathbf{x}_\ell)}{\pi_{\theta_{\text{crd}}}^{\text{ref}}(\mathbf{c}_\ell^w \mid \mathbf{x}_\ell)} - \beta \log \frac{\pi_{\theta_{\text{crd}}}(\mathbf{c}_\ell^l \mid \mathbf{x}_\ell)}{\pi_{\theta_{\text{crd}}}^{\text{ref}}(\mathbf{c}_\ell^l \mid \mathbf{x}_\ell)} \right) \right] \quad (1)$$

Herein, $\pi_{\theta_{\text{crd}}}^{\text{ref}}$ denotes the reference coordinator policy, β controls the strength of preference optimization, and $\mathcal{D}_{\text{crd}}$ is the coordinator preference dataset. This objective increases the likelihood of shared contexts that reduce coordination drift, which supports downstream task progress. Therefore, `Coordinator` is optimized to select shared contexts that lead to more coherent team-level plans and support downstream task progress.

4.2 Agent-Level Hierarchical Framework

`Planner`: Context-Aware Planning. Let τ_ℓ denote the environment timestep at which the ℓ-th planning decision is made. At planning step ℓ, given the assigned role ρ_i, shared context $\mathbf{c}_\ell$, and

local observation $o_{\tau_\ell,i}$, `Planner` generates an agent-specific sub-goal $g_{\ell,i} \sim \pi_{\psi_i}(\cdot \mid \rho_i, \mathbf{c}_\ell, o_{\tau_\ell,i})$. As illustrated in Figure 2(b), `Planner` leverages $\mathbf{c}_\ell$ to produce sub-goals that are consistent with team-level coordination, while respecting the assigned role and local task constraints. The generated sub-goals are provided to `Coordinator`, which performs drift-aware validation and may request revision when coordination quality degrades.

`Planner` is trained with supervised fine-tuning (SFT) on reference sub-goals derived from offline trajectories. Given an offline dataset $\mathcal{D}_{\text{plan}}$, where each example consists of $(\rho_i, \mathbf{c}_\ell, o_{\tau_\ell,i}, g_{\ell,i})$, we optimize the following negative log-likelihood objective.

$$\mathcal{L}_{\text{plan}}(\psi_i) = -\mathbb{E}_{(\rho_i,\mathbf{c}_\ell,o_{\tau_\ell,i},g_{\ell,i})\sim\mathcal{D}_{\text{plan}}}\left[\log \pi_{\psi_i}(g_{\ell,i} \mid \rho_i, \mathbf{c}_\ell, o_{\tau_\ell,i})\right] \tag{2}$$

This objective trains `Planner` to imitate reference sub-goals aligned with coordinated behaviors in $\mathcal{D}_{\text{plan}}$, conditioned on the role, shared context, and local observation.

Theorem 4.1 (Perplexity Reduction over Sub-Goals via Shared Context)**.** *Let $g^\star_{\ell,-i}$ denote the reference sub-goals of agents other than agent i. Then conditioning on the **shared context** $\mathbf{c}_\ell$ cannot increase the perplexity (PPL) of $g^\star_{\ell,-i}$, i.e.,*

$$\text{PPL}(g^\star_{\ell,-i} \mid o_{\tau_\ell,i}, \mathbf{c}_\ell) \le \text{PPL}(g^\star_{\ell,-i} \mid o_{\tau_\ell,i}).$$

Herein, the conditional perplexity is defined as $\text{PPL}(g^\star_{\ell,-i} \mid y) = \exp\Big(H(g^\star_{\ell,-i} \mid y)\Big)$.

This result should be interpreted as an information-theoretic motivation for using shared context. It formalizes that shared context can reduce predictive uncertainty over teammates' reference sub-goals when $\mathbf{c}_\ell$ contains coordination-relevant information.

Proof. See Appendix B.2. □

`Actor`**: Q-Guided Action Execution.** Given the local observation $o_{t,i}$ and the accepted sub-goal $g_{\ell,i}$, `Actor` executes primitive actions according to $a_{t,i} \sim \pi_{\theta_i}(\cdot \mid o_{t,i}, g_{\ell,i})$. `Actor` is responsible for adapting to fine-grained environmental dynamics while following the high-level sub-goal generated by `Planner`. In addition to primitive actions, `Actor` outputs a termination signal, denoted by the `<done>` tag. Once `<done>` is emitted, the signal is sent to `Planner` and `Coordinator`, triggering sub-goal replanning with the latest buffered information. `Coordinator` updates the shared context, and `Planner` generates a new sub-goal conditioned on the revised context.

Algorithm 2 DriCo: Training Procedure

Require: $\pi^{\text{ref}}_{\theta_{\text{crd}}}$, $\{\pi^{\text{ref}}_{\theta_i}\}_{i=1}^N$, critics $\{Q_{\omega_i}\}_{i=1}^N$, $\mathcal{C}$, offline dataset $\mathcal{D}_{\text{crd}}$, $\mathcal{D}_{\text{plan}}$, $\mathcal{D}_{\text{act}}$, epoch $\text{EP}_1, \text{EP}_2, \text{EP}_3$
Init $\pi_{\theta_{\text{crd}}} \leftarrow \pi^{\text{ref}}_{\theta_{\text{crd}}}$, $\{\pi_{\psi_i}\}_{i=1}^N$, $\{\pi_{\theta_i}\}_{i=1}^N \leftarrow \{\pi^{\text{ref}}_{\theta_i}\}_{i=1}^N$
for $e_1 = 1$ to EP_1 **do**
  Update $\pi_{\theta_{\text{crd}}}$ on $\mathcal{D}_{\text{crd}}$ via Eq.(1)
end for `// Coordinator Training`
for $e_2 = 1$ to EP_2 **do**
  Update π_{ψ_i} on $\mathcal{D}_{\text{plan}}$ via Eq. (2)
end for `// Planner SFT`
for $e_3 = 1$ to EP_3 **do**
  Update $\{Q_{\omega_i}\}_{i=1}^N$ on $\mathcal{D}_{\text{act}}$ via Eq. (26)
  Construct $\mathcal{D}^{\text{pref}}_{\text{act}}$ using $\{Q_{\omega_i}\}_{i=1}^N$
  Update $\{\pi_{\theta_i}\}_{i=1}^N$ on $\mathcal{D}^{\text{pref}}_{\text{act}}$ via Eq. (3)
end for `// Actor Preference RL`

To train `Actor`, we adopt a preference-based actor–critic framework with sub-goal-conditioned value learning. Unlike prompting-only action generation, our `Actor` explicitly incorporates sub-goal-conditioned value estimates to optimize low-level execution with respect to long-term task progress. We further introduce a *Q-guided preference optimization mechanism* that aligns low-level action generation with sub-goal-specific value estimates. For notational simplicity, let $\zeta_{t,i} = (o_{t,i}, g_{\ell,i})$ denote the actor input. Each agent maintains a sub-goal-conditioned action-value function $Q_{\omega_i}(\zeta_{t,i}, a)$. During training, action preferences $(a^w_{t,i}, a^l_{t,i})$ are constructed by sampling candidate actor outputs and ranking them with Q_{ω_i}. Given the resulting actor preference dataset $\mathcal{D}^{\text{pref}}_{\text{act}}$, we optimize the actor policy using the following objective.

$$\mathcal{L}_{\text{act}}(\theta_i) = -\mathbb{E}_{(\zeta_{t,i},a^w_{t,i},a^l_{t,i})\sim\mathcal{D}^{\text{pref}}_{\text{act}}}\left[\log\sigma\left(\beta\log\frac{\pi_{\theta_i}(a^w_{t,i} \mid \zeta_{t,i})}{\pi^{\text{ref}}_{\theta_i}(a^w_{t,i} \mid \zeta_{t,i})} - \beta\log\frac{\pi_{\theta_i}(a^l_{t,i} \mid \zeta_{t,i})}{\pi^{\text{ref}}_{\theta_i}(a^l_{t,i} \mid \zeta_{t,i})}\right)\right] \tag{3}$$

This objective encourages `Actor` to favor actions with higher sub-goal-conditioned value estimates, thereby improving the consistency between high-level plans and low-level execution. Details on critic learning, candidate sampling, and agent-specific rewards are provided in Appendix C.

4.3 Communication Cost Analysis

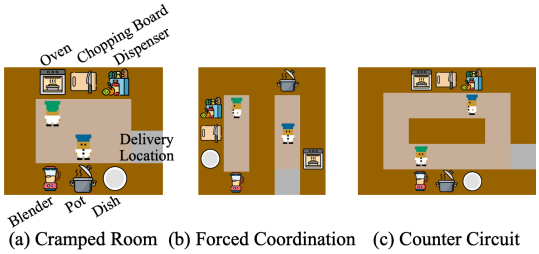
DriCo replaces dense pairwise discussion with sparse coordinator-mediated shared-context updates. Let K_{disc}, K_{up}, and K_{coor} denote the maximum pairwise, agent-to-coordinator, and coordinator-to-agent message sizes, respectively. Dense discussion incurs $C_{\text{disc}} = \mathcal{O}(TN^2K_{\text{disc}})$ over an episode of horizon T. DriCo invokes the `Coordinator` only at sub-goal planning or `Actor`-triggered recovery steps, yielding $C_{\text{coor}} = \mathcal{O}(HN(K_{\text{up}} + K_{\text{coor}}))$, where H is the number of coordinator invocations. When $H \ll T$, this reduces repeated communication relative to dense discussion-based coordination. The detailed derivation is provided in Appendix B.3.

5 Experiments

We evaluate whether DriCo can maintain stable coordination in long-horizon multi-agent interaction. Additional results are provided in Appendix J. Accordingly, our experiments aim to answer three questions: (1) Does DriCo improve long-horizon task performance across LLM-Overcooked difficulty levels? (2) Does DriCo reduce coordination drift? (3) Which components, including shared context, RL training, and intervention, contribute to the gains?

5.1 Experimental Setups

Benchmark. We introduce and evaluate DriCo on **LLM-Overcooked**, a new LLM-oriented benchmark we build for training and evaluating long-horizon multi-agent coordination. Built on Overcooked-AI [3], LLM-Overcooked redesigns the environment for language-conditioned goals, structured recipes, and communication through language. Compared with Collab-Overcooked [34], it increases variation through more diverse layouts and recipes, provides separate training and evaluation environments, and includes reference datasets for learning coordination policies. Additional benchmark details and comparisons are provided in Appendix D.



(a) Cramped Room (b) Forced Coordination (c) Counter Circuit

Figure 3: Example layouts in LLM-Overcooked.

Dataset and Backbones. We train `Coordinator`, `Planner`, and `Actor` on trajectories generated from a unified simulation pipeline, and release the datasets for fair comparison. Our framework is built on the instruction-tuned open-source models Qwen-2.5 7B [41] and Llama-3.1 8B [24] on an NVIDIA RTX PRO 6000 Blackwell GPU. Details are provided in Appendix E and Appendix F.

Metric. We evaluate performance with success rate and progress completeness. Success rate measures the fraction of completed episodes, while progress completeness captures recipe milestone completion within an episode, considering partial progress in long-horizon failures. Details are provided in Appendix G.

5.2 Baselines

We compare DriCo with representative baselines. Additional details are provided in Appendix H.

- **RoCo [23]:** A dialogue-based coordination method that uses LLM-mediated discussion, environment feedback, and re-planning before executing validated sub-task plans.
- **ProAgent [43]:** A proactive planning method that infers teammate intentions, uses memory and skill verification, and corrects beliefs from observed teammate behavior.
- **Collab-Overcooked [34]:** An in-context learning baseline built on the agent prompts provided by Collab-Overcooked for natural-language multi-agent coordination.
- **CoELA [45]:** A modular embodied-agent framework combining perception, memory, communication, planning, and execution for cooperative long-horizon tasks.
- **DriCo (Proposed):** DriCo learns a drift-aware coordinator that constructs shared context, assigns roles, and reduces conflicts, redundant sub-goals, and looping behaviors over long horizons.

5.3 Overall Performance Analysis

Table 1 compares DriCo with baseline methods across LLM-Overcooked difficulty levels. DriCo achieves overall best performance across task layouts and backbone models, with larger gains on harder levels (L4–L6), where long-horizon dependencies make coordination errors more likely to accumulate. While baselines often make partial progress, their success rates drop sharply as task difficulty increases, suggesting difficulty in maintaining stable shared understanding over extended

Table 1: Performance across LLM-Overcooked task difficulty levels. Each entry reports Success Rate [%] / Progress Completeness [%]. Results are grouped by task layout and backbone model.

Task	Backbone	Method	Success Rate [%] / Progress Completeness [%]						
			L1	L2	L3	L4	L5	L6	Avg.
Cramped Room	Qwen-2.5 7B	RoCo [23]	20/43	0/62	0/37	0/31	0/26	0/14	3/36
		ProAgent [43]	17/77	17/45	0/29	0/25	0/17	0/23	6/36
		Collab-Overcooked [34]	60/87	0/31	0/19	0/32	0/28	0/18	10/36
		CoELA [45]	40/95	0/47	0/23	0/28	0/13	0/13	7/37
		DriCo (Proposed)	**80/86**	**100/100**	**40/55**	**40/55**	**20/40**	**0/26**	**47/60**
	Llama-3.1 8B	RoCo [23]	20/80	0/47	0/50	0/47	0/36	0/24	3/47
		ProAgent [43]	17/81	0/51	0/50	0/47	0/23	0/21	3/46
		Collab-Overcooked [34]	0/91	0/25	0/36	0/34	0/11	0/20	0/36
		CoELA [45]	0/75	20/71	0/45	0/36	0/35	0/19	3/47
		DriCo (Proposed)	**100/100**	0/33	**20/42**	**60/69**	0/24	**0/33**	**30/50**
Forced Coordination	Qwen-2.5 7B	RoCo [23]	53/96	6/51	0/32	0/31	0/30	0/22	10/44
		ProAgent [43]	50/83	50/75	0/58	0/44	0/29	0/24	17/52
		Collab-Overcooked [34]	8/45	0/13	0/9	0/8	0/6	0/5	1/14
		CoELA [45]	0/53	0/37	0/44	0/21	0/17	0/22	0/32
		DriCo (Proposed)	**100/100**	**100/100**	**67/91**	**65/60**	**0/32**	**0/34**	**55/70**
	Llama-3.1 8B	RoCo [23]	26/93	0/54	0/32	0/31	0/30	0/22	4/44
		ProAgent [43]	50/78	33/40	0/32	0/36	0/28	0/21	14/39
		Collab-Overcooked [34]	4/33	0/15	0/12	0/11	0/9	0/7	1/15
		CoELA [45]	40/75	0/31	0/29	0/21	0/11	0/10	7/30
		DriCo (Proposed)	**100/100**	**65/83**	**60/70**	**20/40**	**0/34**	**0/26**	**41/59**

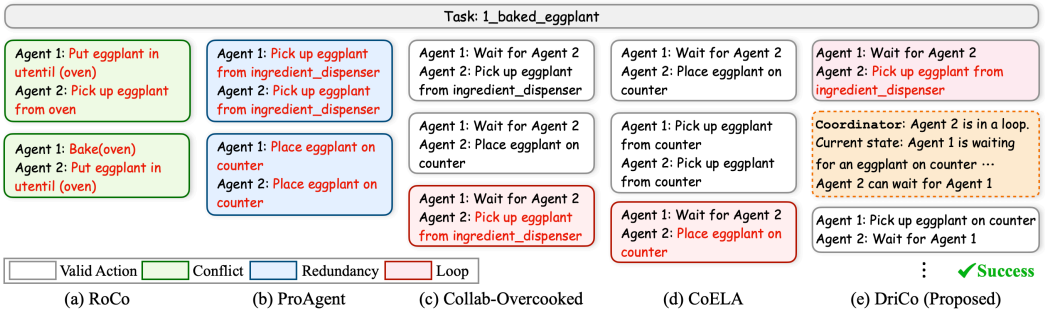


Figure 5: Qualitative case study on `Baked Eggplant` task. Box colors indicate action types: gray denotes valid actions, red denotes conflict, green denotes redundancy, and yellow denotes loop.

episodes. Overall, these results indicate that DriCo improves multi-agent performance by stabilizing the coordination process, rather than only improving individual action generation.

5.4 Coordination Drift Analysis

To evaluate coordination quality, we measure detected and unresolved **coordination drift** across task-difficulty levels. Figure 4 decomposes detected drift into conflict d_ℓ^{conf}, redundancy d_ℓ^{red}, and loop behavior d_ℓ^{loop}, and reports unresolved drift after `Coordinator` intervention. Since drift is also used for training, this analysis serves as a diagnostic of whether `Coordinator` corrects the intended failure modes. Detected drift increases with task difficulty, mainly due to loops, but unresolved drift remains much lower. This suggests that **shared context** and **intervention** recover many long-horizon coordination failures before they propagate.

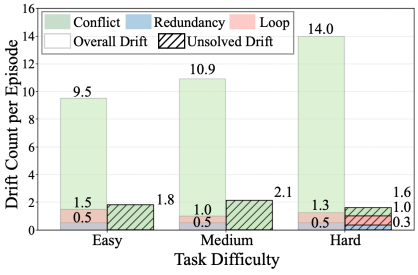


Figure 4: Coordination drift across task levels. Solid bars show detected conflict, redundancy, and loop drift; hatched bars show unresolved drift after `Coordinator` intervention.

5.5 Qualitative Case Studies

We analyze representative coordination failures and recovery behaviors in the `1_baked_eggplant` task. Figure 5 compares baseline drift patterns with DriCo's recovery process. DriCo detects the

Table 2: Ablation results on the Forced Coordination environment across task difficulty levels. Each entry reports Success Rate / Progress Completeness. Component columns indicate whether shared context construction, intervention, and preference-based model training are enabled.

	Components			Success Rate [%] / Progress Completeness [%]						
Variant	**Shared Context**	**Intervention**	**Training**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**Avg.**
w/o Shared Context	✗	✓	✓	70/75	60/51	45/51	20/42	0/31	0/27	33/46
w/o Intervention	✓	✗	✓	92/96	83/90	60/90	58/72	0/24	0/40	49/69
w/o Optimization	✓	✓	✗	100/100	92/96	60/80	60/65	0/20	0/30	52/65
DriCo (Proposed)	✓	✓	✓	**100/100**	**100/100**	**67/91**	**65/60**	**0/32**	0/34	**55/70**

loop state and triggers `Coordinator` **intervention**. In contrast, RoCo [23] produces conflicting oven-related sub-goals, while ProAgent [43] generates redundant ingredient-handling actions. Collab-Overcooked [34] and CoELA [45] fall into repeated wait-and-place behaviors caused by unresolved task dependencies. In DriCo, `Coordinator` reconstructs the shared context, allowing one agent to retrieve the eggplant from the counter while the other waits for the handoff. This case shows that DriCo provides an explicit recovery mechanism that mitigates coordination drift.

5.6 Ablation Studies

Coordinator Components. To assess the contribution of `Coordinator`'s components, we ablate shared context construction, intervention, and preference-based model training in the Forced Coordination environment. As shown in Table 2, removing any component degrades performance. The largest drop occurs without shared context, confirming its importance for maintaining team-level alignment. Removing intervention reduces recovery from emerging drift during long-horizon execution, while removing preference-based training mainly hurts harder tasks, where coordinated context selection and value-aware action execution become more critical. Overall, these results show that the three components are complementary and jointly enable robust long-horizon coordination.

Actor Q-Guided Optimization. We further examine the effect of Q-guided preference optimization for `Actor`. Removing this component preserves the same high-level coordination structure but weakens low-level action selection under each assigned sub-goal. The performance gap becomes larger on harder tasks, suggesting that value-aware execution is especially useful when longer dependency chains require agents to complete sub-goals reliably before replanning.

Table 3: Effect of Q-guided `Actor`.

Variant	Easy	Medium	Hard	Avg.
w/o Q-guided	80/87	40/63	0/32	40/61
DriCo (Proposed)	**100/100**	**66/76**	**0/33**	**55/70**

5.7 Intervention Effect Analysis

We further analyze whether **intervention** helps recover from coordination failures. Figure 6 shows Progress Completeness over the episode horizon, reporting the mean over 3 random seeds with one-standard-deviation shading. DriCo improves more steadily after intervention-triggered points, whereas the no-intervention variant plateaus earlier, indicating that `Coordinator` restores progress by updating the shared context in high-drift states.

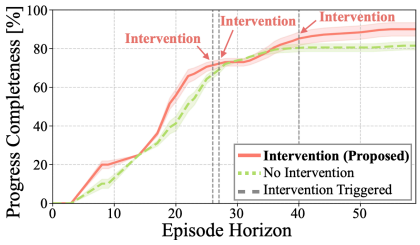


Figure 6: Effect of intervention. Red solid: Intervention; green dashed: No Intervention.

6 Conclusions

We propose **DriCo**, a drift-aware coordination framework that uses coordination drift as an explicit optimization signal for persistent alignment in cooperative multi-agent systems. By integrating shared context construction with hierarchical decision-making, DriCo mitigates long-horizon failure modes—such as conflicts, redundancies, and loop behaviors—that typically emerge. We also introduce **LLM-Overcooked**, a benchmark for evaluating coordination generalization across diverse layouts and recipes. Beyond immediate performance gains, DriCo redefines multi-agent coordination as a learnable, sequential decision-making process rather than a static prompting task. This shift provides a critical foundation for the next generation of planning research, offering scalable pathways for developing decentralized LLM swarms and self-evolving orchestrators capable of robust collaboration in complex, interactive environments.

References

[1] Z. Bi, M. Lu, Y. Li, S. Roy, W. Guan, M. Ziyadi, and X. Wang. OPTAGENT: Optimizing multi-agent LLM interactions through verbal reinforcement learning for enhanced reasoning. In *International Joint Conference on Natural Language Processing and the Asia-Pacific Chapter of the Association for Computational Linguistics (IJCNLP-AACL)*, 2025.

[2] X. Bo, Z. Zhang, Q. Dai, X. Feng, L. Wang, R. Li, X. Chen, and J. Wen. Reflective multi-agent collaboration based on large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[3] M. Carroll, R. Shah, M. K. Ho, T. Griffiths, S. Seshia, P. Abbeel, and A. Dragan. On the utility of learning about humans for human-AI coordination. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[4] B. Chen, G. Li, X. Lin, Z. Wang, and J. Li. BlockAgents: Towards byzantine-robust LLM-based multi-agent coordination via blockchain. In *ACM Turing Award Celebration Conference (ACM-TURC)*, 2024.

[5] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[6] Y. Dang, C. Qian, X. Luo, J. Fan, Z. Xie, R. Shi, W. Chen, C. Yang, X. Che, Y. Tian, X. Xiong, L. Han, Z. Liu, and M. Sun. Multi-agent collaboration via evolving orchestration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[7] A. Estornell and Y. Liu. Multi-LLM debate: Framework, principals, and interventions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[8] P. Feng, Y. He, G. Huang, Y. Lin, H. Zhang, Y. Zhang, and H. Li. AGILE: A novel reinforcement learning framework of LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[9] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

[10] E. Hu, Y. Shen, P. Wallis, Z. Allen, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[11] Q. Kou, M. Li, Z. Liu, L. Qian, Z. Chen, L. Wan, X. Chen, and X. Lan. Offline multi-agent preference-based reinforcement learning with agent-aware direct preference optimization. In *Autonomous Agents and Multiagent Systems*, 2025.

[12] D. Lee and M. Kwon. Episodic future thinking mechanism for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[13] D. Lee and M. Kwon. Temporal distance-aware transition augmentation for offline model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2025.

[14] H. Leong, Y. Li, Y. Wu, W. Ouyang, W. Zhu, J. Gao, and W. Han. AMAS: Adaptively determining communication topology for LLM-based multi-agent system. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[15] C. Li, B. BAO, and Y. Gao. In-context fully decentralized cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[16] H. Li, H. Mahjoub, B. Chalaki, V. Tadiparthi, K. Lee, E. Moradi-Pari, M. Lewis, and K. Sycara. Language grounded multi-agent reinforcement learning with human-interpretable communication. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[17] L. Li, L. Yuan, P. Liu, T. Jiang, and Y. Yu. LLM-assisted semantically diverse teammate generation for efficient multi-agent coordination. In *International Conference on Machine Learning (ICML)*, 2025.

[18] D. Liu, S. Kato, W. Gu, F. Ren, J. Yan, and G. Su. Integrating suboptimal human knowledge with hierarchical reinforcement learning for large-scale multiagent systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[19] H. Liu, L. Chen, Y. Qiao, C. Lv, and H. Li. Reasoning multi-agent behavioral topology for interactive autonomous driving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[20] Y. Liu, C. Xu, L. Liu, Y. Wang, F. Chen, Q. Jia, Y. Zhao, Z. Wang, and X. Li. DeMAC: Enhancing multi-agent coordination with dynamic DAG and manager-player feedback. In *Findings of Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[21] H. Ma, T. Hu, Z. Pu, B. Liu, X. Ai, Y. Liang, and M. Chen. Coevolving with the other you: Fine-tuning LLM with sequential cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[22] H. Ma, S. Wang, Z. Pu, S. Zhao, and X. Ai. Vision-based generic potential function for policy alignment in multi-agent reinforcement learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2025.

[23] Z. Mandi, S. Jain, and S. Song. RoCo: Dialectic multi-robot collaboration with large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[24] Meta AI. Introducing meta llama 3: The most capable openly available llm to date. `https://ai.meta.com/blog/meta-llama-3/`, 2024. Accessed: 2026-01-04.

[25] S. Nayak, A. Orozco, M. Have, J. Zhang, V. Thirumalai, D. Chen, A. Kapoor, E. Robinson, K. Gopalakrishnan, J. Harrison, A. Mahajan, B. Ichtar, and H. Balakrishnan. Long-horizon planning for multi-agent robots in partially observable environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[26] C. Park, S. Han, X. Guo, A. Ozdaglar, K. Zhang, and J. Kim. MAPoRL: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Association for Computational Linguistics (ACL)*, 2025.

[27] D. Qiu, D. Xu, and L. Yue. Reinforcement learning-augmented LLM agents for collaborative decision making and performance optimization. In *International Conference on Frontier Technologies of Information and Computer (ICFTIC)*, 2025.

[28] A. Rath. Agent Drift: Quantifying behavioral degradation in multi-agent LLM systems over extended interactions. *arXiv preprint arXiv:2601.04170*, 2026.

[29] S. Seo, S. Lim, S. Noh, H. Kim, and H. Kang. From assumptions to actions: Turning LLM reasoning into uncertainty-aware planning for embodied agents. In *International Conference on Learning Representations (ICLR)*, 2026.

[30] X. Shen, Y. Liu, Y. Dai, Y. Wang, R. Miao, Y. Tan, S. Pan, and X. Wang. Understanding the information propagation effects of communication topologies in LLM-based multi-agent systems. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[31] H. Singh, R. Das, M. Han, P. Nakov, and I. Laptev. MALMM: Multi-agent large language models for zero-shot robotic manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

[32] S. Singh, Z. Huang, A. Srinivasan, G. Gutow, B. Vundurthy, and H. Choset. Hierarchical planning for long-horizon multi-agent collective construction. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[33] A. Smit, N. Grinsztajn, P. Duckworth, T. Barrett, and A. Pretorius. Should we be going mad? a look at multi-agent debate strategies for LLMs. In *International Conference on Machine Learning (ICML)*, 2024.

[34] H. Sun, S. Zhang, L. Niu, L. Ren, H. Xu, H. Fu, F. Zhao, C. Yuan, and X. Wang. Collab-Overcooked: Benchmarking and evaluating large language models as collaborative agents. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[35] L. Sun, D. Jha, C. Hori, S. Jain, R. Corcodel, X. Zhu, M. Tomizuka, and D. Romeres. Interactive planning using large language models for partially observable robotic tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[36] H. Wang, S. Zhao, J. Wang, Z. Qiang, B. Qin, and T. Liu. Beyond frameworks: Unpacking collaboration strategies in multi-agent systems. In *Association for Computational Linguistics (ACL)*, 2025.

[37] Y. Wang, G. Lucas, B. Becerik-Gerber, and V. Ustun. Implicit behavioral alignment of language agents in high-stakes crowd simulations. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[38] Z. Wang, Y. Wang, X. Liu, L. Ding, M. Zhang, J. Liu, and M. Zhang. AgentDropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In *Association for Computational Linguistics (ACL)*, 2025.

[39] Z. Wu and T. Ito. The hidden strength of disagreement: Unraveling the consensus-diversity tradeoff in adaptive multi-agent systems. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[40] Z. Xi, J. Huang, C. Liao, B. Huang, J. Liu, H. Guo, Y. Yang, R. Zheng, J. Ye, J. Zhang, W. Chen, W. He, Y. Ding, G. Li, Z. Chen, Z. Du, X. Yao, Y. Xu, J. Chen, T. Gui, Z. Wu, Q. Zhang, X. Huang, and Y. Jiang. Agentgym-RL: An open-source framework to train LLM agents for long-horizon decision making via multi-turn RL. In *International Conference on Learning Representations (ICLR)*, 2026.

[41] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

[42] J. Yao, K. Wang, R. Hsieh, H. Zhou, T. Zou, Z. Cheng, Z. Wang, and P. Viswanath. SPIN-bench: How well do LLMs plan strategically and reason socially? In *Conference on Language Modeling (CoLM)*, 2025.

[43] C. Zhang, K. Yang, S. Hu, Z. Wang, G. Li, Y. Sun, C. Zhang, Z. Zhang, A. Liu, S.-C. Zhu, X. Chang, J. Zhang, F. Yin, Y. Liang, and Y. Yang. ProAgent: Building proactive cooperative agents with large language models. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2024.

[44] G. Zhang, Y. Yue, Z. Li, S. Yun, G. Wan, K. Wang, D. Cheng, J. Yu, and T. Chen. Cut the Crap: An economical communication pipeline for LLM-based multi-agent systems. In *International Conference on Learning Representations (ICLR)*, 2025.

[45] H. Zhang, W. Du, J. Shan, Q. Zhou, Y. Du, J. B. Tenenbaum, T. Shu, and C. Gan. Building cooperative embodied agents modularly with large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[46] W. Zhou, M. Mesgar, A. Friedrich, and H. Adel. Efficient multi-agent collaboration with tool use for online planning in complex table question answering. In *Findings of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, 2025.

[47] K. Zhu, H. Du, Z. Hong, X. Yang, S. Guo, Z. Wang, Z. Wang, C. Qian, X. Tang, H. Ji, and J. You. MultiAgentBench : Evaluating the collaboration and competition of LLM agents. In *Association for Computational Linguistics (ACL)*, 2025.

Appendix: DriCo: Drift-Aware Coordination for Long-Horizon Multi-Agent Collaboration

Contents

A Related Works

LLM-based Multi-Agent Collaboration. Recent studies have explored LLM-based multi-agent systems for collaborative problem solving [2, 26, 46, 47], where agents exchange observations, discuss plans, and specialize roles through natural language [7, 14, 30]. Although effective in many settings, these approaches often rely on repeated inter-agent communication to induce coordination, which becomes costly as the number of agents or interaction horizon grows [34, 38]. More importantly, repeated dialogue alone does not guarantee persistent alignment over roles and decisions, particularly under partial observability and dynamically evolving environments [33, 36, 44]. This limitation suggests that effective collaboration requires not more communication, but more structured coordination. Our framework addresses this challenge by organizing coordination around shared context and role assignment, thereby reducing reliance on repeated inter-agent dialogue.

Coordination in Long-Horizon Multi-Agent Decision Making. In long-horizon multi-agent decision-making, the key challenge is not merely eliciting cooperation, but maintaining coordination stability over time [4, 17, 19]. This challenge becomes more pronounced under partial observability and non-stationarity, where agents must remain aligned as local information and task progress evolve [15, 25, 35]. Prior work has explored communication, centralized coordination, and hierarchical control to reduce long-horizon misalignment [5, 6, 18].

A closely related line of work uses centralized coordinators or orchestrators to organize multi-agent collaboration. These methods improve coordination by prescribing workflows, selecting agents, or optimizing communication topology [6, 9, 44]. However, these methods mainly optimize workflow structure, routing efficiency, or communication cost, rather than the gradual degradation of coordination itself. In contrast, we view long-horizon coordination as a sequence of interdependent decisions: role assignments, context updates, and planning choices can affect future progress and recovery from misalignment. This motivates treating coordination as a sequential decision-making problem, while leaving the optimization of such coordination policies to RL-based methods.

Reinforcement Learning for Coordinating LLM Agents. RL provides a natural framework for optimizing sequential decision-making under uncertainty [12, 13], making it well-suited for long-horizon coordination problems in which agents must repeatedly adapt their roles and behaviors over time [22, 31, 32, 40]. Recent work has explored integrating RL with LLM-based agents to improve decision-making, including enhancing planning and action execution through interaction with environments [8, 21]. In multi-agent settings, however, RL is often employed primarily at the execution level, where learned policies are used to produce environment actions, while coordination itself remains largely induced through communication or prompting-based mechanisms [16, 20, 39].

As a result, coordination is rarely optimized explicitly as a learnable policy [1, 29]. This limits the ability of existing approaches to maintain long-term alignment across agents, particularly under partial observability and evolving task dependencies [37, 42]. In contrast, we treat coordination itself as a sequential decision-making problem and leverage RL to directly learn coordination policies that promote stable alignment over extended horizons.

B Additional Theoretical Analysis and Proofs

B.1 Task-General Drift Components and Definition 3.1

Coordination drift is intended to capture task-general failure modes of long-horizon multi-agent collaboration. In many collaborative environments, agents may pursue mutually incompatible intentions, duplicate each other's work, or enter cyclic waiting and replanning patterns. We refer to these recurring coordination failures as conflict, redundancy, and loop, respectively.

General drift verifier interface. DriCo does not assume that coordination drift must be computed by a specific hand-written rule set. More generally, DriCo requires a drift verifier

$$\mathcal{V}_\eta(g_\ell, c_\ell, h_\ell) \to \left(d_\ell^{\text{conf}}, d_\ell^{\text{red}}, d_\ell^{\text{loop}}\right), \tag{4}$$

where $g_\ell = (g_{\ell,1}, \ldots, g_{\ell,N})$ denotes the candidate sub-goals, c_ℓ denotes the shared context, and h_ℓ denotes the recent interaction history up to the corresponding environment timestep τ_ℓ. The verifier maps the current coordination state to non-negative scores for conflict, redundancy, and loop behavior. The DriCo framework only uses the resulting scalar drift score for shared-context ranking, preference construction, and intervention. Therefore, the coordinator optimization and intervention mechanism are agnostic to whether the verifier is implemented with symbolic rules, learned classifiers, LLM-based critics, execution monitors, simulator feedback, or human preference labels.

Verifier construction protocol. The rule-based verifier used in LLM-Overcooked is constructed before evaluation and uses only the training environment ontology, recipe graph, object types, station types, and reference trajectories from the training split. Held-out evaluation layouts, recipes, and trajectories are not used when defining Ξ_{conf}, κ, or $\text{Loop}(\cdot)$. After construction, the verifier is frozen and applied unchanged to all evaluation tasks. This prevents the drift metric from being tuned to evaluation episodes and ensures that drift is measured using task-level constraints rather than test-specific failure patterns.

Domain-adaptable drift components. The three drift components correspond to domain-independent coordination failures rather than Overcooked-specific events. Conflict captures violations of joint feasibility under resource exclusivity, temporal precedence, or mutual exclusion constraints. Redundancy captures semantically equivalent or duplicated sub-goals whose simultaneous execution does not provide additional task progress. Loop captures repeated execution patterns that fail to advance a task milestone over a recent history window.

In structured environments such as LLM-Overcooked, these abstract components can be instantiated using deterministic rules over object states, resource availability, recipe dependencies, normalized sub-goal forms, and action histories. In less structured domains, the same components can instead be estimated from learned failure detectors, LLM-based critics, simulator feedback, execution traces, or human preference labels. Thus, adapting DriCo to a new domain requires specifying a drift verifier over sub-goal compatibility, duplicate work, and progress stagnation; it does not require changing the coordinator objective or the intervention algorithm.

Operational definition used in LLM-Overcooked. We provide additional details on the operational definition of coordination drift used in Definition 3.1. At planning step ℓ, drift is computed from the candidate sub-goals generated by `Planner`, the current shared context maintained by `Coordinator`, and each agent's recent execution history up to the corresponding environment timestep τ_ℓ. Following Definition 3.1, the trigger drift is defined as

$$D_\ell^{\text{trig}} = \lambda_c d_\ell^{\text{conf}} + \lambda_r d_\ell^{\text{red}} + \lambda_l d_\ell^{\text{loop}}, \tag{5}$$

where d_ℓ^{conf}, d_ℓ^{red}, and d_ℓ^{loop} measure conflict, redundancy, and loop behavior, respectively.

For within-step revision, we use the planning drift

$$D_\ell^{\text{plan}} = \lambda_c d_\ell^{\text{conf}} + \lambda_r d_\ell^{\text{red}}, \tag{6}$$

which excludes the loop component because loop drift is computed from previously executed actions and cannot be reduced by resampling the current sub-goals within the same intervention cycle. Instead, loop drift serves as a recovery trigger and is re-evaluated after subsequent actions as the action-history window advances.

Table 4: Domain-adaptable interpretation of coordination drift components.

Component	Domain-independent condition	LLM-Overcooked instantiation
Conflict	Candidate sub-goals are not jointly feasible due to resource exclusivity, temporal precedence, or mutual exclusion constraints.	Two agents target incompatible objects, stations, or recipe-dependent actions.
Redundancy	Multiple agents pursue semantically equivalent sub-goals whose simultaneous execution provides no additional task progress.	Natural-language sub-goals map to the same normalized action-object-target form under $\kappa(\cdot)$.
Loop	Recent execution repeatedly visits similar action patterns without advancing a task milestone.	Repeated wait, pick-place, or failed handoff patterns over a fixed history window.

Conflict. Conflict drift captures pairwise incompatibility between agents' candidate sub-goals under the current shared context.

$$d_\ell^{\text{conf}} = \sum_{i<j} \phi_g(g_{\ell,i}, g_{\ell,j}, \mathbf{c}_\ell) \tag{7}$$

The function $\phi_g(g_{\ell,i}, g_{\ell,j}, \mathbf{c}_\ell)$ returns 1 if the two sub-goals cannot be jointly executed under the current task dependency, object state, or resource accessibility constraints, and returns 0 otherwise. We implement ϕ_g using a predefined incompatibility rule set Ξ_{conf} as follows.

$$\phi_g(g_{\ell,i}, g_{\ell,j}, \mathbf{c}_\ell) = \mathbb{1}\left[\exists \chi \in \Xi_{\text{conf}} \text{ s.t. } \chi(g_{\ell,i}, g_{\ell,j}, \mathbf{c}_\ell) = 1\right] \tag{8}$$

Each rule χ detects a specific coordination conflict, such as competing access to the same object or station, violation of recipe dependency order, infeasible handoff ordering, or mutually blocking sub-goals under the current shared context. In LLM-Overcooked, this includes cases where two agents attempt incompatible operations on the same object or station, where one agent's sub-goal requires an object currently held or blocked by another agent, or where the sub-goals violate the recipe dependency order encoded in the shared context.

Redundancy. Redundancy drift measures unnecessary duplication of candidate sub-goals.

$$d_\ell^{\text{red}} = \sum_{i<j} \mathbb{1}\left[\kappa(g_{\ell,i}) = \kappa(g_{\ell,j})\right] \tag{9}$$

Here, $\kappa(\cdot)$ maps each natural-language sub-goal to a normalized task-level form before comparison. In LLM-Overcooked, this form is an action-object-target tuple, e.g., $(\texttt{pick}, \texttt{eggplant}, \texttt{counter})$. Two sub-goals are counted as redundant when they map to the same tuple, even if their surface forms differ. This captures cases where multiple agents are assigned to the same work although only one execution is needed for task progress.

If a sub-goal cannot be mapped unambiguously, it is assigned to a special `unknown` form and is not counted as redundant unless the compared sub-goals share the same recoverable action-object-target fields. Synonyms and surface-form variants are resolved using the environment ontology. This conservative treatment avoids over-counting redundancy due to parser failures or underspecified LLM outputs.

Loop. Loop drift captures unproductive repeated behavior during low-level execution. At planning step ℓ, let τ_ℓ denote the corresponding environment timestep. For each agent, we examine a recent history window of length k over actions that have already been executed.

$$d_\ell^{\text{loop}} = \sum_{i=1}^{N} \mathbb{1}\left[\text{Loop}(a_{\tau_\ell - k:\tau_\ell - 1, i})\right] \tag{10}$$

When fewer than k actions have been executed before τ_ℓ, the window is truncated to the available action history. The predicate $\text{Loop}(\cdot)$ returns 1 if an agent repeatedly executes the same action or follows a recurring action pattern within the window without completing a new recipe milestone. Examples include repeated waiting, repeated failed pickup attempts, or pick-place cycles that leave

the task state unchanged. To avoid penalizing necessary waiting, $\mathrm{Loop}(\cdot)$ is activated only when the repeated pattern is not associated with a pending environment process or a valid dependency wait under the current shared context.

Unlike conflict and redundancy, loop drift is history-dependent. Therefore, it is used to trigger recovery but is not included in the within-step revision objective in Eq. (6). After a loop-triggered intervention, the revised shared context and regenerated sub-goals are expected to change subsequent actions; the loop component is then re-evaluated as the action-history window advances.

B.2 Proof of Theorem 4.1

Let $g^{\star}_{\ell,-i}$ denote the reference sub-goals of agents other than agent i at planning step ℓ. By the chain rule of conditional entropy,

$$H(g^{\star}_{\ell,-i}, \mathbf{c}_\ell \mid o_{\tau_\ell,i}) = H(\mathbf{c}_\ell \mid o_{\tau_\ell,i}) + H(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}, \mathbf{c}_\ell), \tag{11}$$

and equivalently,

$$H(g^{\star}_{\ell,-i}, \mathbf{c}_\ell \mid o_{\tau_\ell,i}) = H(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}) + H(\mathbf{c}_\ell \mid g^{\star}_{\ell,-i}, o_{\tau_\ell,i}). \tag{12}$$

Equating the two expressions gives

$$H(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}, \mathbf{c}_\ell) = H(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}) - \Big(H(\mathbf{c}_\ell \mid o_{\tau_\ell,i}) - H(\mathbf{c}_\ell \mid g^{\star}_{\ell,-i}, o_{\tau_\ell,i})\Big). \tag{13}$$

The term in parentheses is the conditional mutual information.

$$I(g^{\star}_{\ell,-i}; \mathbf{c}_\ell \mid o_{\tau_\ell,i}) = H(\mathbf{c}_\ell \mid o_{\tau_\ell,i}) - H(\mathbf{c}_\ell \mid g^{\star}_{\ell,-i}, o_{\tau_\ell,i}) \tag{14}$$

Since conditional mutual information is nonnegative, we have

$$H(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}, \mathbf{c}_\ell) = H(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}) - I(g^{\star}_{\ell,-i}; \mathbf{c}_\ell \mid o_{\tau_\ell,i}) \tag{15}$$

$$\leq H(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}). \tag{16}$$

Because $\exp(\cdot)$ is monotonically increasing, exponentiating both sides yields

$$\mathrm{PPL}(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}, \mathbf{c}_\ell) \leq \mathrm{PPL}(g^{\star}_{\ell,-i} \mid o_{\tau_\ell,i}). \tag{17}$$

Thus, conditioning on the shared context cannot increase the perplexity over teammates' reference sub-goals. The inequality is strict when $I(g^{\star}_{\ell,-i}; \mathbf{c}_\ell \mid o_{\tau_\ell,i}) > 0$.

Remark. This result should be interpreted as an information-theoretic motivation for using shared context, rather than a framework-specific guarantee. Under partial observability, an agent's local observation alone may be insufficient to infer the reference sub-goals of its teammates. When the shared context contains coordination-relevant information aggregated from multiple agents, conditioning on $\mathbf{c}_\ell$ can reduce uncertainty over $g^{\star}_{\ell,-i}$. The amount of reduction is characterized by the conditional mutual information $I(g^{\star}_{\ell,-i}; \mathbf{c}_\ell \mid o_{\tau_\ell,i})$.

B.3 Derivation of Communication Cost Analysis

We analyze the communication complexity over one episode. Let T denote the episode horizon, N the number of environment-interacting agents, and H the number of `Coordinator` invocations per episode. Let K_{disc} be the maximum pairwise message size, K_{up} the maximum agent-to-coordinator message size, and K_{coor} the maximum coordinator-to-agent shared-context message size.

For dense pairwise discussion, we suppose that every pair of agents exchanges messages at each timestep. This induces a complete pairwise communication graph with

$$\binom{N}{2} = \frac{N(N-1)}{2} \tag{18}$$

unordered communicating pairs per timestep. Since each pairwise message has size at most K_{disc}, the per-timestep communication cost is

$$\mathcal{O}(N^2 K_{\mathrm{disc}}). \tag{19}$$

Over T timesteps, the total communication cost is

$$C_{\text{disc}} = \mathcal{O}(TN^2 K_{\text{disc}}). \tag{20}$$

For coordinator-mediated communication, `Coordinator` is invoked only during sub-goal planning or `Actor`-triggered recovery. At each invocation, each agent sends local information to `Coordinator`, incurring an uplink cost of

$$\mathcal{O}(NK_{\text{up}}). \tag{21}$$

Then, `Coordinator` sends the shared context or agent-specific coordination information back to the agents, incurring a downlink cost of

$$\mathcal{O}(NK_{\text{coor}}). \tag{22}$$

Thus, the per-invocation communication cost is

$$\mathcal{O}(N(K_{\text{up}} + K_{\text{coor}})). \tag{23}$$

Aggregating over H invocations gives

$$C_{\text{coor}} = \mathcal{O}(HN(K_{\text{up}} + K_{\text{coor}})). \tag{24}$$

This analysis does not require K_{coor} to be independent of N; if the shared-context message grows with the number of agents, that dependence is captured through K_{coor}. Therefore, DriCo should be interpreted as reducing repeated dense pairwise exchange through sparse coordinator-mediated communication rather than as unconditionally changing the asymptotic dependence on N. In practice, when $H \ll T$, DriCo reduces communication relative to discussion-based coordination that performs dense pairwise exchange at every timestep.

C Model Training Process

We provide additional details on the `Coordinator` preference construction, actor-side critic learning, agent-specific reward, and `Actor` preference construction.

Coordinator Preference Construction. For each coordination state $\mathbf{x}_\ell = (\{o_{\tau_\ell,i}\}_{i=1}^N, \{g_{\ell,i}\}_{i=1}^N)$, we construct a candidate shared-context set $\mathcal{B}_\ell^{\mathrm{crd}} = \{\mathbf{c}_\ell^{(m)}\}_{m=1}^{M_{\mathrm{crd}}}$. The candidate set contains the reference context from successful trajectories, contexts sampled from the coordinator policy, and controlled counterfactual context variants. These variants modify coordination-relevant information such as role assignments, task-progress summaries, dependency status, and resource allocation. They are designed to expose the coordinator to plausible misalignment patterns that may arise during long-horizon interaction, such as stale progress information, ambiguous roles, duplicated responsibility, resource contention, or recovery loops.

Importantly, the source of a candidate context is not used as its preference label. A reference context is not automatically treated as preferred, and a counterfactual variant is not automatically treated as dispreferred. Instead, each candidate context is evaluated through its downstream effect on planning. Specifically, we condition `Planner` on $\mathbf{c}_\ell^{(m)}$ and generate the corresponding joint sub-goals. We then compute the induced trigger drift $D_\ell^{\mathrm{trig}}(\mathbf{c}_\ell^{(m)})$ from these generated sub-goals and the recent execution history.

We assign each candidate context the coordination score $s(\mathbf{c}_\ell^{(m)}) = -D_\ell^{\mathrm{trig}}(\mathbf{c}_\ell^{(m)})$, where lower induced drift indicates that the candidate context leads to a more coherent joint plan under the Planner. The loop component is shared across candidate contexts at the same planning step because it is computed from previously executed actions; therefore, it acts as a recovery signal rather than a within-step context-specific planning signal.

We rank candidate contexts by $s(\mathbf{c}_\ell^{(m)})$ and construct pairwise preferences $(\mathbf{c}_\ell^w, \mathbf{c}_\ell^l)$ such that $s(\mathbf{c}_\ell^w) > s(\mathbf{c}_\ell^l)$. The resulting tuples $(\mathbf{x}_\ell, \mathbf{c}_\ell^w, \mathbf{c}_\ell^l)$ form $\mathcal{D}_{\mathrm{crd}}$. Thus, `Coordinator` is optimized to select shared contexts that induce coordinated downstream sub-goals, rather than to distinguish reference contexts from particular counterfactual context types.

`Actor` Input and Valid Action Set. At planning step ℓ, `Planner` produces an accepted sub-goal $g_{\ell,i}$ for agent i. During the execution segment following planning step ℓ, `Actor` receives the local observation and active sub-goal as input, $\zeta_{t,i} = (o_{t,i}, g_{\ell,i})$, where t denotes the environment timestep. At execution time, `Actor` generates a single primitive action or the termination signal `<done>`: $a_{t,i} \sim \pi_{\theta_i}(\cdot \mid \zeta_{t,i})$. We use $\mathcal{A}_{t,i}^{\mathrm{valid}}$ to denote the set of primitive actions that are executable in the environment at timestep t. This set is used to judge whether the generated action is valid, rather than as an explicit action menu given to the model. If the generated action is not in $\mathcal{A}_{t,i}^{\mathrm{valid}}$, it is treated as an invalid action and penalized by the agent-specific reward.

Agent-Specific Reward. The agent-specific reward $r_{t,i}^{\mathrm{agent}}$ is designed to train `Actor` to execute valid primitive actions and emit the termination signal only when the assigned sub-goal is completed. At each environment timestep, the actor output is categorized as a normal primitive action, a valid termination signal, an invalid termination signal, or an invalid action. We define as follows.

$$r_{t,i}^{\mathrm{agent}} = \begin{cases} 0, & \text{correct } \texttt{<done>} \text{ after sub-goal completion,} \\ -10, & \text{incorrect } \texttt{<done>} \text{ before sub-goal completion,} \\ -3, & \text{invalid action,} \\ -1, & \text{valid primitive action.} \end{cases}$$

This reward encourages the actor to complete the assigned sub-goal with valid actions, avoid invalid commands, and emit `<done>` only after completion. It is distinct from the team-level reward used by `Coordinator`.

Critic Learning. Each agent maintains a sub-goal-conditioned action-value function $Q_{\omega_i}(\zeta_{t,i}, a_{t,i})$, which estimates the utility of executing primitive action $a_{t,i}$ under the current observation and active sub-goal. Here, $\mathcal{A}_{t,i}^{\mathrm{valid}}$ denotes the finite set of environment-executable primitive actions available to agent i at timestep t. This set is determined by the simulator state, such as the agent's position, held

object, accessible objects, and station states. Although `Actor` is implemented as an LLM, the critic is defined over this finite symbolic action interface rather than over the unconstrained natural-language generation space.

For each transition $(\zeta_{t,i}, a_{t,i}, r_{t,i}^{\text{agent}}, \zeta_{t+1,i}, d_{t,i}) \in \mathcal{D}_{\text{act}}$, where $d_{t,i}$ indicates whether the current sub-goal terminates, we train the critic with a Bellman regression loss. Using a target critic $\bar{Q}_{\omega_i}$, the target value is

$$y_{t,i} = r_{t,i}^{\text{agent}} + \gamma(1 - d_{t,i}) \max_{a' \in \mathcal{A}_{t+1,i}^{\text{valid}}} \bar{Q}_{\omega_i}(\zeta_{t+1,i}, a'). \tag{25}$$

The maximization in (25) is tractable because $\mathcal{A}_{t+1,i}^{\text{valid}}$ is an enumerable finite set of environment-executable primitive actions. At execution time, the LLM-generated action is parsed into this symbolic interface; outputs that cannot be parsed as valid primitive actions are treated as invalid actions and penalized through $r_{t,i}^{\text{agent}}$.

The critic is optimized by minimizing

$$\mathcal{L}_Q(\omega_i) = \mathbb{E}_{\mathcal{D}_{\text{act}}} \left[\left(Q_{\omega_i}(\zeta_{t,i}, a_{t,i}) - y_{t,i} \right)^2 \right]. \tag{26}$$

The target critic is updated by Polyak averaging

$$\bar{\omega}_i \leftarrow \tau_Q \omega_i + (1 - \tau_Q)\bar{\omega}_i, \tag{27}$$

where τ_Q is the target update coefficient.

`Actor` Preference Construction. During training, we sample M_{cand} candidate outputs from the actor policy for each input $\zeta_{t,i}$.

$$\mathcal{B}_{t,i} = \{a_{t,i}^{(1)}, \ldots, a_{t,i}^{(M_{\text{cand}})}\}, \qquad a_{t,i}^{(m)} \sim \pi_{\theta_i}(\cdot \mid \zeta_{t,i})$$

Each candidate is parsed into the environment action interface and checked against $\mathcal{A}_{t,i}^{\text{valid}}$. Invalid candidates receive the invalid-action penalty. We then use the critic to select the preferred and less-preferred candidates.

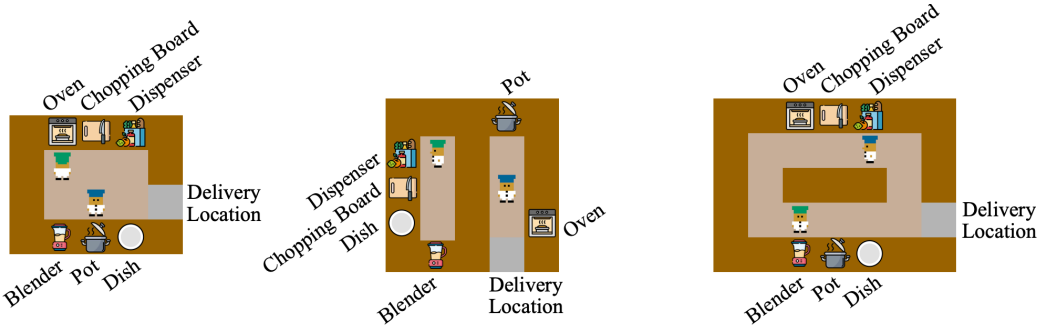
$$a_{t,i}^{w} = \arg\max_{a \in \mathcal{B}_{t,i}} Q_{\omega_i}(\zeta_{t,i}, a), \qquad a_{t,i}^{l} = \arg\min_{a \in \mathcal{B}_{t,i}} Q_{\omega_i}(\zeta_{t,i}, a)$$

The resulting pair $(a_{t,i}^{w}, a_{t,i}^{l})$ is used as an action preference for Eq. (3). At execution time, no candidate comparison is performed; `Actor` generates a single action conditioned on $\zeta_{t,i}$.

D Benchmarks: LLM-Overcooked

D.1 Benchmark Overview

LLM-Overcooked extends Overcooked-AI [3] for training and evaluating LLM-based multi-agent systems in long-horizon collaborative tasks. Unlike the original Overcooked-AI [3] setting, which is primarily designed for low-level policy learning, LLM-Overcooked exposes task goals, recipes, observations, and coordination-relevant information through natural language or structured textual descriptions.



(a) Cramped Room (b) Forced Coordination (c) Counter Circuit

Figure 7: Example layouts in LLM-Overcooked. (a) Cramped Room, (b) Forced Coordination, and (c) Counter Circuit.

As shown in Figure 7, we design diverse layouts that induce different coordination patterns and failure modes in long-horizon interaction.

(a) Cramped Room. In this layout, key objects such as ingredients, cooking stations, and delivery locations are densely placed within a limited space. Multiple agents have overlapping access to most objects, meaning that action capabilities are largely shared rather than strictly separated. While this flexibility allows agents to substitute for each other, it also increases the likelihood of redundant work and interference, such as multiple agents attempting to pick up the same ingredient or blocking each other's paths. This layout primarily evaluates the ability to avoid redundancy and resolve conflicts under high spatial contention.

(b) Forced Coordination. In contrast, this layout enforces a clear division of labor by spatially separating key resources. For example, ingredient sources, preparation areas, and delivery points are distributed across different regions, requiring agents to perform complementary roles. Individual agents cannot complete tasks independently, and successful execution requires explicit handoffs (e.g., one agent prepares ingredients while another delivers dishes). This setting evaluates role assignment, inter-agent dependency handling, and the ability to maintain consistent coordination over multi-step workflows.

(c) Counter Circuit. This layout introduces a central counter that acts as an intermediate buffer for coordination. Agents must place and retrieve items through this shared counter, which mediates indirect interaction between agents. Unlike (a), where agents can directly access most objects, or (b), where roles are spatially separated, the counter enforces structured interaction through a shared bottleneck. This design tests whether agents can use shared intermediate states effectively, manage asynchronous coordination, and avoid deadlocks or inefficient waiting behaviors.

Across all layouts, agents operate under partial observability and must coordinate through sequential decision-making. The environments differ in whether action spaces are shared or role-specialized, and in how coordination dependencies are structured (implicit vs. enforced vs. mediated). This diversity enables systematic evaluation of coordination stability, role adaptation, and failure recovery in long-horizon multi-agent systems.

D.2 Recipe Design and Task Diversity

To support systematic evaluation of long-horizon coordination, we construct a diverse set of recipes that vary in complexity, dependency structure, and coordination requirements. Each recipe defines a sequence of sub-tasks, such as ingredient collection, preparation, cooking, and delivery, with different levels of inter-agent dependency.

We design recipes to capture varying coordination patterns. Simple recipes require minimal coordination and can be completed by a single agent, while more complex recipes involve multiple ingredients, intermediate preparation steps, and strict ordering constraints. These complex recipes require agents to coordinate temporally extended actions, perform role specialization, and maintain consistent task allocation over multiple steps.

In addition, some recipes introduce shared intermediate objects (e.g., partially prepared ingredients or dishes), requiring agents to coordinate through handoffs and shared workspace usage. This enables evaluation of coordination behaviors such as division of labor, dependency handling, and recovery from coordination failures.

Table 5: Representative recipes across difficulty levels in LLM-Overcooked. Higher-level recipes require longer dependency chains and more coordinated preparation.

Difficulty	Level	Recipe	Procedure
Easy	1	`Baked Mushroom`	Mushroom → Bake → Serve
	2	`Boiled Onion Slices`	Onion → Slice → Boil → Serve
Medium	3	`Baked Tomato Soup`	Tomato → Slice → Bake → Cook soup → Serve
	4	`Sliced Broccoli and Mashed Tofu Stew`	Slice broccoli + Mash tofu → Cook stew → Serve
Hard	5	`Mashed Sweet Potato and Chickpea Patty`	Slice sweet potato → Boil → Mash; Chickpea + mashed sweet potato → Bake → Serve
	6	`Cauliflower, Spinach, and Tomato Patty`	Slice cauliflower → Cook → Mash; Slice tomato → Bake with spinach → Serve

To illustrate the increasing task complexity, Table 5 shows representative recipes from all six difficulty levels. Lower-level recipes involve a single ingredient and short preparation sequence, while higher-level recipes introduce additional ingredients, intermediate objects, and longer dependency chains. As the level increases, agents must coordinate more structured preparation steps, handoffs, and final composition, making the tasks progressively more demanding for long-horizon coordination.

Example: Level-6 Recipe.

TASK NAME:
`Cauliflower Spinach and Tomato Patty`
INGREDIENTS:
Cauliflower(1)
Spinach(1)
Tomato(1)
COOKING STEPS:
1. Cut the cauliflower into slices.
2. Place the cauliflower slices in a pot and cook for 3 timesteps.
3. Transfer the boiled cauliflower slices to a blender and stir for 3 timesteps.
4. Cut the tomato into slices.
5. Place the spinach, mashed cauliflower, and tomato slices in the oven and bake for 3 timesteps.
6. Transfer the patty to a dish and serve.

We additionally provide a representative recipe specification to illustrate how task goals, ingredients, and cooking dependencies are defined in LLM-Overcooked.

D.3 Training and Evaluation Environments

To evaluate generalization in long-horizon coordination, LLM-Overcooked provides separate training and evaluation recipe sets across multiple coordination layouts. For in-layout generalization, training and evaluation use the same layout geometry but disjoint recipe sets, requiring agents to transfer coordination strategies to held-out task compositions under the same spatial structure. The benchmark also supports cross-layout evaluation, where models trained on source layouts can be tested on held-out target layouts to assess transfer to unseen coordination structures.

Table 6: Recipe splits for training and evaluation in LLM-Overcooked.

Split	# Recipes	Description
Training	80	Diverse recipes across complexity levels
Test	30	Held-out recipes with unseen ingredient combinations

As summarized in Table 6, LLM-Overcooked contains 80 training recipes and 30 test recipes. The training recipes span varying complexity levels and ingredient combinations, while the test recipes are not used during training. This split prevents agents from relying on memorized recipe-specific strategies. Instead, agents must learn general coordination behaviors that transfer across ingredient combinations, dependency structures, and task lengths. In the main in-layout protocol, because the layouts are shared across training and evaluation, performance differences more directly reflect whether agents can generalize coordination policies to unseen recipes rather than to unseen map geometries.

D.4 Benchmark Comparison

LLM-Overcooked is designed as an LLM-oriented extension of Overcooked-AI [3] for studying long-horizon multi-agent coordination. Table 7 summarizes the main differences between Overcooked-AI [3], Collab-Overcooked [34], and the proposed LLM-Overcooked benchmark.

Table 7: Comparison between Overcooked-AI, Collab-Overcooked, and the proposed LLM-Overcooked benchmark. Check marks indicate that the benchmark explicitly supports the corresponding feature.

Benchmark Feature	Overcooked-AI [3]	Collab-Overcooked [34]	LLM-Overcooked (Proposed)
Grid-based cooperative cooking	✓	✓	✓
Multiple coordination layouts	✓	✓	✓
LLM-oriented language interface	✗	✓	✓
Structured recipe compositions	✗	✗	✓
Separate train–test environments	✗	✗	✓
Reference trajectory datasets	✗	✗	✓

As shown in Table 7, LLM-Overcooked retains the cooperative cooking structure of Overcooked-AI while adapting it to LLM-based multi-agent coordination. Compared with Collab-Overcooked, it adds structured recipes, disjoint train–evaluation configurations, held-out recipe compositions, and reference trajectory datasets, enabling controlled evaluation of long-horizon coordination generalization.

E Dataset Construction

We construct three role-specific supervised datasets from successful trajectories collected in **LLM-Overcooked**. Each trajectory is first parsed into timestep-level records containing the environment state, agent identity, role assignment, recipe information, current sub-goals, executed actions, and transition outcomes. From these records, we derive training instances for the three modules in our framework: the `Coordinator`, `Planner`, and `Actor`. The examples below show the resulting dataset formats using a `baked_eggplant` trajectory, with intermediate timesteps omitted for readability.

`Coordinator` Dataset Format

```
{
  "Task": "Baked_eggplant.",
  "Role Assignment": {
    "Chef": ["oven0", "pot0", "delivery"],
    "Assistant": ["blender0", "chopping_board0", "ingredient_dispenser",
    "dish_dispenser"]
  },
  "Coordinator Samples": [
    {
      "t": 0,
      "Chef space": pot0  oven0  counter
      "Assistant space": chopping_board0  blender0  dish_dispenser
      ingredient_dispenser
      "Subgoal_started_at": scene 0.
      "Kitchen_text":
      {
          "<Chef> holds nothing; <Assistant> holds nothing; <pot0> is
          empty; <chopping_board0> is empty; <oven0> is empty; <blender0>
          is empty; 3 counters can be visited by <Chef>. Their states
          are as follows: counters have nothing.\n"  },
      "Shared_context":
      {
          "The team is preparing an order of baked_eggplant.\n Right
          now, Chef has not started any subgoal yet, while Assistant has
          not started any subgoal yet.\n Role assignment: Chef handles
          oven0, pot0, and delivery; Assistant handles blender0,
          chopping_board0, ingredient_dispenser, and dish_dispenser.\n
          COOKING STEPs:\n 1. Pick up a eggplant.\n 2. Place the eggplant
          in the oven and bake for 3 timesteps.\n 3. Take the baked
          eggplant out of the oven and serve it."  },
    },
    "...",
    {
      "t": 15,
      "Shared_context":
      {
          "The team is preparing an order of baked_eggplant.\n Right
          now, Chef is working to pick up baked_eggplant from oven0
          and deliver it, while Assistant is waiting until the dish is
          delivered.\nCOOKING STEPs:\n1. Pick up a eggplant.\n 2. Place
          the eggplant in the oven and bake for 3 timesteps.\n 3. Take
          the baked eggplant out of the oven and serve it."  },
    }
  ]
}
```

The `Coordinator` dataset, $\mathcal{D}_{\text{crd}}$, trains `Coordinator` to generate a shared context and roles from each agent's observation.

Planner Dataset Format

```
{
  "Task": "Baked_eggplant.",
  "Role Assignment": {
    "Chef": ["oven0", "pot0", "delivery"],
  },
  "Planner Samples": [
    {
      "t": 0,
      "State":
          "Chef holds nothing; Assistant holds nothing; oven0 and
          counters are empty.",
      "Shared Context":
          "The team is preparing baked_eggplant. Both
          agents have not started yet. Chef handles oven0, pot0, and
          delivery, while Assistant handles ingredient_dispenser and
          dish_dispenser.",
      "Recipe": [
          "pick up eggplant",
          "bake eggplant in oven0",
          "serve baked_eggplant"
       ],
      "Sub-goal": "Wait for Assistant to place the next required item
      on the counter.",
    },
    "...",
    {
      "t": 14,
      "State":
          "Chef holds nothing; Assistant holds nothing; oven0 has
          baked_eggplant.",
      "Shared Context":
          "Chef is waiting for oven0 to finish baking,
          while Assistant is waiting until the dish is delivered.",
      "Recipe Progress": "eggplant has been baked; delivery remains",
      "Sub-goal": "Pick up baked_eggplant from oven0 and deliver it.",
    }
  ]
}
```

The Planner dataset, $\mathcal{D}_{\text{plan}}$, trains each role-conditioned Planner to predict the next high-level sub-goal from the current state, recipe information, and coordinator-generated shared context.

Actor Dataset Format


```
{
  "Task": "Baked_eggplant.",
  "Recipe": [
    "pick up eggplant",
    "bake eggplant in oven0",
    "serve baked_eggplant"
  ],
  "Actor Samples": [
    {
      "t": 0,
      "State": "Chef holds nothing; Assistant holds nothing; oven0 and counters
      are empty.",
      "Sub-goal": "Wait for Assistant to place the next required item on the
      counter.",
      "Action": "wait(1)",
      "Done": true,
      "Next State": "Chef holds nothing; Assistant holds nothing; oven0 and
      counters remain empty.",
      "Next Sub-goal": "Wait for Assistant to place the next required item on
      the counter."
    },
    "...",
    {
      "t": 14,
      "State": "Chef holds nothing; Assistant holds nothing; oven0 has
      baked_eggplant.",
      "Sub-goal": "Pick up baked_eggplant from oven0 and deliver it.",
      "Action": "pickup(baked_eggplant, oven0)",
      "Done": false,
      "Next State": "Chef holds a dish with baked_eggplant; oven0 is empty.",
      "Next Sub-goal": "Pick up baked_eggplant from oven0 and deliver it."
    },
    {
      "t": 15,
      "Agent": "Chef",
      "State": "Chef holds a dish with baked_eggplant; Assistant holds
      nothing.",
      "Sub-goal": "Pick up baked_eggplant from oven0 and deliver it.",
      "Action": "deliver_soup()",
      "Done": true,
      "Next State": "Chef holds nothing; Assistant holds nothing; the order
      is delivered.",
      "Next Sub-goal": ""
    }
  ]
}
```


The Actor dataset, $\mathcal{D}_{\text{act}}$, trains each Actor to produce a single executable low-level action conditioned on the current state and active sub-goal, together with a completion flag indicating whether the sub-goal has finished.

In summary, our dataset construction pipeline decomposes each multi-agent trajectory into three levels of supervision. This hierarchical decomposition enables the framework to learn coordination, planning, and action execution from the same underlying trajectory data.

F Model Structure

F.1 Low-Rank Adaptation

We adopt low-rank adaptation (LoRA) [10] for parameter-efficient fine-tuning of DriCo. `Coordinator`, `Planner`, and `Actor` share the same pretrained backbone, while maintaining separate LoRA adapters for their distinct functions. `Coordinator` adapter is optimized for shared context construction, role assignment, and drift-aware intervention; `Planner` adapter is optimized for generating role- and context-conditioned sub-goals; and `Actor` adapter is optimized for sub-goal-conditioned low-level action execution. During training and inference, only the adapter corresponding to the active module is enabled, allowing modular specialization without updating the full backbone.

F.2 Input-Output Examples

We provide representative input-output examples for the two coordination functions of `Coordinator`. We illustrate how `Coordinator` constructs a shared context from recipe information, current sub-goals, and the environment state.

Input/Output Example: `Coordinator` Shared-Context Construction

Task: Build a shared context by summarizing the recipe and each agent's sub-goal.
Input:
You are the coordinator. Build a shared context by summarizing the recipe and each agent's sub-goal.

- Recipe: Tomato Soup requires tomato. Steps: get tomato, cook tomato in pot0, fill dish, deliver.

- Chef's sub-goal: Cook tomato in pot0.

- Assistant's sub-goal: Pick up tomato from ingredient dispenser and place it on the counter.

- State:
Chef space: pot0 oven0 serving_area counter.
Assistant space: ingredient_dispenser dish_dispenser chopping_board blender0 counter.
Chef holds nothing.
Assistant holds nothing.
pot0 is empty.
counter is empty.

Instruction: Write ONE concise paragraph with the following structure:

- Start with: "The team is preparing an order of <dish>."
- Then describe current progress: "Right now, Chef <chef_status>, while Assistant <assistant_status>."
- If a sub-goal is "(not started)", phrase it as "has not started any sub-goal yet".
- If either sub-goal is "(not started)", append the role assignment.
- End with a numbered summary of the recipe steps.

Output:
The team is preparing an order of Tomato Soup. Right now, Chef is working to cook tomato in pot0, while Assistant is working to pick up tomato from the ingredient dispenser and place it on the counter. 1. Get tomato. 2. Cook tomato in pot0. 3. Fill dish. 4. Deliver.

The second example illustrates the critique mode, where `Coordinator` evaluates an agent's proposed sub-goal, detects coordination drift, and provides corrective feedback when the sub-goal is inappropriate.

Input/Output Example: `Coordinator` Intervention

Task: Evaluate whether an agent's proposed sub-goal is appropriate for cooperative cooking.
Input:
You are the Coordinator. Evaluate whether Chef's proposed sub-goal is appropriate for cooperative cooking.

 # Recipe
Tomato Soup requires tomato.
Steps: get tomato, cook tomato in pot0, fill dish, deliver.

 # Current State
Chef space: pot0 oven0 serving_area counter.
Assistant space: ingredient_dispenser dish_dispenser chopping_board blender0 counter.
Chef holds nothing.
Assistant holds nothing.
Counter has tomato.
pot0 is empty.

 # Roles
Chef handles: pot0, oven0, serving_area, counter.
Assistant handles: ingredient_dispenser, dish_dispenser, chopping_board, blender0, counter.

 # Sub-goal Under Review
Chef's sub-goal: "Pick up the tomato from the counter and put it into pot0."
(Assistant's sub-goal: "Pick up the tomato from the counter and put it into pot0.")

 # Decision Criteria
- Approve if the sub-goal aligns with the current state and recipe progress.
- Reject if there is a drift. Drift types:
* loop: Both agents are waiting for each other. Waiting for an active utensil or for a teammate who has not started is NOT a loop.
* redundancy: Both agents are attempting the EXACT same action with the same source, same object, and same target.
* conflict: The sub-goal conflicts with the current state, such as a missing prerequisite, role violation, or competing for the same target.

 # Output Format
If appropriate, respond with:
Appropriate sub-goal.
<one sentence explaining why>

If inappropriate, respond with:
Drift: <loop|redundancy|conflict>
Feedback: <one or two sentences citing the state, the issue, and what Chef should do instead>

Output:
Drift: redundancy
Feedback: Chef and Assistant are attempting the exact same tomato pickup sub-goal from the counter to pot0, but only one agent needs to perform this action. Since Chef can access pot0, Chef should proceed with the pot-side step only after one agent is assigned to handle the tomato pickup.

We also provide examples of `Planner` and `Actor` to illustrate hierarchical decision-making.

Input / Output Example 1: `Planner`

Task: Generate one sub-goal for the Chef.
Input:
You are Chef in LLM-Overcooked.

Output exactly one sub-goal sentence.

Chef can access: pot, oven, counter, serving area.
Chef CANNOT access: ingredient_dispenser, dish_dispenser, chopping_board, blender.

- Shared context:

The team is preparing an order of Tomato Soup. Right now, Chef is waiting to cook tomato in pot0, while Assistant is responsible for fetching tomato and placing it on the counter. 1. Get tomato. 2. Cook tomato in pot0. 3. Fill dish. 4. Deliver.

- Current state:
Chef space: pot0 oven0 serving_area counter.
Assistant space: ingredient_dispenser dish_dispenser chopping_board blender0 counter.
counter has tomato. pot0 is empty. Chef holds nothing.

sub-goal:

Output:
Pick up the tomato from the counter and put it into pot0.

Input / Output Example 2: `Planner`

Task: Generate one sub-goal for the Assistant.
Input:
You are Assistant in LLM-Overcooked.

Output exactly one sub-goal sentence.

Assistant can access: chopping_board, blender, ingredient_dispenser, dish_dispenser, counter.
Assistant CANNOT access: pot, oven, serving area.

-Shared context:

The team is preparing an order of Tomato Soup. Right now, Chef is waiting for tomato at pot0, while Assistant is responsible for fetching tomato and placing it on the counter. 1. Get tomato. 2. Cook tomato in pot0. 3. Fill dish. 4. Deliver.

-Current state:
Chef space: pot0 oven0 serving_area counter.
Assistant space: ingredient_dispenser dish_dispenser chopping_board blender0 counter.
counter is empty. pot0 is empty.
Assistant holds nothing.

sub-goal:

Output:
Pick up a tomato from the ingredient dispenser and place it on the counter.

Input / Output Example: `Actor`

Task: Convert a sub-goal into one executable action.
Input:
You are Actor in LLM-Overcooked.

Given the sub-goal and current state, output exactly ONE executable action and one completion flag. Output ONLY the two lines below, in this exact format. No explanation, no extra lines.

Valid action functions: pickup(item, utensil), place_obj_on_counter(), put_obj_in_utensil(utensil), cook(utensil), bake(utensil), cut(utensil), stir(utensil), fill_dish_with_food(utensil), deliver_soup(), wait(1).

=== SUB-GOAL ===
Pick up the tomato from the counter and put it into pot0.

=== CURRENT STATE ===
Chef is at the counter. counter has tomato. pot0 is empty. Chef is holding nothing.

=== OUTPUT FORMAT ===
action: <one valid action>
done: <true|false>

Output:
action: pickup(tomato, counter)
done: false
Next Input:

=== SUB-GOAL ===
Pick up the tomato from the counter and put it into pot0.

=== CURRENT STATE ===
Chef is at pot0. Chef is holding tomato. pot0 is empty.

Output:
action: put_obj_in_utensil(pot0)
done: true

F.3 Hyperparameters

Table 8: Hyperparameter configurations for `Coordinator`, `Planner`, and `Actor` training.

Hyperparameter	**Coordinator Preference Opt.**	**Planner SFT**	**Actor Preference Opt.**
Backbone / Adaptation			
LoRA rank r	32	32	16
LoRA α_{LoRA}	64	64	32
LoRA dropout	0.05	0.05	0.05
Generation / Decoding			
Max new tokens	128	32	8
Temperature	0.7	0.7	0.7
Top-p	0.9	0.9	0.9
Optimization			
Optimizer	AdamW	AdamW	AdamW
Batch size	8	8	8
Training epochs	3	3	3
Learning rate	1×10^{-5}	1×10^{-5}	1×10^{-5}
Critic learning rate	–	–	1×10^{-5}
Discount factor γ	–	–	0.99
Preference Optimization			
Preference scaling factor β	0.5	–	0.5
Clipping value	5	–	–
Reference policy	$\pi^{\text{ref}}_{\theta_{\text{crd}}}$	–	$\pi^{\text{ref}}_{\theta_i}$
Coordinator / Drift Control			
Drift weights $(\lambda_c, \lambda_r, \lambda_l)$	$(1.0, 1.0, 1.0)$	–	–
Trigger-drift threshold δ	1.0	–	–
Planning-drift threshold δ_{plan}	1.0	–	–
Maximum revision steps $K_{\max}$	3	–	–
Loop detection window	29	–	–
Actor Value Learning / Preference Construction			
Actor candidates M_{cand}	–	–	4
Target critic update coefficient τ_Q	–	–	0.005
Evaluation			
Evaluation episodes M		30	
Random seeds		5	
Maximum episode horizon		120	

G Metric

We evaluate task performance using success rate and progress completeness. For each episode e, success is defined as whether the agents complete and deliver the target recipe within the maximum episode horizon. Given M evaluation episodes, the success rate is

$$\text{SuccessRate} = \frac{1}{M}\sum_{e=1}^{M} \mathbb{1}\left[\text{Success}(e)\right].$$

To capture partial progress in long-horizon tasks, we additionally compute progress completeness based on recipe-specific milestones. For each recipe r, let $\mathcal{K}_r$ denote the ordered set of required milestones, including ingredient acquisition, station placement, cooking or processing, dish filling, and delivery. For an episode e with recipe $r(e)$, progress completeness (ProgComplete) is defined as

$$\text{Progress}(e) = \frac{1}{|\mathcal{K}_{r(e)}|}\sum_{k\in\mathcal{K}_{r(e)}} \mathbb{1}\left[k \text{ is completed in } e\right], \quad \text{ProgComplete} = \frac{1}{M}\sum_{e=1}^{M}\text{Progress}(e).$$

Progress completeness is included because long-horizon tasks may fail near completion despite substantial intermediate progress, which the success rate alone cannot capture.

H Baselines

All baselines are instantiated with the same backbone LLMs used by DriCo, while preserving their original method-specific coordination mechanisms. The backbone LLM is standardized only to control for base-model capability, not to simplify or remove baseline functionality. All methods are evaluated through the same LLM-Overcooked simulation wrapper. We keep the environment interface, observation format, action parser, valid-action constraints, decoding hyperparameters, episode horizon, and evaluation protocol fixed across methods whenever applicable.

Thus, the comparison controls for the LLM backbone and environment interface while allowing each method to use its own native coordination design. This setup is intended to evaluate differences in coordination strategy rather than differences in model scale, prompting infrastructure, or simulator access.

Prompting- and Reasoning-Based Coordination. These methods induce cooperation through LLM prompting, dialogue, memory, verification, modular reasoning, or in-context coordination routines, without directly optimizing an explicit drift-aware coordination policy.

- `RoCo`: RoCo [23] is a dialogue-based multi-robot collaboration framework in which LLM agents exchange observations, discuss task strategies, and generate sub-task plans. The plans are refined using environment feedback such as task constraints, IK feasibility, and collision checking, making RoCo a prompt-based coordination baseline driven by dialogue and replanning rather than learned drift-aware coordination.
- `ProAgent`: ProAgent [43] is an LLM-based cooperative agent for Overcooked-AI that infers teammate intentions from language-converted states and adapts its own high-level skill accordingly. It combines planning, verification, memory, control, and belief correction, serving as a proactive reasoning baseline without an explicit team-level drift-aware coordinator.
- `Collab-Overcooked`: Collab-Overcooked [34] is instantiated as an in-context learning baseline using the agent prompts provided by the original Collab-Overcooked framework for natural-language multi-agent coordination. It relies on prompt-level instructions to induce role assignment, communication, and task-state-aware action selection, but does not update model parameters or introduce an explicit objective for suppressing conflict, redundant actions, or repetitive coordination loops.

Modular Coordination. This category includes methods that organize cooperation through modular agent components, but do not explicitly optimize conflict, redundancy, or loop drift as training signals.

- `CoELA`: CoELA [45] is a modular cooperative embodied language agent composed of perception, memory, communication, planning, and execution modules. It targets decentralized cooperation under partial observability and costly communication, providing a structured communication-and-planning baseline without explicit optimization over coordination drift.

Positioning Relative to Related Directions. DriCo is related to dynamic orchestration of LLM agents [6], long-term agent drift monitoring [28], and multi-agent preference-based reinforcement learning [11], but these directions address complementary aspects of multi-agent systems rather than direct baselines for our setting. Dynamic orchestration focuses on routing or sequencing reasoning agents to improve quality or efficiency, whereas DriCo targets environment-interacting agents whose sub-goals and primitive actions must remain coordinated over long horizons. Agent drift work studies broad behavioral degradation over extended interactions, whereas DriCo operationalizes a narrower coordination-specific drift—conflict, redundancy, and loop—as both a preference signal and an intervention trigger. Preference-based MARL learns from global or agent-wise preferences over trajectory segments, whereas DriCo derives preferences from coordination quality induced by shared contexts. Overall, DriCo complements these directions by using shared-context construction and targeted intervention to recover long-horizon coordination in interactive LLM-based multi-agent environments.

Why these methods are not direct baselines. The related methods discussed above are not direct experimental baselines for LLM-Overcooked. Dynamic orchestration methods focus on routing or sequencing reasoning agents, rather than modeling environment-level sub-goal execution and long-horizon coordination failures. Agent drift work provides monitoring and diagnostic metrics, but not an executable intervention policy. Offline multi-agent preference-based RL methods are designed for symbolic MARL policies with preference-labeled trajectory segments, not for LLM-based shared-context construction and language-conditioned action execution. Thus, we view these methods as complementary research directions rather than directly comparable baselines.

I Notation

Table 9: Basic Environment and Hierarchical Decision Notation

Notation	Description	Notation	Description
$\mathcal{I}$	set of agents	i	i-th agent
$\mathcal{S}$	state space	s_t	environment state at t
$\mathcal{A}_i$	action space of agent i	$a_{t,i}$	primitive action of agent i at t
$\mathcal{A}_{t,i}^{\text{valid}}$	environment-valid primitive action set for agent i at t	$\mathbf{a}_t$	joint primitive action at t
$\mathcal{O}_i$	observation space of agent i	$o_{t,i}$	local observation of agent i at t
$\mathbf{o}_t$	joint observation at t	Ω_i	observation function of agent i
$\mathcal{T}$	state transition function	$\mathcal{R}$	reward function
γ	discount factor	N	number of agents
ℓ	planning step index	τ_ℓ	environment timestep
T_{plan}	set of planning steps	$g_{\ell,i}$	candidate sub-goal of agent i at ℓ
$\mathbf{g}_\ell$	tuple of candidate sub-goals at ℓ	ρ_i	role assigned to agent i
$\mathbf{c}_\ell$	shared context at ℓ	$\mathbf{x}_\ell$	coordination state at ℓ
$\zeta_{t,i}$	actor input, e.g., $(o_{t,i}, g_{\ell,i})$	$\mathcal{C}$	communication buffer
$\pi_{\theta_{\text{crd}}}$	coordinator policy	$\pi_{\theta_{\text{crd}}}^{\text{ref}}$	reference coordinator policy
π_{ψ_i}	planner policy of agent i	π_{θ_i}	actor policy of agent i
$\pi_{\theta_i}^{\text{ref}}$	reference actor policy of agent i	Q_{ω_i}	sub-goal-conditioned action-value function
$\bar{Q}_{\omega_i}$	target action-value function	ω_i	critic parameters of agent i

Table 10: Coordination Drift and Intervention Notation

Notation	Description	Notation	Description
d_ℓ^{conf}	conflict drift at ℓ	d_ℓ^{red}	redundancy drift at ℓ
d_ℓ^{loop}	loop drift at ℓ	D_ℓ^{trig}	trigger drift for intervention
D_ℓ^{plan}	planning drift for within-step revision	$\lambda_c, \lambda_r, \lambda_l$	drift weighting coefficients
δ	intervention threshold for trigger drift	δ_{plan}	revision threshold for planning drift
k	loop detection window size	$K_{\max}$	maximum number of revision steps
$\kappa(\cdot)$	sub-goal normalization function	ϕ_g	pairwise sub-goal incompatibility function
$\mathrm{Loop}(\cdot)$	loop detection predicate	Ξ_{conf}	predefined conflict rule set
χ	individual conflict rule	$\mathcal{M}_\ell$	active drift types at planning step ℓ
$\mathcal{I}_\ell^{\mathrm{aff}}$	affected agents at ℓ	$\mathcal{V}_\eta$	drift verifier

Table 11: Training, Preference, and Communication Notation

Notation	Description	Notation	Description
r_t^{team}	team reward at t	r_ℓ^{team}	team reward aggregated after ℓ
$r_{t,i}^{\mathrm{agent}}$	agent-specific reward for actor training	α	drift penalty weight
β	preference scaling factor	$s(\mathbf{c}_\ell)$	coordination score of shared context $\mathbf{c}_\ell$
H	number of Coordinator invocations per episode	$\mathcal{D}_{\mathrm{crd}}$	coordinator preference dataset
$\mathcal{D}_{\mathrm{plan}}$	planner SFT dataset	$\mathcal{D}_{\mathrm{act}}$	actor transition dataset
$\mathcal{D}_{\mathrm{act}}^{\mathrm{pref}}$	actor preference dataset	–	–
$\mathcal{B}_\ell^{\mathrm{crd}}$	candidate shared-context set	M_{crd}	number of coordinator context candidates
$\mathbf{c}_\ell^w$	preferred shared context	$\mathbf{c}_\ell^l$	less-preferred shared context
$\mathcal{B}_{t,i}$	sampled actor candidate outputs for preference construction	M_{cand}	number of sampled actor candidate outputs
$a_{t,i}^w$	preferred actor output for agent i	$a_{t,i}^l$	less-preferred actor output for agent i
$\mathcal{J}^{\mathrm{crd}}$	coordinator objective	$\mathcal{J}_i^{\mathrm{act}}$	actor objective of agent i
$\mathcal{L}_{\mathrm{crd}}$	coordinator loss	$\mathcal{L}_{\mathrm{plan}}$	planner SFT loss
$\mathcal{L}_{\mathrm{act}}$	actor preference loss	$\mathcal{L}_Q$	critic Bellman regression loss
τ_Q	target critic update coefficient	–	–
K_{disc}	maximum pairwise message size	K_{up}	maximum agent-to-coordinator message size
K_{coor}	maximum coordinator-to-agent message size	C_{disc}	dense pairwise discussion communication cost
C_{coor}	coordinator-mediated communication cost	–	–

J Additional Results

J.1 Additional Experiments and Analyses

This section provides additional evaluation results for the proposed DriCo. These results complement the main experiments by further illustrating the model's coordination performance across additional layouts and component configurations.

Model Performance. We first report DriCo's performance across representative LLM-Overcooked layouts to examine how its coordination behavior changes under different spatial and task constraints.

Table 12: Additional performance of DriCo across LLM-Overcooked layouts and task difficulty levels using Qwen-2.5 7B. Each entry reports Success Rate [%] / Progress Completeness [%], and the last column reports the average over all difficulty levels.

Layout	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**Avg.**
`Cramped Room`	80/86	100/100	40/55	40/55	20/40	0/26	47/60
`Forced Coordination`	100/100	100/100	67/91	65/60	0/32	0/34	55/70
`Counter Circuit`	70/85	60/87	46/47	20/25	0/28	0/20	33/49

Table 12 reports DriCo's performance across three LLM-Overcooked layouts. `Cramped Room` tests coordination under dense object placement and overlapping access, `Forced Coordination` requires role specialization and explicit handoffs, and `Counter Circuit` introduces counter-based bottlenecks for indirect handoffs. Across these regimes, DriCo maintains strong performance and meaningful progress completeness, indicating robust shared-context coordination under diverse long-horizon coordination challenges.

Actor Q-Guided Optimization. We further evaluate the effect of Q-guided preference optimization for the `Actor` on the Forced Coordination environment.

Table 13: Effect of Q-guided `Actor` on the `Forced Coordination` using Qwen-2.5 7B. Each entry reports Success Rate [%] / Progress Completeness [%].

Variant	L1	L2	L3	L4	L5	L6	Avg.
w/o Q-guided	100/100	60/73	40/85	40/40	0/28	0/37	40/61
DriCo (Proposed)	**100/100**	**100/100**	**67/91**	**65/60**	**0/32**	0/34	**55/70**

Table 13 reports per-level success rate and progress completeness. Compared with the variant without Q-guided action selection, the full DriCo model achieves higher performance across all levels, improving the average result from $40/61$ to $55/70$. These results indicate that Q-guided low-level execution improves the reliability of sub-goal completion, especially as task dependencies become longer.

J.2 Analysis of Coordinator Invocation Frequency

To empirically support the sparse communication assumption in Section 4.3, we measure the number of `Coordinator` invocations per episode. Recall that H denotes the number of `Coordinator` invocations per episode, while T denotes the episode horizon.

Table 14: Coordinator invocation frequency on the `Forced Coordination` using Qwen-2.5 7B. We report the mean and standard deviation of the number of `Coordinator` invocations per episode (H).

Metric	L1	L2	L3	L4	L5	L6	Avg.
`Coordinator` invocations H	$36.1_{\pm 3.8}$	$50.3_{\pm 8.3}$	$75.6_{\pm 29.4}$	$20.2_{\pm 7.1}$	$51.1_{\pm 1.6}$	$68.2_{\pm 2.0}$	$50.3_{\pm 13.0}$

Table 14 reports the average `Coordinator` invocation count across task difficulty levels. The invocation count tends to increase on harder tasks, where longer dependency chains and recovery events require more frequent shared-context updates. Since `Coordinator` is invoked only at planning or recovery points rather than at every primitive execution step, these results indicate that Coordinator calls are substantially fewer than per-timestep dense communication in most settings, although harder tasks can require more frequent recovery updates.

J.3 Case Studies

`Coordinator` **Rejection of Redundant Sub-Goals.** We provide an example showing how `Coordinator` intervention prevents redundancy.

Coordinator Intervention Example: Rejecting Redundancy and Refining the Sub-Goal

Initial Assistant sub-goal:

Pick up sweet_potato_slices from chopping_board0 and place it on the counter.

Leader critique:

Drift: redundancy
Feedback: Assistant is working on the same sub-goal as Chef. Assistant should wait for Chef to place the next required item on the counter.

Decision: `Reject`

Refined Assistant sub-goal:

Wait for Chef to place the next required item on the counter.

Coordinator critique:

Appropriate sub-goal.
Explanation: Assistant's sub-goal is appropriate because Assistant is waiting for Chef to place the next required item on the counter.

Decision: `Pass`
Actor output:

```
action: wait(1)
```

Interpretation. The first sub-goal is rejected because it duplicates `Chef`'s current sweet-potato transfer. After incorporating the feedback, `Assistant` switches to a dependency-aware waiting sub-goal. This is accepted because waiting for `Chef`'s counter handoff is necessary for coordinated progress and is not considered a loop.

`Assistant` initially proposes a sub-goal that duplicates `Chef`'s work. `Coordinator` rejects it as redundancy and provides feedback. After refinement, `Assistant` proposes a valid waiting sub-goal and the low-level actor executes `wait(1)`.

`Actor` `<done>` **Triggered Replanning.** As described in Section 4.2, `Actor` produces not only primitive actions but also a termination signal, denoted by the `<done>` tag. When `<done>` is emitted, the signal is passed to `Planner` and `Coordinator`, which then trigger sub-goal replanning using the latest buffered observations and shared context. This mechanism allows the system to terminate sub-goals that are already resolved, indirectly executable through another agent, or invalid under the current environment state.

We provide two representative examples below.

Case 1: `<done>` after a Handoff-Oriented Sub-Goal

Sub-goal. ‘‘`Pass the item in pot to Assistant`’’

Actor output (Chef).

```
action: pass_item_to(assistant, sweet_potato_slices)
done: true
```

Observed behavior. After emitting `done=true`, Chef loads a fallback queue `[wait(1)]`, while Assistant proceeds with `pickup(sweet_potato_slices, pot)`.

Interpretation. This case shows that `Actor` can terminate its own sub-goal once the handoff intent has been established, instead of forcing unnecessary low-level execution. The `<done>` signal allows the control flow to move back to `Planner` and `Coordinator`, while the partner agent continues the relevant execution under the updated shared context.

Case 2: `<done>` for an Invalid Sub-Goal

Sub-goal. ‘‘`Eat the sweet_potato_slices`’’

Actor output (Chef).

```
action: eat(sweet_potato_slices)
done: true
```

Observed behavior. The proposed action is not part of the valid cooking interface. After emitting `done=true`, Chef again falls back to `[wait(1)]`, after which the system transitions to the next sub-goal generation step.

Interpretation. This example illustrates how `Actor` handles an invalid or task-inappropriate sub-goal. Instead of repeatedly executing an infeasible action, `Actor` emits `<done>`, which triggers replanning. `Coordinator` then updates the shared context, and `Planner` generates a new sub-goal conditioned on the revised team state. This prevents the team from remaining stuck on an invalid execution path.

Together, these examples show that the `<done>` signal is not merely a sub-goal completion marker. It also functions as a control mechanism for hierarchical recovery: when the current sub-goal is already resolved, indirectly delegated, or invalid, `Actor` can terminate execution early and hand control back to `Planner` and `Coordinator` for replanning.

K Declaration of LLM Usage

This work uses LLMs as core components of the proposed multi-agent coordination framework. Specifically, `Coordinator`, `Planner`, and `Actor` modules are instantiated with instruction-tuned LLM backbones, including Qwen-2.5 7B [41] and Llama-3.1 8B [24]. The LLMs are adapted using LoRA [10]-based fine-tuning with module-specific adapters. `Coordinator` is optimized with a preference-based objective using coordination drift as preference signals, `Planner` is trained with SFT on reference sub-goals, and `Actor` is trained with a sub-goal-conditioned preference-based actor–critic objective. Thus, LLMs are an essential part of the proposed method, training pipeline, and experimental evaluation.

L Limitations, Future Directions, and Broader Impact

The proposed DriCo framework aims to improve the reliability of LLM-based multi-agent systems by addressing *coordination drift* in long-horizon collaborative tasks. By explicitly modeling conflicts, redundant sub-goals, and looping behaviors, DriCo provides a mechanism for detecting and mitigating coordination failures that can accumulate over time. These capabilities may benefit applications that require persistent team-level alignment, such as collaborative robotics, interactive planning, assistive agents, logistics, and human-agent teaming.

While DriCo establishes a robust foundation for mitigating *coordination drift* through a learned `Coordinator`, several promising avenues remain for future exploration to enhance the scalability, autonomy, and generalizability of multi-agent coordination.

Towards Fully Decentralized and Peer-to-Peer Coordination. DriCo relies on a centralized `Coordinator` to aggregate agent-level information, construct shared context, and trigger drift-aware intervention. This design reduces dense pairwise communication through coordinator-mediated shared-context updates, but it introduces a potential single point of failure and a computational bottleneck as the number of agents increases. Future work could extend DriCo toward decentralized or peer-to-peer coordination, where agents collectively maintain shared context through local message passing without a central arbiter, consensus protocols, or gossip-style updates. Role assignment could be made more adaptive by allowing agents to negotiate or revise roles based on local observations, current sub-goals, and task progress, rather than relying only on centralized assignment. In addition, integrating game-theoretic frameworks into LLM-based reasoning could allow agents to resolve coordination drift locally through strategic bargaining.

Self-Adaptive Drift Detection and Open-World Generalization. The current definition of *coordination drift* focuses on conflict, redundancy, and loop components, which are effective for structured recipe-based tasks in LLM-Overcooked. However, more open-ended environments may involve subtler forms of miscoordination that are difficult to capture with task-specific rules. A promising direction is to learn intrinsic drift signals directly from interaction trajectories, for example, by estimating disagreement among agents' sub-goal distributions, divergence from reference coordination patterns, or uncertainty in team-level progress beyond the rule-based definition in Definition 3.1. Such learned drift measures could make `Coordinator`'s intervention policy more adaptive in unseen environments.

Although LLM-Overcooked includes held-out recipes under shared layouts for evaluating coordination generalization, DriCo is still evaluated within a controlled benchmark family. Future work should study zero-shot coordination with unseen partners, new agent capabilities, and environments with different physical or semantic constraints. This would test whether the learned `Coordinator` can construct useful shared context and recover from *coordination drift* when `Planner` and `Actor` encounter coordination patterns not observed during training.

Dynamic Communication and Information Bottlenecks. Communication and intervention remain important efficiency challenges in long-horizon coordination. Excessive intervention can increase communication overhead and trigger unnecessary replanning, while insufficient intervention can allow coordination drift to accumulate over time. Future versions of DriCo could incorporate a Value of Coordination (VoC) criterion to decide when intervention is worthwhile. By estimating the expected gain in team reward relative to the cost of communication and replanning, `Coordinator` could skip shared-context updates in stable states and intervene only when the expected coordination benefit is high.

Long-Term Safety and Human-Agent Collaboration. As LLM-based agents move into human-centric environments, coordination should remain interpretable, controllable, and aligned with human intent. A promising direction is human-in-the-loop coordination, where a human operator can provide feedback to `Coordinator`, inspect shared context, or override role assignments and interventions when necessary. Future studies should also examine robustness against adversarial or unreliable agents that intentionally or unintentionally induce coordination drift. Developing mechanisms to detect such agents and preserve team stability is critical for deploying LLM-based multi-agent systems in safety-sensitive environments.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have claimed our motivation, scope, and contribution in the Section 1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations can be shown in Appendix L.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Theorem assumptions and proofs can be shown in Section 3, Section 4, and Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have provided dataset construction process, benchmark explaination, and the hyperparameter for the reproducibility of the proposed solution in Appendix D, Appendix E, and Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We have provided the data and code, and described dataset construction process in Appendix E.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Task descriptions and experimental settings can be shown in the Section 5, Appendix G, and Appendix H.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We use error bars to confirm the statistical reliability in the Section 5 and Appendix J.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Computation resource can be shown in the Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics. The work does not involve human subjects, sensitive personal data, or sources of foreseeable harm.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Broader impact can be shown in the Appendix L.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have provided the existing assets in Section 5 and Appendix D.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We have provided the details of dataset in Appendix E and Appendix F.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The usage of LLMs can be shown in the Appendix K.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.