

Reducing Redundant Computation in Multi-Agent Coordination through Locally Centralized Execution

Yidong Bai

Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
ydbai@moegi.waseda.jp

Toshiharu Sugawara

Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
sugawara@waseda.jp

Abstract—In multi-agent reinforcement learning, decentralized execution is a common approach, yet it suffers from the *redundant computation* problem. This occurs when multiple agents redundantly perform the same or similar computation due to overlapping observations. To address this issue, this study introduces a novel method referred to as *locally centralized team transformer* (LCTT). LCTT establishes a *locally centralized execution* framework where selected agents serve as *leaders*, issuing *instructions*, while the rest agents, designated as *workers*, act as these instructions without activating their policy networks. For LCTT, we proposed the *team-transformer* (T-Trans) architecture that allows leaders to provide specific instructions to each worker, and the *leadership shift* mechanism that allows agents autonomously decide their roles as leaders or workers. Our experimental results demonstrate that the proposed method effectively reduces redundant computation, does not decrease reward levels, and leads to faster learning convergence.

Index Terms—Coordination, Cooperation, Multi-agent deep reinforcement learning, Redundant Computation

I. INTRODUCTION

Multi-agent deep reinforcement learning (MADRL) has made remarkable advancements recently [1], [2]. Two frameworks, *centralized training and decentralized execution* (CTDE) [3] and *centralized training and centralized execution* (CTCE) [4], are usually employed for MADRL. In the CTCE model, a centralized network processes the information from all agents and determines their joint actions. However, CTCE is practical only in limited scenarios because of the complex interference of agents' actions and the exponential growth of the joint action space with the number of agents. By contrast, agents in CTDE learn their policies using a centralized network, yet they execute decisions based on their individual observations. This makes the CTDE approach more suitable for real-world applications of multi-agent systems [5], [6].

Yet several problems with CTDE frameworks have long been overlooked. In particular, *redundant computation* is a well-known traditional problem in multi-agent systems (MASs) [7], meaning that the similar computation is done in some agents redundantly. Such redundant computations are caused by overlapping observations, resulting in the associated similar inference in different agents. As depicted in Fig. 1a, agents in *decentralized execution* (DE), in which they observe and process their local information, often have overlapping fields of observation. This overlap means that information

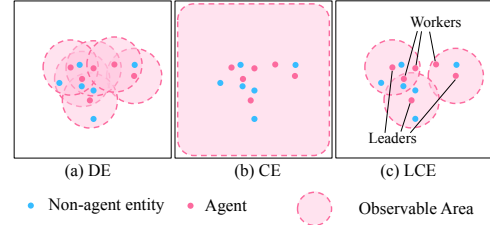


Fig. 1: Example Environments of MADRL.

about various entities in the environment, like other agents, obstacles, and objectives, is processed repeatedly by several agents as they determine their next steps, which seems as a waste of computing, especially in scenarios requiring teamwork and inter-agent communication. By contrast, this issue of redundant computation is avoided in the CTCE framework (Fig. 1b), where all observed data are summed up and provided to a central network.

Appropriate mitigation of duplicate observations can significantly reduce waste in calculations without compromising the effectiveness of MASs. Thus, a hybrid execution framework called the *locally centralized execution* (LCE) as shown in Fig. 1c, in which agents perform less overlapped observations and less redundant processes, is introduced. In conjunction with *centralized training* (CT), the LCE framework designates certain agents as *leaders*. These leaders are responsible for making decisions not only for themselves but also for other agents, termed *workers*, within their field of view. Consequently, workers do not need to process their surroundings or decide on their actions independently. Unlike models that depend on aggregating all agents' observations, the LCE approach cuts down on unnecessary computations and lowers the complexity of decision-making by operating within a significantly smaller joint action space.

We then introduce a *centralized training and localized execution* (CTLCE) framework based on LCE and propose a *localized centralized team transformer* (LCTT) to create target messages to others. Additionally, we also introduce the *redundant observation ratio* R_{da} (≥ 1) to measure the extent of redundant computation within an MAS algorithm. This ratio allows for a direct comparison of the computational expenses

between different deep learning-based methods, assuming they have a comparable number of parameters. To our knowledge, This study is the *first* attempt to address redundant computational issue in MADRL.

The proposed LCTT comprises of LCE, *team transformer* (T-Trans), and *leadership shift* (LS). LCE first establishes a cooperative framework where agents are dynamically organized into leader and worker roles, based on the directionality of instruction messages. Then, T-Trans utilizes a mechnizm similar to the attention [8] to allow leaders to dispatch specific instructions to workers under its control. Third, we employ the *leadership Q-values*, which are calculated for agents to identify their aptness as leaders. Subsequently, we propose LS, which enables a current leader to decide which agents within their observation range are best suited to assume leadership in the following time step, based on their leadership Q-values. This process allows for a fluid transition of leadership roles among agents, ensuring that leadership is always assigned to the most qualified agents throughout the interaction.

The proposed LCTT was implemented upon QMIX [9] and then trained within the CTLCE framework. We conducted its experimental evaluation in a *level-based foraging* (LBF) problem [10], [11], by comparing the performance with the baselines, *multi-agent incentive communication* (MAIC) [10] and QMIX without LCTT. We shows that LCTT achieved faster convergence with comparable rewards owing to the significant reduction of redundancy in computation.

II. RELATED WORK

In addressing challenges within multi-agent learning, one straightforward approach involves utilizing a centralized network that determines joint actions to guide the behavior of all agents. This approach is known as CTCE [12], [13]. However, as the number of agents increases, the joint action space grows exponentially, which restricts the scalability of CTCE to scenarios with many agents due to computational limitations. Another scheme is DTDE [12], in which each agent employs reinforcement learning individually to develop its policy [14]. By training policies independently, DTDE sidesteps the issue of an exponentially expanding joint action space, offering a more scalable solution for multi-agent environments.

However, from the perspective of individual agents, the challenge with DTDE lies in the non-stationary nature of the environment, which changes dynamically due to the actions and policies of other agents that also learn. This often leads to difficulties in achieving convergence. In contrast, CTDE methods [3], [9] learn a shared policy for all agents in a centralized manner while allowing agents to act based on their own local observations. Therefore, CTDE mitigates the problem of non-stationarity and reduces the complexities associated with a huge joint action space. However, DE in CTDE and DTDE encounters an issue of redundant computations. Thus, we propose LCE to mitigate redundant computations and also propose a CTLCE framework, which can be regarded as an intermediate scheme of CTCE and CTDE.

III. BACKGROUND AND PROBLEM

A. Cooperative Dec-POMDP with Communication

Similar to some conventional methods [10], [15]–[17], our model is based on a fully cooperative *decentralized partially observable Markov decision process* (Dec-POMDP) augmented with communication. Let $\mathcal{S}$ be the set of global states and $\mathcal{N} = \{1, \dots, n\}$ represent the set of n agents. We also denote $\mathcal{A}$ as the set of actions that can be executed by $\forall i \in \mathcal{N}$. Discrete time $t \geq 0$ is introduced but we often omit the subscript if t for the sake of simplicity. At every time t , agent $i \in \mathcal{N}$ in state $s \in \mathcal{S}$ obtains a local observation $o_i = O(s, i)$. Then agent i may send a message m_i based on o_i to the agents in the observable area. Concurrently, i might also receive messages from other agents; these messages are denoted as vector $\mathbf{m}_{*,i} = (m_{1,i}, \dots, m_{n,i})$ whose element $m_{j,i}$ denotes the message from j to i ($i, j \in \mathcal{N}$). We set $m_{j,i} = \text{null}$ if message did not arrive from j . Then, referring to $\mathbf{m}_{*,i}$ and o_i , i selects $a_i \in \mathcal{A}$ using its policy $\pi_i(a_i|o_i, \mathbf{m}_{*,i})$ and executes it. After that, i might receive reward $r_i(s, a_i)$ and the state transitions to the next state $s' \in \mathcal{S}$. Agent i attempts to to increase the expected value of the discounted cumulative reward $R_i = \mathbb{E}[\sum_t \gamma^t r_i(s, a_i)]$ as much as possible through learnig of π_i .

We employ DQN [18] to learn Q-function, $Q(o_i, a_i|\theta)$, that enables agents to appropriately decide their actions, where θ is the parameters of the associated deep neural network. Parameters θ is adjusted through learned to minimize the loss function $L^Q(\theta)$.

$$L^Q(\theta) = \mathbb{E}_{o_i, a_i, r_i, o'_i}[(r_i + \gamma \max_{a'_i} \bar{Q}(o'_i, a'_i|\bar{\theta}) - Q(o_i, a_i|\theta))^2], \quad (1)$$

where $\bar{Q}$ denotes the output from the target Q network parameterized by $\bar{\theta}$ which is updated with θ periodically, and a'_i and o'_i are the action and observation at the next time step,

B. Level-Based Foraging (LBF) Environment

In a LBF environment, agents are tasked with collaboratively loading and collecting food. An example snapshot of the LBF is shown in Fig. 2. At the start of the game, n agents (circles in Fig. 2) and β (> 0) foods $\{f_1, \dots, f_\beta\}$ (apples in Fig. 2) are randomly scattered in the environment. Food f_μ is assigned an association level $lv_f(f_\mu) > 0$, reflecting the challenge associated with acquiring that food f_μ . Similarly, agent i is characterized by an associated level, $lv_a(i)$, which represents the i 's capability to forage for food. These levels are expressed by the numbers on foods and agents in in Fig. 2. Agents can observe local areas shown by the lighter color shade of each agent in Fig. 2, communicate with each other, and execute the selected actions every time. This set of actions $\mathcal{A}$ includes moving in four different directions, a “none” action indicating inactivity, and a “loading” action which is used when an agent to attempt to collect food f_μ on the adjacent node.

If $lv_f(f_\mu) > lv_a(i)$, agent i is unable to collect f_μ individually and fails this attempt. For instance, agent j in Fig. 2 whose level is *one*, will fail if it tries to collect food

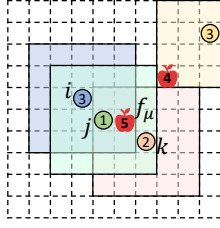


Fig. 2: Example of a level-based foraging environment.

at level of *five* on its own, hence j has to wait for other agents (e.g., k and i) to come up with the food. Then, when $\sum_{i \in \mathcal{N}_{f_\mu}} lv_a(i) \geq lv_f(f_\mu)$, i.e., the sum of the levels of the agents that try to load f_μ simultaneously is larger than or equal to $lv_f(f_\mu)$, they can collect, divide, and load it. Note that $\mathcal{N}_{f_\mu}$ represents the set of agents that executes “loading” f_μ at its neighbor nodes, simultaneously.

When agents load f_μ successfully, they receive the rewards by dividing $lv_f(f_\mu)$ by their levels. Therefore, The rewards of $i \in \mathcal{N}_{f_\mu}$ is

$$r_i(s, a_i) = lv_f(f_\mu) \cdot \frac{lv_a(i)}{\sum_{j \in \mathcal{N}_{f_\mu}} lv_a(j)},$$

which means that the reward for food and the level of that food are the same. Each agent i aims to maximize both their *individual rewards* R_i and *team rewards* $R = \sum_{i \in \mathcal{N}} R_i$. While agents collaborate to maximize R , the pursuit of maximizing R_i may not always align with team cooperation, especially if an additional agent joins the effort to collect food unnecessarily.

C. Problem Statement

Agents i , j , and k in Fig. 2 observe food f_μ and communicate to cooperatively to load it. The information in i ’s observations, containing information about f_μ , i and j , is included in j ’s observations. Likewise, the information for k ’s observations is also included in j ’s observation. This results in the repeated processing of the same information across the networks for these agents, leading to wasted use of computing resources. This issue becomes even more pronounced within the CTDE framework where agents operate under shared network parameters.

Therefore, instead of allowing i , j , and k make decisions separately, a more efficient approach would be enabling agent j , using j ’s observation and network, to coordinate the actions of i and k through communication. This method prevents the need for executing networks in agents i and k redundantly. We posit that such a mechanism can reduce the required computing cost and enhance the overall efficiency of MADRL. Our research introduces a framework in which agents autonomously determine which agents should instruct or be instructed.

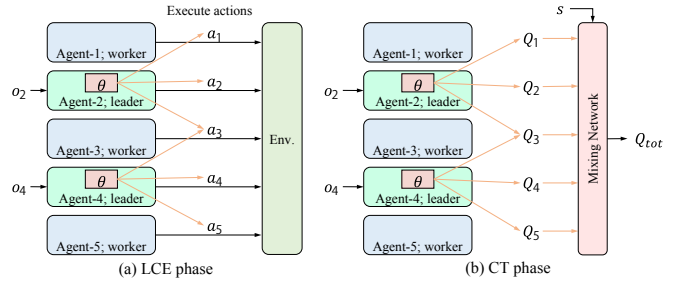


Fig. 3: Example of a CTLCE framework structure, in which Q_{tot} is the global Q-values, s is the global state, and θ is the policy network.

IV. PROPOSED METHOD

A. Redundant Observation Ratio

We propose a metric *redundant observation ratio* R_{dd} for DE to describe the extent of redundant computation in MASSs. This metric quantifies the average number of times the information pertaining to each entity is processed across agents. It is defined as follows.

$$R_{dd} = \frac{\sum_{i \in \mathcal{N}} U_i}{\sum_{e \in E} \delta(e)}, \quad (2)$$

where E is the set of entities, such as all targets, agents, and obstacles, in the environment and U_i represents the number of entities in i ’s observation. Function δ express the observability of entity $e \in E$, i.e., $\delta(e) = 1$ if an agent observes e ; otherwise, $\delta(e) = 0$.

Hence, R_{dd} indicates the extent of overlaps in their observations and thus the associated redundant computations. For instance, R_{dd} are 2.75, 1.17, and 1.0, respectively, in Figs. 1a, 1c, and 1d. Usually, $R_{dd} \geq 1$ holds. Because the observations of all agents are merged and fed to the centralized network in the CE framework (Fig. 1b), we can consider $R_{dd} = 1$.

B. Locally Centralized Execution (LCE)

First, let $l (\leq n)$ agents be leaders and the other $n-l$ agents be workers.

To mitigate redundant observations and computations, we enable leaders to construct instructions of actions for workers in their observable region including itself. The workers act as the leaders’ instructions and thus can eliminate the need to observe their surroundings (and their observations were excluded from Eq. 2). $\mathcal{W}$ and $\mathcal{L}$ are the sets of workers and leaders, respectively. The instruction message, $m_{i,k}$, from i (leader) to k (worker) encapsulates the Q value of k ’s possible actions, $Q_{i,k}(o_i, a_k | \theta)$ for $a_k \in \mathcal{A}$ and observation o_i .

An example structure of the CTLCE framework is shown in Fig. 3, in which agents are able to observe their neighboring two agents. Agents 2 and 4 are leaders and agents 1, 3 and 5 are workers. The actions of agents 1, 3 and 5 are instructed by their adjacent leaders in the LCE phase (Fig. 3a). In CT phase (Fig. 3b), the policy is trained similar to standard QMIX [9].

Algorithm 1 Locally Centralized Execution

```
1: Let  $t = 0$ . We randomly designate  $l$  agents as leaders, and
   the other  $(n - l)$  agents as workers.
2:
3: for  $t = 0, \dots, T$  do
4:   parallel for All leaders  $i \in \mathcal{L}$  do
5:     Obtain local observation  $o_i$ .
6:     parallel for Observed agents  $j$  (including  $i$ ) do
7:       Construct an instruction  $m_{i,j}$ 
8:       Send  $m_{i,j}$  to agent  $j$ 
9:     end parallel for
10:    Leadership Shift: Appoint an observed agent to be
        the leader at the  $t + 1$ .
11:   end parallel for
12:   parallel for All worker  $k \in \mathcal{W}$  do
13:     if  $k$  receives no instruction then
14:       Obtain local observation  $o_k$ .
15:       Construct an instruction for itself,  $m_{k,k}$ 
16:     end if
17:   end parallel for
18:   Calculate average values of instructions as Q-values for
   all agents  $\mathcal{N}$ 
19:   Agents act as the Q-values
20: end for
```

Algorithm 1 shows the pseudocode for LCE. If leaders are observable each other, they construct the instructions for other leaders as well as for themselves. When a agent j receives multiple instruction from various leaders, it averages these instructions to determine the Q values for its actions. In the LCE framework, the number of leaders is typically set to be $0 < l < n$; LCE with $l = 0$ represents DE without any form of communication between agents, and that with $l = n$ indicates DE where there is dense communication across all agents. Finding an appropriate value for l is crucial for the efficiency of the framework and often requires experiments and experience.

All agents within the system share the same neural network architecture, although workers typically do not activate their networks as often as leaders do. Only when a worker does not receive any instruction, the worker observes the surroundings and run its own network to obtain Q-values for itself (Line 12~17 in Alg. 1). LS (Line 10 in Alg. 1) and how leaders construct instructions (Lines 7 & 15 in Alg. 1) are explained in Sections IV-C and IV-D.

C. Team Transformer (T-Trans)

In existing research, teammate modeling [10], [16] has been employed to allow agents to send specific messages to a designated teammate. Unfortunately, this approach is unsuitable for our leaders, as workers do not (always) determine their actions using their own policies. Therefore, we introduce T-Trans so that leaders are able to generate targeted messages for their workers. T-Trans and leadership shift (which will be described below) are employed in the CTLCE framework,

which is referred to as the *locally centralized T-Trans* (LCTT), hereafter.

Fig. 4 shows the T-Trans structure. First, agent i 's observation is organized as a collection of information corresponding to each entity within its observational range: $o_i = (o_{i,1}, \dots, o_{i,i}, \dots, o_{i,|E|})$, and $o_{i,j} = \emptyset$ for any entity that falls outside of i 's observable area. Subsequently, they are passed to a *gate recurrent unit* (GRU) [19] cell (see (a) in Fig. 4) to obtain temporal information. Then they are passed to an attention-like module [8] (see (b) in Fig. 4), to represent the associations among entities. Agent's characteristics and the temporal features are concatenated, and then they forwarded through an action head for calculating Q-values. Additionally, a classifier, which is shown as (c) in Fig. 4, is applied to determine the most appropriate candidate among the agents as the leader for the next time.

In the attention-like module, the calculation of the attention feature is executed as

$$z_i = \text{softmax}\left(\frac{q_i k_i^T}{\sqrt{d_{att}}}\right) v_i, \quad (3)$$

where k_i^T represents the transpose of k_i ; z_i encapsulates a representation unique to each teammate among the agents. Then, z_i is concatenated with the duplicated hidden states and converted into teammate-specific instructions by the fully connected (FC) layer action head.

D. Leadership Shift (LS)

At the start of each episode, we randomly select l (> 0) agents as leaders (Line 1 in Alg. 1). The LS mechanism was introduced to facilitate the dynamic allocation of leadership and to ensure that instructions are sent to workers from appropriate leaders. An MLP ((c) in Fig. 4) is used by leaders to generate the hidden states and calculate the *leadership scores*, $g_i = (g_{i,1}, \dots, g_{i,n})$, where $g_{i,j}$ is the score of i 's assessment for j 's potential as a leader. Based on these scores, a leader nominates one of the agents it observes to take over as a next leader. Algorithm 2 is the pseudocode of LS for agent i .

To train the leadership scores g_i effectively, we propose the *leadership Q-value* to create pseudo-labels for g_i as follows. First, the leadership Q-value for all agent $\forall k \in \mathcal{N}$ is calculated as

$$\bar{Q}_k^L(o_k, o_j, |\bar{\theta}) = \sum_j \bar{Q}_j(o_j, \arg\max_{a_j} \bar{Q}_k(o_k, a_j | \bar{\theta}) | \bar{\theta}), \quad (4)$$

where j is an agent observable by k . Note that o_j is required to calculate $\bar{Q}_k^L$ only in the CT phase. Our method in the LCE phase does not depend on acquiring observations from other agents. Then, the pseudo-labels for g_i , denoted by g_i^* , are obtained using the onehot function:

$$g_i^* = \text{onehot}(g_{i,1}^*, \dots, g_{i,n}^*). \quad (5)$$

where $g_{i,k}^* = \bar{Q}_k^L$ if i can observe k and $g_{i,k}^* = -\infty$, otherwise. After that, g_i is learned through the minimization of the cross-entropy loss:

$$L_i^g(\theta) = H(g_i^*) + D_{KL}(g_i^* || g_i), \quad (6)$$

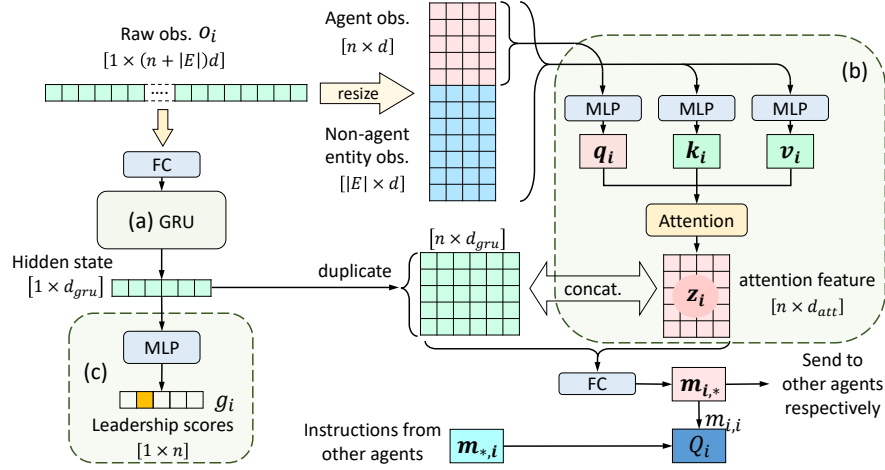


Fig. 4: T-Trans Structure. Parameters d_{att} and d_{gru} are dimensions of features of attention layers and GRU, and d is the dimension of observation of respective entities; k_i and v_i are key and value matrices whose sizes are $(|E| + n) \times d_{att}$. q_i is the $n \times d_{att}$ query matrix.

Algorithm 2 Leadership Shift (LS)

- 1: Calculate the leadership scores $g_i = (g_{i,1}, \dots, g_{i,n})$.
- 2: Choose the agent j that is best suited to be the leader based on $j = \text{argmax}_j g_{i,j}$.
- 3: **if** j is not appointed to be a leader at $t + 1$ **then**
- 4: Send a signal to j to appoint it to be a leader at $t + 1$.
- 5: Appoint i to be a *worker* at the $t + 1$.
- 6: **else**
- 7: Appoint i to be a leader at $t + 1$.
- 8: **end if**

where $D_{KL}(g_i^* || g_i)$ is the Kullback-Leibler divergence of g_i^* from g_i , and $H(g_i^*)$ is the entropy of g_i^* . Note that g_i^* is obtained from the observations at $t + 1$ as the output of the target network with $\bar{\theta}$, whereas g_i is computed using the observations of i at t from the network specified by θ .

Finally, the Q-values generated by the agents are fed into a *mixing network* [9] to combine these values effectively, and the collective learning goal for all parameters is

$$L(\theta) = \sum_{i=1}^n L_i^Q(\theta) + \lambda \sum_{i=1}^n L_i^g(\theta), \quad (7)$$

where λ is a weight hyperparameter and $L_i^Q(\theta)$ is the standard DQN loss function (Eq. 1).

V. EXPERIMENTS

We conducted an experiment to evaluate our method, LCTT, using the LBF environment, as shown in Fig. 2, where the size of the grid world is 10×10 , the number of agents $n = 4$ and the number of food $\beta = 2$. These results are then compared to those with the baselines, QMIX [9] and MAIC [10].

The level of any agent, $lv_a(i)$ is set to an integer randomly selected between 1 and 5. The level of food f_μ is also set to a random integer $1 \leq lv_f(\mu) < \sum_{i \in \mathcal{N}} lv_a(i)$. The visibility

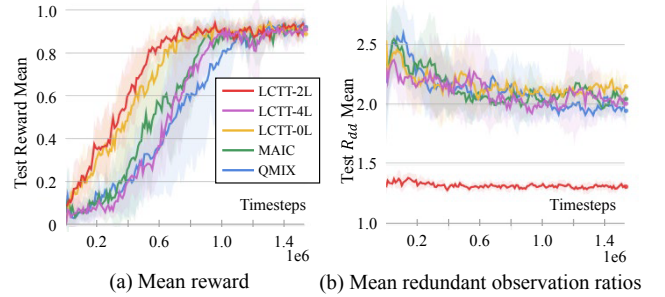


Fig. 5: Experimental results of LCTT and baselines.

for all agents was confined to a 5×5 area. The setup for our experiments matched the conditions found in the official MAIC repository.

Fig. 5 shows the experimental results, in which LCTT- l L means the LCTT with l leaders; for instance, LCTT-1L corresponds to a scenario where one leader instructs others although the leadership can shift among agents. LCTT-0L correspond to a DE setup, while LCTT-4L depicts a scenario where every agent attempts to instruct other agents.

Figure 5a displays the mean rewards obtained by LCTT and baseline methods over timestep in the test phase. While all approaches eventually reached comparable reward upon convergence, the LCTT with two leaders, denoted as LCTT-2L, achieved the fastest convergence. MAIC enables each agent to send incentive messages that can bias the value functions of other agents. In this case, MAIC can be considered a special case of LCTT in which $l = n = 4$. The faster convergence of LCTT-4L compared to MAIC could be attributed to the utilization of the T-Trans structure for message generation, rather than relying on teammate modeling. This is because T-Trans bypasses the need for explicit modeling or an ancillary loss function. QMIX is a classical CTDE method, underpinning both LCTT and MAIC, where agents decide

based solely on their observations without communication. From this perspective, QMIX represents a version of LCTT with $l = 0$, i.e., all agents are workers and make their own decisions, without using T-Trans. The finding that LCTT-0L converged faster than QMIX suggests that the T-Trans framework is more effective than conventional MLP networks in learning. For clarity, the performance curves of LCTT-1L and LCTT-3L were excluded from Fig. 5. They both converged faster than MAIC yet not as fast as LCTT-2L.

Figure 5b presents the mean values of redundant observation ratio, R_{dd} . It's clear that LCTT-2L has a significantly lower redundant observation ratio compared to other methods, suggesting that two leaders of LCTT-2L effectively instruct worker agents in collaborative tasks rather than let workers decide their own actions. For LCTT-4L, LCTT-0L, MAIC, and QMIX, where all agents are required to make decisions, the redundant observation ratios are higher than those seen in LCTT-2L. Moreover, their redundant observation ratios temporarily increased at the beginning of the learning process, reflecting the agents' realization that solo efforts are insufficient for task completion, necessitating collective action. Over time, these ratios then level off to a more stable range as agents learn the importance of spreading out to effectively scout for food sources. Unlike the others, the redundant observation ratio for LCTT-2L remains low throughout, only slightly decreasing as learning progresses, benefiting from the LCE strategy. This strategy ensures that while agents come together, the redundant observation ratio does not rise because workers fall within the leaders' instruction region and cease their individual observations; conversely, as agents spread out, the redundant observation ratio does not fall further since workers exit the leaders' instruction region and resume their observation.

VI. CONCLUSION

This research addresses the issue of redundant computation in multi-agent systems. We introduced a novel metric, the redundant observation ratio, to quantitatively assess the extent of redundant computations. Our findings suggest that reducing redundancy is feasible through a structured system of instruction construction among agents. We developed the LCTT framework, enabling agents to decide which agents should instruct and be instructed by others, and how. Through the experimental results in LBF, we demonstrated that the proposed method significantly reduces redundant computation without causing a reduction in rewards, but instead achieving faster convergence. Thus, when our method is applied to practice, it helps to save a large computational cost.

We believe that it is promising to combine our study with research on multi-agent communication, for example, establishing communication among leaders pave the way for even more significant advancements.

REFERENCES

- [1] C. Yu, X. Wang, X. Xu, M. Zhang, H. Ge, J. Ren, L. Sun, B. Chen, and G. Tan, "Distributed multiagent coordinated learning for autonomous driving in highways based on dynamic coordination graphs," *Ieee transactions on intelligent transportation systems*, vol. 21, no. 2, pp. 735–748, 2019.
- [2] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.
- [3] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Neural Information Processing Systems (NIPS)*, 2017.
- [4] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*. Springer, 2017, pp. 66–83.
- [5] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & sons, 2009.
- [6] T. Sugawara, "A cooperative lan diagnostic and observation expert system," in *1990 Ninth Annual International Phoenix Conference on Computers and Communications*. IEEE Computer Society, 1990, pp. 667–668.
- [7] E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Coherent cooperation among communicating problem solvers," *IEEE Transactions on computers*, vol. 100, no. 11, pp. 1275–1291, 1987.
- [8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, E. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [9] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 7234–7284, 2020.
- [10] L. Yuan, J. Wang, F. Zhang, C. Wang, Z. Zhang, Y. Yu, and C. Zhang, "Multi-agent incentive communication via decentralized teammate modeling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, 2022, pp. 9466–9474.
- [11] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks," *arXiv preprint arXiv:2006.07869*, 2020.
- [12] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, pp. 1–49, 2022.
- [13] L. Han, P. Sun, Y. Du, J. Xiong, Q. Wang, X. Sun, H. Liu, and T. Zhang, "Grid-wise control for multi-agent reinforcement learning in video game ai," in *International Conference on Machine Learning*, 2019, pp. 2576–2585.
- [14] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.
- [15] Z. Ding, T. Huang, and Z. Lu, "Learning individually inferred communication for multi-agent cooperation," *Advances in Neural Information Processing Systems*, vol. 33, pp. 22 069–22 079, 2020.
- [16] Y. Wang, fangwei zhong, J. Xu, and Y. Wang, "Tom2c: Target-oriented multi-agent communication and cooperation with theory of mind," in *International Conference on Learning Representations*, 2022. [Online]. Available: <https://openreview.net/forum?id=2t7CkQXNpuq>
- [17] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation," *Advances in neural information processing systems*, vol. 31, 2018.
- [18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [19] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.