Distributed Multi-Agent Reinforcement Learning with One-hop Neighbors and Compute Straggler Mitigation

Baoqian Wang, *Student Member, IEEE*, Junfei Xie, *Senior Member, IEEE*, Nikolay Atanasov, *Senior Member, IEEE*

***Abstract*— Most multi-agent reinforcement learning (MARL) methods are limited in the scale of problems they can handle. With increasing numbers of agents, the number of training iterations required to find the optimal behaviors increases exponentially due to the exponentially growing joint state and action spaces. This paper tackles this limitation by introducing a scalable MARL method called Distributed multi-Agent Reinforcement Learning with One-hop Neighbors (DARL1N). DARL1N is an off-policy actor-critic method that addresses the curse of dimensionality by restricting information exchanges among the agents to one-hop neighbors when representing value and policy functions. Each agent optimizes its value and policy functions over a one-hop neighborhood, significantly reducing the learning complexity, yet maintaining expressiveness by training with varying neighbor numbers and states. This structure allows us to formulate a distributed learning framework to further speed up the training procedure. Distributed computing systems, however, contain *straggler* compute nodes, which are slow or unresponsive due to communication bottlenecks, software or hardware problems. To mitigate the detrimental straggler effect, we introduce a novel coded distributed learning architecture, which leverages coding theory to improve the resilience of the learning system to stragglers. Comprehensive experiments show that DARL1N significantly reduces training time without sacrificing policy quality and is scalable as the number of agents increases. Moreover, the coded distributed learning architecture improves training efficiency in the presence of stragglers.**

***Index Terms*— Multi-Agent Reinforcement Learning, Scalability, Distributed Computing, Coded Computation.**

I. Introduction

Recent years have witnessed tremendous success of reinforcement learning (RL) in challenging decision making problems, such as robot control and video games. Research efforts are currently focused on multi-agent settings, including cooperative robot navigation [1], multi-player games [2], and traffic management [3]. Direct application of RL techniques in multi-agent settings by running single-agent algorithms simultaneously on each agent exhibits poor performance [4]. This is because, without considering interactions among the agents, the environment becomes non-stationary from the perspective of a single agent.

Multi-agent reinforcement learning (MARL) [5] addresses this challenge by considering all agents and their dynamics collectively when learning the value function and policy of an individual agent. Most effective MARL algorithms, such as multi-agent deep deterministic policy gradient (MADDPG) [4] and counterfactual multi-agent (COMA) [6], adopt this strategy. However, learning a joint-state value or action-value (Q) or policy function is challenging due to the exponentially growing joint state and action spaces with increasing number of agents [7], [8]. Policies trained with joint state-action pairs have poor performance in large-scale settings as demonstrated in recent work [9], [10], because their accurate approximation requires models with extremely large capacity.

MARL algorithms that improve the quality of learned policies for large-scale multi-agent settings often employ value function factorization that factorizes the global value/Q function into individual value/Q functions depending on local states and actions, e.g., as in Value Decomposition Network (VDN) [11], QMIX [12], QTran [13], mean-field MARL (MFAC) [9] or scalable actor critic (SAC) [8]. In addition to value factorization, there are several other methods proposed to enable scalable MARL. MAAC [14] uses an attention module to abstract states of other agents when training an agent's Q function, which reduces the quadratically increasing input space to a linear space. EPC [10] applies curriculum learning to gradually scale MARL up. While these methods achieve great policy performance, the training time can be significant when the number of agents increases because these methods cannot be easily trained in an efficient distributed or parallel manner over multiple computers.

To address the challenge of training policies for large numbers of agents over a distributed computing architecture, we propose a MARL algorithm called Distributed multi-Agent Reinforcement Learning with One-hop Neighbors (DARL1N). DARL1N's *main advantage* over state-of-the-art MARL methods is that it allows distributed training across compute nodes (devices with networking, storage, and computing capabilities) running in parallel, with each compute node simulating only a very small subset of the agent transitions. This is made

We gratefully acknowledge support from NSF CAREER-2048266, NSF CCF-2402689 and ONR N00014-23-1-2353.

Baoqian Wang is with the Boeing AI, The Boeing Company, Huntsville, AL 35808, USA (e-mail: `baoqian.wang@boeing.com`).

Junfei Xie is with the Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182, USA (e-mail: `jxie4@sdsu.edu`). Corresponding author.

Nikolay Atanasov is with the Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA 92093, USA (e-mail: `natanasov@ucsd.edu`).

possible by modeling the agent team topology as a proximity graph and representing the Q function and policy of each agent as a function of its one-hop neighbors only. This structure significantly reduces the representation complexity of the Q and policy functions and yet maintains expressiveness when training is done over varying states and numbers of neighbors. Furthermore, when agent interactions are restricted to one-hop neighborhoods, training an agent's Q function and policy requires transitions only of the agent itself and its potential two-hop neighbors. This enables highly efficient distributed training because each compute node needs to simulate only the transitions of the agents assigned to it and their two-hop neighbors.

RL or MARL policies can be trained over a distributed computing architecture in either a centralized [15], [16] or decentralized [17] manner. Decentralized architectures [17] offer greater resilience to node failures and malicious attacks but introduce significant communication overhead as frequent information exchanges are required. Additionally, achieving global coordination in such systems is inherently challenging: global information must either be inferred under the assumption of globally observable states, which limits its applicability, or obtained through consensus, which is difficult to achieve in large-scale systems. In contrast, centralized architectures with a central controller are more communication-efficient and facilitate easier global coordination, with communication occurring either asynchronously [18] or synchronously [19]. Asynchronous training faces multiple challenges including slow convergence, difficult debugging and analysis, and sometimes subpar quality of learned policies as learners may return stale gradients evaluated with old parameters [19]–[23]. Synchronous training is superior in these aspects but is vulnerable to computing *stragglers* [24] that are common in wireless and mobile networks. These stragglers, which are slow or unresponsive compute nodes caused by communication bottlenecks or software and hardware issues, can result in delays or failures in the training process. Coded computation [25] that employs coding theory to introduce redundant computation can mitigate computing stragglers. While extensively explored in various distributed computation problems such as matrix multiplication [25], linear inverse problems [26], convolution [27], and map reduce [28], its application for MARL remains understudied. In our previous work [16], we explored the merits of coded computation in enhancing resilience and accelerating the training of MADDPG [16] in a distributed manner. Building upon this, in this paper, we propose a coded distributed learning architecture tailored for DARL1N. Unlike the one introduced in [16], where the central controller simulates global environment interactions among all agents and sends simulation data to each learner to train an agent, in the new architecture, each learner directly simulates local environment interactions among a small set of neighboring agents during individual agent training and thus improves distributed computing efficiency and reduces communication overhead.

Contributions: The primary contribution of this paper is a new MARL algorithm called DARL1N, which employs one-hop neighborhood factorization of the value and policy functions, allowing distributed training with each compute node simulating a small number of agent transitions. DARL1N supports *highly-efficient distributed training* and generates *high-quality multi-agent policies for large agent teams*. The second contribution is a novel coded distributed learning architecture for DARL1N called Coded DARL1N, which allows individual agents to be trained by multiple compute nodes simultaneously, enabling *resilience to stragglers*. Our analysis shows that introducing redundant computations via coding theory does not introduce bias in the value and policy gradient estimates, and the training converges similarly to stochastic gradient descent-based methods. Four codes including Maximum Distance Separable (MDS), Random Sparse, Repetition, and Low Density Generator Matrix (LDGM) codes are investigated to introduce redundant computation. Moreover, we conduct comprehensive experiments comparing DARL1N with four state-of-the-art MARL methods, including MADDPG, MFAC, EPC and SAC, and evaluating their performance in different RL environments, including Ising Model, Food Collection, Grassland, Adversarial Battle, and Multi-Access Wireless Communication. We also conduct experiments to evaluate the resilience of Coded DARL1N to stragglers when trained under different coding schemes.

It is noted that DARL1N was first presented in a short conference version [29]. Differing from this version, this journal article further extends DARL1N by introducing a new coded distributed learning architecture to enhance its resilience to stragglers while also improving training efficiency. Theoretical analysis is conducted to elucidate the convergence of Coded DARL1N. Additionally, this journal undertakes a more comprehensive experimental study, encompassing not only the performance of DARL1N but also its new coded variant. It includes additional benchmarks, environments, and evaluation metrics for a more thorough assessment from various aspects.

In the rest of the paper, Sec. II formulates the MARL problem to be addressed and describes the occurrence of stragglers within distributed learning systems. The proposed DARL1N algorithm is then introduced in Sec. III, followed by the coded distributed learning architecture and different coding schemes, which are described in Sec. IV and Sec. V, respectively. Experiment results are presented in Sec. VI. Limitations and future work are discussed in Sec. VII. Finally, Sec. VIII concludes the paper.

II. Problem Statement

In MARL, M agents learn to optimize their behavior by interacting with the environment. Denote the state and action of agent $i \in [M] := \{1, \ldots, M\}$ by $s_i \in \mathcal{S}_i$ and $a_i \in \mathcal{A}_i$, respectively, where $\mathcal{S}_i$ and $\mathcal{A}_i$ are the corresponding state and action spaces. Let $\mathbf{s} := (s_1, \ldots, s_M) \in \mathcal{S} := \prod_{i\in[M]} \mathcal{S}_i$ and $\mathbf{a} := (a_1, \ldots, a_M) \in \mathcal{A} := \prod_{i\in[M]} \mathcal{A}_i$ denote the joint state and action of all agents. At time t, a joint action $\mathbf{a}(t)$ applied at state $\mathbf{s}(t)$ triggers a transition to a new state $\mathbf{s}(t+1) \in \mathcal{S}$ according to a conditional probability density function (pdf) $p(\mathbf{s}(t+1)|\mathbf{s}(t), \mathbf{a}(t))$. After each transition, each agent i receives a reward $r_i(\mathbf{s}(t), \mathbf{a}(t))$, determined by the joint state and action according to the function $r_i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. The objective of each agent i is to design a policy $\mu_i : \mathcal{S} \to \mathcal{A}_i$ to maximize the expected cumulative discounted reward (known as the *value function*):

$$V_i^{\boldsymbol{\mu}}(\mathbf{s}) := \mathbb{E}_{\substack{\mathbf{a}(t)=\boldsymbol{\mu}(\mathbf{s}(t))\\ \mathbf{s}(t)\sim p}}\left[\sum_{t=0}^{\infty}\gamma^t r_i(\mathbf{s}(t),\mathbf{a}(t)) \mid \mathbf{s}(0)=\mathbf{s}\right],$$

where $\boldsymbol{\mu} := (\mu_1, \dots, \mu_M)$ denotes the joint policy of all agents and $\gamma \in (0,1)$ is a discount factor. Alternatively, an optimal policy μ_i^* for agent i can be obtained by maximizing the *action-value (Q) function*:

$$Q_i^{\boldsymbol{\mu}}(\mathbf{s},\mathbf{a}) := \mathbb{E}_{\substack{\mathbf{a}(t)=\boldsymbol{\mu}(\mathbf{s}(t))\\ \mathbf{s}(t)\sim p}}\left[\sum_{t=0}^{\infty}\gamma^t r_i(\mathbf{s}(t),\mathbf{a}(t)) \mid \mathbf{s}(0)=\mathbf{s},\mathbf{a}(0)=\mathbf{a}\right]$$

and setting $\mu_i^*(\mathbf{s}) \in \arg\max_{a_i}\max_{\mathbf{a}_{-i}} Q_i^*(\mathbf{s},\mathbf{a})$, where $Q_i^*(\mathbf{s},\mathbf{a}) := \max_{\boldsymbol{\mu}} Q_i^{\boldsymbol{\mu}}(\mathbf{s},\mathbf{a})$ and $\mathbf{a}_{-i}$ denotes the actions of all agents except i.

To develop a distributed MARL algorithm, we impose additional structure on the MARL problem. Assume that all agents share a common state space, i.e., $\mathcal{S}_i = \mathcal{S}_j$, $\forall i,j \in [M]$ and let $\mathrm{dist} : \mathcal{S}_i \times \mathcal{S}_i \to \mathbb{R}$ be a distance metric on the state space. Note that the distance metric can also be defined over a common state subspace. However, for notation simplicity, a common state space is assumed here. Consider a proximity graph [30] that models the topology of the agent team. A d-disk proximity graph is defined as a mapping that associates the joint state $\mathbf{s} \in \mathcal{S}$ with an undirected graph $(\mathcal{V},\mathcal{E})$ such that $\mathcal{V} = \{s_1, s_2, \dots, s_M\}$ and $\mathcal{E} = \{(s_i,s_j) | \mathrm{dist}(s_i,s_j) \le d, i \ne j\}$. Define the set of *one-hop neighbors* of agent i as $\mathcal{N}_i := \{j | (s_i,s_j) \in \mathcal{E}\} \cup \{i\}$. We make the following assumption about the agents' motion.

Assumption 1: The distance between two consecutive states, $s_i(t)$ and $s_i(t+1)$, of agent i is bounded, i.e., $\mathrm{dist}(s_i(t), s_i(t+1)) \le \epsilon$, for some $\epsilon > 0$.

This assumption is satisfied in many problems where, e.g., due to physical constraints, the agent states can only change by a bounded amount in a single time step.

Define the *potential neighbors* of agent i at time t as $\mathcal{P}_i(t) := \{j | \mathrm{dist}(s_j(t), s_i(t)) \le 2\epsilon + d\}$, which captures the set of agents that may become one-hop neighbors of agent i at time $t+1$. Denote the joint state and action of the one-hop neighbors of agent i by $\mathbf{s}_{\mathcal{N}_i} = (s_{j_1}, \dots, s_{j_{|\mathcal{N}_i|}})$ and $\mathbf{a}_{\mathcal{N}_i} = (a_{j_1}, \dots, a_{j_{|\mathcal{N}_i|}})$, respectively, where $j_1, \dots, j_{|\mathcal{N}_i|} \in \mathcal{N}_i$. Our key idea is to let agent i's policy, $a_i = \mu_i(\mathbf{s}_{\mathcal{N}_i})$, only depend on the one-hop neighbor states $\mathbf{s}_{\mathcal{N}_i}$ instead of all agent states $\mathbf{s}$, and we assume that each agent can obtain its one-hop neighbor states through observation or communication. The intuition is that agents that are far away from agent i at time t have little impact on its current action $a_i(t)$. To emphasize that the output of a function $f : \prod_{i\in[M]} \mathcal{S}_i \mapsto \mathbb{R}$ is affected only by a subset $\mathcal{N} \subseteq [M]$ of the input dimensions, we use the notation $f(\mathbf{s}) = f(\mathbf{s}_{\mathcal{N}})$ for $\mathbf{s} \in \mathcal{S}$ and $\mathbf{s}_{\mathcal{N}} \in \prod_{i\in\mathcal{N}} \mathcal{S}_i$ in the remainder of the paper. We make two additional assumptions on the problem structure to ensure the validity of our policy model.

Assumption 2: The reward of agent i can be fully specified using its one-hop neighbor states $\mathbf{s}_{\mathcal{N}_i}$ and actions $\mathbf{a}_{\mathcal{N}_i}$, i.e., $r_i(\mathbf{s},\mathbf{a}) = r_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$.

This assumption can always be satisfied by setting d to the full environment range. In this case, the one-hop neighbor reward assumption becomes the standard reward definition, which depends on states and actions of all agents and is applicable to general MARL problems. For environments with local reward models, a smaller distance value d can be chosen based on the specific environment configuration. For example, in collision avoidance problems, an agent's reward may depend only on the states and actions of nearby agents that maintain a safe distance. In multi-agent networks or sensing problems, the one-hop neighbors can be those within communication or observation range. Next, we make a similar assumption for agent i's transition model.

Assumption 3: The transition model of agent i depends only on $\mathbf{a}_{\mathcal{N}_i}$ and states $\mathbf{s}_{\mathcal{N}_i}$, i.e., $p_i\left(s_i(t+1) \mid \mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right)$.

The objective of each agent i is to obtain an optimal policy μ_i^* by solving the following problem:

$$\mu_i^*(\mathbf{s}_{\mathcal{N}_i}) = \arg\max_{a_i}\max_{\mathbf{a}_{-i}} Q_i^*(\mathbf{s},\mathbf{a}), \tag{1}$$

where $Q_i^*(\mathbf{s},\mathbf{a}) := \max_{\boldsymbol{\mu}} Q_i^{\boldsymbol{\mu}}(\mathbf{s},\mathbf{a})$ is the optimal action-value (Q) function introduced in the previous section.

The goal of this paper is to develop a MARL algorithm that (i) utilizes policy and value representations that scale favorably with the number of agents M and (ii) allows efficient training on a distributed computing system containing compute stragglers.

III. Distributed multi-Agent Reinforcement Learning with One-hop Neighbors (DARL1N)

This section develops the DARL1N algorithm to solve the MARL problem with proximity-graph structure introduced in Sec. II. DARL1N considers the effect of the one-hop neighbors of an agent in representing its Q and policy functions, which allows updating the Q and policy function parameters using only local one-hop neighborhood transitions.

Specifically, the Q function of each agent i can be expressed as a function of its one-hop neighbor states $\mathbf{s}_{\mathcal{N}_i}$ and actions $\mathbf{a}_{\mathcal{N}_i}$ as well as the states $\mathbf{s}_{\mathcal{N}_i^-}$ and actions $\mathbf{a}_{\mathcal{N}_i^-}$ of the remaining agents that are not immediate neighbors of i:

$$Q_i^{\boldsymbol{\mu}}(\mathbf{s},\mathbf{a}) = Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}). \tag{2}$$

Inspired by the SAC algorithm [8], we approximate the Q value with a function $\tilde{Q}_i^{\boldsymbol{\mu}}$ that depends only on one-hop neighbor states and actions:

$$\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) = \sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} w_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}) Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-})$$

where the weights $w_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}) > 0$ satisfy $\sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} w_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}) = 1$. The approximation error is given in the following lemma.

Lemma 1: If the absolute value of agent i's reward is upper bounded as $|r_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})| \le \bar{r}$, for some $\bar{r} > 0$, the approximation error between $\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ and $Q_i^{\boldsymbol{\mu}}(\mathbf{s},\mathbf{a})$ is bounded as:

$$|\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) - Q_i^{\boldsymbol{\mu}}(\mathbf{s},\mathbf{a})| \le \frac{2\bar{r}\gamma}{1-\gamma}.$$

Proof: See Appendix A. ■

We parameterize the approximated Q function $\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ and the policy $\mu_i(\mathbf{s}_{\mathcal{N}_i})$ by θ_i and ϕ_i, respectively. To handle the varying sizes of $\mathbf{s}_{\mathcal{N}_i}$ and $\mathbf{a}_{\mathcal{N}_i}$, in the implementation, we set the input dimension of $\tilde{Q}_i^{\boldsymbol{\mu}}$ to the largest possible dimension of $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$, and apply zero-padding for agents that are not within the one-hop neighborhood of agent i. The same procedure is applied to represent $\mu_i(\mathbf{s}_{\mathcal{N}_i})$.

To learn the approximated Q function $\tilde{Q}_i^{\boldsymbol{\mu}}$, instead of incremental on-policy updates to the Q function as in SAC [8], we apply off-policy temporal-difference learning with a buffer similar to MADDPG [4]. The parameters θ_i of the approximated Q function are updated by minimizing the following temporal difference error:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}'_l}\}_{\forall l \in \mathcal{N}'_i}) \sim \mathcal{D}_i}\left[\left(\tilde{Q}_i^{\boldsymbol{\mu}}\left(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}\right) - y\right)^2\right]$$
$$y = r_i + \gamma \hat{Q}_i^{\hat{\boldsymbol{\mu}}}\left(\mathbf{s}_{\mathcal{N}'_i}, \mathbf{a}_{\mathcal{N}'_i}\right) \tag{3}$$

where $\mathcal{D}_i$ is a replay buffer for agent i that contains information only from $\mathcal{N}_i$ and $\mathcal{N}'_i$, the one-hop neighbors of agent i at the current and next time steps, and the one-hop neighbors $\mathcal{N}'_l$ for $l \in \mathcal{N}'_i$. Data from the one-hop neighbors of the next-step one-hop neighbors $\mathcal{N}'_i$ are needed to compute the next-step one-hop neighbors actions $\mathbf{a}_{\mathcal{N}'_i}$. To stabilize the training, a target Q function $\hat{Q}_i^{\hat{\boldsymbol{\mu}}}$ with parameters $\hat{\theta}_i$ and a target policy function $\hat{\mu}_i$ with parameters $\hat{\phi}_i$ are used. The parameters $\hat{\theta}_i$ and $\hat{\phi}_i$ are updated using Polyak averaging: $\hat{\theta}_i = \tau\hat{\theta}_i + (1-\tau)\theta_i, \hat{\phi}_i = \tau\hat{\phi}_i + (1-\tau)\phi_i$ where τ is a hyperparameter. In contrast to MADDPG [4], the replay buffer $\mathcal{D}_i$ for agent i only needs to store its local interactions $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}'_l}\}_{\forall l \in \mathcal{N}'_i})$ with nearby agents, where $\{\mathbf{s}_{\mathcal{N}'_l}\}_{\forall l \in \mathcal{N}'_i}$ is required to calculate $\mathbf{a}_{\mathcal{N}'_i}$. Also, in contrast to SAC [8], each agent i only needs to collect its own training data by simulating local two-hop interactions, which reduces agents' experience correlations and allows efficient distributed training as explained in Sec. IV.

Agent i's policy parameters ϕ_i are updated using a gradient

$$\mathbf{g}(\phi_i) = \mathbb{E}_{\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i} \sim \mathcal{D}_i}\left[\nabla_{\phi_i}\mu_i\left(\mathbf{s}_{\mathcal{N}_i}\right)\nabla_{a_i}\tilde{Q}_i^{\boldsymbol{\mu}}\left(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}\right)\right], \tag{4}$$

where again data $\mathcal{D}_i$ only from local interactions is needed.

To implement the parameter updates proposed above, agent i needs training data $\mathcal{D}_i = (\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}'_l}\}_{l \in \mathcal{N}'_i})$ from its one-hop neighbors at the current and next time steps. The relation between one-hop neighbors at the current and next time steps is captured by the following proposition.

Proposition 1: Under Assumption 1, if an agent j is not a potential neighbor of agent i at time t, i.e., $j \notin \mathcal{P}_i(t)$, it will not be a one-hop neighbor of agent i at time $t+1$, i.e., $j \notin \mathcal{N}_i(t+1)$.

Proof: See Appendix B. ■

Proposition 1 allows us to decouple the global interactions among agents and limit the necessary observations to be among one-hop neighbors. To collect training data, at each time step, agent i first interacts with its one-hop neighbors to obtain their states $\mathbf{s}_{\mathcal{N}_i}$ and actions $\mathbf{a}_{\mathcal{N}_i}$, and compute its reward $r_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$. To obtain $\mathbf{s}_{\mathcal{N}'_l}$ for all $l \in \mathcal{N}'_i$, we first determine agent i's one-hop neighbors at the next time step, $\mathcal{N}'_i$. Using Proposition 1, we let each potential neighbor $k \in \mathcal{P}_i$ perform a transition to a new state $s'_k \sim p_k(\cdot|\mathbf{s}_{\mathcal{N}_k}, \mathbf{a}_{\mathcal{N}_k})$, which is sufficient to determine $\mathcal{N}'_i$. Then, we let the potential neighbors $\mathcal{P}_l$ of each new neighbor $l \in \mathcal{N}'_i$ perform transitions to determine $\mathcal{N}'_l$ and obtain $\mathbf{s}_{\mathcal{N}'_l}$.

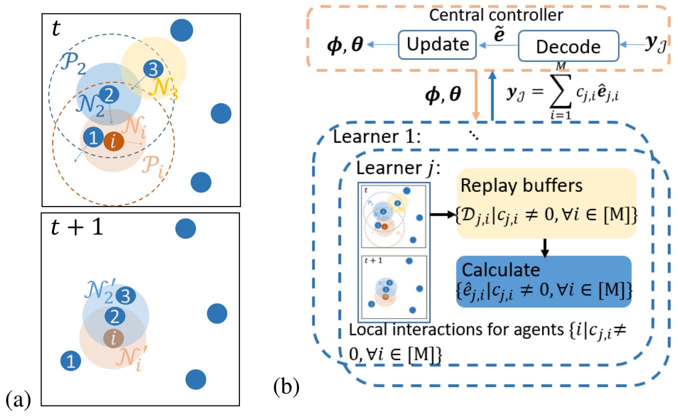

t
$\mathcal{P}_2$
$\mathcal{N}_2$
$\mathcal{N}_3$
$\mathcal{N}_i$
$\mathcal{P}_i$
1
2
3
i
t + 1
$\mathcal{N}'_2$
$\mathcal{N}'_i$
(a)
(b)
Central controller
$\boldsymbol{\phi}, \boldsymbol{\theta}$
Update
$\tilde{\boldsymbol{e}}$
Decode
$\boldsymbol{y}_{\mathcal{J}}$
$\boldsymbol{y}_{\mathcal{J}} = \sum_{i=1}^{M} c_{j,i}\hat{\boldsymbol{e}}_{j,i}$
Learner 1:
Learner j:
Replay buffers
$\{\mathcal{D}_{j,i}|c_{j,i} \neq 0, \forall i \in [M]\}$
Calculate
$\{\hat{e}_{j,i}|c_{j,i} \neq 0, \forall i \in [M]\}$
Local interactions for agents $\{i|c_{j,i} \neq 0, \forall i \in [M]\}$

Fig. 1: (a) One-hop neighbor transitions from one time step to the next in a d-disk proximity graph; (b) Coded distributed learning architecture.

Fig. 1(a) illustrates the data collection process. At time t, agent i obtains $\mathbf{s}_{\mathcal{N}_i}$, $\mathbf{a}_{\mathcal{N}_i}$, and $r_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ for $\mathcal{N}_i = \{i, 1\}$. Then, the potential neighbors of agent i, $\mathcal{P}_i = \{1, 2, i\}$, proceed to their next states at time $t+1$. This is sufficient to determine that $\mathcal{N}'_i = \{i, 2\}$ and obtain $\mathbf{s}_{\mathcal{N}'_i}$. Finally, we let agent 3, which belongs to set $\mathcal{P}_2 = \{i, 1, 2, 3\}$, perform a transition to determine that $\mathcal{N}'_2 = \{i, 2, 3\}$ and obtain $\mathbf{s}_{\mathcal{N}'_2}$.

As each agent only needs to interact with one-hop neighbors to update its parameters, the agents can be trained in parallel on a distributed computing architecture, where each compute node only needs to simulate the two-hop neighbor transitions for agents assigned to it for training.

IV. Coded Distributed Learning Architecture

In this section, we introduce an efficient and resilient distributed learning architecture for training DARL1N. A coded distributed learning architecture, illustrated in Fig. 1(b), consists of a central controller and N compute nodes, called *learners*. The central controller stores a copy of all parameters of the policy ϕ_i, target policy $\hat{\phi}_i$, Q function θ_i, and target Q function $\hat{\theta}_i$, for all $i \in [M]$. In each training iteration, the central controller broadcasts all agents' parameters to all learners, who then calculate and return the gradients required for updating the parameters. In a traditional uncoded distributed learning architecture, each agent is only trained (with its policy and value gradients computed) by a single learner. If any learner becomes slow or unresponsive, i.e., a straggler, the whole training procedure is delayed or may fail. Our coded distributed learning architecture addresses the possible presence of stragglers in the computing system by introducing redundant computations. We let more than one learner train each agent, which not only improves the system resilience to stragglers but also accelerates the training speed, as we show in Sec. VI-B. To describe which learners are assigned to train each agent, we introduce an assignment matrix $\mathbf{C} \in \mathbb{R}^{N \times M}$ with non-zero entries $c_{j,i} \neq 0$

indicating that learner $j \in [N]$ is assigned to train agent $i \in [M]$. The complete set of learners assigned to train an agent i can then be determined by $\{j|c_{j,i} \neq 0, \forall j \in [N]\}$. To construct the assignment matrix $\mathbf{C}$, we apply coding theory as explained in Sec.V.

To calculate the gradients for an agent i, each learner j with $c_{j,i} \neq 0$ simulates transitions to get the interaction data $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}'_l}\}_{l\in\mathcal{N}'_i})$ as described in Sec. III, which are stored in a replay buffer $\mathcal{D}_{j,i}$. After that, learner j calculates the gradients of the temporal difference error needed for updating the Q function parameters θ_i of agent i using (3) and updating the policy parameters ϕ_i using (4).

As the replay buffer $\mathcal{D}_{j,i}$ can have a large size, to improve efficiency, we use a mini-batch $\mathcal{B}_{j,i}$ uniformly sampled from $\mathcal{D}_{j,i}$ to estimate the expectations in (3)-(4). In particular, the temporal difference error in (3) is estimated with:

$$\hat{\mathcal{L}}_j(\theta_i) = \frac{1}{|\mathcal{B}_{j,i}|} \sum_{\substack{(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}'_l}\}_{\forall l \in \mathcal{N}'_i}) \\ \in \mathcal{B}_{j,i}}} \left(\tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) - y\right)^2$$
$$y = r_i + \gamma \hat{Q}_i^{\hat{\boldsymbol{\mu}}}\left(\mathbf{s}_{\mathcal{N}'_i}, \mathbf{a}_{\mathcal{N}'_i}\right). \tag{5}$$

Similarly, the gradients used to update policy parameters are estimated with:

$$\hat{\mathbf{g}}_j(\phi_i) = \frac{1}{|\mathcal{B}_{j,i}|} \sum_{(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) \in \mathcal{B}_{j,i}} \nabla_{\phi_i}\mu_i(\mathbf{s}_{\mathcal{N}_i}) \nabla_{a_i} \tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}). \tag{6}$$

Let $\hat{\mathbf{e}}_{j,i} = [\nabla\hat{\mathcal{L}}_j(\theta_i), \hat{\mathbf{g}}_j(\phi_i)]$ denote the concatenation of estimated gradients. Instead of directly returning the estimated gradients for all agents trained by learner j, i.e., $\{\hat{\mathbf{e}}_{j,i}|\forall i \in [M], c_{j,i} \neq 0\}$, learner j calculates a linear combination of the gradients: $y_j = \sum_{i=1}^{M} c_{j,i}\hat{\mathbf{e}}_{j,i}$ with weights provided by the assignment matrix $\mathbf{C}$ and returns y_j back to the central controller.

At the central controller, let $\mathbf{y}_{\mathcal{J}}$ denote the results that have arrived by a certain time from learners $\mathcal{J} = \{j|y_j \text{ is received}\}$. Moreover, let $\mathbf{C}_{\mathcal{J}} \in \mathbb{R}^{|\mathcal{J}|\times M}$ be a submatrix of $\mathbf{C}$ formed by the j-th row of $\mathbf{C}, \forall j \in \mathcal{J}$. The received gradients $\mathbf{y}_{\mathcal{J}}$ satisfy:

$$\mathbf{y}_{\mathcal{J}} = \mathbf{D}\mathbf{q}$$
$$\mathbf{q} = [\hat{\mathbf{e}}_{1,1}; \hat{\mathbf{e}}_{1,2}; \ldots; \hat{\mathbf{e}}_{N,M-1}; \hat{\mathbf{e}}_{N,M}] \tag{7}$$

where $\mathbf{D} \in \mathbb{R}^{|\mathcal{J}|\times MN}$ is constructed as follows: for i-th row of $\mathbf{D}$, fill the $(1 + (\mathcal{J}_i - 1)M)$-th to $(\mathcal{J}_i M)$-th entries with i-th row of $\mathbf{C}_{\mathcal{J}}$ and set all other entries to 0, $\mathcal{J}_i$ denotes i-th element of $\mathcal{J}$. The vector $\mathbf{q}$ is a concatenation of all the gradients estimated by all learners. The central controller updates the agents' parameters once it receives enough results to decode all estimated gradients, denoted as $\tilde{\mathbf{e}}$. This happens when $\text{rank}(\mathbf{C}_{\mathcal{J}}) = M$, and the decoding equation is given as follows:

$$\tilde{\mathbf{e}} = (\mathbf{C}_{\mathcal{J}}^T\mathbf{C}_{\mathcal{J}})^{-1}\mathbf{C}_{\mathcal{J}}^T\mathbf{y}_{\mathcal{J}}. \tag{8}$$

Alg. 1 summarizes the coded training procedure of DARL1N over a distributed computing architecture, referred to as the Coded DARL1N.

In Coded DARL1N, the gradients $\tilde{\mathbf{e}}$ used by the central controller for parameter updates are stochastic gradients computed by mini-batch samples, which are estimates of the true gradients $\mathbf{e} = [\mathbf{e}_1, \ldots, \mathbf{e}_M]$, where $\mathbf{e}_i = [\nabla\mathcal{L}(\theta_i), \mathbf{g}(\phi_i)]$ with $\mathcal{L}(\theta_i)$ and $\mathbf{g}(\phi_i)$ defined in (4) and (3), respectively. The estimation performance is illustrated in the following theorem.

Algorithm 1: Coded DARL1N

```
   // Central controller:
1  Initialize policy, target policy, Q, and target Q parameters
     $\boldsymbol{\phi} = \{\phi_i, \hat{\phi}_i\}_{i\in[M]}$, $\boldsymbol{\theta} = \{\theta_i, \hat{\theta}_i\}_{i\in[M]}$;
2  Broadcast $\boldsymbol{\phi}, \boldsymbol{\theta}$ to the learners;
3  $\mathbf{y}_{\mathcal{J}} \leftarrow [\,]$;
4  do
5      Listen to channel and collect $y_j$ from the learners:
         $\mathbf{y}_{\mathcal{J}} \leftarrow [\mathbf{y}_{\mathcal{J}}, y_j], j \in [N]$;
6  while $\tilde{\mathbf{e}}$ is not recoverable;
7  Send acknowledgements to learners;
8  Update $\boldsymbol{\phi}, \boldsymbol{\theta}$ with $\tilde{\mathbf{e}}$;
   // Learner $j$:
9  Initialize replay buffer $\mathcal{D}_{j,i}$;
10 for $iter = 1 : max\_iteration$ do
11     Listen to channel;
12     if $\boldsymbol{\phi}, \boldsymbol{\theta}$ received from the central controller then
13         $y_j \leftarrow 0$; $i \leftarrow 1$;
14         while $i \leq M$ and no acknowledgement received do
15             if $c_{j,i} \neq 0$ then
                   // Local interactions:
16                 Perform local interactions to collect training
                     data for agent $i$ and store the data into $\mathcal{D}_{j,i}$;
17                 Sample a mini-batch from $\mathcal{D}_{j,i}$ and calculate
                     $\hat{\mathbf{e}}_{j,i}$ using (5)-(6);
18             $y_j \leftarrow y_j + c_{j,i}\hat{\mathbf{e}}_{j,i}$;
19             $i \leftarrow i + 1$;
20             Send updated $y_j$ to the central controller;
```

Theorem 1: The mini-batch stochastic gradients $\tilde{\mathbf{e}}$ computed by Coded DARL1N are unbiased estimates of the true gradients $\mathbf{e}$, with variance determined by the assignment matrix $\mathbf{C}$.

Proof: See Appendix C. ■

Based on Theorem 1, we can infer that Coded DARL1N converges asymptotically similarly to other stochastic gradient descent-based methods [31].

V. Assignment Matrix Construction and Assessment

The assignment matrix $\mathbf{C}$ affects both the policy variance and computational efficiency of Coded DARL1N. In this section, we explore different schemes, both uncoded and coded, for constructing the assignment matrix, and conduct theoretical analyses on their performance.

A. Assignment Matrix Construction

1) Uncoded Assignment Scheme: In an uncoded distributed training architecture, different learner nodes train different agents exclusively. The assignment matrix can then be constructed as: $\mathbf{C}^{\text{Uncoded}} = [\mathbf{I}_M|\mathbf{0}]^T$, where $\mathbf{I}_M \in \mathbb{R}^{M\times M}$ is an identity matrix.

2) Coded Assignment Schemes: Coded distributed training assigns each agent to multiple learners. Here, we investigate five codes, where the encoding matrices can be directly utilized as the assignment matrix.

- *MDS Code*: An MDS code [32] is an erasure code with the property that any square submatrix of its encoding matrix $\mathbf{C}^{\text{MDS}}$ has full rank. A Vandermonde matrix [16], [33] is commonly used for encoding, with the (j,i)-th entry given by $\mathbf{C}_{j,i} = \alpha_i^{j-1}$, where $\alpha_i \neq 0$, $i \in [M]$, can be any non-zero distinct real numbers.
- *Random Sparse Code*: Compared to an MDS code, a Random Sparse code [34] results in a sparser assignment matrix with the (j,i)-th entry given by:

$$\mathbf{C}_{j,i}^{\text{Random}} = \begin{cases} 0, & \text{with probability } 1-\xi, \\ \zeta, & \text{with probability } \xi. \end{cases} \tag{9}$$

where $\zeta \sim \mathcal{N}(0,1)$, $\xi \in [0,1]$.
- *Repetition Code*: A Repetition code [34] assigns agents to the learners repetitively in a round-robin fashion. The (j,i)-th entry of the assignment matrix is given by:

$$\mathbf{C}_{j,i}^{\text{Repetition}} = \begin{cases} 1, & \text{if } i = (j \bmod M), \\ 0, & \text{else}, \end{cases} \tag{10}$$

where mod is the modulo operator.
- *LDGM Code*: An LDGM code [35] is a special type of a Low Density Parity Check code [36] that constructs a sparser assignment matrix. By applying a systematic biased random code ensemble [35], the LDGM assignment matrix takes the form: $\mathbf{C}^{\text{LDGM}} = [\mathbf{I}_M | \hat{\mathbf{P}}]^T$, where each entry of $\hat{\mathbf{P}}$ is generated independently according to a Bernoulli distribution with success probability $\Pr(\hat{\mathbf{P}}_{i,j} = 1) = \rho$. Note that when $\rho \leq \frac{1}{2}$, the assignment matrix of LDGM code has a low density.

B. Analysis and Comparison of Assignment Schemes

We provide a theoretical analysis and comparison of different assignment schemes from the following aspects: 1) *computation overhead* and 2) *resilience to stragglers*.

1) Computation Overhead: The coded schemes mitigate the impact of stragglers by assigning each agent to multiple learners. The training performed by the extra learners is redundant. To quantify the *computation overhead* introduced by this redundancy, we use the following metric:

$$o_c = \frac{1}{M} \sum_{j=1}^{N} \sum_{i=1}^{M} \mathbb{1}_{\mathbf{C}_{j,i} \neq 0} - 1,$$

where the first term on the right hand side calculates the average number of learners used for training each agent, and $o_c \geq 0$. Using the above metric, the computation overhead of each assignment scheme can be derived as follows:

- *Uncoded*: $o_c^{\text{Uncoded}} = 0$, as each agent is assigned to only one learner in the uncoded scheme.
- *MDS Code*: $o_c^{\text{MDS}} = N - 1$. All entries of the MDS assignment matrix are non-zero, indicating that each agent is assigned to all learners.
- *Random Sparse Code*: o_c^{Random} depends on the parameter ξ, but its expectation is derived as $\mathbb{E}(o_c^{\text{Random}}) = \xi N - 1$.
- *Repetition Code*: $o_c^{\text{Repetition}} = \frac{N}{M} - 1$.
- *LDGM Code*: $\mathbb{E}(o_c^{\text{LDGM}}) = (N - M)\rho$.

Among these schemes, the MDS code incurs the highest computation overhead, while the uncoded scheme results in the lowest. The overhead introduced by the Random Sparse and LDGM codes depend on their parameters, ξ and ρ.

2) Resilience to Stragglers: According to (8), the central controller can update the agents' gradients only after receiving results from enough learners, specifically when $\text{rank}(\mathbf{C}_{\mathcal{J}}) = M$. To evaluate the resilience of assignment schemes to stragglers, we analyze the probability of each scheme being influenced by stragglers under the following assumption.

Assumption 4: In each training iteration, each compute node in a distributed computing system has a probability of $\eta \in [0,1]$ to become a straggler.

The Random Sparse and LDGM codes have randomly generated entries that depend on parameters ξ and ρ, making them hard to analyze theoretically. We focus our analysis on the Uncoded, MDS, and Repetition codes as follows.

Proposition 2: The probability that the Uncoded scheme will be influenced by stragglers is $1 - (1-\eta)^M$.

Proposition 3: The probability that the MDS code will be influenced by stragglers is $\sum_{j=N-M+1}^{N} \binom{N}{j} (1-\eta)^{N-j} \eta^j$.

Proposition 4: The probability that the Repetition code will be influenced by stragglers is $1 - (1 - \eta^{\frac{N}{M}})^M$ given that $\frac{N}{M}$ is positive integer.

The proof of Proposition 2 is direct since the Uncoded scheme is affected by any straggler. The proofs of Propositions 3 and 4 are provided in Appendices D and E, respectively.

VI. Experiments

In this section, we evaluate the DARL1N algorithm and our coding schemes for mitigating compute stragglers.

A. Performance of DARL1N

We conduct a series of comparisons between DARL1N and four state-of-the-art MARL algorithms. For fair comparison, we train DARL1N using a distributed learning architecture with uncoded assignments, and run our experiments on the Amazon EC2 computing clusters [37], which are considered reliable and free of stragglers.

1) Experiment Settings:

a) Environment Configurations: We evaluate DARL1N in five environments, including the Ising Model [9], Food Collection, Grassland, Adversarial Battle [10], and Multi-Access Wireless Communication [8], which cover cooperative and mixed cooperative competitive games. Please refer to [8]–[10] for the description of each environment.

To understand the scalability of our method, we vary the number of agents M and the size of the local state spaces. The specific configurations for the first four environments can be referenced in the conference version [29]. In the Multi-Access Wireless Communication environment, which was not considered in [29], we adopt the setting in [8] and consider a grid of 3×3 agents, with each having a state space of $\mathcal{S}_i = \{0,1\}^z$ to indicate whether there is a packet to send by time step z, where z is set to either $z = 2$ or $z = 10$.

TABLE I: Convergence time and convergence reward of different methods in the Ising Model environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 9$	$M = 16$	$M = 25$	$M = 64$	$M = 9$	$M = 16$	$M = 25$	$M = 64$
MADDPG	62	263	810	1996	460	819	**1280**	1831
MFAC	63	274	851	2003	**468**	814	1276	1751
EPC	101	**26**	**51**	**62**	**468**	**831**	1278	**3321**
EPC Scratch	101	412	993	2995	**468**	826	1275	2503
DARL1N	**38**	102	210	110	465	828	1279	2282

TABLE II: Convergence time and convergence reward of different methods in the Food Collection environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 3$	$M = 6$	$M = 12$	$M = 24$	$M = 3$	$M = 6$	$M = 12$	$M = 24$
MADDPG	**501**	1102	4883	2005	24	24	-112	-364
MFAC	512	832	4924	2013	20	23	-115	-362
EPC	1314	723	2900	8104	**31**	**34**	-16	-87
DARL1N	502	**382**	**310**	**1830**	14	25	**13**	**-61**

b) Neighborhood Configuration: In both the Ising Model and Multi-Access Wireless Communication environments, the agents are arranged in a two dimensional lattice graph, with rewards depending solely on their proximal agents. Consequently, an agent's one-hop neighbors are naturally defined as those directly connected to it, including itself. In the other three environments, the agents are trained to avoid one another. Therefore, the one-hop neighbor distances d are naturally set as the Euclidean safety distances. Specifically, the safety distances (or d) are set to $0.15, 0.2, 0.25, 0.3, 0.35$ when $M = 3, 6, 12, 24, 48$, respectively. Each agent observes its one-hop neighbors to obtain one-hop neighbor states. Other distance metrics that account for velocity can be employed, which is left for future work.

c) Benchmarks: We compare our method with four state-of-the-art MARL algorithms: MADDPG [4], MFAC [9], EPC [10], and SAC [8]. The SAC algorithm only works in the Multi-Access Wireless Communication environment due to the reward assumption.

d) Evaluation Metrics: We measure the performance using two criteria: *training efficiency* and *policy quality*. To measure the training efficiency, we use two metrics: 1) *average training time* spent to run a specified number of training iterations and 2) *convergence time*. The convergence time is defined as the time when the variance of the average total training reward over 90 consecutive iterations does not exceed 2% of the absolute mean reward, where the average total training reward is the total reward of all agents averaged over 10 episodes in three training runs with different random seeds. To measure policy quality, we use *convergence reward*, which is the average total training reward at the convergence time.

e) Computing Configurations: The computing resources are configured in a way so that DARL1N utilizes roughly the same or fewer resources than the baseline methods, as described in [29]. For the Multi-Access Wireless Communication environment, we employ the Amazon EC2 $c5n.large$ instance for DARL1N training, the $z1d.3xlarge$ instance for MADDPG and MFAC, and the $c5.12xlarge$ instance for EPC training. The configurations for the training parameters, as well as the representations of policy and Q functions can be found in [29].

2) Comparative Studies:

a) Ising Model: As shown in Tab. I, when the number of agents is small ($M = 9$), all methods achieve roughly the same reward. DARL1N takes the least amount of time to converge while EPC takes the longest time. When the number of agents increases, it can be observed that the EPC converges immediately and the convergence reward it achieves when $M = 64$ is much higher than the other methods. The reason is that, in the Ising Model, each agent only needs information of its four fixed neighbors, and hence in EPC the policy obtained from the previous stage can be applied to the current stage. The other methods train the agents from scratch without curriculum learning. For illustration, we also show the performance achieved by training EPC from scratch without curriculum learning (labeled as EPC Scratch in Tab. I). The results show that EPC Scratch converges much slower than EPC as the number of agents increases. Note that when the number of agents is 9, EPC and EPC Scratch are the same. Moreover, DARL1N achieves a reward comparable with that of EPC Scratch but converges much faster. From Fig. 2(a), we can observe that DARL1N requires much less time to perform a training iteration than the benchmark methods.

b) Food Collection: Tab. II shows that, in Food Collection, when the problem scale is small, DARL1N, MADDPG and MFAC achieve similar performance in terms of policy quality. As the problem scale increases, the performance of MADDPG and MFAC degrades significantly and becomes much worse than DARL1N or EPC when $M = 12$ and $M = 24$, which is also illustrated in Fig. 3. The convergence reward achieved by DARL1N is comparable or sometimes higher than that achieved by EPC. Moreover, the convergence speed of DARL1N is the highest among all methods in all scenarios. Notably, the convergence time of DALR1N and EPC increases, while that of MADDPG and MFAC decreases as M increases to 24. This occurs because MADDPG and MFAC fail to handle such large-scale networks, causing them to stop learning earlier.

To evaluate the impact of the proposed one-hop neighbor reward formulation on the learning performance, we also present in Fig. 3 the training rewards of DARL1N with a standard reward definition, labeled as DARL1N (Full Range), whose neighbor distance d is set to cover the entire environment, thereby including all agents as one-hop neighbors. The results show that DARL1N (Full Range) achieves performance comparable to EPC but performs worse than DARL1N with a small number of agents considered as one-hop neighbors. This suggests that in the Food Collection environment, agent behavior primarily depends on interactions with a nearby, smaller group of agents. Fig. 2(b) illustrates the training time of DARL1N (Full Range), which increases compared to DARL1N due to the inclusion of more agents in the reward calculations.

Fig. 2(b) also presents the training times of the four benchmarks. Among all methods compared, DARL1N achieves the highest training efficiency and its training time grows linearly as the number of agents increases. When $M = 24$, EPC takes the longest time to train. This is because of the complex policy and Q neural network architectures in EPC, the input dimensions of which grow linearly and quadratically, respectively, with more agents.

To demonstrate DARL1N's applicability to general MARL problems with global reward and transition models, we conduct

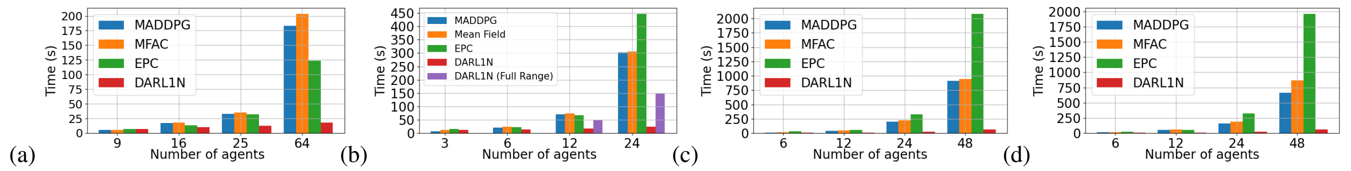


Fig. 2: Average training time of different methods to run (a) 10 iterations in the Ising Model, (b) 30 iterations in the Food Collection, (c) 30 iterations in the Grassland, and (d) 30 iterations in the Adversarial Battle environments.

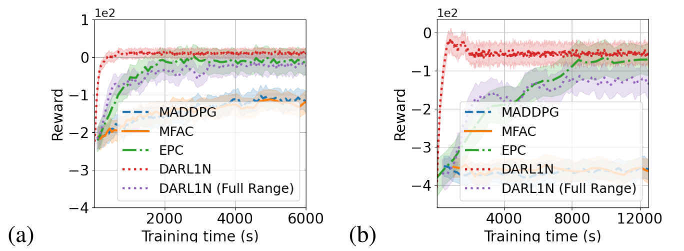


Fig. 3: Average total training reward of different methods in the Food Collection environment when there are (a) $M = 12$, and (b) $M = 24$ agents.

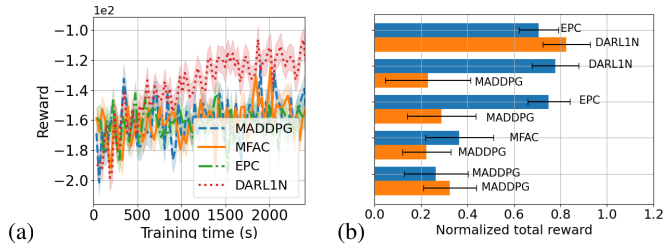


Fig. 4: (a) Average total training reward of different methods in the Food Collection environment with global reward and transition models when there are $M = 8$ agents and (b) Mean and standard deviation of normalized total reward of competing agents trained by different methods in the Adversarial Battle environment with $M = 48$.

a comparison study using a variant of the Food Collection environment where agents must coordinate to exclusively collect all the food. As shown in Fig. 4(a), DARL1N achieves the highest reward level with the fastest training speed. This is due to its distributed learning architecture, which reduces training experience correlation and accelerates training through parallel computing, even without decomposition. In contrast, EPC performs significantly worse, likely because curriculum learning struggles with global agent coordination.

c) Grassland: Similar as the results in the Food Collection environment, the policy generated by DARL1N is equally good or even better than those generated by the benchmark methods, as shown in Tab. III and Fig. 2(c), especially when the problem scale is large. DARL1N also has the fastest convergence speed and takes the shortest time to run a training iteration.

d) Adversarial Battle: In this environment, DARL1N again achieves good performance in terms of policy quality and training efficiency compared to the benchmark methods, as shown in Tab. IV and Fig. 2(d). To further evaluate the performance, we reconsider the last scenario ($M = 48$) and train the good agents and adversary agents using two different methods. The trained good agents and adversarial agents then compete with each other. We apply the Min-Max normalization to measure the normalized total reward of agents

TABLE III: Convergence time and convergence reward of different methods in the Grassland environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 6$	$M = 12$	$M = 24$	$M = 48$	$M = 6$	$M = 12$	$M = 24$	$M = 48$
MADDPG	423	6271	2827	1121	21	11	-302	-612
MFAC	431	7124	3156	**1025**	**23**	9	-311	-608
EPC	4883	2006	3324	15221	12	38	105	205
DARL1N	**103**	**402**	**1752**	5221	18	**46**	**113**	**210**

TABLE IV: Convergence time and convergence reward of different methods in the Adversarial Battle environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 6$	$M = 12$	$M = 24$	$M = 48$	$M = 6$	$M = 12$	$M = 24$	$M = 48$
MADDPG	452	1331	1521	7600	-72	-211	-725	-1321
MFAC	463	1721	1624	6234	-73	-221	-694	-1201
EPC	1512	1432	2041	9210	-75	-215	**-405**	**-642**
DARL1N	**121**	**756**	**1123**	**3110**	**-71**	**-212**	-410	-682

at each side achieved in an episode. To reduce uncertainty, we generate 10 episodes and record the mean values and standard deviations. As shown in Fig. 4(b), DARL1N achieves the best performance, and both DARL1N and EPC significantly outperform MADDPG and MFAC.

e) Multi-Access Wireless Communication: Fig. 5 shows that SAC achieves a higher reward than DARL1N when z takes a small value. However, when z increases, which causes an exponential growth of the state space, DARL1N achieves a much higher reward and converges much faster than SAC. This demonstrates that DARL1N scales better than SAC with the size of the state space.

3) Impact of Neighbor Distance: The parameter of neighbor distance d in DARL1N determines the number of one-hop neighbors of an agent, thereby influencing both training efficiency and policy quality. To evaluate its impact, we conduct experiments using the Grassland environment, considering three good and three adversary agents. The rewards are set to 10 for good agents collecting a grass pellet and -100 for colliding with adversary agents.

The results shown in Fig. 6 indicate that, as the neighbor distance d increases, the training reward increases while the training time arises. This stems from the increased number of one-hop neighbors each agent must consider, thereby requiring each learner to collect and process more data. This phenomenon reveals a trade-off between training efficiency and policy quality controlled by the neighbor distance, which can be properly chosen to achieve a good balance.

B. Performance of Coded Distributed Learning Architecture

In this section, we first conduct numerical studies to evaluate the performance of different assignment schemes described in Sec. V. We then train DARL1N over the proposed coded distributed learning architecture and conduct experimental

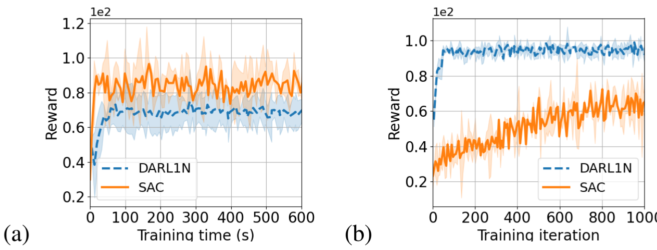


Fig. 5: Average total training reward of DARL1N and SAC in the Multi-Access Wireless Communication environment when (a) $z = 2$ (b) $z = 10$.

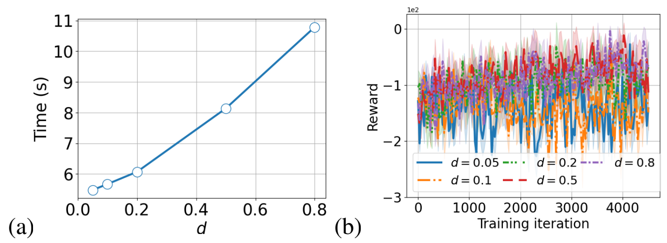


Fig. 6: (a) Average training time of 30 iterations and (b) average total training reward of DARL1N in the Grassland environment when d increases with $M = 6$.

studies to evaluate its performance in mitigating the effect of computing stragglers.

1) Numerical Evaluation: We conduct numerical studies to evaluate the performance of different assignment schemes in the following three aspects.

- *Computation overhead*: Metric (11) is applied, with the mean overhead averaged over 10 experiment runs used for the Random Sparse and LDGM schemes.
- *Resilience to stragglers*: The success rate computed as follows is used. We randomly turn some learners into stragglers that fail to return any results according to Assumption 4. Monte Carlo simulations are then conducted to measure the success rate, which is the ratio of training iterations in which gradients can be successfully estimated with results returned from non-stragglers.
- *Impact on policy quality*: According to Theorem 1, the gradients estimated by Coded DARL1N are unbiased but their variance depends on the assignment matrix. Therefore, we use the variance of the estimated gradients, denoted by $\mathbb{V}[\hat{\mathbf{e}}]$, to assess the impact of assignment schemes on policy quality. Specifically, we vary the number of learners whose results are used by the central controller to estimate the gradients and calculate the average value of $V := \log(\det(\mathbb{V}[\tilde{\mathbf{e}}]))$ over 100 Monte Carlo simulation runs.

Tab. V presents results when $M = 12$ and $N = 24$. The performance of the Random Sparse and LDGM schemes, characterized by parameters ξ and ρ respectively, is evaluated across different parameter values. The results show that the Uncoded scheme has the lowest computation overhead, the smallest variance in most scenarios, but the poorest resilience to stragglers. In contrast, the MDS scheme exhibits the best resilience to stragglers but the largest computation overhead and variance. The Repetition scheme has the smallest variance

TABLE V: Comparison of different assignment schemes in terms of computation overhead, success rate, and average V.

		Computation overhead	**Success rate**			**Average V**		
			$\eta = 0$	$\eta = 0.2$	$\eta = 0.5$	$\lvert\mathcal{J}\rvert = 12$	$\lvert\mathcal{J}\rvert = 18$	$\lvert\mathcal{J}\rvert = 24$
Uncoded		**0**	**1**	0	0	**0**	0	0
MDS		23	**1**	**1**	**1**	124.82	79.75	73.64
Random Sparse	$\xi = 0.2$	4.5	**1**	0.78	0.17	11.48	1.62	-0.68
	$\xi = 0.4$	8.0	**1**	**1**	0.98	16.82	2.93	-1.28
	$\xi = 0.8$	18.3	**1**	**1**	**1**	13.15	1.40	-3.31
Repetition		1	**1**	0.603	0	**0**	0	**-8.32**
LDGM	$\rho = 0.1$	0.91	**1**	0.253	0	2.08	**-1.46**	-4.59
	$\rho = 0.3$	4.41	**1**	0.79	0.11	8.93	1.05	-3.21
	$\rho = 0.5$	5.5	**1**	0.99	0.41	13.13	5.13	-0.31

among all schemes and the lowest computation overhead among coded schemes, though it is relatively less resilient. For the Random Sparse and LDGM schemes, increasing ξ or ρ leads to higher computation overhead and improved resilience to stragglers. However, the Random Sparse scheme generally exhibits larger variance compared to LDGM. In the following experiments, we set $\xi = 0.8$ and $\rho = 0.3$.

2) Experiments: To understand the performance of the coded distributed learning architecture, we train DARL1N using different assignment schemes and evaluate its performance in different straggler scenarios simulated on Amazon EC2.

a) Experiment Settings: We select the Food Collection environment and set the number of agents and learners in all experiments to $M = 12$ and $N = 24$, respectively. To evaluate the impact of stragglers, we vary the straggler probability η in Assumption 4. As Amazon EC2 computing instances are generally stable, we simulate stragglers by having selected compute nodes delay returning results by $\Delta > 0$ amount of time. Evaluations on other computing systems where stragglers are more common, such as wireless and mobile computing systems, are deferred to future work.

b) Experiment Results: We first evaluate the average training time of DARL1N with different assignment schemes. We vary the straggler probability η and the straggler effect Δ. The results are shown in Fig. 7(a) when $\Delta = 1$ and Fig. 7(b) when $\Delta = 4$, where the training time is measured by averaging the time for running 30 training iterations. We can observe that when no stragglers exist ($\eta = 0$), the Uncoded scheme is the most efficient as it has zero computation overhead. The MDS and Random Sparse schemes require a much longer training time than other schemes due to the substantial computation overhead introduced by these schemes. When stragglers exist ($\eta > 0$), the performance of the Uncoded scheme degrades significantly, especially when the straggler effect is significant as shown in Fig. 7(b). Compared to the Uncoded scheme, the LDGM and Repetition schemes are more resilient to stragglers, as indicated by the slower increase in training time. They are also more efficient than the MDS and Random Sparse schemes in most cases. On the contrary, the training time of MDS and Random Sparse does not grow much as Δ increases from 1 to 4, evidencing their high resilience to stragglers. Although they require more training time than other schemes when the straggler effect Δ or the straggler probability η is small, they achieve higher training efficiency when Δ and/or η are large.

To evaluate the impact of different assignment schemes on the quality of trained policies, we measure the training reward achieved by each DARL1N implementation with $\Delta = 1$. Tab. VI summarizes the convergence time, convergence reward,

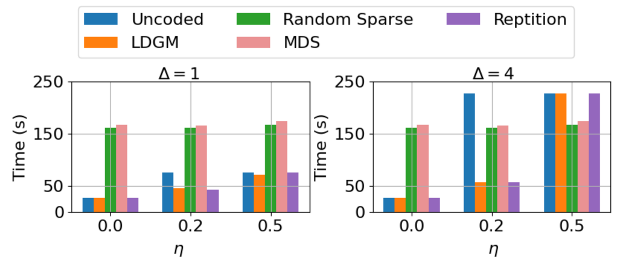


Fig. 7: Average training time of different DARL1N implementations with straggler effect $\Delta = 1$ and $\Delta = 4$, respectively.

TABLE VI: Convergence time, convergence reward and average V of different DARL1N implementations.

Schemes	Convergence Time (s)			Convergence Reward			Average V		
	$\eta = 0$	$\eta = 0.2$	$\eta = 0.5$	$\eta = 0$	$\eta = 0.2$	$\eta = 0.5$	$\eta = 0$	$\eta = 0.2$	$\eta = 0.5$
Uncoded	323	510	521	8	5	10	0	0	0
MDS	502	748	625	-231	-253	-212	82.08	100.10	104.06
Random Sparse	512	820	670	-252	-231	-227	15.28	13.47	12.88
Repetition	331	372	564	11	6	8	**-3.16**	**-4.49**	**-4.15**
LDGM	324	**320**	450	9	**12**	**14**	-0.51	0.14	0.35

and variance V, averaged over 600 training iterations, for different implementations as straggler probability η increases. We can see that the Uncoded scheme converges fast when no stragglers exist ($\eta = 0$) but its convergence speed decreases significantly when stragglers exist ($\eta > 0$). The MDS and Random Sparse schemes achieve the lowest reward and slowest convergence rate, while the LDGM scheme achieves the highest reward and convergence rate in most cases, especially when the straggler probability η is large. The Repetition scheme generally achieves good training reward performance and converges fast when the straggler probability is small. Moreover, we can also see that a larger average V generally leads to a lower reward.

3) Discussion: The experiment results above suggest guidelines for selecting an appropriate assignment scheme of agents to learners. The Uncoded scheme has zero computation overhead and low variance but is the least resilient to stragglers. This makes it suited best for stable distributed systems, such as server-based setups with wired communication. The MDS scheme, while offering the highest resilience, incurs the highest computation overhead and variance, leading to poor policy quality, and making it unsuitable for distributed training using DARL1N. In unstable distributed systems, where stragglers are present, a trade-off between policy quality and resilience must be considered. If policy quality is the priority, the Repetition scheme is an excellent choice. Conversely, if resilience is more critical, the Random Sparse scheme is preferable. For a balanced approach that addresses both aspects, the LDGM scheme is a good option.

VII. Limitations and Future Work

In environments with local reward and transition models, DARL1N needs a suitably chosen distance metric to establish the agent neighborhoods that achieve the right balance between policy quality and ability to distribute the training efficiently. In future work, we will explore learning a neighbor distance metric that adapts to the environment, e.g., based on past episodes or contextual information, to achieve an effective balance between policy reward and training speed. Moreover, the coded distributed learning architecture for DARL1N is designed for a distributed computing system with a stable central controller in place. In future work, we will design a new coded architecture to improve resilience of central controllers such as introducing redundant central controllers using coding theory. Other issues to consider for further improvements include integration of curriculum learning similar as EPC, partially observable states, and software infrastructure to support distributed learning with low-latency communication.

VIII. Conclusion

This paper introduced DARL1N, a scalable MARL algorithm that can be trained over a distribute computing architecture. DARL1N reduces the representation complexity of the value and policy functions of each agent in a MARL problem by disregarding the influence of other agents that are not within one hop of a proximity graph. This model enables highly efficient distributed training, in which a compute node only needs data from an agent it is training and its potential one-hop neighbors. We conducted comprehensive experiments using five MARL environments and compared DARL1N with four state-of-the-art MARL algorithms. DARL1N generates equally good or even better policies in almost all scenarios with significantly higher training efficiency than benchmark methods, especially in large-scale problem settings. To improve the resilience of DARL1N to stragglers common in distributed computing systems, we developed coding schemes that assign each agent to multiple learners. We evaluate properties of MDS, Random Sparse, Repetition, LDGM codes and provide guidelines on selecting suitable assignment schemes under different situations.

References

[1] Y. Jin, S. Wei, J. Yuan, and X. Zhang, "Hierarchical and stable multiagent reinforcement learning for cooperative navigation control," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[2] R. Song, F. L. Lewis, and Q. Wei, "Off-policy integral reinforcement learning method to solve nonlinear continuous-time multiplayer nonzero-sum games," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 704–713, 2016.

[3] G.-P. Antonio and C. Maria-Dolores, "Multi-agent deep reinforcement learning to manage connected autonomous vehicles at tomorrow's intersections," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 7, pp. 7033–7043, 2022.

[4] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 6379–6390.

[5] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in Multi-agent Systems and Applications*, pp. 183–221, 2010.

[6] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[7] K. Gogineni, P. Wei, T. Lan, and G. Venkataramani, "Scalability bottlenecks in multi-agent reinforcement learning systems," *arXiv preprint arXiv:2302.05007*, 2023.

[8] G. Qu, A. Wierman, and N. Li, "Scalable reinforcement learning of localized policies for multi-agent networked systems," in *Learning for Dynamics and Control (L4DC)*. PMLR, 2020, pp. 256–266.

[9] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 5571–5580.

[10] Q. Long, Z. Zhou, A. Gupta, F. Fang, Y. Wu, and X. Wang, "Evolutionary population curriculum for scaling multi-agent reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2020.

[11] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint:1706.05296*, 2017.

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

[13] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5887–5896.

[14] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2961–2970.

[15] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, "Massively parallel methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2015.

[16] B. Wang, J. Xie, and N. Atanasov, "Coding for Distributed Multi-Agent Reinforcement Learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 10 625–10 631.

[17] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 5872–5881.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[19] OpenAI, "A2C," https://openai.com/blog/baselines-acktr-a2c/, 2022, accessed: 2022-01-13.

[20] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013, pp. 1223–1231.

[21] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 19–27.

[22] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 803–812.

[23] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[24] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[25] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, mar 2018.

[26] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[27] B. Zhou, J. Xie, and B. Wang, "Dynamic Coded Distributed Convolution for UAV-based Networked Airborne Computing," in *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022, pp. 955–961.

[28] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2015, pp. 964–971.

[29] B. Wang, J. Xie, and N. Atanasov, "Darl1n: Distributed multi-agent reinforcement learning with one-hop neighbors," pp. 9003–9010, 2022.

[30] F. Bullo, J. Cortés, and S. Martinez, *Distributed control of robotic networks: a mathematical approach to motion coordination algorithms*. Princeton University Press, 2009.

[31] X. Qian and D. Klabjan, "The impact of the mini-batch size on the variance of gradients in stochastic gradient descent," *arXiv preprint arXiv:2004.13146*, 2020.

[32] J. Lacan and J. Fimes, "Systematic mds erasure codes based on vandermonde matrices," *IEEE Communications Letters*, vol. 8, no. 9, pp. 570–572, 2004.

[33] A. Klinger, "The vandermonde matrix," *The American Mathematical Monthly*, vol. 74, no. 5, pp. 571–574, 1967.

[34] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2413–2417.

[35] S. Cai, W. Lin, X. Yao, B. Wei, and X. Ma, "Systematic convolutional low density generator matrix code," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3752–3764, 2021.

[36] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[37] AWS, "Amazon ec2," https://aws.amazon.com/ec2/, 2022, accessed: 2022-01-13.

Appendix

A. Proof of Lemma 1

Consider the Q-value function $Q_i^{\boldsymbol{\mu}}$ of agent i. For two different sets of non-neighbor states $\hat{\mathbf{s}}_{\mathcal{N}_i^-} \neq \mathbf{s}_{\mathcal{N}_i^-}$ and actions $\hat{\mathbf{a}}_{\mathcal{N}_i^-} \neq \mathbf{a}_{\mathcal{N}_i^-}$, we first show that:

$$\begin{aligned}&|Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}) - Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \hat{\mathbf{s}}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \hat{\mathbf{a}}_{\mathcal{N}_i^-})| \\ &\qquad \leq \frac{2\bar{r}\gamma}{1-\gamma}. \end{aligned} \tag{11}$$

Letting $(\mathbf{s}, \mathbf{a})$ and $(\hat{\mathbf{s}}, \hat{\mathbf{a}})$ denote $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-})$ and $(\mathbf{s}_{\mathcal{N}_i}, \hat{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \hat{a}_{\mathcal{N}_i^-})$, respectively, we have:

$$\begin{aligned}
&|Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a}) - Q_i^{\boldsymbol{\mu}}(\hat{\mathbf{s}}, \hat{\mathbf{a}})| \\
&= \Bigg| \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})] \\
&\quad - \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})] \Bigg| \\
&\leq \sum_{t=0}^{\infty} \Big| \mathbb{E}\left[\gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right] \\
&\quad - \mathbb{E}\left[\gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right] \Big| \\
&\overset{(\mathbf{a})}{=} \sum_{t=1}^{\infty} \Big| \mathbb{E}\left[\gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right] \\
&\quad - \mathbb{E}\left[\gamma^t r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right] \Big| \\
&\leq \sum_{t=1}^{\infty} \gamma^t \Big(\big| \mathbb{E}\left[r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right] \big| \\
&\quad + \big| \mathbb{E}\left[r_i(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right] \big| \Big) \\
&\leq \sum_{t=1}^{\infty} 2\gamma^t \bar{r} = \frac{2\bar{r}\gamma}{1-\gamma}
\end{aligned} \tag{12}$$

where $(\mathbf{a})$ derives from the fact that $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ are part of both $(\mathbf{s}, \mathbf{a})$ and $(\hat{\mathbf{s}}, \hat{\mathbf{a}})$. In the above equations, the expectation $\mathbb{E}$ is over state-action trajectories generated by the policy $\boldsymbol{\mu}$ and the transition model p. Then, we have:

$$\begin{aligned}
&\left| \tilde{Q}_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) - Q_i^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a}) \right| \\
&= \Bigg| \sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} \omega_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) \\
&\quad - Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \hat{\mathbf{s}}_{\mathcal{N}_i^-}, \hat{\mathbf{a}}_{\mathcal{N}_i^-}) \Bigg| \\
&\leq \sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} \omega_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) \Big| Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) \\
&\quad - Q_i^{\boldsymbol{\mu}}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \hat{\mathbf{s}}_{\mathcal{N}_i^-}, \hat{\mathbf{a}}_{\mathcal{N}_i^-}) \Big| \leq \frac{2\bar{r}\gamma}{1-\gamma}.
\end{aligned} \tag{13}$$

B. Proof of Proposition 1

If agent $j \notin \mathcal{P}_i(t)$, then based on the definition of potential neighbors, we have $\text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t)) > d + 2\epsilon$. According to the triangle inequality, $\text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t+1)) + \text{dist}(\mathbf{s}_j(t+1), \mathbf{s}_j(t)) \geq \text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t))$, and according to Assumption 1, $\text{dist}(\mathbf{s}_j(t+1), \mathbf{s}_j(t)) \leq \epsilon$. Therefore, $\text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t+1)) > d+\epsilon$. Using the triangle inequality again, we obtain $\text{dist}(\mathbf{s}_i(t+1), \mathbf{s}_j(t+1)) + \text{dist}(\mathbf{s}_i(t+1), \mathbf{s}_i(t)) \geq \text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t+1))$. As $\text{dist}(\mathbf{s}_i(t+1), \mathbf{s}_i(t)) \leq \epsilon$, we have $\text{dist}(\mathbf{s}_i(t+1), \mathbf{s}_j(t+1)) > d$. Therefore, agent j will not be a one-hop neighbor of agent i at time $t+1$.

C. Proof of Theorem 1

The bias of the gradient estimator $\tilde{\mathbf{e}}$ can be calculated using (7) and (8) as follows:

$$\begin{aligned}\mathbb{E}[\tilde{\mathbf{e}}] - \mathbf{e} &= (\mathbf{C}_{\mathcal{J}}^T \mathbf{C}_{\mathcal{J}})^{-1} \mathbf{C}_{\mathcal{J}}^T \mathbb{E}[\mathbf{y}_{\mathcal{J}}] - \mathbf{e} \\ &= (\mathbf{C}_{\mathcal{J}}^T \mathbf{C}_{\mathcal{J}})^{-1} \mathbf{C}_{\mathcal{J}}^T \mathbf{D} \mathbb{E}[\mathbf{q}] - \mathbf{e}. \end{aligned} \tag{14}$$

Since each learner uses the same set of parameters broadcast by the central controller for agent-environment interaction in each training iteration, the replay buffers $\mathcal{D}_{j,i}$, $\forall j \in [N]$, all follow the same distribution as that of $\mathcal{D}_i$. Therefore, we have $\mathbb{E}[\hat{\mathbf{e}}_{j,i}] = \mathbf{e}_i$ and $\mathbf{D}\mathbb{E}[\mathbf{q}] = \mathbf{C}_{\mathcal{J}}\mathbf{e}$ leading to:

$$\mathbb{E}[\tilde{\mathbf{e}}] - \mathbf{e} = (\mathbf{C}_{\mathcal{J}}^T \mathbf{C}_{\mathcal{J}})^{-1} \mathbf{C}_{\mathcal{J}}^T \mathbf{C}_{\mathcal{J}} \mathbf{e} - \mathbf{e} = 0, \tag{15}$$

which shows that $\tilde{\mathbf{e}}$ is an unbiased estimator.

Next, we compute the variance of the gradient estimator $\tilde{\mathbf{e}}$:

$$\begin{aligned}\mathbb{V}[\tilde{\mathbf{e}}] &= \mathbb{V}[(\mathbf{C}_{\mathcal{J}}^T \mathbf{C}_{\mathcal{J}})^{-1} \mathbf{C}_{\mathcal{J}}^T \mathbf{y}_{\mathcal{J}}] \\ &= (\mathbf{C}_{\mathcal{J}}^T \mathbf{C}_{\mathcal{J}})^{-1} \mathbf{C}_{\mathcal{J}}^T \mathbf{D} \mathbb{V}(\mathbf{q}) \mathbf{D}^T ((\mathbf{C}_{\mathcal{J}}^T \mathbf{C}_{\mathcal{J}})^{-1} \mathbf{C}_{\mathcal{J}}^T)^T, \end{aligned} \tag{16}$$

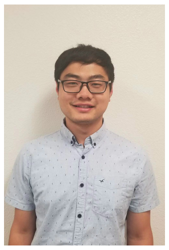
where $\mathbb{V}[\mathbf{q}] = \text{diag}(\mathbb{V}(\hat{\mathbf{e}}_{1,1}), \ldots, \mathbb{V}(\hat{\mathbf{e}}_{N,M}))$ and diag() creates a diagonal matrix with the $\mathbb{V}(\hat{\mathbf{e}}_{j,i})$ as diagonal element. Since $\hat{\mathbf{e}}_{j,i}, \forall i \in [M]$ are independent from each other for each $j \in [N]$. According to (16), we can see that the variance of the gradient estimator $\tilde{\mathbf{e}}$ is impacted by $\mathbf{C}_{\mathcal{J}}$, which is determined by the assignment matrix $\mathbf{C}$ as well as the learners who return their computations promptly.

D. Proof of Proposition 3

The performance of the MDS code scheme will be affected only if the number of stragglers exceeds $N-M$ because $\mathbf{C}^{\text{MDS}}$ has rank M. If there are $W > N-M$ stragglers, the results from non-straggler nodes will be insufficient for the central controller to decode the parameter gradients and it needs to wait for results from the stragglers. Under Assumption 4, W follows a binomial distribution with probability mass function $p(W=w) = \binom{N}{w}(1-\eta)^{N-w}\eta^w$. Therefore, the probability that the performance will be affected by the stragglers is $\sum_{j=N-M+1}^{N} p(W=j) = \sum_{j=N-M+1}^{N} \binom{N}{j}(1-\eta)^{N-j}\eta^j$.

E. Proof of Proposition 4

The assignment matrix $\mathbf{C}^{\text{Repetition}}$ defined in (10) has M linearly independent rows, with each row containing $\frac{N}{M}$ duplicate copies. For j-th copy of i-th linearly independent row, $\forall i \in [M], \forall j \in [\frac{N}{M}]$, we have a learner with index $i+(j-1)M$ needs to send $\mathbf{y}_{i+(j-1)M}$ back to the central controller. The estimated gradients can be decoded when learners with index $i+(j-1)M, \forall i \in [M]$, with any $j \in [\frac{N}{M}]$ send results back to satisfy rank($\mathbf{C}_{\mathcal{J}}$) = M. Under Assumption 4, for a $i \in [M]$, the probability that the learners with index $i+(j-1)M, \forall j \in [\frac{N}{M}]$ are all stragglers that fail to send results back is $\eta^{\frac{N}{M}}$. Furthermore, the probability that there is at least one non-straggler learner for each $i \in [M]$ is $(1-\eta^{\frac{N}{M}})^M$. Therefore, the probability that the performance will be affected by stragglers is then represented with $1-(1-\eta^{\frac{N}{M}})^M$.



Baoqian Wang (S'20) is currently an advanced technologist in intelligent systems at The Boeing Company. He received a Ph.D. degree in Electrical and Computer Engineering from University of California San Diego and San Diego State University in 2023. He received his B.S. degree from Yangtze University, Wuhan China, in 2017, and M.S. degree in Computer Science from Texas A&M University-Corpus Christi. His research interests include distributed computing, reinforcement learning and robotics.

Junfei Xie (S'13-M'16-SM'21) is currently an Associate Professor in the Department of Electrical and Computer Engineering at San Diego State University. She received the B.S. degree in Electrical Engineering from University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2012. She received the M.S. degree in Electrical Engineering in 2013 and the Ph.D. degree in Computer Science and Engineering from University of North Texas, Denton, TX, in 2016. From 2016 to 2019, she was an Assistant Professor in the Department of Computing Sciences at Texas A&M University-Corpus Christi. She is the recipient of the NSF CAREER Award. Her current research interests include large-scale dynamic system design and control, airborne networks, airborne computing, and air traffic flow management, etc.

Nikolay Atanasov (S'07-M'16-SM'23) is an Associate Professor of Electrical and Computer Engineering at the University of California San Diego, La Jolla, CA, USA. He obtained a B.S. degree in Electrical Engineering from Trinity College, Hartford, CT, USA in 2008 and M.S. and Ph.D. degrees in Electrical and Systems Engineering from the University of Pennsylvania, Philadelphia, PA, USA in 2012 and 2015, respectively. Dr. Atanasov's research focuses on robotics, control theory, and machine learning with emphasis on active perception problems for autonomous mobile robots. He works on probabilistic models for simultaneous localization and mapping (SLAM) and on optimal control and reinforcement learning algorithms for minimizing probabilistic model uncertainty. Dr. Atanasov's work has been recognized by the Joseph and Rosaline Wolf award for the best Ph.D. dissertation in Electrical and Systems Engineering at the University of Pennsylvania in 2015, the Best Conference Paper Award at the IEEE International Conference on Robotics and Automation (ICRA) in 2017, the NSF CAREER Award in 2021, and the IEEE RAS Early Academic Career Award in Robotics and Automation in 2023.