Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 56-2 (2023) 3033–3040

Learning Heterogeneous Agent Cooperation via Multiagent League Training ⋆

Qingxu Fu *,** **Xiaolin Ai** *,** **Jianqiang Yi** *,** **Tenghai Qiu** *,**
Wanmai Yuan *** **Zhiqiang Pu** *,**

* *Institute of Automation, Chinese Academy of Sciences.*
** *School of Artificial Intelligence, University of Chinese Academy of Sciences.*
*** *Electronics Technology Group Corporation, Information Science Academy of China. Beijing, China.*

Abstract: Many multiagent systems in the real world include multiple types of agents with different abilities and functionality. Such heterogeneous multiagent systems have significant practical advantages. However, they also come with challenges compared with homogeneous systems for multiagent reinforcement learning, such as the non-stationary problem and the policy version iteration issue. This work proposes a general-purpose reinforcement learning algorithm named Heterogeneous League Training (HLT) to address heterogeneous multiagent problems. HLT keeps track of a pool of policies that agents have explored during training, gathering a league of heterogeneous policies to facilitate future policy optimization. Moreover, a hyper-network is introduced to increase the diversity of agent behaviors when collaborating with teammates having different levels of cooperation skills. We use heterogeneous benchmark tasks to demonstrate that (1) HLT promotes the success rate in cooperative heterogeneous tasks; (2) HLT is an effective approach to solving the policy version iteration problem; (3) HLT provides a practical way to assess the difficulty of learning each role in a heterogeneous team.

Keywords: Heterogeneous System, Reinforcement Learning, Multiagent System.

1. INTRODUCTION

Heterogeneous Multiagent Reinforcement Learning (HMARL) is not a new problem in MARL, but they have not yet been investigated extensively in the literature. A number of studies have covered HMARL, investigating its application in specific domains such as communicationFard and Selmic (2022) and UAV allocationZhao et al. (2019). However, to the best of our knowledge, there are currently no general-purpose heterogeneous reinforcement learning algorithms for cooperative learning within a multi-type, multiagent team.

In general, classic MARL algorithms do not assume the type difference between agents. Nevertheless, most existing research on MARL is primarily focused on homogeneous environments Zheng et al. (2018); Deka and Sycara (2021); Fu et al. (2022b) or heterogeneous benchmarks consisting of only 2 agent types (e.g., most SMAC maps Samvelyan et al. (2019)) to claim the effectiveness of their models. Inhabiting a diverse world, we cannot rely solely on homogenous systems to efficiently solve sophisticated teamwork. Many real multiagent systems require diversified agents to participate to ensure the division of labor and lower costs. A typical example of this is found in colonies of social insects, such as honeybees and ants, where individual agents specialize in different tasks like foraging, nest construction and defense. There are two primary challenges to address when applying MARL algorithms to heterogeneous multiagent tasks:

Firstly, non-stationarity Hernandez-Leal et al. (2017) is one of the difficulties presented by heterogeneous multiagent problems. Neglecting heterogeneous properties between agents can limit the performance of MARL models. In heterogeneous multiagent tasks, different kinds of agents vary in ability, quantity, expected functionality, etc. Notably, the number of each agent type is unbalanced in most scenarios. E.g., in military simulations, high-value aircraft are essential for obtaining battlefield advantage but are very few in number compared with ground agents. This imparity is also common in the world of nature, e.g., in a hive, honey bee queens and workers differ significantly in number. During an MARL training session, agents that are relatively low in number suffer from the high variance in state-value estimation due to insufficient sampling, which results in further non-stationary problems.

Secondly, heterogeneous systems contain more realistic problems that have not yet been studied in the context of decentralized automation systems. Once a team of heterogeneous agents is trained and deployed, maintaining a distributed control system becomes an important issue and can cost considerable resources. In the event of a system failure caused by a single type of agents, upgrading only the policies of the misbehaving agent type is a more rea-

⋆ This work was supported in part by the National Key Research and Development Program of China (2018AAA0102404), the National Natural Science Foundation of China (62073323), the Strategic Priority Research Program of Chinese Academy of Sciences (XDA27030204), the External Cooperation Key Project of Chinese Academy Sciences (173211KYSB20200002), the Science and Technology Development Fund of Macau (No.0025/2019/AKP), and the Beijing Nova Program under Grant 20220484077.

10.1016/j.ifacol.2023.10.1431

sonable and economical approach rather than upgrading the policies of the entire team. This requires agents to have the ability to cooperate with their heterogeneous teammates, even if they have different policy versions. For example, if the flying policy of airborne agents is altered, it is important that other non-flying agents in the same heterogeneous system can acclimate to their teammates' policy shifts without requiring a series of chained policy updates. Such capability is especially essential in automation systems that cannot enable Over-the-Air Programming (OTA) due to reliability or security concerns (such as military robot systems), or systems that cannot afford to suffer a full stop in long-term service (such as traffic and network devices). Therefore, a method to iterate policies without breaking the compatibility between different heterogeneous agents is needed to reduce the cost of maintenance in actual applications of heterogeneous RL algorithms. In this work, we summarize this compatibility issue as the policy version iteration problem.

In this paper, our contributions are as follows: 1. We propose Heterogeneous League Training (HLT), a general-purpose MARL algorithm for heterogeneous multiagent tasks. 2. We demonstrate the superior performance of HLT and the effectiveness of HLT in addressing the policy version iteration problem. 3. Based on HLT, we propose a method for evaluating the difficulty of learning the roles undertaken by different agent types.

2. RELATED WORKS

HMARL is not a new problem in MARL, yet it has not yet been investigated extensively in the literature. While there are works introducing model-based HMARL algorithms on specific domains, e.g., traffic control, medical information processing, and control system Calvo and Dusparic (2018); Fard and Selmic (2022), a general-purpose heterogeneous reinforcement learning is absent as far as we're concerned. On one hand, tasks stressing their heterogeneous property are likely to have solid application backgrounds and concerns on realistic problems. On the other hand, existing simulation and training frameworks cannot support flexible task customizations for heterogeneous systems.

While most RL simulation environments used in multiagent studies only consider homogeneous agent tasks Deka and Sycara (2021); Zheng et al. (2018); Fu et al. (2022b), benchmark environments that involve heterogeneous agent cooperation are also emerging. For instance, Multi-Agent Particle Environment (MAPE) Lowe et al. (2017) is a simple but flexible multiagent environment, designed for less complex tasks, such as predator-prey and speaker-listener tasks. The SMAC benchmark environment proposed in Samvelyan et al. (2019) provides some examples of heterogeneous team combinations, such as 1c3s5z and MMM2. Nevertheless, only a small proportion of the maps (such as MMM2) consider the supportive relationships within team agents.

Existing MARL studies tend to address heterogeneous agent cooperation problems implicitly, without distinguishing them especially from homogeneous ones Rashid et al. (2018, 2020). Firstly, all agents are provided with an identical action space, irrespective of their types. If there are type-specific actions only available to certain agent types, an action masking technique is used to handle these special cases. Secondly, agents are not explicitly modeled according to their types; instead, agents need to discover their abilities and strengths by interacting with the environment and teammates. This practice is successful in many task domains with weak agent ability distinction because agent experience can be shared regardless of agent types. However, in complex heterogeneous problems, the agent policy cannot be generalized across different types, preventing agents from benefiting from shared experiences with agents of different types.

In recent years, the majority of SOTA algorithms based on multiagent systems have been developed without explicit agent-type consideration. In early models, such as IQL (Independently Q-Learning) Kröse (1995), each agent learns its own policy independently. According to Lowe et al. (2017), directly transferring single-agent approaches to multiagent environments results in a non-stationary problem Hernandez-Leal et al. (2017). To address this issue, Lowe et al. (2017) proposes a MADDPG algorithm, which is based on DDPG (Deep Deterministic Policy Gradient Lillicrap et al. (2015)), for stabilizing multiagent RL. Moreover, a value decomposition algorithm VDN Sunehag et al. (2017) proposes a model based on Q-learning. And the VDN model is further improved by Qmix Rashid et al. (2018) and Weighted Qmix Rashid et al. (2020), which use hyper-networks Ha et al. (2016) to decompose the task reward signals. Nevertheless, classic MARL methods ignore the discrepancies in the numbers and abilities of agents from different types, which can negatively impact algorithm performance.

3. PRELIMINARIES

3.1 Heterogeneous MARL.

A heterogeneous MARL task can be described as a Dec-POMDP Oliehoek and Amato (2016) formulated by $\langle A, \Delta, \mathcal{U}, \mathcal{S}, P_t, \mathcal{Z}, P_o, r, \gamma \rangle$, where A is the collection of agents and $N = |A|$ is the total number of agents. $\mathcal{U}$ is a collection of actions within which each agent a_i can choose. $\mathcal{S}$ is the set of environment states and P_t is the transition function. r is the team reward. γ is the discount factor. In a heterogeneous system, $\Delta = \{\delta_1, \ldots, \delta_{n_j}\}$ is the set of agent types and $n_j = |\Delta|$ is the total number of types in the system. Each agent a_i is categorized into a type $\text{Type}(a_i) = \delta_j$. Although the number of agents N can be large, the number of types n_j is considerably smaller ($n_j \ll N$). This is because real-world agents, typically robots, are usually manufactured using standardized

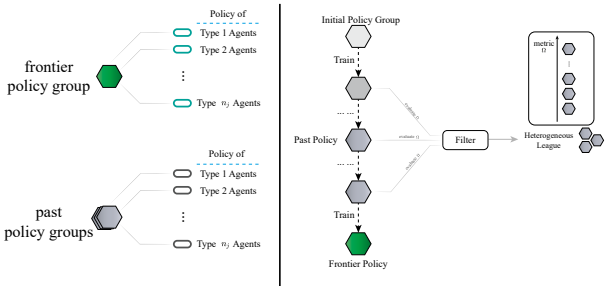


Fig. 1. Heterogeneous League Training framework.

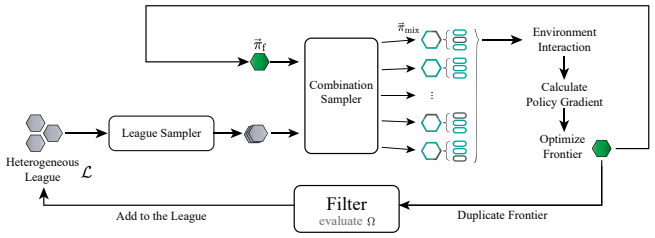


Fig. 2. Policy optimization cycle.

production to reduce costs. Furthermore, the numbers of different types of agents can vary significantly, since it is common that high-value agents are fewer than cheap agents in a functional heterogeneous team.

3.2 League Training in AlphaStar.

League training is first introduced in AlphaStar Vinyals et al. (2019). To enhance the initial capability of agents before optimizing them with reinforcement learning, AlphaStar takes advantage of data from human experts to train agents in a supervised approach, obtaining a population of agents that possess diverse policies. Agents are subsequently trained with reinforcement learning, and intermittently duplicate themselves and freeze these duplications as past players. The agents are trained against each other as well as past selves in history. League training enables agents to learn adaptive and robust policies against the continually developing strategies of opponents. This characteristic also has significant value in heterogeneous MARL for addressing the policy version iteration problem and improving model performance.

4. HETEROGENEOUS LEAGUE TRAINING

We propose a Heterogeneous League Training (HLT) algorithm, an efficient and general-purpose multiagent RL algorithm for heterogeneous cooperative multiagent systems.

This section starts by stating the motivations and model features in Sec 4.1. Next, we briefly introduce the general framework of HLT in Sec 4.2. Then, the details of heterogeneous league training are presented in Sec 4.3 and Sec 4.4. Finally, as an essential structure to achieve adaptive cooperative behaviors, the design of hyper-networks is illustrated in Sec 4.5.

4.1 Motivation and Features.

HLT aims at two goals in heterogeneous multiagent RL:

- Taking advantage of the feature of heterogeneous systems to facilitate cooperation.
- Addressing the policy version iteration problem for real-world automation system.

Motivation The following reasons inspire us to use the **league training** Vinyals et al. (2019) technique to achieve our goals.

(1) The numbers of different types of agents can vary considerably within a heterogeneous team. Episode samples collected by the RL algorithm can provide low-variance experiences for types with numerous agents (T+). On the other hand, some types with comparably fewer agents (T-) receive high-variance experiences. When T- agents fail to keep pace with T+ agents, the learning process becomes unstable. Additionally, T+ agents tend to abandon their reliance on T- agents and instead develop uncooperative policies that exclude assistance from T- agents. By adopting the cooperative league training technique, agents optimize their policies not only from the experiences of cooperating within a fixed team, but also from the experiences of teaming up with foreign agents that possess distinct cooperation skills. More specifically, the cooperative league training technique can address the learning instability problem by encouraging agents to adopt different cooperation strategies depending on the unique traits of their teammates.

(2) Cooperative league training provides an environment where agents are trained to work with a league of heterogeneous teammates equipped with diverse collaboration preferences. By incorporating historical agent policies into the league, agents can learn new cooperative behaviors without forgetting the way to collaborate with old versions of heterogeneous teammates.

Distinctive features of HLT (1) In this paper, we propose a **cooperative** league training algorithm designed to facilitate cooperation among heterogeneous agents. As a comparison, AlphaStar is an example of an **adversarial** multiagent league training algorithm Vinyals et al. (2019); Han et al. (2020) where agents are trained to compete.

(2) The AlphaStar league model requires distributed servers with TPUs to run concurrent matches. But our algorithm only needs a single machine with a single GPU to execute the entire training process. To achieve this, we redesign the procedure of the league training based on the characteristic of heterogeneous systems.

(3) We propose a hyper-network structure, facilitating each agent with the capability to adapt its cooperative behavior according to its type and the performance of its heterogeneous teammates.

4.2 Overall Framework of HLT.

Actors and Critic The HLT algorithm utilizes the actor-critic framework Konda and Tsitsiklis (2000) and follows the Centralized-Training-Decentralized-Execution (CTDE) paradigm Lowe et al. (2017). As a prerequisite, all agents in the heterogeneous team are assumed to possess prior knowledge of their teammates' type identities. The (actor) policy network parameters are shared among agents belonging to the same type. However, agents of different types can freely choose between using shared parameters or independent networks. We use a term **policy group** (denoted as $\boldsymbol{\pi}$) to represent all policies of a heterogeneous team. For generality, policies for all agent types are represented by $\boldsymbol{\pi}$:

$$\boldsymbol{\pi} = \{\pi(\delta_1), \ldots, \pi(\delta_{n_j})\}, \tag{1}$$

where each agent type is represented by $\delta_1, \ldots, \delta_{n_j}$. It is worth noting that each policy $\pi(\delta_j) \in \boldsymbol{\pi}$ is possessed by all agents that belong to type δ_j in the team.

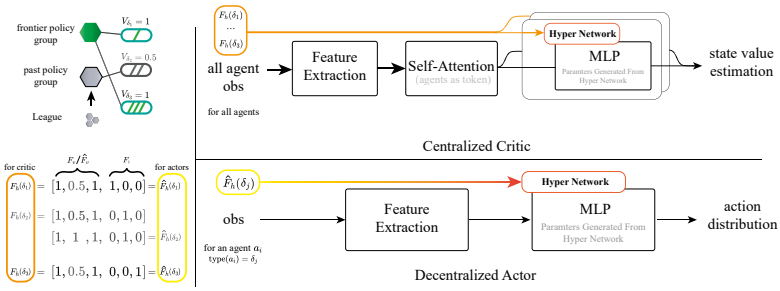


Fig. 3. Hyper network for learning adaptive cooperative behavior.

Algorithm 1 HLT Policy Optimization

Initialize a frontier policy group $\boldsymbol{\pi}_{\mathrm{f}}$ and an empty league $\mathcal{L}$.
for optimization step $M_s = 1, 2, \ldots$ **do**
  Initialize an empty epsiode sample collection $\mathcal{D}$.
  for episode $M_e = 1, 2, \ldots, M_{em}$ **do**
    Generate C_{sel} with the league sampler from $\boldsymbol{\pi}_{\mathrm{f}}$ and $\mathcal{L}$.
    Generate π_{mix} with the combination sampler and C_{sel}.
    Run episode with the mixed policy group π_{mix}.
    Add epsiode samples to $\mathcal{D}$.
  end for
  Calculate the policy gradient with $\mathcal{D}$.
  Update frontier policy networks $\boldsymbol{\pi}_{\mathrm{f}}$.
  if once every M_{st} optimization steps **then**
    Evaluate $\boldsymbol{\pi}_{\mathrm{f}}$ with metric Ω
    Add $\boldsymbol{\pi}_{\mathrm{f}}$ to $\mathcal{L}$ if $\boldsymbol{\pi}_{\mathrm{f}}$ is accepted by the league filter.
  end if
  Update the critic network with $\mathcal{D}$.
end for
End training.

To reduce the computational cost of training, we initialize and optimize only one dynamic learning policy group during league training. This policy group $\boldsymbol{\pi}_{\mathrm{f}}$ is referred to as the **frontier policy group**, or simply just **frontier** for clarity.

The HLT model employs a policy pool, also referred to as a **league** $\mathcal{L}$, to store historical policies visited by agents. In HLT model, a policy pool, or a **league** $\mathcal{L}$, holds historical policies agents have visited. This design bears a resemblance to the league structure of AlphaStar. We only utilize one league structure in the algorithm. As shown in Fig.1, during the training process, a series of past policies are intermittently duplicated from the frontier and added to the league. Once becoming independent and detached from the frontier, the network parameters of the detached policies are frozen so that we only need to compute policy gradient for frontier policy networks instead of for all policy networks in the league. These frozen policies in the league are referred to as **past policy groups**. Let $|\mathcal{L}|$ be the total number of policy groups in the league, the league members (past policy groups) are denoted as:

$$\mathcal{L} = \{\boldsymbol{\pi}_{\mathrm{p},1}, \ldots, \boldsymbol{\pi}_{\mathrm{p},|\mathcal{L}|}\}, \tag{2}$$

where $\boldsymbol{\pi}_{\mathrm{p},1}, \ldots, \boldsymbol{\pi}_{\mathrm{p},|\mathcal{L}|}$ are the past policy groups in the league.

Additionally, while there are different groups of policy networks (as actors) in HLT, there is **only one critic network** for action assessment during the entire training process.

Policy Optimization Iteration As illustrated in Fig. 1, the HLT algorithm begins by initializing random frontier policies $\boldsymbol{\pi}_{\mathrm{f}}$ and an empty league. Upon completion of each policy optimization iteration step, the new frontier policy group will be evaluated by a league filter, determining whether it is eligible for inclusion as a new league member (Sec. 4.4).

Under the condition $|\mathcal{L}| = 0$, agents simply adopt the policies in the frontier $\boldsymbol{\pi}_{\mathrm{f}}$. As the training process progresses, the league is incrementally populated with past policy groups. Once $|\mathcal{L}| > 0$, mixed policies are used for policy optimization. Complete details on policy mixing are provided in Sec.4.3.

4.3 Training with Mixed Policy Groups.

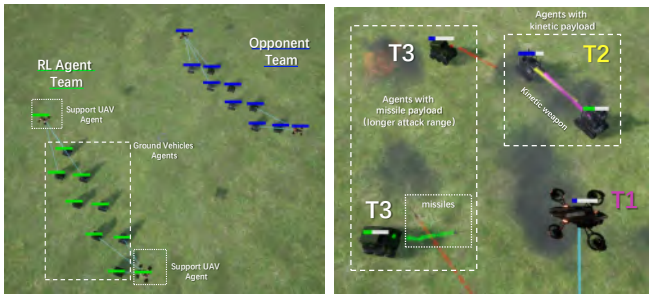
As illustrated in Fig. 2, the frontier policy group is trained to cooperate not only with itself, but also with past policies that visited previously during the learning process. During training the heterogeneous team uses a mixed policy group denoted as $\boldsymbol{\pi}_{\mathrm{mix}}$, which is a hybridization of frontier and past policies. Two samplers are used to generate $\boldsymbol{\pi}_{\mathrm{mix}}$ at the beginning of each training episode.

First, a **league sampler** randomly generates a policy combination C_{sel}. This combination is either frontier-past $C_{\mathrm{sel}} = (\boldsymbol{\pi}_{\mathrm{f}}, \boldsymbol{\pi}_{\mathrm{p},l})$, or frontier-frontier $C_{\mathrm{sel}} = (\boldsymbol{\pi}_{\mathrm{f}}, \boldsymbol{\pi}_{\mathrm{f}})$. The balance between frontier-frontier and frontier-past combinations is adjusted by a constant parameter $p_f \in [0, 1)$:

$$\begin{aligned} P\left[C_{\mathrm{sel}} = (\boldsymbol{\pi}_{\mathrm{f}}, \boldsymbol{\pi}_{\mathrm{p},l}) \mid p_f\right] &= \tfrac{1}{(1-p_f)|\mathcal{L}|} \\ P\left[C_{\mathrm{sel}} = (\boldsymbol{\pi}_{\mathrm{f}}, \boldsymbol{\pi}_{\mathrm{f}}) \mid p_f\right] &= p_f, \end{aligned} \tag{3}$$

where $|\mathcal{L}| > 0$ and $p_f < 1$. When the league is empty ($|\mathcal{L}| = 0$), the league sampler outputs only frontier-frontier combinations, namely $C_{\mathrm{sel}} = (\boldsymbol{\pi}_{\mathrm{f}}, \boldsymbol{\pi}_{\mathrm{f}})$. For instance, when $p_f = 0$, the sampler avoids pure frontier combinations and always samples one of the past policy groups from the league in C_{sel}. Conversely, when $p_f \approx 1$, the sampler tends to exclude the participant of past policies in the league.

Next, C_{sel} is accepted by a **combination sampler**, which eventually determines the policy that each agent executes. For clarity, we use $\boldsymbol{\pi}_{\mathrm{sel}}$ and $C_{\mathrm{sel}} = (\boldsymbol{\pi}_{\mathrm{f}}, \boldsymbol{\pi}_{\mathrm{sel}})$ to represent the selected policy group and the combination, respectively. As shown in Fig. 2, the combination sampler chooses one of the agent types to play the past policies. Each type



(a) A map of UHMP with 2 support UAVs (T1) and 8 ground agents (T2 and T3) on each side.

(b) An illustration of three types of ground agents. Three agent types (T1, T2, T3) have distinctive abilities and responsibilities.

has an equal probability $P(\delta_j)$ of being selected, regardless of the number of agents it possesses:

$$P(\delta_j) = 1/|\Delta|, \forall \delta_j \in \Delta. \tag{4}$$

Finally, let δ_t be the agent type chosen by the combination sampler. Then the mixed policy group $\boldsymbol{\pi}_{\text{mix}}$ that the team executes is represented by:

$$\boldsymbol{\pi}_{\text{mix}} = \{\pi_{\text{mix}}(\delta_1), \dots, \pi_{\text{mix}}(\delta_M)\}, \tag{5}$$

where

$$\pi_{\text{mix}}(\delta_j) = \begin{cases} \pi_f(\delta_j), & \delta_j \neq \delta_t \\ \pi_{\text{sel}}(\delta_j), & \delta_j = \delta_t \end{cases}, \forall \delta_j \in \Delta. \tag{6}$$

This mixed policy group $\boldsymbol{\pi}_{\text{mix}}$ is effective for only one episode. And the league sampler and combination sampler can generate a batch of different $\boldsymbol{\pi}_{\text{mix}}$ to run in parallel. This process is repeated to collect a batch of episodes for frontier policy optimization. The policy gradient (w.r.t. the frontier policy parameters) is calculated from the collected episodes and used to optimize each frontier policy of each agent type.

4.4 League Member Management.

To be able to run on a single machine, the HLT model must control the size of the league. In comparison, researchers of AlphaStar utilize a large past agent pool, which costs a significant amount of memory and time. The AlphaStar approach is not applicable to heterogeneous problems due to the significant cost of computational resources, which exceeds the limitations of most research facilities. Consequently, we developed a league management component to decrease the size of the league. We had the intuition that the league should comprise policies demonstrating a variety of behaviors and unique cooperative abilities; hence, we created a league filter to manage and regulate its size. This filter is illustrated in Figure 2.

In our HLT model, a metric $\Omega \in [0, 1]$ is introduced to compare the similarity of policy groups. This metric can be the success rate, normalized test reward (projecting the minimum reward and maximum reward to 0 and 1, respectively), or any other evaluation scores that reflect the policy performance.

Let $|\mathcal{L}|_{\max}$ be the maximum size of the league. Whenever the size of the pool exceeds the $|\mathcal{L}|_{\max}$ limitation, the league filter deals with policy group candidates with the following procedures:

(1) Evaluate the new input policy group with metric Ω.
(2) Add the policy group into the league, then sort all league policy groups in the league by their Ω scores.
(3) Locate the two policy groups with the closest Ω scores, then remove the newer group between them.
(4) Repeat the last step until the size of the league is reduced to the maximum allowed size $|\mathcal{L}|_{\max}$.

For instance, after a new policy group is added temporarily to an already-full league in step 2), the league will have one extra member that needs to be removed:

$$\mathcal{L} = \{\boldsymbol{\pi}_{\text{p},1}, \dots, \boldsymbol{\pi}_{\text{p},|\mathcal{L}|+1}\}. \tag{7}$$

To locate a policy group to remove, past policy groups are evaluated by metric Ω:

$$\Omega(\mathcal{L}) = \{\omega_1, \dots, \omega_{|\mathcal{L}|_{\max}+1}\} = \left\{\Omega(\boldsymbol{\pi}_{\text{p},1}), \dots, \Omega(\boldsymbol{\pi}_{\text{p},|\mathcal{L}|+1})\right\}, \tag{8}$$

where $\omega_1, \dots, \omega_N, \omega_{|\mathcal{L}|_{\max}+1}$ are evaluation scores. Then the league is sorted according to these scores, then rearrange $\mathcal{L}$ so that:

$$\omega_1 \leq \cdots \leq \omega_N \leq \omega_{|\mathcal{L}|_{\max}+1}. \tag{9}$$

Finally, find a pair of policy groups $(\boldsymbol{\pi}_{\text{p},k}, \boldsymbol{\pi}_{\text{p},k+1})$ with the closest evaluation metric:

$$|\omega_{k+1} - \omega_k| \leq |\omega_{k'+1} - \omega_{k'}|, \quad \forall k' \in [1, |\mathcal{L}|_{\max}]. \tag{10}$$

Considering the order of when group k and group $k+1$ are added into the league, we always remove the newer one between $\boldsymbol{\pi}_{\text{p},k}$ and $\boldsymbol{\pi}_{\text{p},k+1}$. By removing the relatively newer group instead of the older one, we can avoid frequent changes inside the league and prevent potential instability. After removal, the size of the league is reduced back to $|\mathcal{L}|_{\max}$.

4.5 Adaptive Hyper Network.

Cooperating with a variety of partners with different policies is a challenging task. Agents may encounter strong heterogeneous partners that are well-trained and cooperative. However, they may also encounter weak heterogeneous partners that are equipped with immature policies sampled from the league. As a result, it is necessary to equip agents with a neural network structure that is sensitive to the differences of heterogeneous partners. In the HLT model, we use a **hyper-network** structure to address this problem.

As shown in Fig. 3, the hyper-network is a structure to generate network parameters for MLP layers in forward policy networks. Internally, the hyper-network uses two layers of fully connected networks to produce MLP parameters, using an agent-team information vector F_h as input.

This agent-team information vector F_h is determined by agent type δ_j as well as the mixed policy composition $\boldsymbol{\pi}_{\text{mix}}$. It is a concatenation of two parts:

$$F_h(\delta_j, \boldsymbol{\pi}_{\text{mix}}) = \text{concat}\left[\, F_v(\delta_j, \boldsymbol{\pi}_{\text{mix}}),\ F_\delta(\delta_j)\,\right], \tag{11}$$

As it is shown in Fig.3, F_δ is a onehot binary vector to distinguish agent types. E.g., if $\delta_j = \delta_1$, then $F_\delta(\delta_j) = F_\delta(\delta_1) = [1, 0, 0, \dots]$. And $F_v(\delta_j, \boldsymbol{\pi}_{\text{mix}})$ represents the combination of the mixed heterogeneous team by identifying the policy that each agent type executes.

$$F_v(\delta_j, \boldsymbol{\pi}_{\text{mix}}) = (V_{\delta_1}, \dots, V_{\delta_{|\Delta|}}), \tag{12}$$

$$V_{\delta_j} = \begin{cases} 1, & \delta_j \neq \delta_{\text{sel}} \\ \Omega[\pi_{\text{sel}}(\delta_j)], & \delta_j = \delta_{\text{sel}} \end{cases}, \quad \forall \delta_j \in \Delta, \tag{13}$$

(a) Step 4. T2 advancing. (b) Step 8. T3 advancing. (c) Step 11. Circle formation.

Fig. 5. Cooperating with role division. HMARL algorithm controls the green team. The moving direction of agents is represented by circles and red arrows nearby each agent in figures.

where V_δ distinguishes frontier policies (from past policies) with constant $\Omega_{\max} = 1$ for all $\delta_j \neq \delta_{\text{sel}}$. And past policies are represented with the metric score Ω of the policy group they are sampled from. Using this method, V_δ can effectively illustrate the contrast between frontier policies and past policies, since $\Omega[\pi_{\text{sel}}(\delta)] < 1$ unless agents achieve the maximum win rate or average reward (indicating the completion of training).

Specially, we also need to consider the hyper-network input for past policies. As mentioned previously, all past policies are duplications of the frontier policies at a certain stage. Therefore, past policies are always trained under the circumstance where they observe themselves as the frontier. For consistency, past policies still need to observe themselves as frontiers (with $V_\delta = 1$) even after they become a league member. Concretely, differing from F_h, past policies eventually use $\hat{F}_h$ as the hyper-network input:

$$\hat{F}_h(\delta_j, \boldsymbol{\pi}_{\text{mix}}) = \text{concat}\left[\ [1, 1, \ldots, 1],\ F_\delta(\delta_j)\ \right], \qquad (14)$$

Finally, the hyper-network produces a vector of parameters θ_h:

$$\theta_h = \text{HyperNet}[F_h(\delta_j, \boldsymbol{\pi}_{\text{mix}})], \qquad (15)$$

The output vector θ_h is split and then reshaped into the weight matrix and bias vector, which function as parameters of policy MLP layers. The hyper-network, together with the rest of the actor network, is trained in an unsupervised manner with the policy gradient.

5. EXPERIMENTS

5.1 Heterogeneous Benchmark Environment.

The HLT algorithm was implemented and tested within a heterogeneous simulation environment which we designed and named as Unreal-based Hybrid Multiagent Platform (UHMP). As shown in Fig.4a and Fig.4b, we utilize a behavior-tree-based controller to play the opponent of our HLT learner. Two teams share symmetrical configurations. We use a tag 2u-4m-4k to represent this configuration, which indicates the type and number of heterogeneous agents in each team:

Each team comprises two supportive UAVs (δ_1, 2u), which have the ability to attack, repair allies, and mitigate damage from opponents, as well as eight ground units. Half of the ground units (δ_2, 4m) are armed with missile weapons that have a longer attack range, while the other half (δ_3, 4k) are equipped with kinetic weapons that have shorter attack ranges and are incapable of attacking opponent UAVs. $\Delta = \{\delta_1, \delta_2, \delta_3\}$ and $|\Delta| = 3$.

The functionalities of different agents in UHMP are also illustrated in Fig.4b and Fig.4b. Due to the ability difference, agents belonging to one type have to learn policies that are, to an extent, unshareable to other heterogeneous types. Furthermore, due to the imparity of quantity between different types, the learning difficulty of different agent types diverges from one another.

5.2 Experimental Setup.

We run our experiments on a single Linux server with Nvidia GPUs (RTX 8000). Only one GPU was used for each experiment run. We adopt dual-clip PPO Ye et al. (2020), Adam optimizers and policy resonance Fu et al. (2022a) techniques to improve training efficiency. To control the involvement of the heterogeneous league, we set $p_f = 0.1$ for the league sampler by default, so that agents execute mixed policies with the participant of both frontier and past policies in 90% of the episodes. We set $|\mathcal{L}|_{\max} = 5$ because running a large number of past policy groups concurrently can significantly slow down the model calculation. For each optimization step, we use samples collected from $M_{em} = 128$ episodes to calculate the policy gradient. We frequently evaluate the frontier policy group with the winning rate as the metric Ω, and we average the results of 160 episodes in every evaluation for precision. The intermittent evaluation takes place once every $M_{st} = 1600$ training episodes. All sets of experiments were repeated four times with different random seeds.

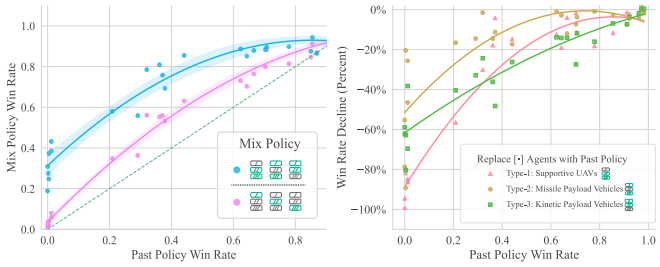
5.3 Performance Evaluation of HLT.

To test the effectiveness of the proposed HLT model, we compare the proposed HLT method with FT-Qmix (Fine-Tuned Qmix) Hu et al. (2021).

While individual observations alone suffice the requirements of our HLT model, FT-Qmix requires an extra state information vector during the centralized training stage. Determining how the additional states are obtained, we use the '-D' suffix to represent a model trained in a Delicately designed state space, or no suffix to represent a model trained in an untuned state space (or not using state information at all).

5.4 The Effectiveness Against Policy Version Iteration.

After completing the training stage, we evaluate the compatibility of the fully trained policy group (namely the eventual frontier policy group). To determine their affinity,



(a) Compatibility between eventual frontier policies and past policies.

(b) Heterogeneous role learning process when mixing the frontier and the past policies. (Frontier exclusive.)

Fig. 6. Compatibility studies.

we assess the performance of mixture policies comprising both past policies and the eventual frontier policy. In this test, we once again leverage the random combination sampler introduced previously to carry out the following tests:

(**1**) In each episode, randomly select one type of agent that executes the past policies while the types of agents execute the frontier policies. (Frontier exclusive test.)

(**2**) In each episode, randomly select one type of agent that executes the frontier policies, while the types of agents execute the past policies. (Frontier inclusive test.)

In the first test, most agents (and agent types) are controlled by the frontier policies. With the frontier policy group being the majority in the team, we can assess how the frontier works with the minority of agents controlled by past policies with various cooperation capabilities. In the second evaluation, we assessed how well the trained frontier policies could adapt in a scenario where most agents follow past policies. In this case, frontier policies controlled a minority of agents and agent types in the team, which challenged their adaptability in the absence of a dominant position.

5.5 Bottleneck Investigation in Heterogeneous Team.

An agent type can become the bottleneck of a heterogeneous team if its role is relatively more difficult to learn. Next, we demonstrate that the HLT model is capable of statistically assessing the learning process of different agent types, and revealing the bottleneck agent type during different learning phases.

Similar to the previous policy version iteration evaluation, we again mix the eventual frontier policies with past policies. A small difference is that the agent type to execute the past policy is specially assigned rather than randomly chosen. This experiment assesses the difficulty of learning diverse roles, and furthermore provides a new perspective on how cooperative heterogeneous joint policies are formed from scratch.

6. RESULTS.

6.1 Main Results.

We compared our HLT method with FT-Qmix on two main metrics, namely win rate and averaged test reward.

Table 1. Performance comparison between methods.

Algorithm	Best Test Win Rate	Best Test Reward
HLT	**98.09%**	**1.782**
FT-Qmix	87.98%	1.452
FT-Qmix-D	91.79%	1.547

Each score is averaged over 160 test episodes, and then averaged again over 4 independent experiments launched by different random seeds. According to Table.1, HLT has the best performance among all models, despite the fact that HLT does not use any state information.

6.2 Heterogeneous Cooperation Policy.

Our observations of the eventual frontier policies trained by HLT indicate a distinct division of agent roles between the different agent types. Figure 5 depicts the situation where green RL agents are poised to make contact with the enemy. The UAVs, or T1 agents, take advantage of their agility by maneuvering back and forth to attack opponents and provide support to ground teammates. In contrast, T2 agents have a longer attack range than T3 but employ a hit-and-run policy. They advance in front of T3 agents when enemies are far away but quickly retreat behind T3 when danger is imminent. T3 agents are skilled in short-range combat. However, as part of a heterogeneous team, they adopt a more restrained policy and remain stationary at the beginning of the simulation. To obtain advantages when opponents approach, T3 agents join forces with T2 agents to form a semi-circle-shaped formation.

6.3 Policy Compatibility.

Fig.6a demonstrates the win rate improvement when the frontier policy group interferes with the past policy groups. The past policy groups have different levels of cooperative ability measured using their Ω metric (policy group win rate). In this experiment, we use past policy groups satisfying $\Omega < 0.90$, which are distinguishable from the frontier policy group with $\Omega \approx 1$. We leverage two tests described in Sec 5.4. In Fig.6a, frontier-exclusive test results are represented by blue samples, and frontier-inclusive test results are shown with red samples. A significant improvement can be observed in the policy group with $\Omega \in (0.2, 0.6)$ in both tests. The participants of the eventual frontier policy promote the win rate notably above the green Ω baseline, suggesting an affinity can be established between the frontier and past policies. When $\Omega > 0.7$, the room for improvement is limited. Nonetheless, frontier policies can still provide compatibility without raising conflicts that ruin the cooperation win rate in those experiments.

6.4 Heterogeneous Role Learning.

The HLT model enables us to evaluate and monitor the learning progress of individual agent types. Fig.6b reveals the win rate decline when agents of a specific type switch to execute past policies instead of the eventual frontier. For example, we can compare the consequence of replacing the policy of T1, T2 and T3 with past policies with $\Omega \approx 0$ (bad random policies) respectively. The worst consequence occurs when we replace the policy of T1 (UAVs), which

causes the most serious consequences (reducing win rate by 90% on average). In comparison, replacing the policy of T2 had a relatively smaller impact. This suggests that well-functioning T1 agents are most indispensable in the frontier policies compared with the other two agent types. However, this result changes as the training proceeds. T1 agents quickly catch up with other agents at $\Omega \approx 0.4$ during the learning process.

On the other hand, T3 agents have the slowest Ω metric ascent among all three agent types, and usually bring the most significant win rate decline when team $\Omega > 0.4$. This indicates that T3 agents are typically bottlenecks in the training process.

7. CONCLUSIONS

This work presents a Heterogeneous League Training algorithm for addressing general heterogeneous multiagent challenges. HLT introduces a lightweight league component that is composed of diverse agent policies accumulated at various training stages, thereby promoting the robustness of agent policies by training agents with heterogeneous teammates exhibiting different cooperation skills.

We demonstrate that the HLT algorithm performs better compared to other methods in cooperative heterogeneous tasks. Additional policy compatibility experiments show that policies trained by HTL are capable of cooperating with immature past policies and promoting their win rate. Moreover, HLT can statistically assess the degree of learning difficulty for different roles in a heterogeneous team, thereby revealing the vulnerabilities of the cooperative system more effectively.

REFERENCES

Calvo, J.A. and Dusparic, I. (2018). Heterogeneous multi-agent deep reinforcement learning for traffic lights control. In *AICS*, 2–13.

Deka, A. and Sycara, K. (2021). Natural emergence of heterogeneous strategies in artificially intelligent competitive teams. In *International Conference on Swarm Intelligence*, 13–25. Springer.

Fard, E. and Selmic, R.R. (2022). Time-delayed data transmission in heterogeneous multi-agent deep reinforcement learning system. In *2022 30th Mediterranean Conference on Control and Automation (MED)*, 636–642. IEEE.

Fu, Q., Qiu, T., Yi, J., Pu, Z., Ai, X., and Yuan, W. (2022a). Solving the diffusion of responsibility problem in multiagent reinforcement learning with a policy resonance approach. *arXiv preprint arXiv:2208.07753*.

Fu, Q., Qiu, T., Yi, J., Pu, Z., and Wu, S. (2022b). Concentration network for reinforcement learning of large-scale multi-agent systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9), 9341–9349. doi:10.1609/aaai.v36i9.21165.

Ha, D., Dai, A., and Le, Q.V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Han, L., Xiong, J., Sun, P., Sun, X., Fang, M., Guo, Q., Chen, Q., Shi, T., Yu, H., Wu, X., et al. (2020). Tstarbot-x: An open-sourced and comprehensive study for efficient league training in starcraft ii full game. *arXiv preprint arXiv:2011.13729*.

Hernandez-Leal, P., Kaisers, M., Baarslag, T., and De Cote, E.M. (2017). A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*.

Hu, J., Jiang, S., Harding, S.A., Wu, H., and wei Liao, S. (2021). Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning.

Konda, V.R. and Tsitsiklis, J.N. (2000). Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, 1008–1014.

Kröse, B.J.A. (1995). Learning from delayed rewards. *Robotics Auton. Syst.*, 15(4), 233–235. doi:10.1016/0921-8890(95)00026-C.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 6382–6393.

Oliehoek, F.A. and Amato, C. (2016). *A concise introduction to decentralized POMDPs*. Springer.

Rashid, T., Farquhar, G., Peng, B., and Whiteson, S. (2020). Weighted qmix: Expanding monotonic value function factorisation. *arXiv e-prints*, arXiv–2006.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 4295–4304. PMLR.

Samvelyan, M., Rashid, T., De Witt, C.S., Farquhar, G., Nardelli, N., Rudner, T.G., Hung, C.M., Torr, P.H., Foerster, J., and Whiteson, S. (2019). The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2186–2188.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J.Z., Tuyls, K., et al. (2017). Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.

Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782), 350–354.

Ye, D., Liu, Z., Sun, M., Shi, B., Zhao, P., Wu, H., Yu, H., Yang, S., Wu, X., Guo, Q., et al. (2020). Mastering complex control in moba games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 6672–6679.

Zhao, X., Zong, Q., Tian, B., Zhang, B., and You, M. (2019). Fast task allocation for heterogeneous unmanned aerial vehicles through reinforcement learning. *Aerospace Science and Technology*, 92, 588–594. doi:10.1016/j.ast.2019.06.024.

Zheng, L., Yang, J., Cai, H., Zhou, M., Zhang, W., Wang, J., and Yu, Y. (2018). Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.