HISTORY MATTERS: STRENGTHEN COLLABORATION THROUGH DELEGATION FOR MULTI-AGENT REINFORCEMENT LEARNING SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents play an increasingly important role in decision-making systems. In this paper, we consider collaborative heterogeneous multi-agent systems (MAS), where their members have different capabilities and operating costs. We study how agents can delegate tasks to one another so that certain research tasks can be completed effectively under resource-constrained scenarios. We first develop a multi-agent reinforcement learning-based (MARL) delegation training that enables agents to make sequential delegation decisions while minimizing the total execution cost. We then extend this approach to MARL with prescribed delegation topologies. Furthermore, we introduce two new frameworks for collaboration and delegation in multi-agent systems. The first framework proposes that an agent's policy depends not only on the current state of the underlying Markov decision process but also on the interaction history, including previous joint actions. This history-dependent formulation can improve coordination even in fully observable environments, where conventional MARL methods typically restrict policies to depend only on the current state. The second framework proposes a novel, potentially multi-dimensional monetary mechanism to facilitate the collaboration and delegation for MAS.

1 INTRODUCTION

AI agents have emerged as a promising approach for solving complex, multi-step problems. Large language models (LLMs) often serve as the reasoning engines of these agents, enabling them to interpret tasks, generate plans, use external tools, and coordinate specialized models or other agents. As agentic systems become more capable, they are also increasingly likely to contain multiple models with different capabilities, ownership relationships, and operating costs.

Despite their potential, deploying LLM-based agents can be expensive. Reliance on API-based models may incur substantial and unpredictable costs due to massive token usage. Using a premium model for every decision is often unnecessary: Smaller or less expensive models may be sufficient for routine tasks, while stronger models may be needed only for difficult or high-consequence decisions. Thus, an effective multi-agent system must determine both how to solve a task and which agent should execute each step.

Existing work addresses this cost-performance trade-off through model routing, multi-agent orchestration, and MARL-based collaboration. FrugalGPT, RouteLLM, and AutoMix select or cascade models according to expected quality, task difficulty, and invocation cost (Chen et al., 2023; Ong et al., 2025; Aggarwal et al., 2024), while Router-R1 and MasRouter use learned controllers to make sequential routing, role-assignment, and model-selection decisions (Zhang et al., 2025b; Yue et al., 2025). MAGRPO and CoLLM instead apply MARL to improve the task-level policies of LLM agents operating within predefined collaboration protocols (Liu et al., 2026b;a). Graph-based frameworks such as GPTSwarm and G-Designer optimize computational or communication connections among agents (Zhuge et al., 2024; Zhang et al., 2025a).

Our focus is complementary but distinct: we learn a delegation layer over agents that already possess pretrained task policies. Delegation is an explicit agent action rather than a decision imposed entirely by a centralized router. This formulation allows us to study which agent should execute,

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

who may delegate to whom, and how execution decisions depend on agent costs and prior interactions. Accordingly, we formulate task delegation among LLM-based agents as a MARL problem, enabling the delegation layer to adaptively and efficiently assign each task to the most appropriate agent, which then executes its own pretrained policy.

Delegation also raises questions on agents' willingness to collaborate. Whether an agent accepts a costly task may depend on whether another agent previously cooperated, fulfilled a delegation, or provided compensation, rather than because of the current physical state alone. Although histories are commonly used in partially observable environments to infer hidden state, we consider a **different role** for history: past joint actions and inter-agent interactions are strategically relevant even when the environmental state is fully observable. Such histories can support reciprocity, trust, and intertemporal incentives that may not be expressible by policies restricted to the current Markov state that were traditionally considered in fully observable MGs Shi et al. (2024); Zhang et al. (2023); Bai & Jin (2020).

Thus it is natural to ask "Which agents should be trusted to perform the tasks using their own pretrained policies?" A related question is that "When should agents trust their peers and delegate the work of solving the tasks to their peers?" In this paper, we investigate how agents should delegate tasks to other agents in a collaborating environment. Our contributions include the following:

1. Our work formulates delegation itself as a sequential multi-agent decision problem. Agents with heterogeneous pretrained capabilities choose whether to execute, delegate, or remain inactive; We propose a formulation in which AI agents delegate tasks to their peers subject to management and delegation constraints. The framework is designed to reduce operational costs and may also support the development of safer and more reliable multi-agent systems Overman & Bayati (2026).

2. We introduce a history-dependent policy formulation tailored to collaboration and delegation. The policies may depend on the public interaction history, including previous joint actions and states, rather than only on the current environmental state. We show that conditioning on previous behavior can support reciprocal collaboration outcomes.

3. To make history-dependent policies practical in MARL, we propose a virtual money-transfer framework that supports collaboration and delegation. An agent's money balance provides a compact summary of its past interactions, including previous actions and transfers. The virtual money is separate from the environmental rewards that measure the agent's actual task performance. By tracking the balances and constraining transfers, the system can encourage collaborations that lead to higher environmental rewards across time steps.

Related works: Overman & Bayati (2026) formulates an oversight game in which human supervisors decide when to intervene and when to rely on a supervised agent's pretrained policy during MARL training. Its human–agent delegation mechanism is designed for safety guarantees. In contrast, our work studies delegation among agents as a cost-allocation problem, with the primary objective of minimizing execution costs while maintaining effective task performance.

Schmid et al. (2018) studies action trading, where selfish agents can trade their actions in exchange for reward, and shows empirically that such action trading can increase the overall reward. Our money transfer framework differs in two aspects: First, the virtual money is separate from environmental rewards and records past interactions. This distinction is useful when task-level rewards are not transferable, while an internal budget can still be exchanged to facilitate delegation. Second, payments depend on the joint action, rather than only on the receiver's individual action.

The formulation of our transferable money system is also different from the series of peer-incentivation (PI) works because our formulation treats money as a component of the state variable, while PI type of works, such as *DRIVE* Altmann et al. (2026) and *LIO* Yang et al. (2020), usually treat incentives primarily as reward-shaping signals for policy learning or even rewards themselves. **Our money transfer system is useful even when environmental rewards are non-transferable.**

A line of reputation-based MARL research, including Anastassacos et al. (2021); Ren et al. (2025), also incorporates the effect from past interactions. However, the information resulted by past interactions in those works is either formulated as an explicit social norm or a learned social evaluation, and the resulting reputation score directly reshapes its environmental reward. In contrast, our money variable is neither a judgment of an agent's social standing nor a reward-shaping term. It is a trans-

ferable and budget-constrained balance. For more related works in routing across LLMs and MARL for LLM collaboration, please see Appendix A.

2 PROBLEM FORMULATION

Consider a collaborative Markov game (MG) with n agents:

$$\mathcal{M} = \{\mathcal{N}, \mathcal{S}, \mathcal{A}, P, R, \gamma, \{\mu_i\}_{i=1}^n, T, K\}, \tag{1}$$

where $\mathcal{N}$ is the set of agents, with $|\mathcal{N}| = n$, $\mathcal{S}$ is the shared state space, $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the joint agent-group action space, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function that depends on the joint action of the group, $\gamma \in [0, 1)$ is the discount factor, T is the number of steps in each episode and K is the number of episodes. Each agent is individually pre-trained in the same environment and equipped with its own learned policy $\mu_i : \mathcal{S} \times \mathcal{A}_i \to [0, 1], i = 1 \dots n$. The stronger the agent i is in intelligence, the better its pre-learned policy μ_i.

We consider this collaborative MG built on top of these pre-trained policies. This MG can be viewed as an extra "wrapper" layer between the pre-training of LLM agents and the deployment of these agents. The output of this MARL interface is $\boldsymbol{\pi} : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which is the joint group policy used to delegate tasks to specific agents. Notice that the delegation policy $\boldsymbol{\pi}$ is independent of the pre-trained policies $\{\mu_i\}_{i=1}^n$.

Define each agent's one-step reward as $R_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Then the collaborative one-step reward is $R(s_t, \boldsymbol{a}_t) = \sum_{i \in \mathcal{N}} R_i(s_t, \boldsymbol{a}_t)$. The collaborating group aims to maximize the objective function

$$J(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}, P}\Big[\sum_{t=1}^{T} \gamma^{t-1} \sum_{i \in \mathcal{N}} R_i(s_t, \boldsymbol{a}_t)\Big]. \tag{2}$$

On top of the collaborative game, we further introduce different delegation structures that are applicable in different scenarios. First, the **management constraints**, meaning that some agents have ownership of or control over the actions of fellow agents. Second, the **delegation constraints**, which generalize management constraints by allowing any agent to delegate tasks to any other agent. Third, the most general and realistic setting is **money trading**, in which agents incentivize others to undertake delegated tasks by allocating a portion of their held money as payment.

2.1 MANAGEMENT CONSTRAINTS

In the simplest case, assume that the stronger agent has one-way control over its fellow agent, and thus can mandatorily delegate work through this management structure. We take a 3 agent system as an example. Name the three agents A, B, and C. Suppose that agent A has control over agent B, and agent B has control over agent C. Although $A(B)$ is a more capable agent than $B(C)$, $B(C)$ is more cost-efficient in simple tasks than its corresponding superior. Therefore, when dealing with simple tasks, superiors have incentives to delegate tasks to their fellow agents. Regardless of whether each agent chooses to delegate or adopt its pre-trained action, there is only one agent actually executing at each step.

We assume that agents have heterogeneous capabilities and execution costs. At each time step, the current state contains sufficient information for agents to identify the task. Each agent knows everybody's capabilities and costs for the given task. Thus, before choosing whether to execute or delegate, agent i can determine which agents are able to complete the task and can compare their expected performance and execution costs. In this formulation, we adopt the full-information assumption and treat these profiles as known to all agents.

We formulate the delegation game as follows $\{\{A, B, C\}, \mathcal{S}, \mathcal{A}, R, P, \gamma, \{\mu_A, \mu_B, \mu_C\}\}$: The shared state space $\mathcal{S}$ is the same state space where agents were previously trained. For every agent i, its action space is augmented by its own delegation action space and the no-operation action, $\mathcal{A}_i' = \mathcal{A}_i \cup \mathcal{D}_i \cup \{\text{no-op}\}$, with $\mathcal{D}_i$ defined as $\mathcal{D}_A = \{d_{A \to B}\}, \mathcal{D}_B = \{d_{B \to C}\}, \mathcal{D}_C = \emptyset$. Each delegation action $d_{i \to j}$ means the agent i delegates the tasks to agent j. The expanded joint action space is the product of expanded action spaces $\mathcal{A}' = \prod_{i \in \{A,B,C\}} \mathcal{A}_i'$. The transition probability function, discount factor are defined the same way as those of (1).

$\{\mu_A, \mu_B, \mu_C\}$ are the pre-learned policies. For example, for each LLM agent, $\mu_i(\tilde{a}_i|s)$ denotes its fixed task-execution policy. This policy is induced by the agent's underlying pre-trained language model and available tools. Given a task state or context s, μ_i determines a distribution over the responses or environmental actions available to agent i.

Sequential Execution Rules: At each time step t, only one agent actually executes the delegated task. If agent i selects to perform, its action is sampled from its own pre-trained policy $a_{\mu_i} \sim \mu_i$. The execution action a_{exec} of the group is chosen in the following way:

$$a_{exec} = \begin{cases} a_{\mu_A}, \text{ if } \boldsymbol{a} = (a_{\mu_A}, \cdot, \cdot), \\ a_{\mu_B}, \text{ if } \boldsymbol{a} = (d_{A\to B}, a_{\mu_B}, \cdot), \\ a_{\mu_C}, \text{ if } \boldsymbol{a} = (d_{A\to B}, d_{B\to C}, a_{\mu_C}). \end{cases}$$

Here "$\cdot$" means any action of the corresponding agent. This execution rule follows exactly the hierarchical relations between agents A, B and C.

The reward function of each agent R_i consists of two parts: environmental reward $\bar{R}$ and individual execution or delegation cost $\mathcal{C}_i$. The environmental reward $\bar{R} : \mathcal{S} \times \mathcal{A}_{exec} \to \mathbb{R}$ is generated in correspondence to the executed action a_{exec}, and is the same for any agent i in the group.

The other part of the reward function is the individual execution or delegation cost $\mathcal{C}_i : \mathcal{A}_i \cup \mathcal{D}_i \cup \{\text{no-op}\} \to \mathbb{R}$. If agent i chooses to execute, then the execution cost is generated according to the specific execution cost $\mathcal{C}_i(a_{\mu_i})$ (For example, the execution cost of each agent can be represented by the API usage cost of the corresponding LLM.); If agent i chooses to delegate the task, there is a relatively smaller delegation cost $\mathcal{C}_i(d_{i\to j})$; If no-op is chosen, $\mathcal{C}_i(\text{no-op}) = 0, \forall i \in \mathcal{N}$.

Therefore, one-step reward function of each agent i is formulated as $R_i(s, \boldsymbol{a}) = \alpha\bar{R}(s, a_{exec}) - \beta\mathcal{C}_i(a_i)$, where α and β are scaling factors, since the environmental reward and the money cost differ in units. The objective of the agent group is still to maximize (2).

2.2 Delegation Constraints

In a more general case, when delegation is assigned in interactive ways between every two pairs of agents, delegation constraints can be represented through a directed graph. For example, an arrow from Agent A to Agent B represents A choosing to delegate to B, and vice versa. Compared to subsection 2.1, we omit the strict hierarchical relation in delegation rules.

With the same three-agent example as in 2.1, we let all other settings be the same but modify the action spaces and the execution rules. The individual delegation action space $\mathcal{D}_i$ of each agent i is now expanded, and any agent can delegate the task to any other agent in the group: $\mathcal{D}_A = \{d_{A\to B}, d_{A\to C}\}, \mathcal{D}_B = \{d_{B\to A}, d_{B\to C}\}, \mathcal{D}_C = \{d_{C\to A}, d_{C\to B}\}$

Networked Execution Rules: Multiple agents can execute at the same time. The execution action is now a 3-element vector $\boldsymbol{a}$, with each element indicating whether and with what action an agent is executing. Here $(\boldsymbol{a}_{\text{exec}})_i$ denotes the i-th element of the execution action.

$$(\boldsymbol{a}_{\text{exec}})_i = a_{\mu_i}, \text{if} \begin{cases} \exists\, j \neq i : \ a_j = d_{j\to i} \\ a_i \sim \mu_i. \end{cases}$$

This execution rule means that an agent performs if and only if some other agent delegates the task to it and it does not delegate to any other agent. For initialization, at the beginning of every episode, an oracle delegates the task to a selected subset of agents. The individual reward remains the same as that of 2.1, and the group objective is the same as (2).

In realistic human societies, whether a delegation relationship can be formed often depends on an individual's willingness to accept a costly task. To capture this feature, we allow each agent's willingness to accept such a task to depend on its previous interactions with other agents. In particular, an agent may be more willing to trust a peer and accept a delegated task if they have chosen collaborative joint actions in the past. We therefore model each agent's policy as depending on the history of past states and joint actions, rather than only on the current state.

2.3 POLICIES DEPENDENT ON PAST INTERACTIONS

In this subsection, we consider a class of history-dependent policies that differs from the conventional Markov policies commonly used in the MARL literature Shi et al. (2024); Zhang et al. (2023); Bai & Jin (2020). Under a Markov policy, each agent selects its action based only on their current state. By contrast, under a history-dependent policy, an agent may condition its action on the entire past interactions of the game.

Consider a finite dynamic game with n participants $\mathcal{G} = \{\mathcal{S}, \mathcal{A}_1 \times \cdots \times \mathcal{A}_n, P, T, r\}$. Let $x_t \in \mathcal{S}$ denote the state at time t, and let $\boldsymbol{a}_t = (a_t^1, \ldots, a_t^n)$ denote the joint action at time t, $1 \leq t \leq T$. We define the public history available at time t by $h_t = (x_1, \boldsymbol{a}_1, x_2, \boldsymbol{a}_2, \ldots, x_{t-1}, \boldsymbol{a}_{t-1}, x_t)$. The public history h_t contains all past states, all past joint actions, and the current state. For each agent i, a deterministic history-dependent policy profile is a sequence of policies $\{\pi_t^i\}_{t=1}^T$, where each history dependent policy maps from the public history up to time t to the action space of agent i, $\pi_t^i : \mathcal{H}_t \to \mathcal{A}_i,\ t = 1, \ldots, T$, where $\mathcal{H}_t$ denotes the set of histories up to time t. A Markov policy is a special case of history-dependent policies, where the policy depends only on the current state and is irrelevant to past trajectories: $\pi_t^i(h_t) = \mu_t^i(x_t)$.

The key advantage of a history-dependent policy is that it allows agents to reward, punish, or condition their actions based on the previous behavior of other agents. We have a motivating example in Appendix B. This example illustrates that history-dependent policies can support reciprocity mechanisms that are not expressible by Markov policies. It can even incentivize equilibrium policies with higher returns. Since future actions can depend on past joint actions, one agent may be incentivized to take a costly action early in the game if another agent can condition its later behavior on that action. In this sense, history-dependent policies may support equilibria with higher returns than those obtained under the Markov policy restriction.

Remark. Notice that the regular Markov policy space is a subspace of the history-dependent policy space, so the return of a history-dependent policy is at least as good as that of a Markov policy.

Algorithm: We develop a *Two-step Nash Value Iteration Algorithm* to find the overall deterministic Nash equilibrium policy that favors agent B. In the case when there are multiple pure Nash equilibria, depending on the agents' need, Algorithm 1 outputs the policy profile that favors the return of the chosen agent i. Please refer to Appendix C for details of this algorithm. We prove that the output of this algorithm is a history-dependent pure-strategy Nash equilibrium.

Theorem 1. *Consider a finite two-step dynamic game. If the two-step continuation game admits at least one pure-strategy Nash equilibrium, and that at least one induced one-step game admits a pure-strategy Nash equilibrium, then Algorithm 1 returns a pure-strategy subgame-perfect Nash equilibrium policy profile.*

Proof. We prove by showing that none of the agent i prefers to deviate from the output policy π^* at any time step $t = 1$ and $t = 2$. Please see Appendix D for details of this proof. □

Extension of Algorithm 1 to Finite n-step Games: The two-step algorithm can be extended to an n step dynamic game by induction. First, solve the final two-step continuation game, consisting of times $n-1$ and n using Algorithm 1. This generates the set of deterministic history-dependent Nash equilibrium continuation policies for the last two steps.

Next, apply each continuation policy for the previous step $n-2$. For every possible history h_{n-2}, compute the immediate reward plus the expected equilibrium value generated by the selected continuation policy. Then find the Nash equilibria of this induced one-step game and combine each equilibrium action with the continuation policy. This gives the equilibrium policies for the last three steps. Repeat this procedure backward until the algorithm yields the deterministic history-dependent Nash equilibrium policies for the entire n-step dynamic game.

2.4 DELEGATION THROUGH TRANSFERABLE MONEY

We now consider a more realistic setting in which agents can form delegation contracts through transferable money. The key feature of this framework is that money is separate from the environmental rewards that agents aim to maximize. The reward function represents the real-world utility obtained by each agent from the environment, while money only serves as an internal medium as

a record of past joint actions and states, and money enables delegation and coordination in a distributed multi-agent reinforcement learning.

Consider a Markov game (1). Let s_t denote the environmental state at time t, and let $\mathcal{A}_i$ be the environmental action space of agent i. To improve this regular Markov game with a money transfer mechanism, each agent is equipped with a money balance state $w_t^i \geq 0, \forall t, i$, and define the augmented individual state as $\tilde{s}_t^i = (s_t, w_t^i)$. Now the individual state records both the environmental state and the leftover money balance of agent i. More generally, the money can be multi-dimensional instead of a scalar.

At each time step t, each agent i chooses a physical action a_t^i and a money transfer action $\boldsymbol{p}_t^i$. This physical action is mapped from the environmental policy $a_t^i \sim \pi_i^{\text{env}}(\tilde{s}_t^i)$.

Money Transfer Actions The money transfer action of each agent i is $\boldsymbol{p}_t^i = [p_t^{ij}]_{j \neq i} \in \mathbb{R}_+^{n-1}$. Its domain is determined by a prescribed contract: $\text{Cr}^i : \mathcal{A} \to \mathbb{R}^{n-1}$ that maps from each *joint action* to a list of fixed prices agent i is willing to pay for other agents. At time step t, if agent j performs the corresponding physical action a_j at step t, agent i pays money $p_t^{ij} = \text{Cr}_j^i(\boldsymbol{a})$ to agent j (we can also make the amount depend on historical actions). Thus, the money transfer actions $\boldsymbol{p}_t^i$ are chosen according to the current augmented state and the current joint action (it can also depend on historical actions or trajectories): $\boldsymbol{p}_t^i \sim \pi_i^{\text{tr}}(\cdot|\tilde{s}_t, \boldsymbol{a}_t)$. Agent i's transfer policy is therefore denoted by $\pi_i^{\text{tr}} : \tilde{s}_t \times \mathcal{A} \to \mathbb{R}_+^{n-1}$. This policy determines the vector of payments $[p_t^{ij}]_{j \neq i}$ made by agent i to all other agents. The full policy of agent i is therefore $\pi_i = (\pi_i^{\text{env}}, \pi_i^{\text{tr}})$, and the joint policy is $\boldsymbol{\pi} = (\boldsymbol{\pi}^{\text{env}}, \boldsymbol{\pi}^{\text{tr}})$. The agent's objective is still purely environmental: $J(\boldsymbol{\pi}) = \mathbb{E}^{\boldsymbol{\pi}}[\sum_{t=1}^{T} \sum_{i=1}^{N} \gamma^{t-1} R_i(s_t, \boldsymbol{a}_t)]$, and money does not directly enter the reward function.

Insufficient Money Balance When agent i wants to pay more than its current balance, i.e., $\|\boldsymbol{p}_t^i\|_1 > w_t^i$, then the desired joint action is not feasible, and agent i needs to turn to an affordable joint action. When w_t^i drops to 0, then no transfer is allowed.

3 Numerical Analysis

This section presents the numerical results tied to the mechanisms introduced in the problem formulations. We begin with management and delegation structures on MARL. The delegation policy is trained for solving GSM8K (Cobbe et al., 2021), a public grade-school math word-problem benchmark with reference answers, using one shared solver stack and changing only the interaction rule between the two subsections. Protocol knobs that matter for reproducibility—prices, caches, seeds, hyper-parameters, and compute—are collected in Appendix E. The remaining subsections treat history-dependent policies and money transfer separately.

Shared GSM8K stack. We use three frozen solvers of different strength and cost on this math benchmark: Nemotron 3 Ultra as solver A, Llama 3.2 3B as solver B, and Llama 3.2 1B as solver C. Only a centralized DQN coordinator π is trained; A, B, and C stay frozen. Synthetic token prices make solver A expensive and solver C cheap (Appendix E). Before training the MARL interface, we cache each solver's GSM8K answers and offline correctness labels once; during training π reuses those fixed offline answers and labels instead of calling the solvers or a live labeler at every step, which keeps the runs reproducible (Appendix E). Each episode packs 32 questions, allows up to three attempts per question (fail penalty -1), and runs for 1000 episodes with $\gamma = 0.99$, cost weight $\lambda = 100$, and handoff fee $\alpha = 0.05$. Unless noted otherwise, we report mean$\pm$std of return, accuracy, and cost across seeds 0–19.

The *always-Ultra* baseline is a no-handoff policy that sends every question to solver A alone, under the same caches and cost rules as the learned policy in that subsection.

3.1 Management Constraints on GSM8K

Here we put the hierarchy of Section 2.1 on GSM8K in a restricted form. The coordinator may use only three routes along the chain $A \to B \to C$: (i) solver A answers alone; (ii) A hands the question to solver B, and B answers; or (iii) the question is passed $A \to B \to C$, and solver C

Late-train metric	DQN+map	DQN no map	always-Ultra
Return	27.56±0.32	27.65±0.28	23.90±0.30
Accuracy	0.982±0.002	0.982±0.002	0.971±0.002
Cost ($/ep)	0.033±0.002	0.032±0.002	0.063±0.002
Cum. cost ($/1000)	31.1±0.7	31.1±0.7	60.8±0.2

Table 1: Management Constraints: Late-training metrics (last 100 episodes; mean±std over 20 seeds).

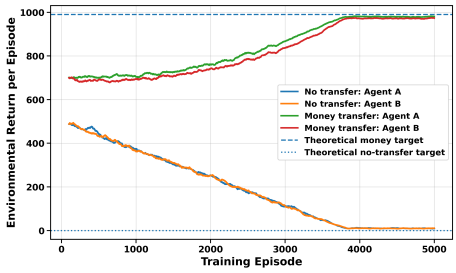


Figure 1: Returns of independent Q-learning with money transfer Mechanism.

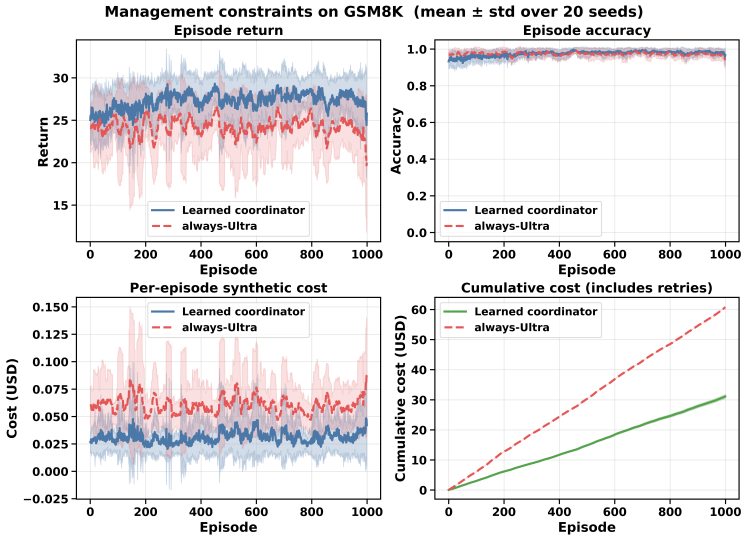


Figure 2: Management constraints on GSM8K: mean±std over 20 seeds for return, accuracy, per-episode cost, and cumulative cost (MARL learned coordinator vs always-Ultra).

answers. Exactly one solver runs on each attempt. Skipping straight from A to C (allowed in the theory section) is turned off here, so every handoff must follow the chain in order.

The step reward credits a correct answer under the offline labels, charges for the solve and for each handoff, and applies a failure penalty after three misses:

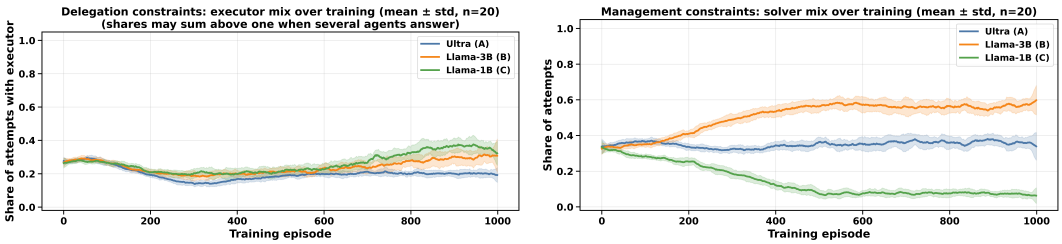
$$r \;=\; R_{\text{corr}} \;-\; \lambda\, C_{\text{solve}}(a_{\text{exec}}) \;-\; \lambda\, C_{\text{edge}}(n_e, a_{\text{exec}}) \;+\; R_{\text{fail}},$$

where $C_{\text{edge}} = n_e\, \alpha\, C_{\text{solve}}$ with $\alpha = 0.05$, and R_{fail} is 0 on ordinary steps and -1 after three failed attempts (so the leading "+" adds a negative penalty; see Appendix E).

Table 1 and Figure 2 show the results in the late MARL training. The *always-Ultra* baseline has return 23.90±0.30 at the cost of $0.063±0.002 per episode. The MARL learned coordinator reaches 27.56±0.32 return at the cost of $0.033±0.002 per episode (about $31.1 vs. $60.8 cumulative over 1000 episodes), with accuracy 0.982±0.002 versus 0.971±0.002. That is roughly +3.7 return at about half the synthetic cost. A paired Wilcoxon signed-rank test over the 20 seed-wise late-train means finds higher return and lower cost than always-Ultra in all 20 seeds (two-sided $p = 2^{-19}$). Figure 3b tracks which solver is called. Late policies lean on solver B, still ask solver A on a nontrivial share, and almost never rely on solver C.

3.2 Delegation Constraints on GSM8K

We keep the same GSM8K stack and swap to the freer delegation rule indicated in section 2.2. The coordinator first chooses a non-empty kickoff set among the seven options $\{\{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}$, and $\{A, B, C\}\}$. Then, each of A, B, and C chooses one of the three actions: answer, pass to the first other agent, or pass to the second other agent. Packing kickoff with



(a) Delegation Constraints.

(b) Management Constraints.

Figure 3: Share of attempts using Ultra (A), Llama 3B (B), and Llama 1B (C) over training (mean±std over 20 seeds).

Late-train metric	DQN+map	DQN no map	always-Ultra	Mgmt. π
Return	25.20±0.42	25.21±0.33	23.31±0.30	28.70±0.18
Accuracy	0.961±0.005	0.961±0.004	0.971±0.002	0.986±0.001
Cost ($/ep)	0.043±0.002	0.043±0.002	0.068±0.002	0.024±0.001
Cum. cost ($/1000)	45.5±0.6	45.4±0.6	66.7±0.2	23.9±0.1

Table 2: Delegation constraints: Late-training metrics (last 100 episodes; mean±std over 20 seeds; handoff fee $\alpha = 0.05$). Mgmt. π is the frozen management coordinator on the same packets.

the three local choices gives $7 \times 3^3 = 189$ joint discrete actions. An agent answers only if it was handed the task (by kickoff or by a peer pass) and it chooses to answer; Multiple agents may answer in the same attempt. We count the attempt as successful if any agent is correct. We charge the sum of those agents' solve costs, and add $\alpha = 0.05$ edge fee (each kickoff, handoff, and peer pass).

Besides *always-Ultra*, we roll out the frozen management coordinator from section 3.1 on the same packets for a matched comparison. *Always-Ultra* is evaluated in the same graph environment, so it also pays the kickoff edge fee when solver A is started.

Table 2 and Figure 4 show results in the late training. Against *Always-Ultra*, which returns 23.31 ± 0.30 at the cost of $\$0.068 \pm 0.002$ per episode, the graph coordinator reaches 25.20 ± 0.42 return at the cost of $\$0.043 \pm 0.002$/episode, with accuracy 0.961 ± 0.005 versus 0.971 ± 0.002. There is about 1.9 increased return at roughly 37% lower per-episode cost. The cumulative synthetic cost over 1000 episodes falls from about $\$66.7$ to $\$45.5$. The same Wilcoxon test again finds a higher return and lower cost than *Always-Ultra* in all 20 seeds with two-sided $p = 2^{-19}$. On matched packets, the management coordinator is still ahead. It returns 28.70 ± 0.18, with accuracy 0.986 ± 0.001, at the cost $\$0.024 \pm 0.001$ per episode. Therefore, the graph delegation beats never handing off, while the hierarchy policy remains stronger on return at a lower cost in these runs.

Figure 3a shows the executor mix. Compared with management routing, the graph policies call solver C more often, and sometimes several agents answer the same question. That freer pattern helps explain why return and accuracy sit a bit below the hierarchy policy, even though the cost stays below *Always-Ultra*.

3.3 Interactions of Distributed Agents with the Money Transfer Mechanism

We compare independent Q-learning (IQL) with and without money transfer in a two-agent sequential game. Each episode has 20 steps and training lasts 5000 episodes. Agents A and B alternate as the active agent, with A active on odd-numbered steps and B on even-numbered steps; only the active agent's action affects the physical reward.

The state is $s = (\text{phase}, t)$, where $\text{phase} \in \{s_1, s_2\}$ identifies the active agent and $t \in \{0, \ldots, 20\}$ is the time index. Each agent chooses Null or Yes, corresponding respectively to not helping or helping the other agent. The phase alternates deterministically after every step.

The active agent chooses whether to incur a small personal cost to generate a large benefit for the other agent. When agent A is active, the reward function is as follows, and the reward function for B is symmetric when it is active.

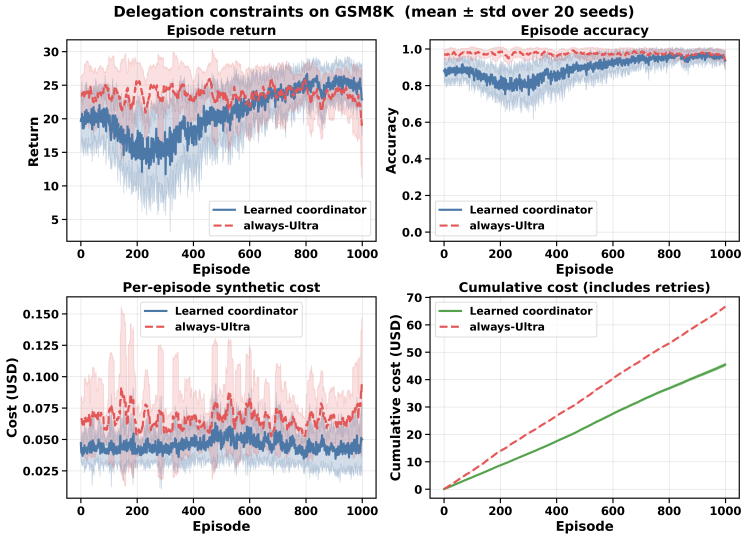


Figure 4: Delegation constraints on GSM8K: mean±std over 20 seeds for return, accuracy, per-episode cost, and cumulative cost (learned coordinator vs always-Ultra).

$$(r_t^A, r_t^B) = \begin{cases} (0,0), \text{ if } A \text{ chooses Null} \\ (-1,100), \text{ if } A \text{ chooses Yes} \end{cases} ; (r_t^A, r_t^B) = \begin{cases} (0,0), \text{ if } B \text{ chooses Null} \\ (100,-1), \text{ if } B \text{ chooses Yes} \end{cases} .$$

The returns of the independent Q-learning with a money transfer mechanism are shown in Figure 1. *1. Independent Q-learning (IQL)* Agents A and B maintain separate Q-functions and independently select their actions using an ε-greedy policy. Each agent updates its action values using only its own environmental reward, while treating the other agent's behavior as part of the environment. After the 5000-episode training, the learned policy is that both agents refuse to collaborate. On agent A's turns, agent A always chooses the action Null. On agent B's turns, agent B always chooses the action Null as well. This non-cooperative joint policy results in a 0 episodic environmental return.

2. IQL with Money Transfer With the new formulation, we consider an extended independent Q-learning with money transfers. The environment is augmented with transferable money balances, and therefore a state consists of $s = (\text{phase}, \text{t}, \boldsymbol{w})$, where $\boldsymbol{w} = (w_1, w_2)$ is the money balances of both agents. Each agent starts with an initial balance of 1 unit. Before the active agent chooses its physical action, the other agent acts as the payer and decides whether to offer a fixed payment 1 unit for the Yes action. An offer is made only when the payer has sufficient funds: $w_i \geq 1$. If the payment is made, the corresponding amount is transferred from the payer's balance to the active agent's balance, and the contract requires the active agent choosing the action Yes.

After 5000 training episodes, the learned joint policy is that on agent A's turns, agent B transfers one unit to A. By contract, agent A chooses the action Yes to help the agent B. The physical and money policies are symmetric on agent B's turns. For every episode with 20 time steps, both agents helped at all 10 of their active stages and obtained an environmental return of 990. Thus, the money-transfer framework successfully supported the fully cooperative outcome.

Three-agent collaboration through money transfer Through experiments, we show that money transfer can significantly boost collaboration between agents, compared with the no-money-transfer scheme. Please see Appendix Section F for details.

4 Conclusion

In this paper, we study the delegation problem among a heterogeneous multi-agent group. We also proposed frameworks of collaboration/delegation for multi-agent systems with history-based policies and money-based policies. In terms of limitations, this work has not addressed the pricing of the joint actions based on history, even though we have shown that certain pricing of the joint actions can help the agents achieve collaborations. More efficient algorithms are needed for finding the Nash equilibria and coarse correlated equilibria using history-dependent policies.

AI USE STATEMENT

In this work, we used generative AI tools to generate experiment code. That code was reviewed manually before use. We used the AI tools to help polish the writing of the paper. We have not used generative AI tools to draft or edit the paper, to generate synthetic datasets, to develop the theoretical model, to formulate mathematical claims, to write proofs, to propose the hypotheses tested here, to translate text, to clean datasets, or to carry out qualitative data analysis. Survey design and transcription are not applicable to this work. The numerical results come from our training runs. We take responsibility for the final content of this work.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. AutoMix: Automatically mixing language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4164. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ecda225cb187b40ea8edc1f46b03ffda-Abstract-Conference.html.

Philipp Altmann, Thomy Phan, Maximilian Zorn, Claudia Linnhoff-Popien, and Sven Koenig. Dynamic incentivized cooperation under changing rewards, 2026. URL https://arxiv.org/abs/2601.06382.

Nicolas Anastassacos, Julian García, Stephen Hailes, and Mirco Musolesi. Cooperation and reputation dynamics with reinforcement learning, 2021. URL https://arxiv.org/abs/2102.07523.

Yu Bai and Chi Jin. Provable self-play algorithms for competitive reinforcement learning. *arXiv preprint arXiv:2002.04017*, 2020.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023. doi: 10.48550/arXiv.2305.05176. URL https://arxiv.org/abs/2305.05176.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. RouterBench: A benchmark for multi-LLM routing system. *arXiv preprint arXiv:2403.12031*, 2024. doi: 10.48550/arXiv.2403.12031. URL https://arxiv.org/abs/2403.12031.

Dima Ivanov, Paul Dütting, Inbal Talgam-Cohen, Tonghan Wang, and David C. Parkes. Principal-agent reinforcement learning: Orchestrating AI agents with contracts, 2024. URL https://arxiv.org/abs/2407.18074.

Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, Zhen Wang, and Shuyue Hu. LLMRouterBench: A massive benchmark and unified framework for LLM routing. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37733–37754, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1881. URL https://aclanthology.org/2026.findings-acl.1881/.

Shuo Liu, Tianle Chen, Ryan Amiri, and Christopher Amato. Learning decentralized LLM collaboration with multi-agent actor critic. *arXiv preprint arXiv:2601.21972*, 2026a. doi: 10.48550/arXiv.2601.21972. URL https://arxiv.org/abs/2601.21972.

Shuo Liu, Zeyu Liang, Xueguang Lyu, and Christopher Amato. LLM collaboration with multi-agent reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(38): 32150–32158, 2026b. doi: 10.1609/aaai.v40i38.40487. URL https://ojs.aaai.org/index.php/AAAI/article/view/40487.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html.

William Overman and Mohsen Bayati. The oversight game: Learning to cooperatively balance an ai agent's safety and autonomy, 2026. URL https://arxiv.org/abs/2510.26752.

Tianyu Ren, Xuan Yao, Yang Li, and Xiao-Jun Zeng. Bottom-up reputation promotes cooperation with multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2502.01971.

Kyrill Schmid, Lenz Belzner, Thomas Gabor, and Thomy Phan. Action markets in deep multi-agent reinforcement learning. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis (eds.), *Artificial Neural Networks and Machine Learning – ICANN 2018*, pp. 240–249, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01421-6.

Laixi Shi, Eric Mazumdar, Yuejie Chi, and Adam Wierman. Sample-efficient robust multi-agent reinforcement learning in the face of environmental uncertainty, 2024. URL https://arxiv.org/abs/2404.18909.

Jiachen Yang, Ang Li, Mehrdad Farajtabar, Peter Sunehag, Edward Hughes, and Hongyuan Zha. Learning to incentivize other learning agents, 2020. URL https://arxiv.org/abs/2006.06051.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24611–24624, 2022. URL https://arxiv.org/pdf/2103.01955.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. MasRouter: Learning to route LLMs for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.757. URL https://aclanthology.org/2025.acl-long.757/.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-Designer: Architecting multi-agent communication topologies via graph neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76678–76692. PMLR, 2025a. URL https://proceedings.mlr.press/v267/zhang25cu.html.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-R1: Teaching LLMs multi-round routing and aggregation via reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/ceaa137fce916aba5c65fceb1309088b-Abstract-Conference.html.

Kaiqing Zhang, Sham M. Kakade, Tamer Basar, and Lin F. Yang. Model-based multi-agent rl in zero-sum markov games with near-optimal sample complexity. *Journal of Machine Learning Research*, 24(175):1–53, 2023.

Tong Zhu, Baiting Chen, Jin J. Zhou, Hua Zhou, Sriram Sankararaman, and Xiaowu Dai. Competitive multi-agent delegation for LLM reasoning, 2025. URL https://openreview.net/forum?id=nDdpp0285M. Submitted to ICLR 2026; OpenReview forum id nDdpp0285M.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Related Work

A.1 Cost-Aware Routing Across Language Models

A growing body of work studies how to exploit differences in the capabilities and costs of available language models. FrugalGPT considers prompt adaptation, model approximation, and cascades of language models to reduce inference cost while maintaining response quality (Chen et al., 2023). RouteLLM learns a router from preference data to select between a stronger, more expensive model and a weaker, less expensive model (Ong et al., 2025). AutoMix first queries a smaller model and uses self-verification together with a partially observable decision process to determine whether the query should be escalated to a larger model (Aggarwal et al., 2024). RouterBench and LLMRouterBench provide standardized datasets and evaluation frameworks for studying routing performance and performance–cost trade-offs across collections of language models (Hu et al., 2024; Li et al., 2026). These approaches primarily treat routing as a selection problem in which a centralized controller assigns a query to one or more passive models. The routed models do not independently choose whether to execute, delegate, or reject a task. Our management formulation shares the objective of assigning simple tasks to inexpensive agents and difficult tasks to more capable agents, but it represents execution and delegation as actions within a Markov game. The resulting policy therefore describes not only which model should be used, but also which agent initiates the delegation and which delegation paths are feasible under the organizational structure Recent work has extended routing from single-step model selection to sequential decision making. Router-R1 formulates multi-model routing and aggregation as an RL problem in which an LLM router interleaves internal reasoning with model-invocation actions and receives a cost-aware reward (Zhang et al., 2025b). MasRouter expands the routing problem to multi-agent systems by jointly determining a collaboration mode, assigning fixed roles, and selecting an LLM for each role using a cascaded controller (Yue et al., 2025). These methods are particularly close to our cost-performance objective because they adaptively determine which model capabilities should be invoked rather than relying on a fixed cascade. Nevertheless, the router in these systems remains a centralized orchestrator that constructs or controls the multi-agent workflow. In our framework, delegation is instead endogenous to the participating agents. An agent may exercise its pretrained policy, delegate to another agent, or remain inactive, and the validity and outcome of these choices depend on the actions of the other agents. This formulation supports questions that are not represented by centralized routing, including whether a delegated agent accepts execution, how authority constrains delegation, how reciprocal behavior develops over repeated interactions, and how agents compensate one another for costly actions.

A.2 Multi-Agent Reinforcement Learning for LLM Collaboration

Recent studies have directly formulated LLM collaboration as a cooperative MARL problem. Liu et al. introduce Multi-Agent Group Relative Policy Optimization, or MAGRPO, which jointly fine-tunes multiple LLM agents using group-level feedback in multi-agent, multi-turn writing and coding tasks (Liu et al., 2026b). Their algorithm addresses the limitation that independently pretrained language models are not necessarily optimized to coordinate and demonstrates that collaborative behavior can be improved through a shared multi-agent training objective. In another work, CoLLM-CC (Liu et al., 2026a) uses a centralized critic during training, whereas CoLLM-DC uses decentralized critics. The comparison shows that Monte Carlo optimization and decentralized value estimation can perform well in relatively short-horizon, dense-reward tasks, while a centralized critic is more effective in long-horizon or sparse-reward settings. This work establishes centralized training with decentralized execution as a promising approach for optimizing LLM-agent collaboration and is directly relevant to the algorithmic implementation of multi-step delegation. MAGRPO and CoLLM are the closest prior works to ours in their use of MARL for multiple LLM agents. However, the learning targets are different. These methods primarily optimize the agents' language-generation policies within a specified collaboration and execution protocol. Participating agents produce responses or actions according to their assigned roles, and MARL improves how effectively those agents collaborate. Our primary formulation instead learns a meta-policy over agents that already possess pretrained task policies. The meta-policy determines whether a particular pretrained policy should be exercised and whether execution should be delegated to another agent. Thus, MAGRPO and CoLLM optimize behavior within a collaborative arrangement, whereas our framework optimizes the delegation and activation structure governing which underlying capability is used. In

other words, our framework can be utilized in conjunction with MAGRPO and CoLLM. The treatment of heterogeneity also differs. Existing LLM-MARL experiments may pair agents with different model architectures or complementary roles, but they do not primarily formulate heterogeneity as a capability; cost trade-off in which invoking a stronger policy incurs a larger execution cost. Our formulation explicitly associates the underlying agents with different policy quality and operating costs and learns when the additional capability of an expensive agent justifies its use. Moreover, our delegation policies may be restricted by management or delegation relationships, while the LLM-MARL methods above assume a predefined protocol describing which agents participate and how their outputs are combined. Finally, the two lines of work assign different roles to interaction history. In decentralized LLM collaboration, local trajectories and centralized information are useful for handling partial observability and estimating long-horizon returns. In our history-dependent game, previous actions, delegation choices, and transfers are also strategically meaningful. Agents may condition future behavior on whether another agent previously cooperated, accepted a costly task, or honored a payment. This distinction allows history to support reciprocity and intertemporal incentives even when the current environmental state is fully observable Closer to our terminology of *delegation* and *contracts*, Ivanov et al. combine deep RL with principal–agent contract theory so a principal can steer agents through outcome-contingent payments (Ivanov et al., 2024), while Zhu et al. propose COMMAND, in which a principal LLM competitively delegates reasoning tasks to multiple agent LLMs (Zhu et al., 2025). Relative to these lines, we focus on cost-aware management and graph-style delegation among frozen heterogeneous solvers, together with history-dependent reciprocity and transferable money inside a Markov-game wrapper.

B Motivating Example of History-dependent Policies

Motivating example: Consider a two-agent dynamic game with two agents A and B, with two time steps $T = 2$, and with a state space $\mathcal{S} = \{s_1, s_2\}$. The state at time t is denoted by x_t. The action spaces are $\mathcal{A}_1 = \{a_{A,1}, a_{A,2}\}$, $\mathcal{A}_2 = \{a_{B,1}, a_{B,2}\}$. The joint action is written as $\boldsymbol{a}_t = (a_t^A, a_t^B)$.

At the first time step, agent A's action determines the reward, while at the second time step, agent B's action determines the reward:

$$r_1(x_1, \boldsymbol{a}_1) = \begin{cases} (0,0), & \text{if } a_1^A = a_{A,1}, \\ (-0.1, 1), & \text{if } a_1^A = a_{A,2}, \end{cases}; r_2(x_2, \boldsymbol{a}_2) = \begin{cases} (0,0), & \text{if } x_2 = s_1 \text{ and } a_2^B = a_{B,1}, \\ (9,0), & \text{if } x_2 = s_1 \text{ and } a_2^B = a_{B,2}, \\ (0,10), & \text{if } x_2 = s_2 \text{ and } a_2^B = a_{B,1}, \\ (9,10), & \text{if } x_2 = s_2 \text{ and } a_2^B = a_{B,2} \end{cases}$$

where $a_{A,2}$ is costly for agent A, but beneficial for agent B. $a_{B,2}$ benefits agent A, while agent B's own second-step reward is independent of whether it chooses $a_{B,1}$ or $a_{B,2}$. The state transition is given by $\mathbb{P}(x_2 = s_2 \mid x_1 = s_1, \boldsymbol{a}_1) = 0.5, \mathbb{P}(x_2 = s_1 \mid x_1 = s_1, \boldsymbol{a}_1) = 0.5, \forall \boldsymbol{a}_1 \in \mathcal{A}$.

History-dependent Nash equilibrium. Under the history-dependent policy formulation, the following policy profile is one deterministic Nash equilibrium:

$$\pi_1^A(h_1) = a_{A,2}, \pi_2^A(h_2),\ \pi_1^B(h_1) \text{ any action in } \mathcal{A}_1, \mathcal{A}_2,\ \pi_2^B(h_2) = \begin{cases} a_{B,1}, & \text{if } a_1^A = a_{A,1}, \\ a_{B,2}, & \text{if } a_1^A = a_{A,2}. \end{cases}$$

This policy is history-dependent because agent B's second-period action depends on the past joint action $\boldsymbol{a}_1$, not only on the current state x_2.

Under this policy profile, agent A chooses the costly action $a_{A,2}$ at the first time step. In response, agent B chooses $a_{B,2}$ at the second time step, which rewards agent A. The expected returns are $V_0^A = -0.1 + 0.5(9 + 9) = 8.9$, and $V_0^B = 1 + 0.5(0 + 10) = 6$.

Markov Nash equilibrium. Now suppose that agents are restricted to Markov policies. Then agent B's second-period policy can depend only on the current state x_2, but not on agent A's first-period action. Therefore, the history-dependent reward mechanism described above is no longer available.

An example Markov equilibrium is given by

$$\pi_1^A(x_1) = a_{A,1}, \qquad \pi_2^A(x_2) \text{ can be any action in } \mathcal{A}_1,$$
$$\pi_1^B(x_1) \text{ can be any action } \mathcal{A}_2, \qquad \pi_2^B(x_2) = a_{B,1}.$$

Algorithm 1 Two-step Nash Equilibrium Value Iteration

Require: Initial state x_1; state space $\mathcal{S}$; action spaces $\mathcal{A}_A$ and $\mathcal{A}_B$; transition probabilities $P(x_2 \mid x_1, \boldsymbol{a}_1)$; rewards r_t^i and for $i \in \{A, B\}, t \in \{1, 2\}$.

Ensure: A deterministic history-dependent Nash equilibrium policy profile that maximizes agent B's value.

Define the set of feasible two-step histories:

$$\mathcal{H}_2 = \{(\boldsymbol{a}_1, x_2) : \boldsymbol{a}_1 \in \mathcal{A}, x_2 \in \mathcal{S}\}$$

for each history $h_2 = (\boldsymbol{a}_1, x_2) \in \mathcal{H}_2$ **do**

Define the two-step normal-form game by

$$Q_2^i(h_2, \boldsymbol{a}_2) = r_2^i(x_2, \boldsymbol{a}_2), \qquad i \in \{A, B\}.$$

Compute the set of pure Nash equilibria

$$\mathcal{E}_2(h_2) = \Big\{ \boldsymbol{a}_2 \in \mathcal{A} : \ Q_2^i\big(h_2, (a_2^i, \boldsymbol{a}_2^{-i})\big) \geq Q_2^i\big(h_2, (\widetilde{a}_2^i, \boldsymbol{a}_2^{-i})\big), \quad \forall \widetilde{a}_2^i \in \mathcal{A}_i, \forall i \in \{1, 2, \ldots, N\} \ \Big\}$$

end for

Enumerate all deterministic continuation selectors

$$\sigma_2 : \mathcal{H}_2 \longrightarrow \mathcal{A}$$

satisfying

$$\sigma_2(h_2) \in \mathcal{E}_2(h_2), \qquad \forall h_2 \in \mathcal{H}_2$$

Initialize the set of candidate equilibrium policy profiles: $\mathcal{C} \leftarrow \varnothing$

for each continuation selector $\sigma_2 \in \mathcal{S}_2$ **do**

for each joint action $\boldsymbol{a}_1 \in \mathcal{A}$ at $t = 1$, **do**

Compute the continuation value

$$W_i^{\sigma_2}(x_1, \boldsymbol{a}_1) = \sum_{x_2 \in \mathcal{S}} P(x_2 \mid x_1, \boldsymbol{a}_1) r_2^i(x_2, \sigma_2(\boldsymbol{a}_1, x_2)), \forall i$$

Compute the induced utilities at $t = 1$

$$Q_1^{i,\sigma_2}(x_1, \boldsymbol{a}_1) = r_1^i(x_1, \boldsymbol{a}_1) + W_i^{\sigma_2}(x_1, \boldsymbol{a}_1)$$

end for

Find the set of pure Nash equilibria of the induced game at $t = 1$:

$$\mathcal{E}_1(\sigma_2) = \big\{ \boldsymbol{a}_1 : \ Q_1^{i,\sigma_2}\big(x_1, (a_1^i, \boldsymbol{a}_1^{-i})\big) \geq Q_1^{i,\sigma_2}\big(x_1, (\widetilde{a}_1^i, \boldsymbol{a}_1^{-i})\big), \quad \forall \widetilde{a}_1^i \in \mathcal{A}_i, \forall i \ \big\}$$

for each $\boldsymbol{a}_1 \in E_1(\sigma_2)$ **do**

Construct the equilibrium policy profile

$$\pi = \big(\boldsymbol{a}_1, \sigma_2\big)$$

Set

$$V_i^{\pi}(x_1) = Q_1^{i,\sigma_2}(x_1, \boldsymbol{a}_1), \qquad \forall i, \forall x_1 \in \mathcal{S}, \forall \pi$$

Add π to $\mathcal{C}$

end for

end for

Select

$$\pi^* = \arg\max_{\pi \in \mathcal{C}} \|V_i^{\pi}\|_1$$

return π^* and its equilibrium value $\{V_i^{\pi,*}\}_{i=1}^N$

Under this Markov policy profile, agent A does not choose the costly first-period action, since agent B's future action cannot be conditioned on whether A previously helped B. The expected returns are $V_0^A = 0 + 0.5(0 + 0) = 0$, and $V_0^B = 0 + 0.5(0 + 10) = 5$. In fact, we cannot find any equilibrium for history-independent policies where agent B achieves a reward bigger than 6.

We numerically verified that among all the Markov deterministic equilibria policies, there are 3 possible outcomes in terms of value: $\{(0, 0, 5.0), (4.5, 5.0), (9.0, 5.0)\}$, where within each tuple, the first element is the return of agent A and the second element is the return of agent B. Meanwhile, among all the history-dependent deterministic equilibria, there are 5 possible outcomes: $\{(0.0, 5.0), (4.4, 6.0), (4.5, 5.0), (8.9, 6.0), (9.0, 5.0)\}$.

C Algorithm Finding History-dependent Policies [1]

D PROOF OF THEOREM 1

Proof. We prove by showing that no agent i prefers to deviate from the output policy π^* at any time step $t = 1$ and $t = 2$.

Without loss of generality, suppose that agent i changes its policy in both steps. Instead of adopting

$$\pi_1^{i,*}(x_1) = a_1^{i,*} \text{ and}$$
$$\pi_2^{i,*}(x_2, \boldsymbol{a}_1) = a_2^{i,*}$$

The agent takes

$$\widetilde{a}_1^i \text{ and } \widetilde{a}_2^i, \widetilde{a}_t^i \neq a_t^{i,*}, \text{ and } \widetilde{a}_t^i \in \mathcal{A}_i, t \in \{1, 2\}$$

separately in two steps. We denote the deviated joint action at time step t as $\tilde{\boldsymbol{a}}_t = (\widetilde{a}_t^i, \boldsymbol{a}_t^{-i})$.

Then, there will be a worse or equal reward for the second time step for agent i for every possible history h_2, i.e.,

$$Q_2^i\big(h_2, \tilde{\boldsymbol{a}}_2\big) \leq Q_2^i\big(h_2, \boldsymbol{a}_2^*\big).$$

Then, the continuation values for agent i become worse for every possible first-step action $\boldsymbol{a}_1$,

$$\begin{aligned} r_2^i(x_2, \tilde{\boldsymbol{a}}_2) &\leq r_2^i(x_2, \boldsymbol{a}_2^*) \\ \Rightarrow W_i^{\widetilde{\sigma}}(x_1, \boldsymbol{a}_1) &\leq W_i^{\sigma^*}(x_1, \boldsymbol{a}_1), \end{aligned} \tag{3}$$

where $\widetilde{\sigma}(\boldsymbol{a}_1, x_2) = \tilde{\boldsymbol{a}}_2$, and $\sigma^*(\boldsymbol{a}_1, x_2) = \boldsymbol{a}_2^*$.

The agent i also deviates from $a_1^{i,*}$ to $\widetilde{a}_1^i$, then we compare the utilities of it at $t = 1$ if it deviates from $\boldsymbol{a}_1^*$ found by Algorithm 1.

$$\begin{aligned} & Q_1^{i,\sigma^*}(x_1, \boldsymbol{a}_1^*) - Q_1^{i,\widetilde{\sigma}}(x_1, \tilde{\boldsymbol{a}}_1) \\ &= r_1^i(x_1, \boldsymbol{a}_1^*) + W_i^{\sigma^*}(x, \boldsymbol{a}_1^*) - r_1^i(x_1, \tilde{\boldsymbol{a}}_1) - W_i^{\widetilde{\sigma}}(x_1, \tilde{\boldsymbol{a}_1}) \\ &= r_1^i(x_1, \boldsymbol{a}_1^*) + W_i^{\sigma^*}(x, \boldsymbol{a}_1^*) - W_i^{\widetilde{\sigma}}(x_1, \boldsymbol{a}_1^*) + W_i^{\widetilde{\sigma}}(x_1, \boldsymbol{a}_1^*) \\ &- r_1^i(x_1, \tilde{\boldsymbol{a}}_1) - W_i^{\widetilde{\sigma}}(x_1, \tilde{\boldsymbol{a}_1}) \\ &= [\big(r_1^i(x_1, \boldsymbol{a}_1^*) + W_i^{\widetilde{\sigma}}(x_1, \boldsymbol{a}_1^*)\big) - \big(r_1^i(x_1, \widetilde{\boldsymbol{a}}_1) + W_i^{\widetilde{\sigma}}(x_1, \tilde{\boldsymbol{a}_1})\big)] \\ &+ \big(W_i^{\sigma^*}(x, \boldsymbol{a}_1^*) - W_i^{\widetilde{\sigma}}(x_1, \boldsymbol{a}_1^*)\big) \end{aligned}$$

Since $\boldsymbol{a}_1^*$ is the Nash equilibrium action of the first step found by the algorithm, we know that by adopting the same continuation selector $\widetilde{\sigma}$, the first big term is greater than 0, and we already proved that the second term is greater than 0 in (3). Therefore

$$Q_1^{i,\widetilde{\sigma}}(x_1, \tilde{\boldsymbol{a}}_1) \leq Q_1^{i,\sigma^*}(x_1, \boldsymbol{a}_1^*),$$

and we prove that no agent i wants to deviate from the Nash equilibrium policy π^* found by Algorithm 1. □

E IMPLEMENTATION DETAILS FOR THE GSM8K STUDIES

This appendix collects protocol details for Subsections 3.1–3.2 that are omitted from the main text for space. Nothing here changes the reported numbers; it only records how those numbers were produced (needed for reproducibility).

Why GSM8K. We evaluate on GSM8K (Cobbe et al., 2021) (`main` split), a public grade-school math word-problem benchmark with reference solutions. We use GSM8K because (i) free-form answers can be checked against a gold solution, (ii) the corpus is large enough to carve out a fixed ability-map bank, a training pool, and the official test holdout, and (iii) accuracy varies sharply across our three frozen solvers, so cost–accuracy trade-offs are meaningful.

Hyper-parameter	Value
Optimizer / learning rate	Adam / 10^{-3}
Discount γ	0.99
Cost weight λ / handoff fee α	100 / 0.05
Episodes $\times$ questions/episode	1000×32
Max attempts per question / fail penalty	$3 / -1$
Hidden size / batch / replay buffer	128 / 32 / 80,000
Target sync / ε schedule	every 200; $1 \to 0.05$ over 20k steps
Seeds	0–19

Table 3: Final DQN / environment hyper-parameters for the GSM8K management and delegation studies.

Solvers and synthetic prices. We use three frozen solvers: Nemotron 3 Ultra (A), Llama 3.2 3B Instruct (B), and Llama 3.2 1B Instruct (C). Only the coordinator π is trained (centralized DQN). Synthetic token prices (\$/1M input/output) are $\$1.00/\$3.00$ for Ultra, $\$0.25/\$1.50$ for Llama 3B, and $\$0.05/\$0.10$ for Llama 1B; these are experimental parameters, not live provider invoices.

Offline preprocessing. Before any RL training, each solver answers eligible GSM8K items once, and we record measured token usage. Separately, we ask Nemotron 3 Ultra to mark each stored answer as correct or incorrect relative to the GSM8K reference solution. This is an offline labeling step used to measure solver accuracy and to score cached training episodes. Listings 1–2 summarize the corresponding preprocessing and training entrypoints; the full scripts are provided in the code-and-data supplementary ZIP (not via a web URL). Using Ultra both as one of the three solvers and as this offline labeler is a practical choice rather than a modeling requirement. A separate model (for example GPT-5.5 or GLM-5.2) could have done the same labeling job; we used Ultra because it was available to us at negligible additional cost, and because checking free-form answers against the GSM8K reference is a narrow comparison task. On a hand-checked sample of 200 items, Ultra's labels agreed with our judgments in every case, so we used it for the full offline labeling pass. On a held-out development bank, the resulting accuracies are about 96.7% for Ultra, 84.5% for Llama 3B, and 53.7% for Llama 1B. Tokens spent on this offline labeling are excluded from the step cost used by π. From GSM8K `main`, 1000 official-train items form a fixed ability map (empirical success rates by difficulty and solver). The remaining 6473 training items are used to train π, and the official test set (1319) is held out. Cached answers and correctness labels are reused throughout training so that policies facing the same question see the same solver outputs. In particular, we do not call the solvers or a live labeler at every RL step: π reads the already-cached offline answers and labels, which makes the GSM8K coordinator experiments reproducible.

Training protocol, seeds, and baseline. An episode contains 32 questions. A question may be retried up to three times; after three failures the environment advances with penalty -1. We use $\gamma = 0.99$, cost weight $\lambda = 100$, and 1000 training episodes. Each GSM8K study reports mean$\pm$std over 20 independent runs with seeds $\{0, 1, \ldots, 19\}$; for seed s we set the environment, NumPy, and PyTorch RNGs from s before training. Figures in the main text use the ability-map observation; a paired no-map run gives nearly the same aggregates and is summarized in the tables. Both GSM8K studies use a handoff fee rate $\alpha = 0.05$ (defined per subsection). The *always-Ultra* baseline sends every question to Ultra with no handoff to B or C, under the same caches and the same cost accounting as the learned policy in that subsection.

Hyper-parameters and selection. Table 3 lists the final coordinator settings shared by both GSM8K studies (delegation uses 189 discrete joint actions instead of 3). During development we fixed the environment protocol early (episode length 32, three retries, $\lambda = 100$, $\alpha = 0.05$, $\gamma = 0.99$) after small sanity runs and retained a standard DQN schedule (Adam learning rate 10^{-3}, ε decay $1 \to 0.05$ over 20,000 gradient steps, replay buffer 80,000, target sync every 200 updates, hidden size 128). We did not run a large grid search; all reported seeds use this single final configuration.

Compute. Coordinator training ran on NVIDIA A100-SXM4-80GB GPUs under Linux, using Python 3.10, PyTorch 2.5.1 (CUDA 12.1 wheels), NumPy, Gymnasium, and HuggingFace `transformers` for local Llama solves and caches. Each seed used a single A100; offline Ultra labeling used API calls and is excluded from the step cost charged to π.

```
# Offline GSM8K preprocessing (conceptual; full scripts in code release)
items = load_gsm8k_main()                 # Cobbe et al., GSM8K main
label, train, test = split(items,
    label_n=1000, seed=0)                 # test = official 1319
for solver in [Ultra, Llama3B, Llama1B]:
    cache[solver] = answer_once(label + train + test)  # store tokens
for (q, ans) in all_cached_pairs(cache):
    judge[q, ans] = Ultra_marks_vs_gold(q, ans)        # offline only
ability_map = success_rates(label, judge)              # difficulty x solver
# RL samples from train caches; judge tokens excluded from step cost
```

Listing 1: Offline preprocessing used before both GSM8K RL studies.

```
# Coordinator training entrypoints (seed S in 0..19; PYTHONPATH=src)
python experiments/exp21_management/run_train.py \
  --config configs/exp21/cache_dqn_v1_run3.yaml --seed S
python experiments/exp22_delegation/run_train.py \
  --config configs/exp22/dqn_v2_edge_fee_with_map.yaml --seed S
# Warm/label helpers: run_warm_cache.py, run_label.py,
# run_build_ability_map_local.py (same experiments/ tree)
```

Listing 2: Training and preprocessing entrypoints for the GSM8K studies.

Management reward terms. In Subsection 3.1, the shared step reward is

$$r \;=\; R_{\text{corr}} \;-\; \lambda\, C_{\text{solve}}(a_{\text{exec}}) \;-\; \lambda\, C_{\text{edge}}(n_e, a_{\text{exec}}) \;+\; R_{\text{fail}}.$$

Here $R_{\text{corr}} \in \{0, 1\}$ is one if the chosen solver's cached answer is correct under the offline Ultra labels above, and zero otherwise; $C_{\text{solve}}(a_{\text{exec}})$ is that solver's synthetic token cost; n_e is the number of handoffs on the chosen route (0, 1, or 2); $C_{\text{edge}}(n_e, a_{\text{exec}}) = n_e\, \alpha\, C_{\text{solve}}(a_{\text{exec}})$ with $\alpha = 0.05$ is a small fee for each handoff; $\lambda = 100$ scales the dollar costs into the reward; and $R_{\text{fail}} = -1$ only after three failed attempts on the same question. In short, we pay for answering and also pay a little for each pass down the chain.

Delegation cost accounting. In Subsection 3.2, we count the attempt as successful if any answering agent is correct under the offline Ultra labels, and we charge the sum of those agents' solve costs. In addition, each kickoff handoff and each peer pass pays a small edge fee of $\alpha = 0.05$ times the recipient's expected solve cost, matching the handoff fee used in the management study. Always-Ultra is evaluated in the same graph environment, so it also pays the kickoff edge fee when Ultra is started.

Code and entrypoints. Listings 1–2 summarize the offline pipeline and DQN entrypoints; the full scripts are in the code-and-data supplementary ZIP.

F 3-Agent Collaboration through Money Exchange

F.1 Delegation Through Transferable Money

We construct a three-agent environment to study whether money transfers can help independently trained agents coordinate their actions. The environment contains three agents, denoted Agent 1, Agent 2, and Agent 3, and three jobs, denoted Job 1, Job 2, and Job 3.

Each agent is associated with one job having the same index. Agent 1 benefits from the completion of Job 1, Agent 2 from Job 2, and Agent 3 from Job 3. However, any agent is allowed to perform any of the three jobs. The agents also differ in how costly it is for them to perform each job. Thus, the job whose completion benefits an agent is not necessarily the job that the agent can perform most efficiently.

At the beginning of every time step, one instance of each job becomes available. Each agent must choose exactly one of the three jobs to perform, and all jobs expire at the end of that time step. Since each of the three agents can choose any of the three jobs, there are $3^3 = 27$ possible joint actions. A job is considered completed if at least one agent chooses it.

The cost of performing each job is defined as follows:

Worker	Job 1	Job 2	Job 3
Agent 1	3	8	1
Agent 2	1	3	8
Agent 3	8	1	3

This cost structure gives each agent one job with cost 1, one with cost 3, and one with cost 8. The cost means reduction in an agent's received environmental reward if that agent performs the job. For example, Agent 1 is associated with Job 1, but its cheapest job to perform is Job 3. Similarly, Agent 2 performs Job 1 most cheaply (smallest environmental reward reduction), and Agent 3 performs Job 2 most cheaply.

The environmental reward depends on both job completion and the cost of the action taken. Agent i receives an environmental reward of 6 if Job i is completed during that time step, regardless of which agent completed it. The agent then incurs the cost of the job that it personally chose. The actual environmental reward received by agent i is given by

$$r_i^{\text{env}} = 6 \cdot \mathbf{1}[\text{Job } i \text{ is completed}] - C_{i,a_i},$$

where a_i denotes the job selected by Agent i, and C_{i,a_i} is the corresponding cost.

For example, consider the joint action in which every agent performs the job with the same index:

$$(\text{Agent } 1 \to \text{Job } 1, \text{Agent } 2 \to \text{Job } 2, \text{Agent } 3 \to \text{Job } 3).$$

We refer to this as the *matching assignment*. All three jobs are completed, and each agent incurs a cost of 3. Therefore, each agent receives

$$6 - 3 = 3$$

per step, giving a total environmental reward of 9 across the three agents.

The environment also contains a more efficient joint action:

$$(\text{Agent } 1 \to \text{Job } 3, \text{Agent } 2 \to \text{Job } 1, \text{Agent } 3 \to \text{Job } 2).$$

We refer to this as the *minimum-cost assignment*. Here, every job is again completed, but each agent performs the job for which its cost is 1. Each agent therefore receives

$$6 - 1 = 5$$

per step, giving a total environmental reward of 15.

These are not the only possible outcomes. The 27 joint actions also include cases in which multiple agents choose the same job, one or more jobs remain incomplete, or agents perform jobs with costs of 3 or 8. The matching and minimum-cost assignments are highlighted because they provide two useful reference points: the matching assignment gives each agent a reward of 3, whereas the minimum-cost assignment gives every agent the higher reward of 5.

In the experiments below, agents trained without money consistently learn the matching assignment rather than the minimum-cost assignment. We therefore introduce transferable money as an additional coordination mechanism and study whether prescribed payments and observable balances can make the higher-reward assignment learnable.

Importantly, money is separate from the environmental reward. Agent i is trained only to maximize r_i^{env}. Receiving money does not directly increase an agent's reward, and making a payment does not directly reduce it. Instead, money can influence future behavior through the information and constraints created by the agents' balances.

Each agent starts an episode with a money balance of 6. When money is transferred, the sender's balance decreases and the receiver's balance increases. The updated balances are carried into the next time step and are included in the agents' observations. In addition, a payment that would cause an agent's balance to become negative is not allowed. Thus, earlier transfers both affect the balances observed in later steps and determine which later payments are feasible. Money is useful only if this transfer history helps the learned policy obtain higher environmental reward in subsequent interactions.

The possible money transfers are specified by a fixed *contract*. Conceptually, a contract is a rule that specifies how much money should be transferred between agents for each possible joint action. The contract does not determine which jobs the agents choose. The agents choose their jobs using their learned policies; the contract specifies only the payments associated with those choices.

Because there are 27 possible joint actions, a contract is represented as a 27×3 integer matrix. Each row corresponds to one joint action. The three columns represent transfers along the cyclic agent pairs

$$\text{Agent } 1 \to \text{Agent } 2, \qquad \text{Agent } 2 \to \text{Agent } 3, \qquad \text{Agent } 3 \to \text{Agent } 1.$$

A positive value denotes a payment in the stated direction, whereas a negative value reverses the direction. For example, a value of 3 in the first column represents a payment of 3 from Agent 1 to Agent 2, whereas a value of -3 represents a payment of 3 from Agent 2 to Agent 1.

At each time step, the stored contract is first checked against the agents' current balances. Since the complete contract is part of the observation, this feasibility check is applied to all 27 rows before the agents select their actions. For each row, the three specified transfers are considered simultaneously. If all resulting balances remain nonnegative, that row is feasible and remains unchanged.

If the transfers in a row would cause one or more agents to have a negative balance, the agent with the most negative resulting balance is removed from the payment calculation. Ties are broken in the order Agent 1, Agent 2, and Agent 3. All payments involving that agent are set to zero, and the remaining payments are checked again for feasibility. This process is repeated until the remaining payments are feasible. If fewer than two agents remain, all payments in that row are set to zero for the current time step.

This procedure does not modify the stored contract itself. Instead, it creates an *edited contract* for the current time step based on the agents' current balances. The agents observe this edited contract before choosing their jobs. Once the agents select a joint action, the payments in the corresponding row of the edited contract are executed. The balances are then updated, and the new balances become part of the observation at the next time step. At that next step, the feasibility procedure is again applied to the original stored contract using the updated balances.

When the stored contract contains only zeros, no money is transferred and all balances remain fixed at 6. In this no-money baseline, there is therefore no transfer history to observe, and the agents observe only normalized time t/T. In the money experiments, each agent observes the three current balances, normalized time t/T, and the edited 27×3 contract.

F.2 Learning Setup and Experimental Protocol

F.2.1 Reinforcement Learning Setup

We next study whether the money-transfer mechanism described above changes what independently learning agents are able to coordinate on. The three agents interact in the same environment, but each learns its own policy and maximizes only its own environmental reward. There is no shared training reward.

Agent i learns a policy

$$\pi_i(a_i \mid o_i),$$

where $a_i \in \{1, 2, 3\}$ specifies which job the agent chooses. Each agent has its own actor network and critic network. The critic for Agent i estimates that agent's expected discounted environmental return; it does not predict the sum of the three agents' rewards.

We use Multi-Agent Proximal Policy Optimization (MAPPO), a PPO-based multi-agent reinforcement learning algorithm (Yu et al., 2022). Each agent has its own actor and critic networks. The actor maps the agent's observation to a categorical distribution over the three job actions, and the critic estimates the corresponding discounted environmental return.

Both actor and critic are feed-forward networks with LayerNorm on the input, followed by three ReLU hidden layers of width 128. The actor outputs three logits, one for each job, and the critic outputs a scalar value estimate.

We compare two experimental conditions. In the *no-money condition*, the agents interact with the job environment without monetary transfers. No balances or payment rules are used, and each agent observes only the normalized time t/T.

In the *money condition*, the policies are trained under one fixed prescribed payment rule (contract). As described in the previous subsection, each agent observes the three current balances, normalized time t/T, and the feasible payment table obtained from the prescribed contract at the current balances.

The environmental reward is the only reward used for learning in both conditions. Money is never added to or subtracted from the PPO reward. Its effect is indirect: payments change the balances, which changes the next observation and may also change which payments are feasible later. Thus, the comparison asks whether the payment mechanism changes the joint behavior learned by independently optimizing agents.

At each time step in the money condition, the current balances are first used to determine the feasible version of the prescribed payment table. The agents observe this information and select their jobs. Their environmental rewards are determined from the resulting joint action. The payment associated with that joint action is then applied, the balances are updated, and the updated balances become part of the next observation. During training, actions are sampled from each policy's categorical distribution; during evaluation, each agent selects the job with the largest policy logit.

F.2.2 Reward Cases

We consider two environmental reward definitions. The first is the base reward introduced in the previous subsection. The second changes the agents' individual incentives while preserving the same maximum total environmental reward.

Case 1: Base Reward. Under the base reward, Agent i receives a completion reward of 6 if Job i is completed and subtracts the cost of the job that it personally selects:

$$r_i^{\text{env}} = 6 \cdot \mathbf{1}[\text{Job } i \text{ is completed}] - C_{i,a_i}.$$

As described above, the matching assignment gives each agent a reward of 3 per step, while the minimum-cost assignment gives each agent a reward of 5 per step. Over a 1000-step episode, these correspond to

$$W = 9000$$

and

$$W = 15000,$$

respectively.

Case 2: Count-Weighted Reward. To test whether payment rules that work under the base reward remain effective when the agents' individual incentives change, we also consider a count-weighted reward:

$$r_i^{\text{env}} = 6k_i - C_{i,a_i},$$

where k_i is the number of agents that choose Job i.

Under this reward, the completion reward received by Agent i increases with the number of agents choosing Job i. For example, if two agents choose Job 1, Agent 1 receives a completion reward of 12; if all three agents choose Job 1, Agent 1 receives 18. The agents performing Job 1 do not receive this additional reward unless Job 1 is their associated job. Each agent still subtracts only the cost of the job that it personally selects.

This modification changes the agents' individual incentives. Under the base reward, assigning more than one agent to an already completed job does not increase the completion reward. Under the count-weighted reward, additional agents choosing Job i increase Agent i's reward. We use this modified reward as a stress test of whether previously effective payment rules continue to support the minimum-cost assignment after the individual reward structure changes.

Importantly, the maximum possible total environmental reward remains unchanged. Since each of the three agents chooses exactly one job,

$$k_1 + k_2 + k_3 = 3.$$

Therefore,

$$\sum_{i=1}^{3} r_i^{\text{env}} = 6(k_1 + k_2 + k_3) - \sum_{i=1}^{3} C_{i,a_i} = 18 - \sum_{i=1}^{3} C_{i,a_i}.$$

The minimum total action cost remains 3, so the maximum total reward is still 15 per step, or

$$W_{\max} = 15000$$

over one episode. Thus, the modified reward changes individual incentives without changing the maximum-reward reference outcome.

F.2.3 Training and Evaluation

Each episode lasts

$$T = 1000$$

time steps. At the beginning of every episode in the money condition, the balances are reset to

$$(6, 6, 6).$$

Training uses eight parallel copies of the environment. For a given payment rule, all eight copies use the same fixed rule, while the learned policy parameters are shared across the copies.

For the base-reward experiment, each parallel environment experiences approximately 25 episodes during training. For the count-weighted reward experiment, each parallel environment experiences approximately 125 episodes. Because the second experiment uses a longer training period, its results should not be interpreted as differing from the first experiment only because of the reward definition.

The main PPO parameters are

$$\gamma = 0.99, \qquad \lambda = 0.95, \qquad \epsilon_{\text{clip}} = 0.2,$$

with Adam learning rate 3×10^{-4}.

After training, the policies are frozen and evaluated without further learning. Actions are selected greedily, and each reported seed score is the mean of five evaluation episodes.

For Agent i, define its environmental return over one episode as

$$R_i = \sum_{t=1}^{T} r_{i,t}^{\text{env}}.$$

We report the total episodic environmental reward

$$W = \sum_{i=1}^{3} R_i = \sum_{t=1}^{T} \sum_{i=1}^{3} r_{i,t}^{\text{env}}.$$

The quantity W measures the quality of the joint outcome, but it is not the objective optimized by any individual agent. Each agent learns only from its own environmental reward.

F.2.4 Prescribed Payment Rules

We evaluate 1000 fixed payment rules. For each payment rule, policy training is repeated from scratch with five independent random seeds. Thus, the base-reward money experiment contains

$$1000 \times 5 = 5000$$

independent policy-training runs.

Every payment entry is an integer in

$$\{-6, -5, \ldots, 5, 6\}.$$

The 1000 payment rules were constructed to cover different ways in which transfers could depend on the agents' joint action. Twelve simple rules were specified manually as reference cases. These

include the zero-payment rule, rules that use the same transfer pattern for every joint action, rules that activate payments only for selected joint actions, and rules that permit transfers only along selected directions.

The remaining rules were generated automatically. Some use the same payment pattern across many or all joint actions, some assign payments only to selected joint actions, and others allow payment entries to vary across the joint-action table. The purpose is to evaluate payment structures ranging from simple and highly regular to more action-specific rules.

The collection was built progressively rather than sampled completely before any policies were trained. After the first 50 payment rules, later groups of 50 included modified versions of payment structures that had already produced the maximum-reward joint action. At least 10% of the entries in these later groups were generated in this way; from payment-rule index 512 onward, the proportion was at least 30%.

The modifications included changing payment magnitudes by small integer factors, removing selected payment rows, or perturbing a small number of entries by one or two units. This progressive construction changes only which payment rules are included in the collection. Once policy training begins for a particular rule, that rule remains fixed throughout the training run.

F.2.5 Numerical Results: Base Reward

We first evaluate the no-money condition under the base reward. The policies observe only t/T. Across 30 independent seeds, all learned policies remain at the matching assignment, producing total episodic environmental reward in the interval

$$[9000, 10000).$$

We then train policies under the 1000 prescribed payment rules, using five independent seeds for each rule. Figure 5 shows the resulting 5000 seed-level outcomes. Most trained policies remain near the no-money result: 4298 seeds fall in $[9000, 10000)$, and 408 fall below 9000. However, some policies achieve substantially better outcomes. A total of 183 seeds reach the theoretical maximum

$$W = 15000,$$

while another 111 lie between 12000 and 15000.

Key result. Compared with the no-money outcome of $W = 9000$, successful money-enabled runs reach $W = 15000$, corresponding to a 66.7% increase in total episodic environmental reward. This improvement is shared equally by all three agents: each agent's episodic environmental return increases from 3000 in the matching assignment to 5000 in the minimum-cost assignment, or equivalently from 3 to 5 per time step.

The key finding is that some prescribed payment rules successfully enable the agents to reach the minimum-cost assignment, whereas the no-money condition remains at the matching assignment. In particular, 183 money-enabled seeds reach the maximum total episodic environmental reward of $W = 15000$. Thus, the results demonstrate that appropriately designed contracts can substantially improve coordination and lead to the maximum-reward outcome.

These results establish that appropriately structured transferable-money payment rules can change the learned outcome toward a higher-reward collaborative solution. In the present work, the payment rules are prescribed rather than optimized automatically. Automatically discovering high-performing contracts is left for future work.

Table 4 summarizes the same experiment at the payment-rule level. The figure treats each of the 5000 independently trained seeds as one observation. The table first averages the five seed scores associated with each payment rule and then bins the resulting 1000 payment-rule means.

F.2.6 Numerical Results: Count-Weighted Reward

We next test whether payment rules that performed well under the base reward remain effective under the count-weighted reward.

Total episodic environmental reward	Number of payment rules
below 9000	135
[9000, 10000)	766
[10000, 11000)	32
[11000, 12000)	19
[12000, 13000)	19
[13000, 14000)	19
[14000, 15000)	10
15000	0

Table 4: Distribution of the 1000 prescribed payment rules under the base reward. Each payment rule is represented by the mean total episodic environmental reward of five independently trained seeds.**The key takeaway is that there are good payment rules which enable the agents to collaborate in (close to) the best possible way.**

To do this, we select 20 of the original 1000 payment rules. Ten come from the high-performing end of the base-reward experiment, preferring rules whose five-seed mean exceeded 14000. Ten come from the low-performing end, with five-seed means at or below 9000.

For each selected payment rule, five new policy sets are initialized and trained from scratch under the count-weighted reward. This produces

$$20 \times 5 = 100$$

new money-enabled training runs.

For comparison, five no-money policies are also trained under the count-weighted reward while observing only t/T. All five remain in the

$$[9000, 10000)$$

interval.

Among the 100 money-enabled seeds, 45 reach

$$W = 15000,$$

38 fall in $[9000, 10000)$, and the remaining 17 lie between 11000 and 15000.

The high- and low-performing payment rules remain qualitatively separated. Nine of the ten rules selected from the high-performing group reach 15000 on all five newly trained seeds, while the tenth remains close to that value. In contrast, the rules selected from the low-performing group remain concentrated near the 9000 outcome.

Key result. Under the count-weighted reward, payment rules that previously performed well again support the maximum-reward outcome of $W = 15000$, compared with the no-money outcome near $W = 9000$. The improvement is therefore not restricted to the original reward definition: the same high-performing payment structures continue to support the maximum-reward assignment after the agents' individual incentives are changed.

This comparison should not be interpreted as showing that the count-weighted reward itself improves learning, because the reward definition and training duration both differ from the first experiment. The purpose of this experiment is narrower: to test whether the distinction between previously high-performing and low-performing payment rules persists when the agents are retrained from scratch under a modified individual reward structure.

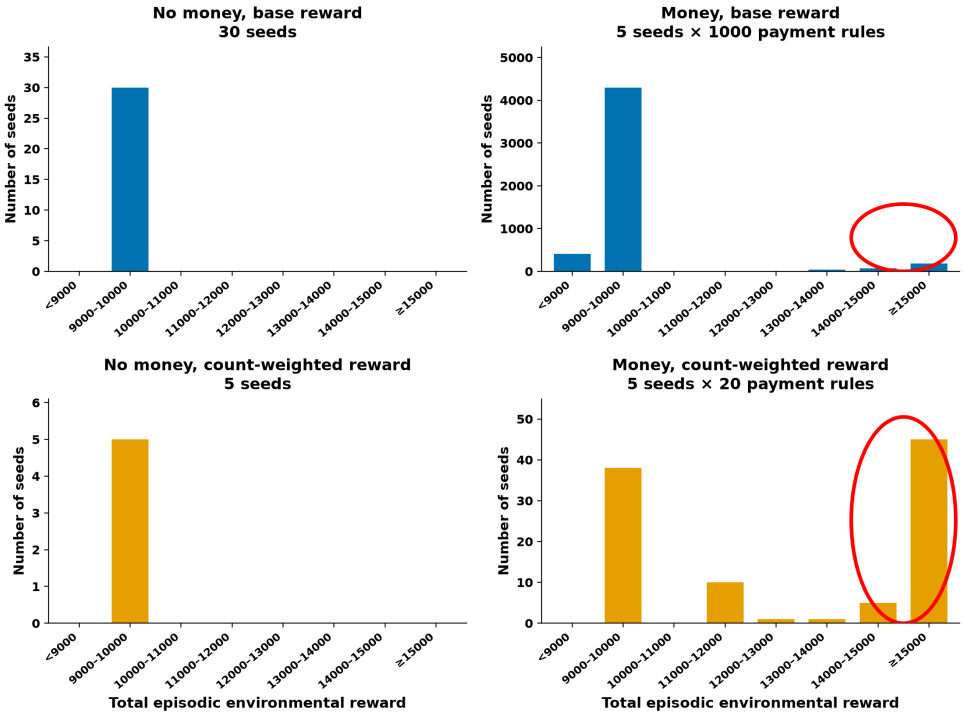


Figure 5: Distribution of total episodic environmental reward across independently trained policy seeds. Each plotted seed score is the mean of five greedy evaluation episodes of 1000 steps after training. Each panel uses its own vertical scale. Top left: no-money condition under the base reward, 30 seeds. Top right: money condition under the base reward, with five independent policy-training seeds for each of 1000 fixed payment rules (5000 seeds). Bottom left: no-money condition under the count-weighted reward, 5 seeds. Bottom right: money condition under the count-weighted reward, with five newly trained seeds for each of 20 selected payment rules (100 seeds). The red circles mark the bars from 14000 through 15000. These circled seeds show that there exist payment rules under which the agents reach the minimum-cost assignment and the maximum possible total episodic environmental reward. The last bin is exactly 15000; no seed exceeds this value. Table 4 instead reports payment-rule-level results by averaging the five base-reward seeds associated with each payment rule.