REINFORCEMENT NETWORKS: NOVEL FRAMEWORK FOR COLLABORATIVE MULTI-AGENT REINFORCEMENT LEARNING TASKS.

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern AI systems often comprise multiple learnable components that can be naturally organized as graphs. A central challenge is the end-to-end training of such systems without restrictive architectural or training assumptions. Such tasks fit the theory and approaches of the collaborative Multi-Agent Reinforcement Learning (MARL) field. We introduce Reinforcement Networks, a general framework for MARL that organizes agents as vertices in a directed acyclic graph (DAG). This structure extends hierarchical RL to arbitrary DAGs, enabling flexible credit assignment and scalable coordination while avoiding strict topologies, fully centralized training, and other limitations of current approaches. We formalize training and inference methods for the Reinforcement Networks framework and connect it to the LevelEnv concept to support reproducible construction, training, and evaluation. We demonstrate the effectiveness of our approach on several collaborative MARL setups by developing several Reinforcement Networks models that achieve improved performance over standard MARL baselines. Beyond empirical gains, Reinforcement Networks unify hierarchical, modular, and graph-structured views of MARL, opening a principled path toward designing and training complex multi-agent systems. We conclude with theoretical and practical directions — richer graph morphologies, compositional curricula, and graph-aware exploration. That positions Reinforcement Networks as a foundation for a new line of research in scalable, structured MARL.

1 INTRODUCTION

Modern AI systems increasingly comprise multiple learnable components that must coordinate to solve complex tasks. Prominent examples include large-language-model (LLM) workflows for tool use and orchestration, Retrieval-Augmented Generation (RAG) pipelines, and multi-agent LLM systems. These systems are naturally expressed as *graphs* of interacting modules: nodes represent learnable components (e.g., retrievers, planners, controllers), and edges encode information flow and control dependencies. This view is well-grounded in the literature on graph-structured learning and computation, where graphs serve as a unifying abstraction for modular architectures and message passing Zhou et al. (2020a); Khemani et al. (2024). In practice, real-world workflows are frequently modeled as *directed acyclic graphs* (DAGs), which make dependencies explicit, enable topological scheduling, and support scalable orchestration in scientific and data-intensive computing Verucchi et al. (2023). Recent LLM systems adopt precisely these ideas: LLM-driven workflow generators and orchestrators (e.g., WorkflowLLM; automated DAG construction for enterprise workflows) formalize complex LLM pipelines as DAGs for reliable execution and optimization Fan et al. (2025); Xu et al. (2024), and DAG-structured plans have been shown to improve task decomposition and concurrency in embodied LLM-agent settings Gao et al. (2024). RAG, now a standard pattern for knowledge-intensive tasks, is also naturally modular—combining retrievers, rerankers, and generators in a graph that can be adapted and optimized end-to-end Gao et al. (2023); Gupta et al. (2024); Zhao et al. (2024).

From learnable systems to collaborative MARL. Although these graph-structured AI systems are typically engineered as pipelines, their components often have objectives and behaviors that

interact nontrivially. This invites a learning paradigm in which modules are treated as *agents* optimizing (possibly aligned) goals under partial information and non-stationarity—the core setting of cooperative multi-agent reinforcement learning (MARL). Foundational work in cooperative MARL formalizes centralized-training-decentralized-execution (CTDE), credit assignment, and coordination as key challenges Amato (2024); Huh & Mohapatra (2024). Methods for credit assignment (e.g., LICA) demonstrate that joint optimization can be effective without fully centralized control at execution time Zhou et al. (2020b). Concurrently, hierarchical reinforcement learning (HRL) provides temporal abstraction via options and multi-level decision-making Sutton et al. (1999), offering a bridge from pipeline hierarchies to principled multi-agent learning. Together, these strands suggest a unified perspective: graph-structured, learnable systems can be cast as collaborative MARL problems in which nodes (modules/agents) coordinate over a computation graph.

Leveled architectures and the LevelEnv abstraction. A recent development that operationalizes this bridge is the *LevelEnv* concept introduced by the TAG framework for decentralized hierarchical MARL Paolo et al. (2025). LevelEnv abstracts each hierarchy level as the "environment" for the level above, standardizing information exchange while preserving loose coupling between levels. This yields hierarchies of arbitrary depth with heterogeneous agents and improves sample efficiency and final performance on standard MARL benchmarks—all without requiring fully centralized training or rigid two-level manager–worker structures Paolo et al. (2025). LevelEnv thus provides a reproducible and modular interface to construct, train, and evaluate leveled multi-agent systems that align closely with how modern LLM workflows and RAG pipelines are actually built.

Why DAGs for multi-agent systems? Representing multi-agent, multi-module systems as DAGs confers several advantages. First, acyclicity enables *topological ordering* and *parallelization* of independent subgraphs, improving scalability Verucchi et al. (2023). Second, DAGs make *credit assignment paths* explicit: upstream decisions can be attributed to downstream outcomes along directed edges, facilitating learning signals that respect causal and temporal dependencies (an idea mirrored in hierarchical task/dependency models and HTN planning) Georgievski & Aiello (2015); Chen et al. (2021). Third, DAG formalisms avoid deadlocks and circular dependencies during training and execution, simplifying stability analysis and orchestration Verucchi et al. (2023). Finally, DAGs are a flexible superset of hierarchical trees: they permit shared substructures and multi-parent dependencies that commonly arise in tool-using LLM agents and modular RAG systems Fan et al. (2025); Xu et al. (2024); Gao et al. (2023).

Our research. We build on these observations to introduce *Reinforcement Networks*, a general framework that organizes collaborating agents as nodes in a DAG. Our formulation unifies hierarchical, modular, and graph-structured views of MARL: it supports flexible credit assignment and scalable coordination without strict topologies or fully centralized training, and it connects directly to the LevelEnv interface Paolo et al. (2025) for reproducible construction, training, and evaluation. We demonstrate empirical gains over classical MARL baselines in collaborative settings and outline theoretical and practical directions—richer graph morphologies, compositional curricula, and graph-aware exploration—toward scalable, structured MARL.

2 Related Works

Multi-Agent Reinforcement Learning Research interest in multi-agent systems has seen substantial growth (Nguyen et al., 2020; Oroojlooy & Hajinezhad, 2023). Current understanding posits that the primary driver of innovation in this domain is the mechanisms known as autocurricula. These processes, engendered by the dynamics of cooperation and competition among adaptive agents, establish a natural framework for continuous learning. To facilitate the experimental advancement of this paradigm, an ecosystem of supporting tools has emerged, including PettingZoo (Terry et al., 2021), which provides a unified interface for environments, and BenchMARL (Bettini et al., 2024), which addresses the challenges of methodological fragmentation and reproducibility in research.

Multi-agent reinforcement learning (MARL) methodologies can be grouped into several categories based on their coordination strategies. The first category comprises independent learning paradigms, characterized by decentralized execution without explicit communication protocols, wherein each

agent perceives the environment as a partially observable system incorporating other agents as environmental components. Notable instantiations of this paradigm include Independent Proximal Policy Optimization (IPPO) (De Witt et al., 2020), Independent Q-Learning (IQL) (Tan, 1997), and Independent Soft Actor-Critic (ISAC) (Bettini et al., 2024) algorithms, constituting direct multi-agent extensions of their single-agent counterparts: Proximal Policy Optimization (PPO) (Schulman et al., 2017), Q-Learning (Watkins & Dayan, 1992), and Soft Actor-Critic (SAC) (Haarnoja et al., 2018) respectively.

A secondary category involves parameter-sharing architectures, distinguished by distributed agents in utilizing common parametric representations through shared critics or value functions. This methodological framework is implemented in Multi-Agent Proximal Policy Optimization (MAPPO) (Yu et al., 2022), Multi-Agent Soft Actor-Critic (MASAC) (Bettini et al., 2024), and Multi-Agent Deep Deterministic Policy Gradient (MADDPG) (Lowe et al., 2017) algorithms.

Another category encompasses explicit communication frameworks, which facilitate inter-agent information exchange through either consensus-based mechanisms (Cassano et al., 2020) in distributed network topologies or learnable communication protocols (Foerster et al., 2016; Jorge et al., 2016). These approaches address coordination challenges through structured information sharing paradigms.

The fundamental challenge in MARL systems concerns environmental non-stationarity arising from concurrent policy adaptation across all learning agentsx. Consequently, the past experiences stored in the replay become obsolete quickly (Foerster et al., 2016), necessitating specialized algorithmic solutions. The predominant paradigm of Centralized Learning with Decentralized Execution partially addresses this issue by using shared learning components during training (Oroojlooy & Hajinezhad, 2023). However, this architecture imposes constraints that limit its applicability in lifelong learning scenarios that require continuous adaptation.

Hierarchical Reinforcement Learning The hierarchy principle underlies the intelligent behavior observed in nature. The key advantages of the hierarchical approach lie in improved credit assignment through abstraction-based value propagation, as well as enabling more semantically meaningful environment exploration through the application of temporal and spatial abstraction (Hutsebaut-Buysse et al., 2022). The enhanced environment exploration capabilities provided by hierarchy represent one of the main advantages of hierarchical reinforcement learning (HRL) compared to ”flat” RL methods (Nachum et al., 2019). In addition, HRL reduces overall computational complexity through task decomposition while maximizing sub-problems reuse, which collectively accelerates the learning procedure.

Various methodological approaches have been proposed to address the challenges of HRL. The Options framework utilizes Semi-Markov Decision Processes (SMDPs) to define temporally-extended actions (“options”), which bundle policies, termination conditions and initiation sets (Sutton et al., 1999). Later developments introduced end-to-end training, as seen in Option-Critic (Bacon et al., 2017). Alternatively, Feudal RL employs a manager-worker architecture where managers issue intrinsic goals to low-level workers (Dayan & Hinton, 1992; Vezhnevets et al., 2017). A fundamental challenge common to these approaches is the non-stationarity of low-level policies during training, which complicates value function estimation at the higher level. Model-based methods, such as CSRL (Li et al., 2017), have been developed to address this specific issue.

HRL with MARL introduces additional complexity. This stems from the necessity to manage multi-agent dependencies, which in turn introduces additional sources of non-stationarity into the learning process. The TAG framework (Paolo et al., 2025) proposes specific methodologies to address these challenges. However, the authors maintain a strong connection to the TAME approach (Levin, 2021) restricting the architecture to a layered digraph, thus limiting model application. Our work extends the authors' approach by addressing its limitations.

3 METHODOLOGY

In this section we describe the mathematical fundamentals of Reinforcement Networks and the processes of topology transformation, training, and inference.

3.1 FUNDAMENTALS

Multi-Agent Reinforcement Learning Math concepts operated in Multi-Agent Reinforcement Learning setup:

- S^{env} - state space describing the environment
- $\{w_i\}_{i=1}^{k}$ - agents operating in the environment
- A - joint action space, A_i - action space of i-th agent
- $p : S \times A \to \Delta(S)$ - transition function
- $p_0 \in \Delta(S)$ - distribution over initial state
- $\{R_i\}_{i=1}^{k}$, where $r_i^{env} : S \times A_i \to \mathbb{R}$ - reward of the i-th agent

3.2 PROPOSED MODEL

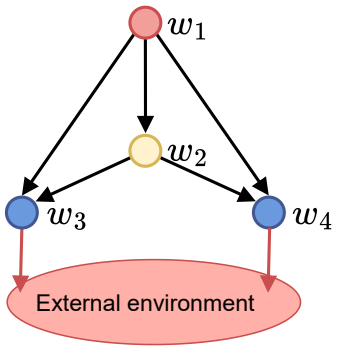

Figure 1: An example of DAG configuration used for agent system hierarchy. Considering agent ω_2 colored yellow, its subordinate agents V_2^- are marked blue and the superior agents are marked red. Edges of the DAG represent agents' couples where direct communication is present. Red arrows depict interaction between *motors* and the real environment.

Reinforcement Networks Consider a directed acyclic graph $G = (V, E)$, where $V = \{w_1, \ldots, w_N\}$ stands for the set of vertices representing the agents, and E is the set of directed edges. We introduce the following notation to formally characterize the local connectivity of each vertex $w_i \in V$: $I_i^+ = [j \mid (w_j, w_i) \in E]$, $\quad I_i^- = [j \mid (w_i, w_j) \in E]$. Here, I_i^+ denotes the indices list of vertices that have an outgoing edge incident to w_i, whereas I_i^- denotes the indices list of vertices that have an incoming edge incident to ω_i. The corresponding cardinalities are designated as $l_i^+ = |I_i^+|$, $\quad l_i^- = |I_i^-|$. Furthermore, we define the sets of agents associated with these indices as $V_i^+ = \{\omega_j \mid j \in I_i^+\}$, $\quad V_i^- = \{\omega_j \mid j \in I_i^-\}$.

The sink nodes of the graph, for which $V_i^- = \varnothing$, interact directly with the external environment and consequently receive rewards and observations from it. We denote the set of these agents by $V_0 \subseteq V$ and refer to them as *motors*.

In this formulation, we refer to a vertex w_j as a *superior agent* of w_i if there exists an edge $(w_j, w_i) \in E$, and as a *subordinate agent* if there exists an edge $(w_i, w_j) \in E$. This terminology enables a natural hierarchical interpretation of the directed acyclic structure.

For the traversal along an edge, we use the verb *pass*, whereas for the reverse direction we use the verb *return*. Similar to TAG (Paolo et al., 2025), for a given agent w_i the set of subordinate agents V_i^- plays the role of the environment, returning both the reward and the observation in response to the agent's actions.

Each agent w_i is represented by a tuple $\langle M_i, O_i^-, A_i, O_i^+, \pi_i, \phi_i, \psi_i, R_i \rangle$, which is defined as follows:

- M_i - the message space, i.e., the set of messages that the agent may return to its superior agents.
- O_i^- - the observation space, defined as $O_i^- = \prod_{j \in I_i^-} M_j$, corresponding to the list of messages passed by subordinate agents.
- A_i - the action space used to influence subordinate agents.
- O_i^+ - the space of instructions, i.e., the actions passed by superior agents, defined as $O_i^+ = \prod_{j \in I_i^+} A_j$.
- $\pi_i : O_i^- \times O_i^+ \to \Delta(A_i)$ - the agent's policy. The conditional distribution $\pi_i(a_i \mid o_i^-, a_i^+)$ depends both on the observations obtained from subordinate agents and the instructions provided by superior agents. Introducing the notation $O_i \coloneqq O_i^- \times O_i^+$, the policy can equivalently be written as $\pi_i(a_i \mid o_i), \quad o_i \in O_i$.
- $\phi_i : O_i^- \times \mathbb{R}^{l_i^-} \to M_i$ - the communication function. It determines the message describing current environment state to be transmitted upward. Here, $\mathbb{R}^{l_i^-}$ represents the vector of rewards returned by subordinate agents. Thus, $m_i = \phi_i(o_i^-, r_i^-)$. For source vertices with $V_i^+ = \varnothing$, the choice of ϕ_i has no impact on the overall system dynamics.
- $\psi_i : O_i^- \times \mathbb{R}^{l_i^-} \to \mathbb{R}$ - the proxy-reward function. The value of this function is returned to superior agents as an element of the reward list. For source vertices with $V_i^+ = \varnothing$, the choice of ψ_i has no impact on the overall system dynamics.
- $R_i : \mathbb{R}^{l_i^-} \to \mathbb{R}$ is the aggregation function, which interprets the list of received rewards.

We use the following notation throughout:

$$a_i^+ = [a_j \mid j \in I_i^+], \quad o_i^- = [m_j \mid j \in I_i^-], \quad r_i^- = [r_j \mid j \in I_i^-],$$

where the elements are assumed to be ordered according to related lists I_i^+, I_i^-. Finally, $\Delta(A_i)$ denotes the probability simplex over the action space A_i.

3.3 Upstream and downstream inference

We describe the exchange of information both among agents and between the environment and the agent system. Figure 2 illustrates the information flow across the entire system, while Figure 3 provides a visualization from the perspective of an individual agent. This mechanism generalizes the standard agent-environment interaction in reinforcement learning to hierarchical multi-agent systems. A single timestep of an agent corresponds to one cycle of interaction, beginning with processing observations, followed by sampling an action, and concluding with receiving a new observation as the environment's response to the chosen action. At each timestep, information flow is decomposed into two phases: upstream inference and downstream inference. Information from the environment—namely observations and rewards—is transmitted to an agent against the direction of outgoing edges, whereas the agent's action is propagated downwards along the edges of the graph. It is important to note that subordinate agents function as an environment for a given agent. For *motors*, as long as no subordinate agents are present, the external environment assumes this role.

Sharing experience. During upstream inference, agents' observations and rewards are propagated upward through the graph. The functions ϕ and ψ generate the messages and rewards that are passed to superior agents. This design enables efficient knowledge sharing. At the same time, it enables partial hiding of the environment state depending on the recipient's abstraction level or task perspective.

In the beginning of a timestep, agent w_i possesses the information received at the previous step as a response from the environment:

$$o_i^-(t) = (m_j(t))_{j \in I_i^-},$$
$$r_i^-(t-1) = (r_j(t-1))_{j \in I_i^-}.$$

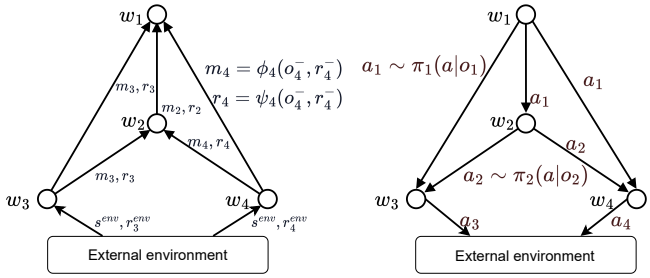

Figure 2: Example of inference in the system. Left: downstream inference. Right: upstream inference.

Given that agent w_i generates a message and a reward to be passed to its superior agents:

$$m_i(t) = \phi_i\big(o_i^-(t), r_i^-(t-1)\big), \qquad r_i(t-1) = \psi_i\big(o_i^-(t), r_i^-(t-1)\big).$$

These values then become elements of the input received by the agents of V_i^+. This process propagates experience upward in the hierarchy, enabling higher-level agents to incorporate information from subordinate agents.

Interaction with the environment. For a given vertex w_i, the set V_i^- serves as the environment for this agent. For *motors* external environment instead of empty sets of subordinate agents is used.

At the start of the second phase of step t, which is considered in this paragraph, agent w_i possesses the information received at the end of the previous step from the environment and during the previous phase from its superior agents, respectively:

$$o_i^-(t) = (m_j(t))_{j \in I_i^-},$$

$$o_i^+(t) = (a_j(t))_{j \in I_i^+},$$

and selects an action according to

$$a_i(t) \sim \pi_i(a \mid o_i^-(t), o_i^+(t)) = \pi_i(a|o_i(t)).$$

In response to this action, the environment represented by the set V_i^- returns two vectors: new observations and rewards,

$$o_i^-(t+1) = (m_j(t+1))_{j \in I_i^-}, \qquad r_i^-(t) = (r_j(t))_{j \in I_i^-}.$$

All subordinate agents must perform at least one step of execution to generate $m_j(t+1)$ and $r_j(t)$ for all $j \in I_i^-$.

To interpret the resulting list of rewards, we utilize the agent's reward aggregation function $R_i : \mathbb{R}^{l_i^-} \to \mathbb{R}$. The reward of agent w_i for executing action $a_i(t)$ is then defined as

$$r_i^{\text{target}}(t) \coloneqq R_i(r_i^-(t)).$$

The function R_i may be specified in various ways (e.g., mean, maximum, weighted sum), providing flexibility and opening avenues for further investigation.

System initialization The system inference begins by sampling an initial state $s_0 \sim p_0$ from the external environment. This state is provided as $o_i^-(0)$ to all motors $w_i \in V_0$. We initialize $r_i(-1) = 0$ for all motors. With these initial values, the motors possess all the information required to perform upstream inference at $t = 0$. Once the motors execute, they trigger a cascade of observations and rewards through the superior agents, which in turn return their directives. Using these signals, the motors interact with the external environment and advance to the next timestep.

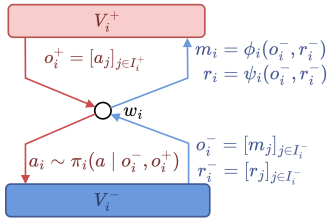

Figure 3: Information flow from the perspective of agent w_i. Rounded rectangles indicate V_i^+ (red) and V_i^- (blue). Actions propagate downstream (red), while messages and rewards flow upstream (blue). One timestep corresponds to a counterclockwise cycle starting at the upper-right corner.

Multiple time scales. It is worth noting that hierarchical graph structure naturally allows different agents to operate on distinct though consistent time scales, reflecting the varying reaction speeds of agents at different depths. Specifically, for a single time step of agent w_i, one can execute T steps for the subordinate levels ($s = 1, \ldots, T$), producing sequences

$$(m_j(s))_{s=1}^T, \quad (r_j(s))_{s=1}^T$$

for all $j \in I_i^-$. At each internal step, the actions from superior agents $a_j^+(1)$ are reused, keeping the top-level instructions constant throughout the execution of the internal sequence of steps.

The resulting sequences of rewards and observations are then aggregated over time to form the inputs for w_i, for example:

$$m_j(t) = \frac{1}{T} \sum_{s=1}^{T} m_j(s), \qquad r_j(t) = \sum_{s=1}^{T} r_j(s).$$

3.4 Learning process

We propose using reinforcement learning to train both agents' policies as well as their communication and proxy-reward functions. Accordingly, the following Markov Decision Processes (MDPs) are defined for each agent. In what follows, we take agent w_i as a reference.

Each component of the agent—policy, communication, and proxy-reward function—can be formalized as a separate MDP. This formulation allows us to treat learning each component consistently within the reinforcement learning framework, while capturing the interactions among them.

Policy MDP. The MDP for the agent's policy is defined as $\langle O_i, A_i, p_{\pi_i}, p_{\pi_i}^R, p_{\pi_i}^0 \rangle$, where:

- $O_i = O_i^- \times O_i^+ = \prod_{j \in I_i^-} M_j \times \prod_{j \in I_i^+} A_j$ is the state space.
- A_i is the action space.
- $p_{\pi_i} : O_i \times A_i \to \Delta(O_i)$ is an unknown transition function.
- $p_{\pi_i}^R : O_i \times A_i \to \Delta(\mathbb{R})$ is the reward distribution, determined by R_i, subordinate agents, and the external environment.
- $p_{\pi_i}^0 \in \Delta(O_i)$ is the initial state distribution, derived from the external environment and the communication functions of subordinate agents.

A trajectory for learning the policy of agent w_i is

$$\big(o_i(0), a_i(0), r_i^{\text{target}}(0), o_i(1), \ldots, o_i(T)\big).$$

Communication MDP. Let $D_i = O_i^- \times \mathbb{R}^{l_i^-}$. The communication MDP is $\langle D_i, M_i, p_{\phi_i}, p_{\phi_i}^R, p_{\phi_i}^0 \rangle$, with:

- state space: D_i

- action space: M_i
- transition function: $p_{\phi_i} : D_i \times M_i \to \Delta(D_i)$.
- reward distribution: $p^R_{\phi_i} : D_i \times M_i \to \Delta(\mathbb{R})$ as the reward distribution, determined by R_i, proxy-reward functions, and the external environment.
- $p^0_{\phi_i} \in \Delta(D_i)$ as the initial state distribution, derived from the external environment and subordinate agents' communication functions.

A trajectory for learning the communication function of agent w_i is

$$\big(d_i(0), m_i(0), r_i^{\text{target}}(0), d_i(1), \ldots, d_i(T)\big),$$

where $d_i(t) = (o_i^-(t), r_i^-(t-1))$.

Proxy-Reward Function. Finally, for learning the proxy-reward function, the trajectory is

$$\big(d_i(0), r_i(0), r_i^{\text{target}}(0), d_i(1), \ldots, d_i(T)\big),$$

with $\mathbb{R}$ as the action space.

This MDP-based formulation enables the usage of various RL and MARL algorithms for learning each component while accounting for interactions between the agent's policy, communication, and proxy-reward function.

4 Application

In the previous section, we formulated the Reinforcement Networks framework and described the inference procedure of the model. We now present one possible approach—among the wide variety of potential implementations—for realizing this framework. Our method builds upon the LevelEnv framework introduced in (Paolo et al., 2025).

4.1 Layered digraph

The notion of *outgoing depth* of a vertex in the graph G is defined by induction. First, for each sink vertex w_s, we set $d_s := 0$. Next, consider a vertex w_i. Assume that the outgoing depths d_j have already been defined for all $w_j \in V_i^-$. We then define $d_i := \max_{j \in I_i^-} d_j + 1$. In other words, the outgoing depth d_i corresponds to the maximum length of a directed path from w_i to any sink of the acyclic graph G.

We now introduce *identity node*. Such nodes effectively act as transparent intermediaries within the graph, simply forwarding information and actions without modification. Formally, an identity node w_n is a vertex satisfying the following conditions:

- $l_n^- = 1$;
- $\phi_n(o_n^-, r_n^-) = (o_n^-, r_n^-)$, i.e., the communication and reward-proxy functions simply forwards the observations and reward from the subordinate agent;
- $\pi_n(o_n^-, a_n^+) = a_n^+$, i.e., the policy is deterministic and directly replicates the actions received from superior agents;
- $M_n = O_n^-$ and $A_n = A_n^+$, so that the message and action spaces coincide with the observation and instruction spaces, respectively.

We define the *edge expansion* (by one node) operation for an edge $(w_i, w_j) \in E$ within a graph $G = (V, E)$, where $G \in \mathcal{G}$ and $\mathcal{G}$ denotes the set of DAGs. Formally, let $N = |V|$ and define a new identity node w_{N+1} such that $I^+_{N+1} = \{i\}, \quad I^-_{N+1} = \{j\}$. The updated vertex set is $V' = V \cup \{w_{N+1}\}$, and the edge set is modified as $E' = \big(E \setminus \{(w_i, w_j)\}\big) \cup \{(w_i, w_{N+1}), (w_{N+1}, w_j)\}$. The resulting graph is defined as $G' = (V', E') \in \mathcal{G}$.

The expansion of an edge (w_i, w_j) by $k \geqslant 2$ nodes is defined recursively as a sequence of single-node expansions. Specifically, first we expand the edge (w_i, w_j) with identity node w_{N+1}, then at step $t = 1, \ldots, k-1$, each edge (w_{N+t}, w_j) is expanded by inserting an identity node.

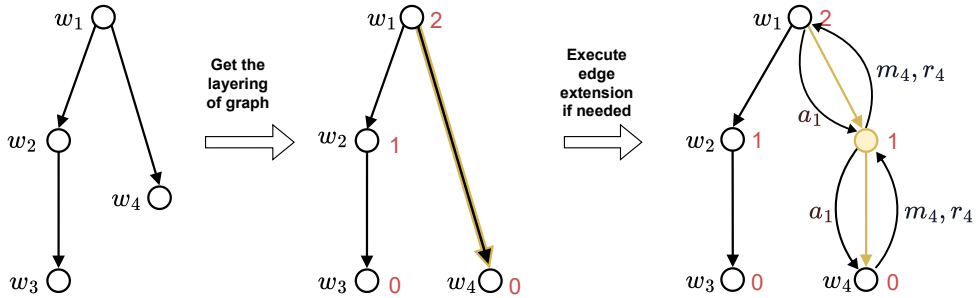

Figure 4: Transformation of a directed acyclic graph (DAG) into a *layered digraph*. Each vertex is assigned to one layering component. The index of the latter is written in red. The edge (w_1, w_4) connects vertices of layers 2 and 0, respectively. The edge expansion operation is applied to this edge by inserting a single identity node w_5, resulting in the edges (w_1, w_5) and (w_5, w_4). In the resulting graph (V', E'), for all $(w_i, w_j) \in E'$, it holds that $d_i - d_j = 1$.

Namely, the operation inserts a new vertex w_{N+1} along the edge (w_i, w_j) that merely forwards received messages and actions without modification. See Figure 4 for an illustration.

A *layered digraph* is a directed acyclic graph (DAG) in which vertices are assigned to discrete layers such that every edge connects a vertex on one layer to a vertex on the immediately lower layer. Any directed acyclic graph (DAG) can be transformed into a functionally equivalent *layered digraph*. To achieve this, it suffices to apply the edge expansion operation to every edge $(w_i, w_j) \in E$ for which $d_i - d_j > 1$, inserting $(d_i - d_j - 1)$ intermediate vertices.

The partitioning of a directed acyclic graph (DAG) into layers can be accomplished using algorithms such as the Longest Path algorithm, the Coffman-Graham algorithm, or the ILP algorithm of Gansner et al. (for an overview, see (Healy & Nikolov, 2001)). In this context, the edge expansion operation serves as an instrument to convert any DAG into a layered form, ensuring that each edge connects vertices on consecutive layers. The aforementioned algorithms can then be applied to determine an optimal placement of vertices across the layers, minimizing the total width or other layout-related objectives. However, all the graph sinks are restricted to be placed on the same level for the following application.

4.2 Connection to *LevelEnv*

Transforming the DAG into a layered digraph enables a direct implementation of the *LevelEnv* abstraction (Paolo et al., 2025), where each layer acts as an environment for the superior level while functioning as a set of agents with respect to the subordinate one.

At layer L_l, each agent selects its action based on observations and rewards received from subordinate agents in V_i^- together with directives provided by superior agents in V_i^+. The individual actions are then assembled into a joint action vector and passed to L_{l-1}, where masking ensures that each subordinate agent processes only the relevant components. Using its local observations and rewards, each subordinate agent produces outputs via the communication function ϕ (messages) and the function ψ (rewards). These outputs are aggregated into layer-wide vectors and returned upward. At L_l, masking redistributes the returned signals to the appropriate agents, after which ϕ and ψ are applied again to produce the aggregated outputs for the superior layer.

Thus, layers exchange information exclusively through unified vectors, while masking enforces the graph structure by restricting each agent to signals from its designated neighbors. This design simultaneously preserves agent-level independence and allows each layer to operate on its own temporal and spatial scale.

5 EXPERIMENTS

In conducting our experiments, we employed two graph configurations: **3PPO**, taken from (Paolo et al., 2025), and its modification **3PPO-full**. The 3PPO configuration represents a complete binary tree of height 3. In 3PPO-full, we augment this structure by adding edges from the root node to each leaf node. All experiments were conducted on the VMAS balance task.

For each configuration, we constructed three variants: (1) **-none**, where communication and proxy-reward functions are not learned (proxy-reward is a simple sum of rewards and observations are passed upward unchanged); (2) **-comm**, where the communication function is learned via PPO while the proxy-reward remains a sum of rewards; and (3) **-both**, where both communication and proxy-reward functions are learned using PPO. For the 3PPO configuration, we additionally implemented a modification **-comm-autoenc**, where the communication function employs an autoencoder trained simultaneously with the policy. Results are averaged over 10 runs for each configuration.

As can be observed from the results, the utilization of learnable communication functions leads to improved model performance. However, simultaneously learning both communication and proxy-reward functions destabilizes the training process. Training the proxy-reward function did not result in increased sample efficiency or performance improvements. Conversely, the introduction of additional edges in the 3PPO-full configuration mitigated the negative impact of the sharp increase in the number of learnable functions.

6 CONCLUSION

In this work, we proposed a novel, flexible, and scalable approach to constructing solutions for collaborative MARL tasks. Our research opens a wide range of directions for further developing the theory of Reinforcement Networks. Below, we outline several promising avenues for future work.

Optimal topology construction. Developing algorithms and methods to identify optimal DAG topologies tailored to specific MARL tasks is a key challenge. Progress in this area would support both researchers and practitioners in designing more effective systems.

Automatic hyperparameter optimization. Hyperparameter tuning remains a fundamental challenge in Reinforcement Learning. Exploring automated strategies in the context of Reinforcement Networks offers significant potential for improving training efficiency and stability.

Proxy-reward and communication function training. Advancing the theory and practice of training proxy-reward and communication functions is, in our view, one of the most promising directions for enhancing both the robustness and performance of our approach.

LLM-based agents. Another important direction is to investigate how our methods can be applied to the tuning of LLM-based agents. This raises a number of open questions, ranging from the design of communication and proxy-reward functions to optimizing training and inference efficiency.

AUTHOR CONTRIBUTIONS

If you'd like to, you may include a section for author contributions as is done in many journals. This is optional and at the discretion of the authors.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

Christopher Amato. An introduction to centralized training for decentralized execution in cooperative multi-agent reinforcement learning, 2024. URL https://arxiv.org/abs/2409.03052.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Matteo Bettini, Amanda Prorok, and Vincent Moens. Benchmarl: Benchmarking multi-agent reinforcement learning. *Journal of Machine Learning Research*, 25(217):1–10, 2024. URL http://jmlr.org/papers/v25/23-1612.html.

Lucas Cassano, Kun Yuan, and Ali H Sayed. Multiagent fully decentralized value function learning with linear convergence rates. *IEEE Transactions on Automatic Control*, 66(4):1497–1512, 2020.

Kevin Chen, Nithin Shrivatsav Srikanth, David Kent, Harish Ravichandar, and Sonia Chernova. Learning hierarchical task networks with preferences from unannotated demonstrations. In *Conference on Robot Learning*, pp. 1572–1581. PMLR, 2021.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. WorkflowLLM: Enhancing workflow orchestration capability of large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=3Hy00Wvabi.

Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

Zeyu Gao, Yao Mu, Jinye Qu, Mengkang Hu, Shijia Peng, Chengkai Hou, Lingyue Guo, Ping Luo, Shanghang Zhang, and Yanfeng Lu. Dag-plan: Generating directed acyclic dependency graphs for dual-arm cooperative planning. *arXiv preprint arXiv:2406.09953*, 2024.

Ilche Georgievski and Marco Aiello. Htn planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156, 2015.

Shailja Gupta, Rajesh Ranjan, and Surya Narayan Singh. A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions. *arXiv preprint arXiv:2410.12837*, 2024.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Patrick Healy and Nikola S Nikolov. How to layer a directed acyclic graph. In *International symposium on graph drawing*, pp. 16–30. Springer, 2001.

Dom Huh and Prasant Mohapatra. Multi-agent reinforcement learning: A comprehensive survey, 2024. URL https://arxiv.org/abs/2312.10256.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1): 172–221, 2022.

Emilio Jorge, Mikael Kågebäck, Fredrik D Johansson, and Emil Gustavsson. Learning to play guess who? and inventing a grounded language as a consequence. *arXiv preprint arXiv:1611.03218*, 2016.

Bharti Khemani, Shruti Patil, Ketan Kotecha, and Sudeep Tanwar. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, 11(1):18, 2024.

Michael Levin. Technological approach to mind everywhere: An experimentally-grounded framework for understanding diverse bodies and minds. *Frontiers in Systems Neuroscience*, 16, 2021. URL https://api.semanticscholar.org/CorpusID:245553743.

Zhuoru Li, Akshay Narayan, and Tze-Yun Leong. An efficient approach to model-based hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Ofir Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv:1909.10618*, 2019.

Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

Afshin Oroojlooy and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 53(11):13677–13722, 2023.

Giuseppe Paolo, Abdelhakim Benechehab, Hamza Cherkaoui, Albert Thomas, and Balázs Kégl. Tag: A decentralized framework for multi-agent hierarchical reinforcement learning. *Under review by the International Conference on Machine Learning (ICML)*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Ming Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *International Conference on Machine Learning*, 1997. URL https://api.semanticscholar.org/CorpusID:281306197.

Jordan Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

Micaela Verucchi, Ignacio Sañudo Olmedo, and Marko Bertogna. A survey on real-time dag scheduling, revisiting the global-partitioned infinity war. *Real-Time Systems*, 59(3):479–530, 2023.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International conference on machine learning*, pp. 3540–3549. PMLR, 2017.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

Jia Xu, Weilin Du, Xiao Liu, and Xuejun Li. Llm4workflow: An llm-based automated workflow model generation tool. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 2394–2398, 2024.

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. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*, 2024.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020a.

Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for cooperative multi-agent reinforcement learning, 2020b. URL `https://arxiv.org/abs/2007.02529`.

A Appendix

A.1 Implementation and Experiment Setup

This appendix details the hierarchical agent and training configuration used in our experiments. The implementation is based on `PyTorch`, `PettingZoo` (parallel API), and a lightweight PPO variant (`SimplePPO`). The agent composes three control levels (top/middle/bottom) managed by a `Hierarchy` orchestrator.

A.2 System Overview

We instantiate a three-level hierarchy with learnable, continuous communication vectors (ϕ) and optional proxy rewards (ψ). Unless stated otherwise, communication learning is enabled and proxy rewards are disabled:

- **Bottom level:** one `SimplePPO` per environment agent $\texttt{agent}_i$. Each bottom agent acts in the environment and receives a scalar reward.
- **Middle level:** two `SimplePPO` controllers (`middle_ppo_1`, `middle_ppo_2`); each governs half of the bottom agents and outputs a discrete directive for each subordinate.
- **Top level:** a single `SimplePPO` (`top_ppo`) that issues directives to the two middle controllers. Communication and proxy rewards are *disabled* at this level.

For the common case of four environment agents, the topology is:

Action Frequencies. Each level advances once per environment step by default: $\texttt{ifreq\_bottom} = \texttt{ifreq\_mid} = \texttt{ifreq\_top} = 1$.

A.3 Interfaces: Observation, Action, Directives, Communication

Let A be the number of environment agents and C the communication size.

- **Bottom level**
  - *Observation*: provided by the environment ($\mathcal{O}_{\text{env}}(i)$ for agent i); concatenation is enabled in the hierarchy when needed.
  - *Action*: environment action space for agent i.
  - *Directive space* (from middle): $\mathrm{Discrete}(5)$ per bottom agent.
  - *Communication* (to middle, if enabled): $\mathrm{Box}(-\infty, \infty; C)$.
  - *Reward*: scalar (1-dim).
- **Middle level**
  - *Observation*: if communication learning is enabled, a concatenation of C-dim vectors from each subordinate $\Rightarrow$ $\mathrm{Box}(-\infty, \infty; C \times \#\text{subordinates})$. Otherwise, concatenated raw env observations from subordinates.
  - *Action*: a $\mathrm{Discrete}(5)$ directive per subordinate, emitted as a keyed dictionary (one entry per subordinate).

  - *Directive space* (from top): $\mathrm{Discrete}(5)$.
  - *Communication* (to top, if enabled): $\mathrm{Box}(-\infty, \infty; C)$.
  - *Reward*: vector with one component per subordinate.
- **Top level**
  - *Observation*: if communication is enabled, $\mathrm{Box}(-\infty, \infty; C \times 2)$ (two middle controllers); otherwise the sum of middle observation sizes.
  - *Action*: a $\mathrm{Discrete}(5)$ directive per middle controller (keyed dictionary).
  - *Directive space*: none (top receives no directives).
  - *Communication*: none (disabled at this level).
  - *Reward*: vector with one component per middle controller.

A.4 GRAPH LAYERING

Pseudocode of the proposed DAG conversion to layered digraph algorithm might be found at 1.

Algorithm 1 Graph Transformation with Outgoing Depth Computation

Input: Directed graph $G = (V, E)$
Output: Layered digraph $G' = (V', E')$ with identity nodes

procedure OUTGOINGDEPTH(G)
  for all $w_i \in V$ **do**
    $d_i \leftarrow$ **None**
  end for
  for all $w_i \in V$ **do**
    if $d_i =$ **None then**
      DFS-VISIT(w_i)
    end if
  end for
end procedure
procedure DFS-VISIT(w_i)
  $d_i \leftarrow 0$
  ChildrenDepths $\leftarrow$ [ ]
  for all $w_j \in I_i^-$ **do** ▷ I_i^- = set of children of w_i
    if $d_j =$ **None then**
      DFS-VISIT(w_j)
    end if
    ChildrenDepths.append(d_j)
  end for
  if ChildrenDepths $\neq$ [ ] **then**
    $d_i \leftarrow \max(\text{ChildrenDepths}) + 1$
  end if
end procedure
procedure EDGEEXTENSION(G, w_i, w_j)
  $N \leftarrow |V|$
  introduce new node w_{N+1}
  $A_{N+1} \leftarrow A_j$ ▷ copy attribute of w_j
  $M_{N+1} \leftarrow M_i$ ▷ copy attribute of w_i
  $E' \leftarrow (E \setminus \{(w_i, w_j)\}) \cup \{(w_i, w_{N+1}), (w_{N+1}, w_j)\}$
  $V' \leftarrow V \cup \{w_{N+1}\}$
  return $G' = (V', E')$
end procedure
procedure TRANSFORMGRAPH(G)
  OUTGOINGDEPTH(G) ▷ compute outgoing depths d_i for all $w_i \in V$ with the algorithm you like
  $V' \leftarrow V$, $E' \leftarrow E$
  for all $(w_i, w_j) \in E$ **do**
    if $d_i - d_j > 1$ **then**
      $k \leftarrow 1$
      $(V', E') \leftarrow$ EDGEEXTENSION($(V', E'), w_i, w_j$)
      while $k < d_i - d_j - 1$ **do**
        $(V', E') \leftarrow$ EDGEEXTENSION($(V', E'), w_{|V'|}, w_j$)
        $k \leftarrow k + 1$
      end while
    end if
  end for
  return (V', E')
end procedure