# Privacy-Engineered Value Decomposition Networks for Cooperative Multi-Agent Reinforcement Learning

Parham Gohari, Matthew Hale, and Ufuk Topcu

***Abstract*— In cooperative multi-agent reinforcement learning (Co-MARL), a team of agents must jointly optimize the team's long-term rewards to learn a designated task. Optimizing rewards as a team often requires inter-agent communication and data sharing, leading to potential privacy implications. We assume privacy considerations prohibit the agents from sharing their environment interaction data. Accordingly, we propose Privacy-Engineered Value Decomposition Networks (PE-VDN), a Co-MARL algorithm that models multi-agent coordination while provably safeguarding the confidentiality of the agents' environment interaction data. We integrate three privacy-engineering techniques to redesign the data flows of the VDN algorithm—an existing Co-MARL algorithm that consolidates the agents' environment interaction data to train a central controller that models multi-agent coordination—and develop PE-VDN. In the first technique, we design a distributed computation scheme that eliminates Vanilla VDN's dependency on sharing environment interaction data. Then, we utilize a privacy-preserving multi-party computation protocol to guarantee that the data flows of the distributed computation scheme do not pose new privacy risks. Finally, we enforce differential privacy to preempt inference threats against the agents' training data—past environment interactions—when they take actions based on their neural network predictions. We implement PE-VDN in StarCraft Multi-Agent Competition (SMAC) and show that it achieves $80\%$ of Vanilla VDN's win rate while maintaining differential privacy levels that provide meaningful privacy guarantees. The results demonstrate that PE-VDN can safeguard the confidentiality of agents' environment interaction data without sacrificing multi-agent coordination.**

## I. INTRODUCTION

Cooperative multi-agent reinforcement learning (Co-MARL) is a machine learning problem in which multiple agents work together to optimize performance in a common task. The agents interact with an environment that rewards their actions as a team and must learn a decision-making scheme that maximizes the team's long-term rewards through trial and error [25]. Co-MARL algorithms can extend the capabilities of single-agent reinforcement learning algorithms to complete complex tasks that involve multiple agents; for instance, in self-driving vehicles where reinforcement learning is a popular approach for autonomous driving [2], Co-MARL may enable a fleet of autonomous vehicles to cooperate and reduce traffic congestion [4].

We study modeling multi-agent coordination in Co-MARL systems with privacy in mind. Effective coordination often requires inter-agent communication and data sharing. However, sharing data may have privacy ramifications in situations where agents represent privacy-sensitive entities or handle privacy-sensitive information. For example, sharing a self-driving vehicle's environment interaction data may reveal commuting patterns and sensitive locations such as home, places of worship, and nightlife activities [9].

We assume that any data sharing that reveals the agents' interactions with the environment violates privacy. The presence of privacy-sensitive information in the agents' environment interaction data could complicate Co-MARL algorithms that rely on sharing them. Centralized training algorithms [18] are major examples in which a central node consolidates the environment interaction data of all agents and trains a central controller that determines the team's actions. The central controller accounts for the dynamics of how the training of one agent affects others and effectively models multi-agent coordination. By requiring the agents to share their environment interaction data, centralized training methods expose the agents' sensitive information to various privacy risks such as data breaches and unauthorized access. Accordingly, we raise the following research question:

> Is it possible to redesign the data flows of centralized training algorithms to safeguard the confidentiality of the agents' environment interaction data without losing multi-agent coordination?

We show that it is indeed possible to design such an algorithm and propose Privacy-Engineered Value Decomposition Networks (PE-VDN). Instead of relying on a central node that consolidates the agents' environment interaction data, PE-VDN establishes multi-agent coordination by creating peer-to-peer communication channels. Additionally, the algorithm incorporates additional privacy-enhancing techniques to ensure that its established information flows do not undermine the confidentiality of the agents' privacy-sensitive environment interaction data.

We develop PE-VDN based on the Value Decomposition Networks (VDN) algorithm in [21]—referred to as Vanilla VDN, hereafter. Vanilla VDN is a centralized training and decentralized execution (CTDE) Co-MARL algorithm that uses a specially structured function approximator to estimate the team's action values and compute optimal actions. The function approximator consists of multiple branches of neural networks, each of which is designated for one agent. These

P. Gohari is with the Department of Electrical and Computer Engineering, and Robert Strauss Center for International Security and Law, University of Texas at Austin, Austin, TX, USA pgohari@utexas.edu. M. Hale is with the Faculty of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL, USA matthewhale@ufl.edu. U. Topcu is with the Faculty of Aerospace Engineering and Engineering Mechanics, University of Texas at Austin, Austin, TX, USA utopcu@utexas.edu

This work was supported by AFOSR under Grant FA9550-19-1-0169 and ONR under Grant N00014-21-1-2502. M. Hale was additionally supported by NSF under CAREER Grant 1943275 and AFRL under Grant FA8651-23-F-A008.

neural networks use their designated agent's environment interaction data as input, and the summation of their outputs estimates the team's action value. The coupling of the neural networks during training takes multi-agent coordination into account. Once training concludes, the algorithm distributes the final neural network branches back to their designated agents so that they can use them to make their own decisions at runtime. This decentralized execution feature is well-suited to our privacy goals because it eliminates the need for sharing any data during execution.

We use the Vanilla VDN algorithm as PE-VDN's starting point and incorporate three privacy-engineering techniques to modify data flows and satisfy our privacy goals. First, we decentralize the vanilla algorithm's training. We demonstrate that the gradients that Vanilla VDN computes for centralized training are only coupled by a summation term that aggregates the output of all neural network branches in the action-value function approximator. We propose an equivalent distributed computation scheme for the gradients that enables the agents to locally maintain and optimize their dedicated neural network branches themselves. The resulting decentralized training algorithm computes the same gradients as Vanilla VDN and only requires the agents to share their neural network outputs with each other; hence, it mitigates the privacy risks of sharing environment interaction data.

In the second privacy measure, we integrate a privacy-preserving multi-party computation protocol with the distributed computation of the gradients. The protocol enables the agents to compute the summation term that couples their gradients while hiding the value of their neural network outputs from each other. Since the agents only need the summation term for decentralized training, revealing their neural network outputs for computing the summation unnecessarily exposes information that correlates with their sensitive environment interaction data. The protocol obfuscates the neural network outputs with correlated random numbers that act as encryption keys, then splits the resulting values into encrypted peer-to-peer messages. The protocol guarantees that as long as no party gains access to all of the peer-to-peer messages that an agent emits, the agents' neural network outputs remain a secret.

The third and last privacy-protection layer that we design for PE-VDN protects the agents' environment interaction data against indirect inference threats. The agents choose their actions based on their internal neural network's predictions and unintentional information leaks can occur when neural networks release their predictions. For example, so-called membership and attribute inference attacks, and inversion attacks are all known to make accurate inferences about the training data of a neural network by mere input and output observations [29]. In PE-VDN, the neural networks that determine the agents' actions are trained with environment interaction data. Our third privacy-engineering technique preempts inference threats against the agents' environment interaction data.

The DP-SGD algorithm [1] is a privacy-preserving training algorithm that can disrupt inference attacks against the training dataset of neural networks. DP-SGD enforces differential privacy during training and helps protect the confidentiality of the training data when external entities interact with the trained models [7]. Intuitively, DP-SGD guarantees that two training datasets that differ in a single record produce approximately statistically indistinguishable neural network parameters. If the agents train their internal neural networks with DP-SGD, then the plausible-deniability guarantees that DP-SGD provides can reduce the risks of indirect information leakage about the agents' past environment interaction data.

The DP-SGD algorithm can be seamlessly integrated with our last two privacy-engineering techniques; however, its differential privacy analysis does not readily apply to deep reinforcement learning algorithms. DP-SGD was originally developed for supervised machine learning algorithms with static training datasets, whereas in deep reinforcement learning, a stream of environment interaction data continuously enters a replay buffer and is used for training. We provide a theoretical analysis that leverages DP-SGD's Moments Accountant method [1] and Maximum-Overlap Parallel Composition [19] to compute the differential privacy level of DP-SGD when applied to deep reinforcement learning.

We implement a Python library for PE-VDN and test it in the StarCraft Multi-Agent Competition (SMAC) suite [16]. In the numerical results, we dissect the different privacy-engineering components and study the trade-offs between privacy, precision, and performance. Our results show that within the acceptable differential privacy range of 0 and 10 [13], the agents can achieve $80\%$ of Vanilla VDN's win rate.

## II. PRELIMINARIES

In this section, we cover some technical background on Co-MARL's problem formulation and review Vanilla VDN.

### A. Dec-POMDP

Co-MARL algorithms typically model multi-agent cooperation using decentralized partially observable Markov decision processes (Dec-POMDPs) [6]. A Dec-POMDP $\mathcal{G}$ is a tuple $\mathcal{G} = \langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{Z}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{N}$ is the set of agents and $N = |\mathcal{N}|$; $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ are the sets of all possible states, actions, and observations, respectively; $\mathcal{Z} : \mathcal{S} \mapsto \mathcal{O}$ is an observation function; $\mathcal{P} : \mathcal{S}^N \times \mathcal{A}^N \times \mathcal{S}^N \mapsto [0, 1]$ is the environment's transition probabilities; $\mathcal{R} : \mathcal{S}^N \times \mathcal{A}^N \times \mathcal{S}^N \mapsto \mathbb{R}$ is the reward function; and $\gamma \in [0, 1)$ is the discount factor.

A team policy, denoted $\boldsymbol{\pi} := (\pi_i)_{i \in \mathcal{N}}$, determines the actions that each of the agents must take at every environment observation. Similar to the notation used for the team policy, we use bold symbols to denote team actions and observations. In POMDPs, the policy typically incorporates a *history* of past environment observations and actions. Let $\boldsymbol{\tau}_t = \langle (\boldsymbol{o}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{o}_{t-1}, \boldsymbol{a}_{t-1}), \boldsymbol{o}_t \rangle$ denote the team's history at time $t$ where, for all $k \leq t$, $\boldsymbol{a}_k \in \mathcal{A}^N$ and $\boldsymbol{o} \in \mathcal{O}^N$. Then, $\boldsymbol{\pi}(\boldsymbol{a}_t \mid \boldsymbol{\tau}_t)$ is the probability that the team takes action $\boldsymbol{a}_t$ when its history is $\boldsymbol{\tau}_t$.

Given a team policy $\boldsymbol{\pi}$, a value function $V^{\boldsymbol{\pi}}$ evaluates the expected total reward that the policy accumulates, i.e.,

$$V^{\boldsymbol{\pi}}(\boldsymbol{s}) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \mid \boldsymbol{s}_1 = \boldsymbol{s}\right], \quad (1)$$

where $\boldsymbol{a}_t \sim \boldsymbol{\pi}$ and $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$. An action-value function $Q^{\boldsymbol{\pi}}$ is similarly defined as

$$Q^{\boldsymbol{\pi}}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}\left[\mathcal{R}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V^{\boldsymbol{\pi}}(\boldsymbol{s}') \mid \boldsymbol{s}' \sim \mathcal{P}(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})\right]. \quad (2)$$

The goal of solving a Dec-POMDP is to find an optimal action-value function $Q^*$ and a corresponding optimal policy $\boldsymbol{\pi}^*$ such that $Q^{\boldsymbol{\pi}^*}(\boldsymbol{s}, \boldsymbol{a}) = Q^*(\boldsymbol{s}, \boldsymbol{a}) = \sup_{\boldsymbol{\pi}} Q^{\boldsymbol{\pi}}(\boldsymbol{s}, \boldsymbol{a})$. In Co-MARL, the goal is to find an optimal policy without knowing the underlying transition probabilities.

### B. Deep Q-Learning

Deep Q-Learning (DQN) [11] is a variant of tabular Q-learning that uses deep neural networks to approximate the Q-function. By using neural networks, DQN can handle high-dimensional and continuous state spaces. DQN uses a *replay buffer* to train the neural network that approximates the Q-function. The replay buffer is a fixed-length database of the agent's past environment interaction data in the form of $\langle s_t, a_t, s_{t+1}, \mathcal{R}(s_t, a_t, s_{t+1})\rangle$, or in short $\langle s, a, s', r\rangle$. The replay buffer is typically filled as follows: the most recent data replaces the oldest entry. However, other methods that use specific heuristics to identify the entry that must be replaced with incoming data exist as well [3].

With $E$ denoting a minibatch of the replay buffer, DQN minimizes the Bellman error loss function defined as follows:

$$\ell(\theta) = \sum_{\langle s,a,s',r\rangle \in E} \left(r + \gamma \max_{a'} Q^{\pi}\left(s', a'; \theta^-\right) - Q^{\pi}(s, a; \theta)\right)^2, \quad (3)$$

where $\theta^-$ is called the target parameters. These parameters are copied from $\theta$ periodically to stabilize training.

### C. Multi-Agent DQN

Both tabular Q-learning and DQN can be used to solve Co-MARL problems. Since the agents in Co-MARL are rewarded together as a team, a single Q-function can represent the action values of the entire team, i.e., the team can be treated as a single agent with a multi-dimensional action. Such bundling of the agents reduces Co-MARL to single-agent reinforcement learning and is the main basis of centralized training methods. Bundling agents requires a consolidated replay buffer to support DQN. In the consolidated replay buffer, all parameters except the rewards are replaced with their team versions—team states and team actions and thus $\langle \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}, r\rangle$ is used instead of $\langle s, a, s', r\rangle$. We also replace all state values $s$ with histories $\tau$ to support partial observability. The resulting loss function is

$$\sum_{\langle \boldsymbol{\tau},\boldsymbol{a},\boldsymbol{\tau'},r\rangle \in E} \left(r + \gamma \max_{\boldsymbol{a'}} Q^{\boldsymbol{\pi}}\left(\boldsymbol{\tau'}, \boldsymbol{a'}; \theta^-\right) - Q^{\pi}(\boldsymbol{\tau}, \boldsymbol{a}; \theta)\right)^2. \quad (4)$$

Once training concludes, the trained Q-function for the team supports the decision-making of all agents. In this case, the agents must aggregate their environment observations to execute the learned policy too. Therefore, this algorithm is an instance of centralized training and centralized execution.

Alternatively, each agent may attribute the team rewards to itself and use DQN to train a policy independently. This approach is often called independent Q-learning. Independent training allows the agents to execute their policy without having to share their environment observations; however, as opposed to centralized training, independent training does not take multi-agent cooperation into account but it often scales better with the number of agents [25].

In CTDE methods, the agents use centralized training to learn the team's optimal Q-function but then decompose it into a set of local Q-functions that allow for decentralized execution. The agents in CTDE methods choose the action that maximizes their local Q-function; therefore, similar to independent Q-learning, the agents need not share data to execute the team's policy. The difference between these local Q-functions and those obtained via independent training, however, is that the former takes multi-agent coordination into account by design during training.

CTDE methods typically assume that the agents' individually optimal actions amount to the optimal action for the team. This so-called decentralizability assumption justifies decomposing the team's Q-function into local Q-functions and supports decentralized execution. The formal definition of decentralizability is as follows:

*Definition 1:* A reinforcement learning task is decentralizable if there exists a collection of local action-value functions $\{Q_i\}_{i\in\mathcal{N}}$ such that, for all team histories $\boldsymbol{\tau}$, team actions $\boldsymbol{a}$, and agents $i \in \mathcal{N}$,

$$\left(\arg\max_{\boldsymbol{a}} Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{a})\right)_i = \arg\max_{a_i} Q_i(\tau_i, a_i). \quad (5)$$

### D. Vanilla VDN

The Vanilla VDN algorithm leverages decentralizability by approximating the team's central Q-function with the summation of some local $Q_i$ functions. That is, the algorithm assumes that there exist $\{Q_i\}_{i\in\mathcal{N}}$ such that, for all joint policies $\boldsymbol{\pi}$, histories $\boldsymbol{\tau}$, and actions $\boldsymbol{a}$,

$$Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{a}) = \sum_{i\in\mathcal{N}} Q_i(\tau_i, a_i). \quad (6)$$

This additivity assumption implies decentralizability in (5); however, not all decentralizable tasks satisfy additivity.

In the centralized training phase of Vanilla VDN, the agents must send their environment interaction data to a central node to create a consolidated replay buffer and train the team's Q-function. The optimizer first draws a minibatch of the consolidated replay buffer, denoted $E$. Let, $e = \langle \boldsymbol{\tau}, \boldsymbol{a}, \boldsymbol{\tau'}, r\rangle \in E$ be an element of the minibatch, where $\boldsymbol{\tau} = (\tau_i)_{i\in\mathcal{N}}$, $\boldsymbol{a} = (a_i)_{i\in\mathcal{N}}$, and $\boldsymbol{\tau'} = (\tau_i')_{i\in\mathcal{N}}$ denote the team's history, action, and next-step history, respectively. With $\theta_i$ denoting the parameters of agent $i$'s dedicated portion of the

Q-function, Vanilla VDN's objective function is as follows:

$$\ell_{VDN}(e;\{\theta_i\}_{i\in\mathcal{N}}) = \left(r + \sum_{i\in\mathcal{N}} \left(\gamma \max_{a'} Q_i\left(\tau_i', a'; \theta_i^-\right) - Q_i\left(\tau_i, a_i; \theta_i\right)\right)\right)^2. \tag{7}$$

Vanilla VDN uses backpropagation to compute the gradient of the objective function with respect to each $\theta_i$ and uses gradient descent methods to update the Q-function's weights and biases. That is, for all neural network branches $\{\theta_i\}_{i\in\mathcal{N}}$, the optimizer performs the update

$$\theta_i \leftarrow \theta_i - \alpha \sum_{e\in E} \frac{\partial \ell_{\text{VDN}}(e;\theta_1,\ldots,\theta_N)}{\partial \theta_i}, \tag{8}$$

where $\alpha$ is the learning rate. Once centralized training concludes, the optimizer distributes the agents' designated portions—namely $Q_i(\cdot,\cdot;\theta_i)$ for each $i \in \mathcal{N}$—back to the agents to support decentralized execution.

## III. Related Work

Our main research objective is to design a Co-MARL algorithm with proper handling of the agents' environment interaction data in the name of privacy. In this section, we briefly review the key research themes that are relevant to our research objective.

The survey in [10] and its references to previous related surveys provide an overview of privacy and security for multi-agent machine learning. These surveys point to numerous privacy threats against distributed learning systems and are closely related to our privacy goals. Furthermore, these surveys refer to federated learning, secure multi-party computation, and differential privacy as three prominent defensive mechanisms for general machine learning tasks. We incorporate all three mechanisms in the design of PE-VDN.

Next, we review existing works in designing privacy-aware deep reinforcement learning agents. The single-agent deep reinforcement learning algorithm in [23] is a differentially private algorithm that protects the confidentiality of the agent's rewards. The work in [22] develops a single-agent tabular reinforcement learning algorithm that satisfies joint differential privacy—a relaxation of conventional differential privacy. As opposed to the first work, we consider the multi-agent setup, protect the agents' environment interaction data which includes rewards, and instead of developing a customized differential privacy mechanism that perturbs the objective function, we use verified and open-source libraries. Compared with the second algorithm, we consider a multi-agent setup, take continuous observations into account, and do not use differential privacy relaxations. The single-agent algorithm in [30] generalizes enforcing joint differential privacy to continuous state and action spaces; however, it is restricted to linear function approximators as opposed to this work's use of deep neural networks.

The decentralized multi-agent deep reinforcement learning algorithms in [14] and [26] closely relate to the privacy goals that we consider. However, the algorithms do not consider team rewards and assume that the agents are rewarded individually. Moreover, the algorithms do not address the privacy risks associated with the inter-agent communication frameworks that they propose.

## IV. Privacy-Engineering Vanilla VDN

We develop three privacy-engineering techniques to redesign Vanilla VDN's information flows and satisfy our privacy objectives with respect to environment interaction data. In this section, we describe each of the three privacy-engineering techniques that we use to develop PE-VDN.

### A. Decentralized Training

The Vanilla VDN algorithm's update rule in (8) requires the loss function's gradients with respect to the parameters of all neural network branches within the function approximator. Recall that, in Vanilla VDN, these branches are dedicated to specific agents for decentralized execution. Evaluating the gradient of the loss function in (7), for every branch $\theta_i$ and minibatch sample $e$, we can write

$$\frac{\partial \ell_{\text{VDN}}(e;\theta_1,\ldots,\theta_N)}{\partial \theta_i} = \underbrace{-2\left(r + \sum_{i\in\mathcal{N}} \left(\gamma \max_{a'} Q_i\left(\tau_i', a'; \theta_i^-\right) - Q_i\left(\tau_i, a_i; \theta_i\right)\right)\right)}_{A} \cdot \underbrace{\frac{\partial Q_i(\tau_i, a_i; \theta_i)}{\partial \theta_i}}_{B}. \tag{9}$$

From the gradient expression and its separation into terms A and B, we can observe that the A term is the only factor that couples the gradients of the branches and is a function of all of the agents' environment interaction data. The B term, however, is the gradient of the $i^{\text{th}}$ branch, and computing it only requires the parameters of that branch and its dedicated agent's environment interaction data.

In PE-VDN, the agents maintain and train their dedicated branch of the VDN themselves. They compute their local gradients by cooperating with other agents to compute the A term in (9) together, which is the coupling term that accounts for multi-agent coordination. We now show how the agents can cooperate to compute the A term without sharing their environment interaction data.

We require every agent $i$ to compute the message

$$m_i = \gamma \max_{a'} Q_i\left(\tau_i', a'; \theta_i^-\right) - Q_i\left(\tau_i, a_i; \theta_i\right) \tag{10}$$

and share it with all other agents. Each $m_i$ value can be computed using the $i^{\text{th}}$ agent's environment interaction data and the current parameters of its local $Q_i$ function. If every agent broadcasts its message $m_i$ to the other agents and locally computes its B term, it can then update its $Q_i$ function just as Vanilla VDN would have updated that agent's dedicated branch. The resulting distributed computation scheme eliminates the need for sharing environment interaction data with a central node.

### B. Privacy-Preserving Multi-Party Summation

Although the summation in the A term can be easily computed by requiring every agent to broadcast their $m_i$ messages, doing so may lead to unintentional information leakage about the agents' environment interaction data. The $m_i$ values are functions of their corresponding agent's privacy-sensitive environment interaction data and correlate with them. We now show how the agents can compute the desired summation in term A in (9) while hiding their $m_i$ values from one another.

We use a privacy-preserving multi-party computation technique called secret sharing [17] to compute the A term while hiding the underlying $m_i$ values. Secret sharing refers to the process of dividing a secret into $n$ pieces such that the observation of the pieces reveals no information about the underlying secret unless a sufficient number of them are available. Precisely, a $(k, n)$-secret sharing of $s$ guarantees that observers with access to up to $k-1$ shares can learn no information about $s$.

Additive secret sharing is an efficient $(n, n)$-secret sharing scheme [24] that satisfies additive homomorphism, i.e., the summation of the shares equals the summation of the secrets under the guarantee that the individual shares reveal no information about the secret. In computing the A term in the gradient expression in (9), a privacy-preserving summation protocol based on additive secret sharing can be used to protect the agents' contribution to the summation. More details about how we integrate additive secret sharing with decentralized training can be found in the extended version of this paper [5].

### C. Training with Differential Privacy

When the agents choose actions based on their neural network's action-value predictions, they may unintentionally leak information about their environment interaction data as well. It has been observed that the predictions of neural networks could mirror specific relationships in the training dataset that were not intended to be exposed [28]. For example, text-generative neural networks may complete certain input prompts with memorized phrases within their training datasets [8]. Similarly, the actions of deep reinforcement learning agents—including independent training algorithms in Co-MARL—may reveal certain sensitive characteristics of their training experience as well [12].

A machine learning algorithm that trains a neural network must map a training dataset to a set of parameters that determine the neural network's weights and biases. Under fixed hyperparameters and pseudo-randomization seeds, training algorithms are deterministic mappings. These deterministic mappings may produce outputs that external observers can accurately associate with certain training data. Privacy-enhancing techniques based on differential privacy can disrupt inference privacy threats against the training datasets [28]. Differentially private training algorithms can provably generate model parameters that are approximately statistically indistinguishable from those trained with datasets that differ in only one element. This guarantee establishes a plausible deniability argument against the accuracy of privacy threats that base their attacks on the learned parameters of a neural network or its predictions.

The DP-SGD algorithm [1] is a differentially private supervised learning algorithm for neural networks and enforces differential privacy by repeatedly injecting Gaussian noise into the gradients. The algorithm's so-called Moments Accountant subroutine tracks the differential privacy level of the composition of the iterations throughout the training.

As opposed to typical gradient descent algorithms that draw fixed-length minibatches, DP-SGD uses Poisson sampling, which refers to a sampling method that chooses each of the minibatch elements with a fixed probability. Moreover, DP-SGD clips the gradients to a fixed threshold $C$. That is, with $g_x$ denoting the gradient for sample $x$, DP-SGD uses the mapping $\text{Clip}_C(g_x) := g_x \cdot \max\left(1, \|g_x\|_2/C\right)^{-1}$.

We integrate DP-SGD with our earlier decentralization and the privacy-preserving multi-party summation protocol contributions as follows: instead of performing Poisson sampling on a static labeled dataset as in supervised learning, we perform Poisson sampling on the stream of environment interaction data that gets loaded onto the agents' replay buffers. The integration of the DP-SGD concludes the design of PE-VDN.

The application of DP-SGD to the contents of the agents' replay buffers makes the existing differential privacy analysis of DP-SGD ill-suited to PE-VDN. Specifically, the differential privacy analysis of the DP-SGD algorithm via the Moments Accountant method [1] only applies to the contents of the replay buffer for a single iteration, not the agents' entire collection of environment interaction data. As the last remaining puzzle piece, we now state a theorem that computes the differential privacy level of PE-VDN.

*Theorem 1:* Fix a Poisson sampling rate $q$, noise variance $\sigma^2$, and $\delta$ as DP-SGD parameters. Let $(\epsilon(T), \delta)$ be the differential privacy level that the Moments Accountant method computes for DP-SGD after $T$ iterations. Then, applying DP-SGD updates to PE-VDN with buffer size $B$, once the replay buffers have been fully populated achieves $(\epsilon(B), \delta)$-differential privacy.

*Proof:* By the end of the training, let $\mathcal{D} = \{d_1, \ldots, d_T\}$ denote the set of all sequences of environment interaction data that have been loaded onto an agent's replay buffer in sequential order. That is, $d_1$, is the first sequence, $d_2$ is the second sequence, and so forth. Define

$$
\begin{aligned}
D_1 &= \{d_1, d_2, \ldots, d_B\} \\
&\vdots \\
D_B &= \{d_B, d_{B+1}, \ldots, d_{2B-1}\}
\end{aligned}
$$

At the $i^{\text{th}}$ iteration, the algorithms apply DP-SGD to $\mathcal{D} \cap D_i$. The contents of each $D_i$ are at most used in the algorithm for $B$ iterations, and once a sequence $d \in \mathcal{D}$ is replaced in the replay buffer with a new one, it is no longer a part of the training. The composition theorem in [19] can leverage such special set overlaps to provide better differential privacy

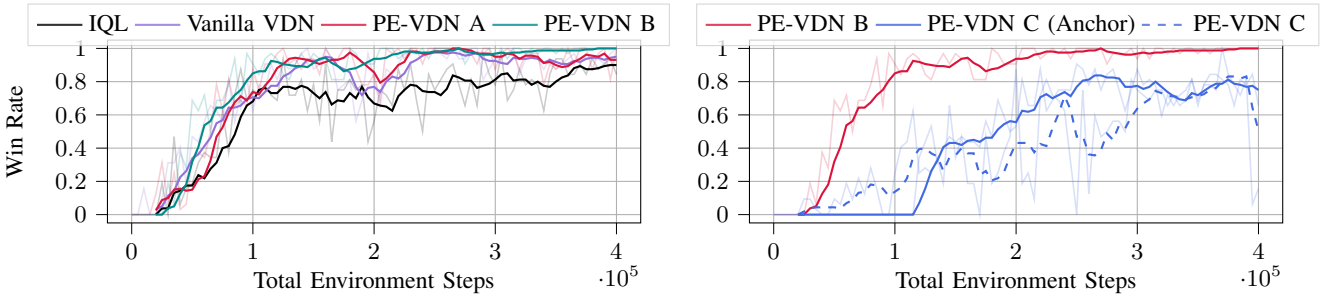


Fig. 1. Comparison of the win rate of different algorithms in SMAC's `3m` environment. The left plot compares win rates of Vanilla VDN, Independent Q-Learning (IQL), PE-VDN with only decentralized training (PE-VDN A), and PE-VDN with decentralized training and privacy-preserving multi-party summation (PE-VDN B) algorithms. The right plot demonstrates the win rate of PE-VDN with differential privacy (PE-VDN C). The solid line represents the anchor model's win rate and the volatile dashed line is the win rate of the running model.

bounds. Consider the overlap set

$$X := \left\{ I \subseteq \{1, \dots, T\} \mid \bigcap_{i \in I} D_i \neq \varnothing \right\}. \tag{11}$$

Each index set $I \in X$ has at most $B$ elements, which corresponds to $B$ consecutive sequences of environment interaction data. By Theorem 5 of [19], the differential privacy of the overall algorithm is upper bounded by that of the composition of $B$ iterations of DP-SGD updates, which can be computed via the Moments Accountant method. ■

## V. Numerical Experiments

Based on the three privacy-engineering techniques described above, we implement a Python library for PE-VDN[1]. We use the StarCraft Multi-Agent Competition (SMAC) suite [16], a machine learning application programming interface (API) for using Co-MARL to learn how to play StarCraft II. Our experiments take place in SMAC's `3m` environment in which three agents must cooperate to kill three pre-trained enemies.

In the experiments, we dissect the different components of PE-VDN to empirically evaluate their effects on team rewards. First, we consider the decentralization of Vanilla VDN's training. We do not expect that decentralization will affect performance because the distributed computation scheme in Section IV-A (PE-VDN A) computes the same gradients as Vanilla VDN. We also do not anticipate any significant performance drop due to the privacy-preserving multi-party summation protocol (PE-VDN B) because the only source of inaccuracy that it introduces is the encoding function's quantization error.

We periodically evaluate the agents' win rate during training and plot the results in Figure 1. The win rates confirm that Vanilla VDN, PE-VDN A, and PE-VDN B with `PRECISION` $= 5$ all perform similarly. The superior performance of Vanilla VDN, PE-VDN A, and PE-VDN B compared with Independent Q-Learning in Figure 1 indicates that the agents indeed benefit from cooperation.

In the next experiment, we evaluate the effects of enforcing differential privacy via the DP-SGD algorithm. DP-SGD's injection of noise into the gradients may affect performance negatively. In the supervised learning setup where DP-SGD has been primarily studied, it has been reported that the performance of the models trained with DP-SGD is more sensitive to the choice of training hyperparameters than non-differentially private counterparts [13]. In particular, hyperparameter tuning may play a key role in model accuracy, training stability, and sample complexity [13]. Clipping gradients may add bias, perturbing the gradients may deflect the parameters away from local minima and destabilize training, and achieving meaningful $\epsilon$ and $\delta$ values—$\epsilon < 3$ and $\delta < 1/n^{1.1}$, where $n$ is the training dataset's size [13]—may require large training datasets.

In our implementations, we use the Opacus [27] toolbox which is an open-source library for DP-SGD. We use Opacus' built-in Moments Accountant method to track PE-VDN's differential privacy levels as stated in Theorem 1. Without differential privacy, we used the Adam optimizer with a learning rate of $5 \cdot 10^{-4}$. In light of the guidelines in [13], we found that using the SGD optimizer with weight decay $0.01$, momentum $0.9$, and a significantly higher learning rate $5 \cdot 10^{-3}$ performs better for DP-SGD. Moreover, we use an *anchoring technique* to improve the training's stability further. In this technique, once the periodic win-rate evaluation returns rates that are beyond a specific threshold, we save the parameters as an anchor model. Then we repeatedly increase the threshold and penalize the rest of the training with the current parameters' distance from the anchor. With these techniques, Figure 1 shows the performance of PE-VDN under $(2.90, 4.9 \cdot 10^{-4})$-differential privacy and benchmarks it with PE-VDN B with Poisson sampling.

## VI. Conclusions & Future Work

In this work, we proposed the PE-VDN algorithm which incorporates three privacy-engineering techniques to protect the confidentiality of the agents' environment interaction data in Vanilla VDN. In particular, we re-engineered the data flows of Vanilla VDN to achieve the following: decentralized training without compromising multi-agent coordination, privacy-preserving inter-agent communication and computation, and differentially private neural network training.

[1]All codes and instructions are provided here.

For future work, we aim to develop similar decentralization and privacy-preserving communication schemes for more sophisticated centralized training and decentralized execution algorithms such as QMIX [15] and QTRAN [20]. These algorithms improve upon the performance of the VDN algorithm by relaxing the additivity assumption in (6). In particular, the QTRAN algorithm only requires decentralizability as stated in (5).

## REFERENCES


[1] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016. doi:10.1145/2976749.2978318.

[2] Szilárd Aradi. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 2022. doi:10.1109/TITS.2020.3024655.

[3] Giang Dao and Minwoo Lee. Relevant experiences in replay buffer. In *IEEE Symposium Series on Computational Intelligence, SSCI*. IEEE, 2019. doi:10.1109/SSCI44817.2019.9002745.

[4] J. Alexander Fax and Richard M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 2004. doi:10.1109/TAC.2004.834433.

[5] Parham Gohari, Matthew Hale, and Ufuk Topcu. Privacy-engineered value decomposition networks for cooperative multi-agent reinforcement learning. 2023. URL: https://corelab.mae.ufl.edu/papers/CDC2023_PrivacyEngineeredVDN_Extended.pdf.

[6] Shen Guicheng and Wang Yang. Review on Dec-POMDP model for MARL algorithms. In *Smart Communications, Intelligent Algorithms and Interactive Methods: Proceedings of 4th International Conference on Wireless Communications and Applications ICWCA*. Springer, 2022. doi:10.1007/978-981-16-5164-9_5.

[7] Trung Ha, Tran Khanh Dang, Tran Tri Dang, Tuan Anh Truong, and Manh Tuan Nguyen. Differential privacy in deep learning: An overview. In *2019 International Conference on Advanced Computing and Applications, ACOMP*. IEEE Computer Society, 2019. doi:10.1109/ACOMP.2019.00022.

[8] Bargav Jayaraman, Esha Ghosh, Huseyin A. Inan, Melissa Chase, Sambuddha Roy, and Wei Dai. Active data pattern extraction attacks on generative language models. 2022. arXiv:2207.10802, doi:10.48550/arXiv.2207.10802.

[9] Yijing Li, Xiaofeng Tao, Xuefei Zhang, Junjie Liu, and Jin Xu. Privacy-preserved federated learning for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 2022. doi:10.1109/TITS.2021.3081560.

[10] Chuan Ma, Jun Li. Kang Wei, Bo Liu, Ming Ding, Long Yuan, Zhu Han, and H. Vincent Poor. Trusted AI in multi-agent systems: An overview of privacy and security for distributed learning. 2022. arXiv:2202.09027.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015. URL: https://doi.org/10.1038/nature14236.

[12] Xinlei Pan, Weiyao Wang, Xiaoshuai Zhang, Bo Li, Jinfeng Yi, and Dawn Song. How you act tells a lot: Privacy-leaking attack on deep reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2019. URL: http://dl.acm.org/citation.cfm?id=3331715.

[13] Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H. Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Thakurta. How to dp-fy ML: A practical guide to machine learning with differential privacy. 2023. arXiv:2303.00654, doi:10.48550/arXiv.2303.00654.

[14] Chao Qu, Shie Mannor, Huan Xu, Yuan Qi, Le Song, and Junwu Xiong. Value propagation for decentralized networked deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/8a0e1141fd37fa5b98d5bb769ba1a7cc-Abstract.html.

[15] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 2020. URL: http://jmlr.org/papers/v21/20-081.html.

[16] Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob N. Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. 2019. URL: http://dl.acm.org/citation.cfm?id=3332052.

[17] Adi Shamir. How to share a secret. *Communications of the ACM*, 1979. doi:10.1145/359168.359176.

[18] Piyush K. Sharma, Rolando Fernandez, Erin G. Zaroukian, Michael R. Dorothy, Anjon Basak, and Derrik E. Asher. Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training. In *Artificial intelligence and machine learning for multi-domain operations applications III*. SPIE, 2021.

[19] Josh Smith, Hassan Jameel Asghar, Gianpaolo Gioiosa, Sirine Mrabet, Serge Gaspers, and Paul Tyler. Making the most of parallel composition in differential privacy. *Proceedings of Privacy Enhancing Technologies*, 2022. doi:10.2478/popets-2022-0013.

[20] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Hostallero, and Yung Yi. QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, Proceedings of Machine Learning Research. PMLR, 2019. URL: http://proceedings.mlr.press/v97/son19a.html.

[21] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2018. URL: http://dl.acm.org/citation.cfm?id=3238080.

[22] Giuseppe Vietri, Borja Balle, Akshay Krishnamurthy, and Zhiwei Steven Wu. Private reinforcement learning with PAC and regret guarantees. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, Proceedings of Machine Learning Research. PMLR, 2020. URL: http://proceedings.mlr.press/v119/vietri20a.html.

[23] Baoxiang Wang and Nidhi Hegde. Privacy-preserving q-learning with functional noise in continuous spaces. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/6646b06b90bd13dabc11ddba01270d23-Abstract.html.

[24] Lizhi Xiong, Wenhao Zhou, Zhihua Xia, Qi Gu, and Jian Weng. Efficient privacy-preserving computation based on additive secret sharing. 2020. arXiv:2009.05356.

[25] Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. 2020. arXiv:2011.00583.

[26] Yujian Ye, Yi Tang, Huiyu Wang, Xiao-Ping Zhang, and Goran Strbac. A scalable privacy-preserving multi-agent deep reinforcement learning approach for large-scale peer-to-peer transactive energy trading. *IEEE Transactions on Smart Grid*, 2021. doi:10.1109/TSG.2021.3103917.

[27] Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-friendly differential privacy library in pytorch. 2021. arXiv:2109.12298.

[28] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 2021. doi:10.1016/j.knosys.2021.106775.

[29] Xiaoyu Zhang, Chao Chen, Yi Xie, Xiaofeng Chen, Jun Zhang, and Yang Xiang. A survey on privacy inference attacks and defenses in cloud-based deep neural network. *Computer Standards & Interfaces*, 2023. doi:10.1016/j.csi.2022.103672.

[30] Xingyu Zhou. Differentially private reinforcement learning with linear function approximation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2022. doi:10.1145/3508028.