The Dissertation Committee for Parham Gohari
certifies that this is the approved version of the following dissertation:

Engineering Artificial Intelligence Systems for Privacy

Committee:

Ufuk Topcu, Supervisor

Haris Vikalo

Sandeep Chinchali

Mathew Hale

Adam Klein

Engineering Artificial Intelligence Systems for Privacy

by
Parham Gohari

Dissertation

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

The University of Texas at Austin
August 2023

Epigraph

What starts here changes the world.

—University of Texas at Austin

Acknowledgments

I extend my heartfelt gratitude to all those who supported and encouraged me throughout this remarkable journey. My deepest thanks to my supervisor, Ufuk Topcu, for allowing me the freedom to explore my research interests and guiding me with unwavering support. I am truly appreciative of my dissertation committee for generously sharing their insights and offering valuable feedback that greatly influenced my work. I would like to express a special thank you to Matthew Hale, whose introduction to the world of privacy and mentorship have been instrumental in shaping my path. Thank you, Adam Klein and the Strauss Center for International Security and Law for their role in broadening my perspective on privacy beyond the realm of engineering. To my friends and family, your unwavering support has been a constant source of inspiration and strength, making this journey more meaningful. My gratitude knows no bounds, and I am truly honored to have had the opportunity to learn and grow in the presence of such remarkable individuals.

Abstract

Engineering Artificial Intelligence Systems for Privacy

Parham Gohari, PhD
The University of Texas at Austin, 2023

SUPERVISOR: Ufuk Topcu

This dissertation addresses the increasing need for privacy-aware artificial intelligence (AI) systems and investigates engineering approaches that strike a balance between maximizing performance and minimizing potential privacy threats. The privacy analysis methodology followed throughout this study uses the Contextual Integrity Theorem's framing of privacy as "appropriate information flows" to find a common vocabulary between the social concept of privacy and mathematical AI algorithms. The analysis begins with a technical privacy policy that describes the information flows that are appropriate for a given AI task. Once these permissible information flows are determined, the next step is to develop engineering approaches to optimize performance within the confinements of the technical privacy policy, which we refer to as privacy engineering. We study privacy engineering for AI systems in two sequential decision-making domains, namely policy synthesis and reinforcement learning. In both domains, agents must strategize their decisions to achieve an ultimate long-term goal, such as a robot navigating to a predetermined location. For policy synthesis, we study privacy engineering in the context of Markov decision processes (MDPs), which are abstract models of an environment consisting of states, actions, transition probabilities, and rewards. The privacy priority considered for this problem is to protect the confidentiality of the MDP's transition probabilities.

Such a privacy priority is particularly relevant if the disclosure of the environment model to unauthorized parties could be harmful, such as the models that businesses develop to predict competitive market trends. For this problem, we identify differential privacy as an appropriate technical mechanism to achieve the set privacy goals and make two main contributions. First, we introduce the Dirichlet mechanism for enforcing differential privacy on simplex-valued data—which includes transition probabilities in MDPs with finite states and actions. Then, we use the Dirichlet mechanism to develop a differentially private policy synthesis algorithm. For the reinforcement learning problem, we study privacy engineering in the context of cooperative multi-agent reinforcement learning in which a team of agents must learn a common task through trial and error. For this problem, we assume that information disclosures about the agents' individual interactions with the environment violate privacy. We demonstrate that numerous existing algorithmic solutions rely on sharing environment interactions. Consequently, we introduce alternative privacy-engineered algorithms that establish permissible data-sharing frameworks according to the set technical privacy policy. The contributions of this dissertation demonstrate that privacy and AI can indeed be reconciled via privacy engineering. The findings highlight future research opportunities to design and implement AI algorithms with privacy as a priority.

Table of Contents

List of Tables 10
List of Figures 11
Chapter 1: Dirichlet Mechanism 14
1.1 Introduction 14
1.2 Preliminaries 18
1.2.1 Notation 18
1.2.2 Differential Privacy 18
1.2.3 Dirichlet Mechanism 19
1.3 Dirichlet Mechanism for Differential Privacy of Identity Queries . . . 20
1.3.1 Computing δ 20
1.3.2 Computing ϵ 26
1.4 Accuracy Analysis 32
1.5 Conclusion 33
Chapter 2: Privacy-Preserving Policy Synthesis 34
2.1 Introduction 34
2.2 Related Works 37
2.3 Preliminaries 38
2.4 Privacy-Engineered Policy Synthesis Algorithm 39
2.5 Cost of Privacy 41
2.5.1 Finite-Horizon MDPs 43
2.5.2 Infinite-Horizon MDPs 47
2.6 Computational Complexity 50
2.6.1 Finite-Horizon MDPs 51
2.6.2 Infinite-Horizon MDPs 51
2.7 Numerical Results 52
2.8 Conclusion 54
Chapter 3: Impact of Neural Network Architecture on Vulnerability to Membership Inference Attacks 55
3.1 Introduction 55
3.2 Preliminaries 59
3.2.1 Recurrent vs. Feed-Forward Architecture 59
3.2.2 RNN Applications Considered 60

3.3 Methods . . . 62
3.3.1 Threat Model and Assumptions . . . 62
3.3.2 Designing MIAs . . . 64
3.3.3 Connection with the Existing MIAs . . . 67
3.4 Vulnerability to Privacy Threats . . . 68
3.4.1 Vulnerability Factors . . . 68
3.4.2 Experimental Setup . . . 70
3.4.3 Numerical Results . . . 72
3.5 Preempting Privacy Threats . . . 78
3.5.1 Defense via Regularization . . . 78
3.5.2 Defense via Differential Privacy . . . 80
3.6 Conclusion . . . 90
Chapter 4: Privacy Engineering Co-MARL Algorithms . . . 91
4.1 Introduction . . . 91
4.2 Preliminaries . . . 95
4.2.1 Dec-POMDP . . . 95
4.2.2 Deep Q-Learning . . . 96
4.2.3 Multi-Agent DQN . . . 97
4.3 Engineering VDN, QMIX, and QTRAN for Privacy . . . 98
4.3.1 VDN . . . 98
4.3.2 QMIX . . . 105
4.3.3 QTRAN . . . 111
4.4 Enforcing Differential Privacy . . . 115
4.5 Numerical Results . . . 118
4.6 Conclusion . . . 122
Chapter 5: Conclusions and Future Research Opportunities . . . 123
Works Cited . . . 125

List of Tables

3.1 Empirical sensitivity estimated by Algorithm 3. 90

List of Figures

1.1 An example where $|W| = 3$ to compare the values of ϵ in Theorem 1.6 with those computed based on the simplifications proposed in Lemma 1.7. At each level of δ, first γ is optimized according to the optimization problem in (1.23), then the optimal γ is substituted in the expressions for the original and approximated values. Left: $\eta = 0.15$, $\bar{\eta} = 0.15$, and $k = 6.7$. Middle: $\eta = 0.20$, $\bar{\eta} = 0.20$, and $k = 5.1$. Right: $\eta = 0.25$, $\bar{\eta} = 0.25$, and $k = 4.1$ 32

2.1 From left to right: the first two plots show all the value functions that are used to compute and validate the cost of privacy for Examples 1 and 2. The first plot corresponds to Example 1 and the second plot shows the results for Example 2. The third plot shows the cost of privacy itself for both examples. 53

2.2 The corporation's investment model with four possible startups to acquire, given as an MDP. 53

3.1 Schematic of the execution of an MIA. 63

3.2 Left: a schematic of training shadow models. The skull above the shadow model's training dataset and learning algorithm indicates that the two components may differ between the victim and the attacker's side. Right: training the MIA using the output of the shadow model and side-information. 65

3.3 The MiniGrid-MultiRoom-N4-v0 environment. The agent (red triangle) must find the goal state (green square) while observing the highlighted box that surrounds it. The environment consists of multiple floor maps, two of which are shown above. 71

3.4 Comparing the vulnerability of RNNs and FFNNs to MIAs in three representative machine learning tasks: image classification (left), machine translation (center), and deep reinforcement learning (right). The first row plots training and validation performance vs. number of epochs; the second row plots the average prediction entropy vs. number of epochs; and the third row plots attack accuracy vs. number of epochs. 73

3.5 An illustration of the decision boundaries of MIAs with respect to cross-entropy loss and entropy of the confidence scores. The top row shows the results for RNNs and the bottom row shows the results for FFNNs. 75

3.6 Comparing model memorization in RNNs and FFNNs. The top row plots the training and validation performance of the models when trained sequentially with a collection of ordered batches of training data. The saw-tooth pattern in the training performance is common in sequential training of machine learning models and is due to the *catastrophic interference* phenomenon (McCloskey and Cohen, 1989). The validation lines are smoothed out by measuring performance at the end of every 10 epochs. The bottom row plots the attack accuracy of the MIAs with respect to the individual epochs in the order in which they were introduced to the victims during training. 77

3.7 Effects of regularization on attack accuracy. From the left to the right column: image classification, machine translation, and deep reinforcement learning. The MIAs are trained separately for each regularization value tested and the shadow models use the same parameters as their victims. 78

3.8 The privacy-utility trade-off of DP-SGD and GPM. The top row corresponds to the utility loss caused by DP-SGD and the second row corresponds to the utility loss caused by the GPM. The RNNs consistently trade off more utility than FFNNs for both DP-SGD and GPM. 89

4.1 The computational graph of the forward pass in Vanilla VDN with two agents. Parameters annotated with hats do not carry gradient attributes. Q_{logits} denotes a vector that contains the action values for all available actions. one_hot$(\cdot)$ converts an integer index to a one-hot indicator vector. The computational graph of $Q_{\text{target}} = r + \gamma \sum_{i \in \mathcal{N}} \max_a Q_i\left(\tau_i', a; \theta_{Q_i}^-\right)$ is omitted because its gradients are detached. 100

4.2 The computational graph of the backward pass in Vanilla VDN with two agents. 101

4.3 The forward pass of the agents in PE-VDN. $Q_{\text{sum}} = \sum_{i \in \mathcal{N}} Q_i\left(\tau_i, a_i; \theta_{Q_i}\right)$ and $Q_{\text{target}} = r + \gamma \sum_{i \in \mathcal{N}} \max_a Q_i\left(\tau_i', a; \theta_{Q_i}^-\right)$, which use the first and the second part of the m_i-messages, respectively. 102

4.4 The backward pass of the agents in PE-VDN. Each agent's backward pass in PE-VDN is equivalent to the backward pass of its corresponding branch in Vanilla VDN. 103

4.5 A two-agent example of the computational graph of Vanilla QMIX's forward pass of the loss function. In this example, the mixing function consists of two layers, each of which has its upstream hypernetworks with parameters $\theta_h^{[1]}$ and $\theta_h^{[2]}$. The first layer of the mixing function has the exponential linear unit (elu) as its activation function. The computational path to $Q_{\text{target}} = r + \max_{\boldsymbol{a}'} Q_{\text{tot}}(\boldsymbol{\tau}', \boldsymbol{a}')$ is similar to that of Q_{tot} but not depicted due to its lack of contribution to the gradients. 107

4.6 Incorporating encoders to process partial observations for the input of hypernetworks instead of processing global state information. 108

4.7 An example of the computational graph that each agent creates for the forward pass of the loss function in PE-QMIX. In this example, the mixing function comprises two layers and the first layer is activated by the elu function. 110
4.8 The computational graph of the forward pass of ℓ_{td} in PE-VDN. . . . 113
4.9 The computational graph of the forward pass of ℓ_{opt} in PE-QTRAN. . 114
4.10 The computational graph of the forward pass of ℓ_{nopt} in PE-QTRAN. 115
4.11 A screenshot of SMAC's `3m` environment. 118
4.12 Comparison of the team's win rate in IQL and PE-VDN. 119
4.13 Comparison of the team's win rate in IQL and PE-QMIX. 120
4.14 Comparison of the team's win rate in IQL and PE-QTRAN. 120
4.15 Comparison of the team's win rate in PE-VDN under $(2.90, 4.9 \cdot 10^{-4})$-differential privacy and PE-VDN without differential privacy protections. The win rates correspond to the anchor models. 121
4.16 Comparison of the team's win rate in PE-QMIX under $(2.90, 4.9 \cdot 10^{-4})$-differential privacy and PE-QMIX without differential privacy protections. The win rates correspond to the anchor models. 122

Chapter 1: The Dirichlet Mechanism[1]

Abstract

This chapter introduces a novel privacy-engineering technique for vectors that belong to the unit simplex. Such vectors can be seen in many sequential decision-making applications, such as the transition probabilities and the policy in Markov decision process. We use differential privacy as the underlying mathematical framework for these developments. The introduced mechanism is a probabilistic mapping that maps a vector within the unit simplex to the same domain according to a Dirichlet distribution. The mechanism is well-suited for inputs within the unit simplex because it always returns a privatized output that is also in the unit simplex. We establish a trade-off between the strength of privacy and the variance of the mechanism output, and we introduce a concentration parameter to balance the trade-off between them. This chapter lays the groundwork for the privacy-preserving policy-synthesis algorithm that we present in Chapter 2.

1.1 Introduction

In many decision-making problems, a policy-maker forms a control policy based on privacy-sensitive information, such as personal information collected from the individuals in a network. The collection and processing of such data often raises

[1]The research presented in this chapter has been previously published in the following papers: (Gohari et al., 2020b, 2021). Parham Gohari formulated the problem, derived the theoretical results, conducted the numerical experiments, and wrote the paper.

privacy concerns (Chatel et al., 2021). In some applications, privatizing sensitive data has been achieved by adding carefully calibrated noise to sensitive data and functions thereof (Wang et al., 2017; Nozari et al., 2018; Hale et al., 2018). These noise-additive approaches are well-suited to some classes of numerical data, though sensitive data may take a form ill-suited to them. For example, developments in (Jones et al., 2019) explored symbolic control systems in which additive noise cannot be meaningfully implemented.

In this chapter, we privatize data inputs that belong to the unit simplex, i.e., the set of vectors with non-negative entries that sum to one. Such vectors are seen in many sequential decision-making problems. For example, in MDPs, the goal is to find a total-reward-maximizing policy (Puterman, 1994). Finite state and action sets give rise to discrete, finitely supported transition probability distributions, which can be formalized as vectors with non-negative entries summing to one. Processing transition probabilities in applications such as autonomous driving (Brechtel et al., 2011a) and smart power grids (Misra et al., 2013a) may lead to unauthorized disclosure of the underlying consumers' behavior patterns. Thus, there is a need to handle such privacy-sensitive transition probabilities with care, and this use represents one application of privatizing sensitive data in the unit simplex. Existing noise-additive approaches will not, in general, produce a privatized vector in the unit simplex, and we therefore propose a new approach to privacy for this context.

In this chapter, we use differential privacy as the underlying mathematical framework for privacy. Differential privacy, first introduced in (Dwork et al., 2006), is designed to protect the exact values of sensitive pieces of data, while preserving their usefulness in aggregate statistical analyses. Two desirable properties of differential privacy are (i) that it is immune to post-processing (Dwork and Roth, 2014), in the sense that arbitrary post-hoc transformations of privatized data do not weaken its privacy guarantees, and (ii) that it is robust to side information, in that gaining additional information about data-producing entities does not weaken its privacy guarantees by much (Kasiviswanathan and Smith, 2014). As a result, differential

privacy has been frequently used as the mathematical formulation of privacy in both computer science and, more recently, in control theory (Nissim and Wood, 2018; Cortés et al., 2016; Han and Pappas, 2018).

As one of the main contributions of this dissertation, we introduce a differential privacy mechanism that privatizes a vector within the unit simplex. By a mechanism, we mean a probabilistic mapping from some pre-defined domain to a pre-defined range that is used to privatize sensitive data. This chapter develops a novel mechanism using the Dirichlet distribution, and we therefore call it the Dirichlet mechanism. The Dirichlet distribution is a multivariate distribution supported on the unit simplex, which makes it a natural choice for this setting because its outputs are always elements of the unit simplex.

In our developments, we use probabilistic differential privacy, which is known to imply that the conventional form of differential privacy also holds (Götz et al., 2012). Then, we show that the Dirichlet mechanism satisfies probabilistic differential privacy for identity queries. By an identity query, we mean privatizing a single vector within the unit simplex. In the course of proving these privacy guarantees, based on the assumptions we provide, we prove the log-concavity of the cumulative distribution function of a Dirichlet distribution. The proof that we present may be of independent interest in ongoing research on convexity analysis of special functions such as (Karp, 2016). In this vein, we prove a generalization of Theorem 6 of (Prékopa, 1973) which has been used in other works for stochastic programming (Prékopa, 1971).

Following the convention in the differential privacy literature, we also analyze the accuracy of the output of the mechanism (Dwork and Roth, 2014; McSherry and Talwar, 2007). In particular, we evaluate the accuracy of the Dirichlet mechanism in terms of the expected value and the variance of its outputs. Similar to additive noise methods, the Dirichlet mechanism output has the same expected value as its input, which implies that its privatized outputs obey a distribution centered on the underlying sensitive data. We show that there exists a trade-off between the privacy

and the variance of the output of the mechanism. The derived expression for the output variance shows how to tune the worst-case variance by scaling the input by a parameter that we introduce in the mechanism definition.

We emphasize that additive noise privacy mechanisms are ill-suited to privacy on the unit simplex because they add noise of infinite support. As a result, such mechanisms will output a vector that does not belong to the unit simplex; attempting to normalize the noise would result in its distribution not being one known to provide differential privacy. It is for these reasons that we develop the Dirichlet mechanism. Although its form appears quite different from existing mechanisms, they are related through membership in a broad class of probability distributions. In particular, the Laplacian, Gaussian, and exponential mechanisms all use distributions belonging to a parameterized family of exponential distributions. The outputs of the Dirichlet distribution are equivalent to a normalized vector of i.i.d. exponential random variables, which means their distribution also belongs to the exponential family. This connection reveals why we should expect the Dirichlet mechanism to be well-suited to differential privacy, and the developments of this study formalize and confirm this intuition.

We also point out here that the exponential mechanism is another widely used differentilly private mechanism which can be used for sensitive data ill-suited to additive approaches (Dwork and Roth, 2014). However, the exponential mechanism can be computationally demanding to implement for privacy applications with many possible outputs. The output space here is the unit simplex, which contains uncountably many elements. The resulting complexity of such an implementation therefore makes it infeasible (Vadhan, 2017), especially in large dimensions, and we avoid it here.

1.2 Preliminaries

1.2.1 Notation

In this section we establish notation used throughout the chapter. We represent the real numbers by $\mathbb{R}$ and the positive reals by $\mathbb{R}_+$. For a positive integer n, let $[n] := \{1, \ldots, n\}$. We denote the unit simplex in $\mathbb{R}^n$ by $\Delta(n)$, where

$$\Delta(n) := \left\{ x \in \mathbb{R}_+^n \mid \sum_{i=1}^{n} x_i = 1, \text{ for all } i \in [n] \right\}.$$

We use $\Delta(n)^\circ$ to represent the interior of $\Delta(n)$. Letting $W \subseteq [n-1]$, with $|W| \geq 2$, we define the set

$$\Delta_{n,W}^{(\eta,\bar{\eta})} := \left\{ p \in \Delta(n)^\circ \mid \sum_{i \in W} p_i \leq 1 - \bar{\eta}, p_i \geq \eta \text{ for all } i \in W \right\}.$$

We impose the following assumption on η and $\bar{\eta}$. Assume that in $\Delta_{n,W}^{(\eta,\bar{\eta})}$, $\eta > 0$, $\bar{\eta} > 0$, and $\eta + \bar{\eta} < 1$.

Letting $p \in \mathbb{R}^n$, we use the notation $p_{(i,j)}$ to denote the vector $(p_i, p_j)^T \in \mathbb{R}^2$, where $(\cdot)^T$ is the transpose of a vector, and $p_{-(i,j)} \in \mathbb{R}^{n-2}$ to denote the vector p with i^{th} and j^{th} entries removed. $\mathbb{P}[\cdot]$ denotes the probability of an event. For a random variable, $\mathbb{E}[\cdot]$ denotes its expectation and $\mathrm{Var}[\cdot]$ denotes its variance. We use the notation $|\cdot|$ for the cardinality of a finite set. $||\cdot||_1$ denotes the 1-norm of a vector. We also use special functions

$$\Gamma(z) := \int_0^\infty x^{z-1} \exp(-x) \mathrm{d}x, \qquad z \in \mathbb{R}_+,$$

$$\mathrm{beta}(a,b) := \int_0^1 t^{a-1}(1-t)^{b-1} \mathrm{d}t, \qquad a, b \in \mathbb{R}_+.$$

1.2.2 Differential Privacy

Intuitively, differential privacy guarantees that two *nearby* inputs to a privacy mechanism will generate statistically similar outputs. In differential privacy, the notion of "nearby" is formally defined by an adjacency relation, and we define adjacency over the unit simplex as follows.

Definition 1.1. For a constant $b \in (0, 1]$, constants $\eta, \bar{\eta}$, and fixed set $W \subseteq [n-1]$, two vectors $p, q \in \Delta_{n,W}^{(\eta,\bar{\eta})}$ are said to be b-adjacent if there exist indices $i, j \in W$ such that $p_{-(i,j)} = q_{-(i,j)}$ and $||p - q||_1 \leq b$.

In words, two vectors are adjacent if they differ in two entries by an amount not more than b. Ordinarily, differential privacy considers sensitive data differing in a single entry, e.g., one entry in a database (Dwork and Roth, 2014). However, it is not possible to do so for an element of the unit simplex because changing only a single entry would violate the condition that vectors' entries sum to one. We therefore consider privacy with the above adjacency relation. Differential privacy itself is defined next.

Definition 1.2. (Probabilistic differential privacy; (Machanavajjhala et al., 2008)) Let $b \in (0, 1]$ and $W \subseteq [n-1]$ be given. Fix a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. A mechanism $\mathcal{M} : \Delta_{n,W}^{\eta,\bar{\eta}} \times \Omega \mapsto \Delta(n)$ is said to be probabilistically (ϵ, δ)-differentially private if, for all $p \in \Delta_{n,W}^{\eta,\bar{\eta}}$, we can partition the output space $\Delta(n)$ into two disjoint sets $\Omega_1(W), \Omega_2(W)$, such that $\mathbb{P}[\mathcal{M}(p) \in \Omega_2(W)] \leq \delta$, and for all $q \in \Delta_{n,W}^{\eta,\bar{\eta}}$ b-adjacent to p, and for all $x \in \Omega_1(W)$,

$$\log\left(\frac{\mathbb{P}[\mathcal{M}(p) = x]}{\mathbb{P}[\mathcal{M}(q) = x]}\right) \leq \epsilon.$$

Probabilistic differential privacy is known to imply conventional differential privacy (Machanavajjhala et al., 2008), and we will use probabilistic differential privacy as a means to show that conventional differential privacy holds.

1.2.3 Dirichlet Mechanism

The main contribution of this chapter is to present a differentially private mechanism that, without any need of projection, maps elements of $\Delta(n)$ to $\Delta(n)$. In order to do so, we first introduce the Dirichlet mechanism. A Dirichlet mechanism with concentration parameter $k \in \mathbb{R}_+$, denoted by Dir_k, takes as input a vector

$p \in \Delta(n)^\circ$ and outputs $x \in \Delta(n)$ according to the Dirichlet probability distribution function (PDF) centered on p, i.e.,

$$\mathbb{P}[\mathrm{Dir}_k(p) = x] = \frac{1}{\mathrm{B}(kp)} \prod_{i=1}^{n-1} x_i^{kp_i - 1} \left(1 - \sum_{i=1}^{n-1} x_i\right)^{kp_n - 1},$$

where

$$\mathrm{B}(kp) := \frac{\prod_{i=1}^{n} \Gamma(kp_i)}{\Gamma\left(k \sum_{i=1}^{n} p_i\right)} \tag{1.1}$$

is the multi-variate beta function.

We later use the parameter k to adjust the trade-off that we establish between the accuracy and the privacy level of the Dirichlet mechanism. Next, we establish the privacy guarantees that the Dirichlet mechanism provides.

1.3 Dirichlet Mechanism for Differential Privacy of Identity Queries

We begin by analyzing identity queries under the Dirichlet mechanism. Here, a sensitive vector p is directly input to the Dirichlet mechanism to make it approximately indistinguishable from other adjacent sensitive vectors. To show the level of privacy that holds, we first bound δ, then bound ϵ.

1.3.1 Computing δ

Fix $W \subseteq [n-1]$. In accordance with Definition 1.2, we partition the output space of the Dirichlet mechanism into two sets $\Omega_1(W), \Omega_2(W)$ defined by

$$\Omega_1(W) := \{x \in \Delta(n) \mid x_i \geq \gamma \text{ for all } i \in W\} \tag{1.2}$$

and

$$\Omega_2(W) := \Delta(n) \setminus \Omega_1(W), \tag{1.3}$$

where $\gamma \in (0, 1)$ is a parameter that defines these sets.

Our goal is to show that the Dirichlet mechanism output belongs to Ω_1 with high probability. Letting $p \in \Delta_{n,W}^{(\eta,\bar{\eta})}$. In the next lemma we show how to calculate $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1]$.

Lemma 1.1. *Let $W \subseteq [n-1]$ be a given set of indices which is used to construct $\Delta_{n,W}^{(\eta,\bar{\eta})}$, let $p \in \Delta_{n,W}^{(\eta,\bar{\eta})}$ and let*

$$\mathcal{A}_r := \left\{ x \in \mathbb{R}^{r-1} \;\middle|\; \sum_{i\in[r-1]} x_i \leq 1, x_i \geq \gamma \text{ for all } i \in W \right\},$$

for all $r \geq |W| + 1$. Then, for a Dirichlet mechanism with parameter $k \in \mathbb{R}_+$, we have that

$$\mathbb{P}\left[\mathrm{Dir}_k(p) \in \Omega_1\right] = \frac{\displaystyle\int_{\mathcal{A}_{|W|+1}} \left(\prod_{i\in W} x_i^{kp_i-1}\right)\left(1-\sum_{i\in W} x_i\right)^{k(1-\sum_{i\in W} p_i)-1} \prod_{i\in W} \mathrm{d}x_i}{\mathrm{B}(k\tilde{p}_W)},$$

where $\tilde{p}_W \in \Delta_{|W|+1}$ is equal to p after removing entries with indices outside W, with an additional entry equal to $1 - \sum_{i\in W} p_i$ appended as its final entry.

Proof. Without loss of generality, take $W = [|W|]$. In order to find $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1(W)]$, we need to integrate the Dirichlet PDF over the region $\mathcal{A}_n$. Therefore, we need to evaluate the $(n-1)$-fold integral

$$\frac{\displaystyle\int_{\mathcal{A}_n} \left(\prod_{i=1}^{n-1} x_i^{kp_i-1}\right)\left(1-\sum_{i=1}^{n-1} x_i\right)^{kp_n-1} \mathrm{d}x_{n-1}\ldots\mathrm{d}x_1}{\mathrm{B}(kp)}. \tag{1.4}$$

Using a method similar to the one adopted in (Rao and Sobel, 1980), let $y := \sum_{i=1}^{n-2} x_i$. Then we can rewrite (1.4) as

$$\frac{1}{\mathrm{B}(kp)} \int_{\mathcal{A}_{n-1}} \int_0^{1-y} \prod_{i=1}^{n-1} x_i^{kp_i-1}(1-y-x_{n-1})^{kp_n-1}\mathrm{d}x_{n-1}\ldots\mathrm{d}x_1. \tag{1.5}$$

Now let $u := \frac{x_{n-1}}{1-y}$ and take the inner integral with respect to u. Then (1.5) becomes

$$\frac{1}{\mathrm{B}(kp)} \int_{\mathcal{A}_{n-1}} \prod_{i=1}^{n-2} x_i^{kp_i-1}(1-y)^{k(p_{n-1}+p_n)-1} \int_0^1 u^{kp_{n-1}-1}(1-u)^{kp_n-1}\mathrm{d}u\ \mathrm{d}x_{n-2}\ldots\mathrm{d}x_1.$$

From the definition of the beta function, we have

$$\int_0^1 u^{kp_{n-1}-1}(1-u)^{kp_n-1}\mathrm{d}u = \mathrm{beta}(kp_{n-1}, kp_n).$$

Using the gamma function representation of beta functions, i.e.,

$$\mathrm{beta}(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)},\ a,b \in \mathbb{R}_+, \tag{1.6}$$

and (1.1), we find that $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1]$ is equal to

$$\frac{1}{\mathrm{B}(kp)}\frac{\Gamma(kp_{n-1})\Gamma(kp_n)}{\Gamma(k(p_{n-1}+p_n))}\int_{\mathcal{A}_{n-1}}\prod_{i=1}^{n-2}x_i^{kp_i-1}\left(1-\sum_{i=1}^{n-2}x_i\right)^{k(p_{n-1}+p_n)-1}\mathrm{d}x_{n-2}\ldots\mathrm{d}x_1.$$

Using the same trick, for the next step, let $y := \sum_{i=1}^{n-3} x_i$ and $u := \frac{x_{n-2}}{1-y}$. Then $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1]$ is equal to

$$\frac{1}{\mathrm{B}(kp)}\frac{\Gamma(kp_{n-2})\Gamma(kp_{n-1})\Gamma(kp_n)}{\Gamma(k(p_{n-2}+p_{n-1}+p_n))}\int_{\mathcal{A}_{n-2}}\prod_{i=1}^{n-3}x_i^{kp_i-1}\left(1-\sum_{i=1}^{n-3}x_i\right)^{k(p_{n-2}+p_{n-1}+p_n)-1}\mathrm{d}x_{n-3}\ldots\mathrm{d}x_1.$$

We continue to adopt the same change of variable strategy until we are left with an integral over the region $\mathcal{A}_{|W|+1}$, which concludes the proof. □

Lemma 1.1 shows that instead of an $(n-1)$-fold integral of the Dirichlet PDF, the computations can be reduced to a $|W|$-fold integral. However, the expression still depends on the input vector p, which is undesirable and generally incompatible with differential privacy. The reason is that (ϵ, δ)-differential privacy must be a guarantee for all adjacent input data and not for a specific data point. In the next lemma, we show that $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1(W)]$ is a log-concave function of p over the region $\Delta_{n,W}^{(\eta,\bar{\eta})}$, which we will use to derive a bound for δ that holds for all p of interest.

Lemma 1.2. *Let $W \subseteq [n-1]$ be a given set of indices which is used to construct $\Delta_{n,W}^{(\eta,\bar{\eta})}$ and let Dir_k be the Dirichlet mechanism with parameter k and input p. Then, $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1(W)]$ is a log-concave function of p over the domain $\Delta_{n,W}^{(\eta,\bar{\eta})}$.*

Proof. We first state a result from (Prékopa, 1973) which we later use to prove Lemma 1.2.

Lemma 1.3 (Theorem 3 in (Prékopa, 1973))**.** *Let $f_1, \ldots, f_k$ be non-negative and Borel measurable functions defined on $\mathbb{R}^n$ and let*

$$r(t) = \sup_{\lambda_1 x_1 + \cdots + \lambda_k = t} f_1(x_1) \ldots f_k(x_k), \quad t \in \mathbb{R}^n,$$

where $\lambda_1, \ldots, \lambda_k$ are positive constants satisfying the equality $\lambda_1 + \cdots + \lambda_k = 1$. Then, the function $r(t)$ also Borel measurable

$$\int_{\mathbb{R}^n} r(t)dt \geq \left(\int_{\mathbb{R}^n} f_1^{\frac{1}{\lambda_1}}(t)\mathrm{d}t \right)^{\lambda_1} \ldots \left(\int_{\mathbb{R}^n} f_k^{\frac{1}{\lambda_k}}(t)\mathrm{d}t \right)^{\lambda_k}.$$

We next review the definition of log-concave functions. A function $g : \mathbb{R}^n \mapsto \mathbb{R}$ is said to be log-concave if for all $x_1, x_2 \in \mathbb{R}^n$ and $\theta \in [0, 1]$, we have that

$$g(\theta x_1 + (1-\theta)x_2) \geq (g(x_1))^{\theta}(g(x_2))^{1-\theta}.$$

This condition is equivalent to

$$g(t) \geq \sup_{\theta u + (1-\theta)v = t} g(u)^{\theta} g(v)^{1-\theta}. \tag{1.7}$$

Note that g is log-concave if and only if $\log g$ is concave. Next, for $x \in \mathbb{R}^n$ and $p \in \Delta_{n,W}^{(\eta,\bar{\eta})}$ let $f : \mathbb{R}^n \times \Delta_{n,W}^{(\eta,\bar{\eta})} \mapsto [0, 1]$ be defined as

$$f_W(x, p) = \frac{\prod_{i \in W} x_i^{kp_i - 1} \left(1 - \sum_{i \in W} x_i\right)^{k(1 - \sum_{i \in W} p_i) - 1}}{B(kp_W)}.$$

For a fixed $p \in \Delta_{n,W}^{\eta,\bar{\eta}}$, let

$$f_1(x) := f_W(x, p).$$

The function $f_1(x)$ is the Dirichlet probability distribution function with parameter $a \in \mathbb{R}^W$ where $a := k\tilde{p}_W$. Since $p \in \Delta_{n,W}^{\eta,\bar{\eta}}$, we have that $a_i \geq 1$, for all $i \in [|W|]$.

Therefore, f_1 is a log-concave function [Equation 4.4 of (Prékopa, 1971)]. Then, by (1.7),

$$f(t_x, p) \geq \sup_{\alpha u_x + (1-\alpha) v_x = t_x} f(u_x, p)^{\alpha} f(v_x, p)^{1-\alpha}, \tag{1.8}$$

for all $p \in \Delta_{n,W}^{(\eta,\bar{\eta})}$, $t_x, u_x, v_x \in \mathbb{R}^n$ and $\alpha \in [0,1]$. Similarly, for a fixed $x \in \mathbb{R}^{|W|}$, let

$$f_2(p) := f_W(x, p).$$

Evaluating the Hessian of $\log f_2(p)$, let

$$\bar{\psi} := \psi^{(1)} \left(k \left(1 - \sum_{i \in [|W|]} x_i \right) \right),$$

where $\psi^{(1)}$ is the trigamma function. Then,

$$-\frac{(\nabla^2 \log f_2(p))_{i,j}}{k^2} = \begin{cases} \psi^{(1)}(kp_i) + \bar{\psi} & i = j, \text{and } i, j \in W \\ \bar{\psi} & i \neq j, \text{and } i, j \in W \\ 0 & i \notin W \text{and/or } j \notin W \end{cases}.$$

The trigamma function is positive for all positive arguments, and all arguments are positive here. Therefore, the Hessian matrix is a sum of two negative semi definite matrices and as a result, negative semi definite itself. The aforementioned argument results in log-concavity of $f_2(p)$. Therefore, using (1.7),

$$f(x, t_p) \geq \sup_{\beta u_p + (1-\beta) v_p = t_p} f(x, u_p)^{\beta} f(x, v_p)^{1-\beta} \tag{1.9}$$

for all $x \in \mathbb{R}^n$, $t_p, u_p, v_p \in \Delta_{n,W}^{(\eta,\bar{\eta})}$ and $\beta \in [0,1]$. Next, let $\lambda \in [0,1]$. Choose $\tilde{u}_x, \tilde{v}_x, \tilde{u}_p, \tilde{v}_p$ such that

$$\lambda \tilde{u}_x + (1-\lambda)\tilde{v}_x = t_x,$$
$$\lambda \tilde{u}_p + (1-\lambda)\tilde{v}_p = p.$$

Assigning u_x to x in (1.9), we find

$$\begin{aligned} f(u_x, p) &\geq \sup_{\beta u_p + (1-\beta) v_p = p} f(u_x, u_p)^{\beta} f(u_x, v_p)^{1-\beta} \\ &\geq f(u_x, \tilde{u}_p)^{\lambda} f(u_x, \tilde{v}_p)^{1-\lambda}. \end{aligned} \tag{1.10}$$

Similarly, we can write

$$\begin{aligned} f(v_x, p) &\geq \sup_{\beta u_p + (1-\beta) v_p = p} f(v_x, u_p)^{\beta} f(v_x, v_p)^{1-\beta} \\ &\geq f(v_x, \tilde{u}_p)^{\lambda} f(v_x, \tilde{v}_p)^{1-\lambda}. \end{aligned} \tag{1.11}$$

Revisiting (1.8), using (1.10) and (1.11), we can write

$$\begin{aligned} f(t_x, p) \quad &\geq \quad \sup_{\alpha u_x + (1-\alpha) v_x = t_x} f(u_x, p)^{\alpha} f(v_x, p)^{1-\alpha} \\ &\geq \quad \sup_{\lambda \tilde{u}_x + (1-\lambda) \tilde{v}_x = t_x} f(\tilde{u}_x, p)^{\lambda} f(\tilde{v}_x, p)^{1-\lambda} \\ &\geq \quad \sup_{\lambda \tilde{u}_x + (1-\lambda) \tilde{v}_x = t_x} \Big(f(\tilde{u}_x, \tilde{u}_p)^{\lambda^2} f(\tilde{v}_x, \tilde{v}_p)^{\lambda(1-\lambda)} \\ &\qquad f(\tilde{v}_x, \tilde{u}_p)^{(1-\lambda)\lambda} f(\tilde{v}_x, \tilde{v}_p)^{(1-\lambda)^2} \Big). \end{aligned} \tag{1.12}$$

The second line in (1.12) is true because the feasible region of the second optimisation problem is a subset of that of the first one. Note that

$$\lambda \tilde{u}_x + (1-\lambda)\tilde{v}_x = \lambda^2 \tilde{u}_x + \lambda(1-\lambda)\tilde{u}_x + \lambda(1-\lambda)\tilde{v}_x + (1-\lambda)^2 \tilde{v}_x.$$

Since $\lambda^2 + \lambda(1-\lambda) + \lambda(1-\lambda) + (1-\lambda)^2 = 1$, Theorem 1.3 applies. Therefore, for an arbitrary $\mathcal{A} \subseteq \mathbb{R}^n$, we can write

$$\begin{aligned} \int_{\mathcal{A}} f(t_x, p) dt_x \geq \left(\int_{\mathcal{A}} f(u_x, \tilde{u}_p) du_x \right)^{\lambda^2} \left(\int_{\mathcal{A}} f(u_x, \tilde{v}_p) du_x \right)^{\lambda(1-\lambda)} \\ \left(\int_{\mathcal{A}} f(v_x, \tilde{u}_p) dv_x \right)^{(1-\lambda)\lambda} \left(\int_{\mathcal{A}} f(v_x, \tilde{v}_p) dv_x \right)^{(1-\lambda)^2}. \end{aligned}$$

By renaming the variables t_x, u_x and v_x to x inside the integrals and merging the similar terms into one, we find

$$\int_{\mathcal{A}} f(x, p) dx \geq \left(\int_{\mathcal{A}} f(x, \tilde{u}_p) dx \right)^{\lambda} \left(\int_{\mathcal{A}} f(x, \tilde{v}_p) dx \right)^{(1-\lambda)},$$

where $\lambda \tilde{u}_p + (1-\lambda)\tilde{v}_p = p$. Therefore, $\int_{\mathcal{A}} f(x, p) dx$ is log-concave which concludes the promised results. □

Revisiting the definition of $\Omega_1(W), \Omega_2(W)$ in (1.2) and (1.3), we find that

$$\begin{aligned}\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_2(W)] &= 1 - \mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1(W)] \\ &\leq 1 - \min_p \mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1(W)] = \delta.\end{aligned} \tag{1.13}$$

From this, we see that bounding δ can be done by minimizing $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1(W)]$, an explicit form of which was given in Lemma 1.1. In Lemma 1.2, we established the log-concavity of the function that we seek to minimize. As a result, instead of minimizing $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1(W)]$ over the entire continuous domain of $\Delta_{n,W}^{(\eta,\bar{\eta})}$, we can only consider the extreme points. Note that the set of points within $\Delta_{n,W}^{(\eta,\bar{\eta})}$ form a polyhedron with at most $|W|(|W|+1)/2$ vertices. As the minimum of an unsorted list of n entries can be found in linear time, the time complexity of finding $\min \mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1]$ is $\mathcal{O}(|W|^2)$. Next, we develop analogous bounds for ϵ.

1.3.2 Computing ϵ

As above, fix $\eta, \bar{\eta} \in (0,1)$, $b \in (0,1]$, and $W \subseteq [n-1]$. Then, for a given $k \in \mathbb{R}_+$, bounding ϵ requires evaluating the term

$$\log\left(\frac{\mathbb{P}[\mathrm{Dir}_k(p) = x]}{\mathbb{P}[\mathrm{Dir}_k(q) = x]}\right),$$

for all $x \in \Omega_1(W)$ and b-adjacent p and q in $\Delta_{n,W}^{(\eta,\bar{\eta})}$. Let $i, j \in W$ be the indices in which p and q differ. Using the definition of the Dirichlet mechanism, we find

$$\begin{aligned}\log\left(\frac{\mathbb{P}[\mathrm{Dir}_k(p) = x]}{\mathbb{P}[\mathrm{Dir}_k(q) = x]}\right) &= \log\left(\frac{\mathrm{B}(kq)\prod_{i=1}^{n} x_i^{kp_i-1}}{\mathrm{B}(kp)\prod_{i=1}^{n} x_i^{kq_i-1}}\right) \\ &= \log\left(\frac{\Gamma(kq_i)\Gamma(kq_j)}{\Gamma(kp_i)\Gamma(kp_j)}\left(\frac{x_i}{x_j}\right)^{k(p_i-q_i)}\right).\end{aligned} \tag{1.14}$$

Note that if either x_i or x_j goes to 0, then the term in (1.14) would be unbounded. Recalling that the indices in which p and q can differ are restricted to the set W, we find that the values at these indices must be bounded below by η, and therefore the

ratios of interest remain bounded as well. Lemma 1.4 below will provide an explicit value of ϵ

Lemma 1.4. *Let W be a given set of indices which is used to construct $\Delta_{n,W}^{(\eta,\bar{\eta})}$ and Dir_k be a Dirichlet mechanism with parameter k. Then, for all $x \in \Omega_1(W)$ we have that*

$$\log\left(\frac{\mathbb{P}[\mathrm{Dir}_k(p)=x]}{\mathbb{P}[\mathrm{Dir}_k(q)=x]}\right) \leq \log\left(\frac{\mathrm{beta}(k\eta, k(1-\bar{\eta}-\eta))}{\mathrm{beta}(k(\eta+\frac{b}{2}), k(1-\bar{\eta}-\eta-\frac{b}{2}))}\right) + \frac{kb}{2}\log\left(\frac{1-(|W|-1))\gamma}{\gamma}\right),$$

where the parameter $\gamma \in (0,1)$ takes the same value of γ used to compute δ in Section 1.3.1.

Proof. We first prove a lemma which we later use to prove Lemma 1.4.

Lemma 1.5. *Let W be a given set of indices which is used to construct $\Delta_{n,W}^{(\eta,\bar{\eta})}$ and let p, q be any b-adjacent vectors in $\Delta_{n,W}^{(\eta,\bar{\eta})}$ with their i^{th} and j^{th} entries different. Then, for a constant $k \in \mathbb{R}_+$, we have that*

$$\frac{\mathrm{beta}(kq_i, kq_j)}{\mathrm{beta}(kp_i, kp_j)} \leq \frac{\mathrm{beta}(kq_i, k(1-\bar{\eta}-q_i))}{\mathrm{beta}(kp_i, k(1-\bar{\eta}-p_i))}.$$

Proof. Let $c := p_i + p_j = q_i + q_j$. Then, using (1.6), we have that

$$\frac{\mathrm{beta}(kp_i, kp_j)}{\mathrm{beta}(kq_i, kq_j)} = \frac{\Gamma(kq_i)\Gamma(k(c-q_i))}{\Gamma(kp_i)\Gamma(k(c-p_i))} = \frac{\Gamma(kq_j)\Gamma(k(c-q_j))}{\Gamma(kp_j)\Gamma(k(c-p_j))}. \tag{1.15}$$

Using the definition of digamma function, $\psi^{(0)}(\cdot)$, we have

$$\frac{\partial}{\partial x}\left[\frac{\Gamma(x-a)}{\Gamma(x-b)}\right] = \frac{\Gamma(x-a)[\psi^{(0)}(x-a) - \psi^{(0)}(x-b)]}{\Gamma(x-b)}. \tag{1.16}$$

As the digamma function is strictly increasing on interval $(0, +\infty)$, the derivative in (1.16) is positive if and only if $x-b < x-a$, which is true if and only if $a < b$. Returning to (1.15), we will construct an upper bound using the first identity if

$q_i < p_i$ and we will construct an upper bound using the second identity if $q_j < p_j$. For correctness, suppose $q_i < p_i$. Then

$$\frac{\text{beta}(kp_i, kp_j)}{\text{beta}(kq_i, kq_j)} = \frac{\Gamma(kq_i)\Gamma(k(c-q_i))}{\Gamma(kp_i)\Gamma(k(c-p_i))} \leq \frac{\text{beta}(kq_i, k(1-\bar{\eta}-q_j))}{\text{beta}(kp_i, k(1-\bar{\eta}-p_i))}.$$

The other case will work identically. □

Next, let

$$\begin{aligned} v := \quad & \max_{p,q,x\in\mathbb{R}^2} \quad \log\left(\frac{\Gamma(kq_i)\Gamma(kq_j)}{\Gamma(kp_i)\Gamma(kp_j)}\left(\frac{x_i}{x_j}\right)^{k(p_i-q_i)}\right) \\ & \text{subject to} \quad |p_i - q_i| \leq \frac{b}{2}, \\ & \qquad\qquad\quad p_i + p_j = q_i + q_j, \\ & \qquad\qquad\quad p_i + p_j \leq 1 - \bar{\eta}, \\ & \qquad\qquad\quad p_{(i,j)} \in [\eta, 1-\bar{\eta}-\eta]^2, \\ & \qquad\qquad\quad q_{(i,j)} \in [\eta, 1-\bar{\eta}-\eta]^2, \\ & \qquad\qquad\quad x_{(i,j)} \in [\gamma, 1-(|W|-1)\gamma]^2, \end{aligned} \tag{1.17}$$

and let $\mathcal{C}$ denote the set of feasible points of the optimization problem in (1.17); we note that the first two constraints enforce adjacency, while the others encode $p, q \in \Delta_{n,W}^{(\eta,\bar{\eta})}$ and $x \in \Omega_1(W)$.

By sub-additivity of the maximum, we have

$$v \leq \max_{p,q,x\in\mathcal{C}} \log\left(\frac{\Gamma(kq_i)\Gamma(kq_j)}{\Gamma(kp_i)\Gamma(kp_j)}\right) + \max_{p,q,x\in\mathcal{C}} \log\left(\frac{x_i}{x_j}\right)^{k(p_i-q_i)}. \tag{1.18}$$

Now, with $v_1 := \max_{p,q,x\in\mathcal{C}} \log\left(\frac{x_i}{x_j}\right)^{k(p_i-q_i)}$, we find

$$v_1 \leq \max_{p,q,x\in\mathcal{C}} |k(p_i - q_i)| \left|\log\left(\frac{x_i}{x_j}\right)\right| = \frac{kb}{2}\log\left(\frac{1-(|W|-1)\gamma}{\gamma}\right).$$

Next, let $c := p_i + p_j = q_i + q_j$ and substitute q_j, p_j with $c - q_i$ and $c - p_i$

respectively. Let

$$\begin{aligned} v_2 := \quad & \max_{p_i,q_i,c} \quad \log\left(\frac{\Gamma(kq_i)\Gamma(k(c-q_i))}{\Gamma(kp_i)\Gamma(k(c-p_i))}\right) \\ & \text{subject to} \quad |p_i - q_i| \leq \frac{b}{2}, \\ & \qquad\qquad\quad c \in [2\eta, 1-\bar{\eta}], \\ & \qquad\qquad\quad p_i \in [\eta, 1-\bar{\eta}-\eta], \\ & \qquad\qquad\quad q_i \in [\eta, 1-\bar{\eta}-\eta], \end{aligned}$$

where the constraints again encode adjacency of p and q and their containment in $\Delta_{n,W}^{(\eta,\bar{\eta})}$.

Then, from Lemma 1.5 and Equation (1.6), we have that

$$\begin{aligned} v_2 \leq \quad & \max_{p_i,q_i} \quad \log\left(\frac{\text{beta}(kq_i, k(1-\bar{\eta}-q_i))}{\text{beta}(kp_i, k(1-\bar{\eta}-p_i))}\right) \\ & \text{subject to} \quad |p_i - q_i| \leq \frac{b}{2}, \\ & \qquad\qquad\quad p_i \in [\eta, 1-\bar{\eta}-\eta], \\ & \qquad\qquad\quad q_i \in [\eta, 1-\bar{\eta}-\eta]. \end{aligned} \tag{1.19}$$

Evaluating the gradient of the objective function in the optimization problem in (1.19), it can be shown that the Karush-Kuhn-Tucker (KKT) conditions of optimality are not satisfied in the interior of the set of feasible points except for points that lie on the line $p_i = q_i$. However, since the KKT conditions are only necessary conditions (see chapter 11 of (Boyd and Vandenberghe, 2014)), satisfying them does not imply optimality which is indeed the case here.

Evaluating points on the boundary of the feasible region shows that KKT conditions are also not satisfied, thus, the only points remaining are (p_i, q_i)'s in the set

$$\begin{aligned} \left\{ \left(\eta + \frac{b}{2}, \eta\right), \left(1-\bar{\eta}-\eta-\frac{b}{2}, 1-\bar{\eta}-\eta\right), \right. \\ \left. \left(\eta, \eta+\frac{b}{2}\right), \left(1-\bar{\eta}-\eta, 1-\bar{\eta}-\eta-\frac{b}{2}\right) \right\}. \end{aligned} \tag{1.20}$$

Note that since $\text{beta}(a,b) = \text{beta}(b,a)$, the points in the first row give equal positive objectives and the points in the second row have equal negative objectives. Hence, we can choose the first point in Equation (1.20) to find

$$v_2 = \log\left(\frac{\text{beta}(k\eta, k(1-\bar{\eta}-\eta))}{\text{beta}(k(\eta+\frac{b}{2}), k(1-\bar{\eta}-\eta-\frac{b}{2}))}\right). \tag{1.21}$$

Substituting v_1 and v_2 in (1.18) concludes the proof. □

We now state the main theorem of this section, which formally establishes the (ϵ, δ)-differential privacy of the Dirichlet mechanism for identity queries.

Theorem 1.6. *Fix $\eta, \bar{\eta} \in (0,1)$ and $b \in (0,1]$, and consider b-adjacent vectors $p, q \in \Delta_{n,W}^{(\eta,\bar{\eta})}$. Let $W \subseteq [n-1]$ be a given set of indices which is used to construct $\Delta_{n,W}^{(\eta,\bar{\eta})}$. Then the Dirichlet mechanism with parameter $k \in \mathbb{R}_+$ is (ϵ, δ)-differentially private, where*

$$\epsilon = \log\left(\frac{\text{beta}(k\eta, k(1-\bar{\eta}-\eta))}{\text{beta}(k(\eta+\frac{b}{2}), k(1-\bar{\eta}-\eta-\frac{b}{2}))}\right) + \frac{kb}{2}\log\left(\frac{1-(|W|-1)\gamma}{\gamma}\right),$$

and $\delta = 1 - \min_{p \in \Delta_{n,W}^{(\eta,\bar{\eta})}} \mathbb{P}[\text{Dir}_k(p) \in \Omega_1(W)]$.

Proof. The expression for ϵ results immediately from Lemma 1.4 and the expression for δ is a direct result of (1.13). □

The expression given for ϵ in Theorem 1.6 contains a ratio of beta functions. In the following lemma we present upper and lower bounds for beta functions in terms of simpler functions which we will use to provide a simplified upper bound for ϵ.

Lemma 1.7. *Let $a, b > 1$. Then*

$$\exp(2-a-b) \leq \text{beta}(a,b) \leq \frac{a+b-1}{(2a-1)(2b-1)}. \tag{1.22}$$

Proof. From Jensen's inequality we have that

$$\phi\left(\int_a^b f\ dx\right) \leq \int_a^b \phi(f)\ dx,$$

where f is integrable over the domain of interest and ϕ is a convex function. Since the exponential function is convex, we have that

$$\text{beta}(a,b) = \int_0^1 \exp\left(\log\left(x^{a-1}(1-x)^{b-1}\right)\right)dx \geq \exp\left(\int_0^1 \log\left(x^{a-1}(1-x)^{b-1}\right)dx\right).$$

Evaluating the integral, we find

$$\exp\left(\int_0^1 \log\left(x^{a-1}(1-x)^{b-1}\right)dx\right) = \int_0^1 (a-1)\log(x)+(b-1)\log(1-x)dx = 2-(a+b).$$

Then

$$\text{beta}(a,b) \geq \exp(2-(a+b)).$$

The upper bound follows from the identity that

$$2\alpha\beta \leq \alpha^2+\beta^2, \text{ for all } \alpha,\beta \in \mathbb{R}.$$

Substituting α, β with x^{a-1} and y^{b-1} in the integral representation of the beta function results in the introduced upper bound. □

Note that if a mechanism is ϵ_1-differentially private, it is also ϵ_2-differentially private for all $\epsilon_2 \geq \epsilon_1$. Therefore, if the upper bound for ϵ after simplification of beta functions is still within an acceptable range, e.g., $\delta \leq 0.05$ and $\epsilon \leq 5$ (McSherry and Talwar, 2007; Bonomi et al., 2012; Hsu et al., 2014), then using the approximate value of ϵ does not substantially harm the interpretation of the Dirichlet mechanism's protections. In Figure 1.1, for three instances of $(\eta, \bar{\eta}, k)$ and $b = 0.1$, we show how the approximation captures the behavior of ϵ. All three cases show that the approximation causes an offset to the exact value of ϵ, and the level of offset decreases with the value of the original ϵ.

Next, we point out that the parameter γ, which is used in the definition of Ω_2, is not a parameter of the mechanism, in the sense that changing γ does not change

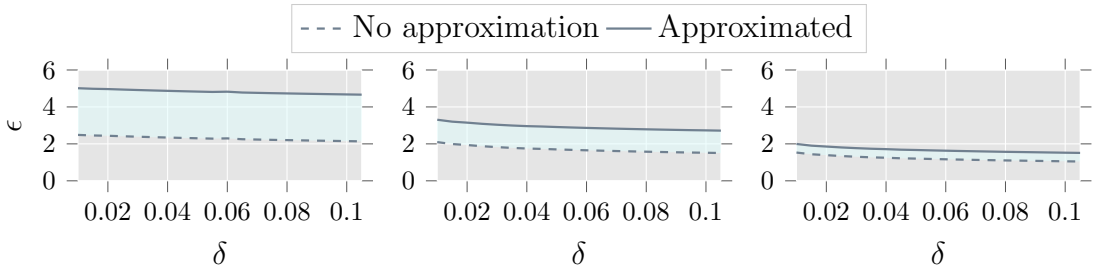

Figure 1.1: An example where $|W| = 3$ to compare the values of ϵ in Theorem 1.6 with those computed based on the simplifications proposed in Lemma 1.7. At each level of δ, first γ is optimized according to the optimization problem in (1.23), then the optimal γ is substituted in the expressions for the original and approximated values. Left: $\eta = 0.15$, $\bar{\eta} = 0.15$, and $k = 6.7$. Middle: $\eta = 0.20$, $\bar{\eta} = 0.20$, and $k = 5.1$. Right: $\eta = 0.25$, $\bar{\eta} = 0.25$, and $k = 4.1$

the mechanism itself. Instead, γ balances the trade-off between privacy level and the probability of failing to guarantee that privacy level, i.e., changing γ can decrease ϵ in exchange for increasing δ and vice versa.

In some cases, we are given the highest probability of privacy failure, δ, that is acceptable, and one must maximize the level of privacy subject to that upper bound. Let $\hat{\delta}$ denote maximum admissible value of δ. Then we are interested in minimizing ϵ while obeying $\delta \leq \hat{\delta}$. Using Theorem 1.6 to substitute ϵ, let V be the set of vertices of $\Delta_{n,W}^{(\eta,\bar{\eta})}$. Then we must solve

$$\min_{\gamma} \left\{ \gamma : \mathbb{P}\left[\mathrm{Dir}_k(p) \in \Omega_1(W)\right] \geq 1 - \hat{\delta}, \forall p \in V \right\}. \tag{1.23}$$

Note that the feasible region of the above optimization problem is a convex set because the function $\mathbb{P}[\mathrm{Dir}_k(p) \in \Omega_1(W)]$ is a strictly decreasing function of γ. Therefore, ϵ can be optimized for a given $\hat{\delta}$ using off-the-shelf convex optimization tool-boxes.

1.4 Accuracy Analysis

We briefly analyze the accuracy of the Dirichlet mechanism by two metrics. First, in terms of the expected location of the mechanism output on the unit simplex and second in terms of the variance of the output vector.

Lemma 1.8. *Let $x \in \Delta(n)$ be the output of a Dirichlet mechanism with input $p \in \Delta(n)^\circ$ and parameter $k \in \mathbb{R}_+$. Then we have that $\mathbb{E}[x_i] = p_i$ and*

$$\mathrm{Var}[x_i] = \frac{p_i(1-p_i)}{k+1} \leq \frac{1}{4(k+1)}. \tag{1.24}$$

Proof. Let $\bar{p} = \sum_{r=1}^{n} kp_r$. Using equation (49.9) in (Kotz et al., 2000) we can write

$$\mathbb{E}[x_i] = \frac{kp_i}{\bar{p}} = p_i \quad \text{and} \quad \mathrm{Var}[x_i] = \frac{kp_i(\bar{p} - kp_i)}{\bar{p}^2(\bar{p}+1)}.$$

Since input p belongs to the unit simplex, we have that $\bar{p} = k$. Substituting $\bar{p}$ with k concludes the promised results. The variance of the output depends on the input data p_i. However, we can find the worst-case variance by maximizing the expression for the variance which occurs at $p_i = 0.5$. Hence, we have that

$$\mathrm{Var}[x_i] \leq \frac{1}{4(k+1)}.$$

□

1.5 Conclusion

In this chapter we introduced a mechanism used for privatizing data inputs that belong to the unit simplex. We used the Dirichlet distribution to probabilistically map a vector within the unit simplex to itself. We proved that the Dirichlet mechanism is differentially private with high probability. We also provided simulation results validating that the privacy bounds and the accuracy of the mechanism are within ranges typically considered in the differential privacy literature.

Chapter 2: Privacy-Preserving Policy Synthesis for Markov Decision Processes[1]

Abstract

In this chapter, we develop a privacy-engineered policy synthesis algorithm for MDPs. The technical privacy policy considered in this chapter sets the privacy objective as providing theoretical guarantees that the synthesized policy for the underlying MDP does not leak information about its transition probabilities. We identify differential privacy as the appropriate privacy-engineering technique to provide the theoretical privacy guarantees that the technical privacy policy requires. We develop a deferentially private policy synthesis algorithm by utilizing the Dirichlet mechanism to obfuscate the transition probabilities of the MDP and using the privatized transition probabilities to synthesize a policy using dynamic programming. We show that there is a trade-off between the level of differential privacy and the synthesized policy's performance, and we demonstrate how the adjustment of the Dirichlet mechanism's concentration parameters can balance the trade-off.

2.1 Introduction

In many decision-making problems, agents desire to protect sensitive information that drives their actions from eavesdroppers and adversaries, such as applications

[1]The research presented in this chapter has been previously published in (Gohari et al., 2020a). Parham Gohari formulated the problem, derived the theoretical results, conducted the numerical experiments, and wrote the paper.

in autonomous driving or smart power grids (Glancy, 2012; Guan et al., 2018). In these applications, as well as in many other sequential decision-making problems, choosing actions can be cast as a policy-synthesis problem wherein the environment is modeled as a Markov decision process (MDP) (Brechtel et al., 2011b; Misra et al., 2013b). The goal in a policy-synthesis problem is to find a reward-maximizing control policy based on the transition probabilities of the underlying MDP. In this chapter, we study the problem of synthesizing a policy that protects the privacy of the transition probabilities.

Transition probabilities in an MDP govern the dynamics of the environment and may carry information that should be protected during policy synthesis. For example, suppose that through market research, a corporation discovers a niche in the market and decides to invest. Such an investment may alert competitors to the discovered niche and leads to other firms making similar decisions. Previous works in economics have associated higher market shares with profitability (Szymanski et al., 1993; Prescott et al., 1986). Therefore, competitors' entrance to the market may be harmful to the investing corporation. As a result, it is often crucial for a decision-maker to choose actions that do not reveal its knowledge about its environment dynamics.

The main contribution of this chapter is to develop a policy-synthesis algorithm that enforces differential privacy for transition probabilities with adjustable privacy and utility. We define the utility of a privacy-preserving policy synthesis algorithm to be the value function associated with the policy, which in an MDP is its expected total reward (Puterman, 1994). Utility loss due to privacy is a common phenomenon, and we follow the convention in the differential privacy literature to analyze the utility of the privacy-preserving algorithm by comparing it to its non-private counterpart (Dwork and Roth, 2014; McSherry and Talwar, 2007).

In order to show that the algorithm enforces differential privacy, we exploit the fact that differential privacy is immune to post-processing (Dwork and Roth, 2014).

By immunity to post-processing, we mean that arbitrary functions of the output of a differentially private algorithm do not weaken its privacy guarantees. The algorithm first privatizes the transition probabilities via the Dirichlet mechanism introduced in Chapter 2. We then use dynamic programming to synthesize a policy based on the privatized transition probabilities. Since the dynamic programming stage is an act of post-processing on the output of a differentially private mechanism, its output preserves the differential privacy provided to transition probabilities.

We employ the Dirichlet mechanism for privatization because it preserves the unique structure of the transition probabilities, i.e., vectors with non-negative components that sum to one. Using traditional differentially private mechanisms that add infinite-support noise to transition probabilities are ill-suited to this work as they break the structure of transition probabilities. For example, they can result in a transition probability vector with negative components. Although normalization may seem a fitting solution in order to project the perturbed vector back onto the unit simplex, we avoid normalization because it makes it difficult to quantify utility.

We introduce the "cost of privacy" as a measure of the utility of the algorithm. We define the cost of privacy to be the difference between the expected total rewards of the policy with privacy and that of the same policy without privacy. Since we perturb the transition probabilities to enforce differential privacy, the output of the dynamic-programming stage is susceptible to suboptimality, which the cost of privacy quantifies.

We bound the cost of privacy for both finite- and infinite-horizon MDPs. For finite-horizon MDPs, we show that we can compute the cost of privacy in polynomial time via a backward-in-time recursive algorithm. For the case of infinite-horizon MDPs, we show that an algorithm similar to policy evaluation converges to the cost of privacy asymptotically. We further show that the number of iterations required to approximate the cost of privacy is polynomial in problem parameters. This work enables a decision-maker to control the level of privacy based on the utility loss that

they are willing to tolerate.

In order to empirically validate the expressions that we introduce for the cost of privacy, we run the algorithm on two example MPDs. The first example is a small MDP that models a corporation's investment. The second example is an MDP with a larger state and action space, with its transition probabilities generated randomly. We run the algorithm at a range of privacy levels and visualize our results by plotting the cost of privacy versus privacy level. The results illustrate the trade-off that we establish between the strength of data privacy and utility. Furthermore, we observe that the bounds we provide for the cost of privacy are meaningful in the sense that they empirically provide a close approximation to the cost of privacy.

2.2 Related Works

The works in (Zhang et al., 2019; Venkitasubramaniam, 2013; Wang and Hegde, 2019) study the problem of learning a policy in an MDP while enforcing differential privacy. The key difference between this study and the works above is that we protect the transition probabilities which belong to the probability simplex, whereas the other works protect the sensory data that are scalars. We emphasize that although scalars can be readily privatized using traditional differentially private mechanisms, transition probabilities need to be treated specially to ensure that they remain non-negative and sum to one.

The problems of robust and distributionally robust MDPs are related to this study. Robust policy synthesis in an MDP is the problem of synthesizing a policy that mitigates uncertainties present in transition probabilities (Nilim and Ghaoui, 2005; Iyengar, 2005). Distributionally robust MDPs assume that the planner has access to a probability measure over the uncertainty sets (Xu and Mannor, 2012).

We base the cost of privacy bounds on a concentration bound that we derive for the output of the Dirichlet mechanism. Finding the worst-case cost of privacy coincides with lower bounding the value of a distributionally robust policy where

the uncertainties in the transition probabilities adhere to the concentration bound of the Dirichlet mechanism. Despite the fact that the solutions take similar forms, this research studies the value loss due to privacy, whereas the problem of robust policy synthesis tries to compute a policy that mitigates the effect of uncertainties.

2.3 Preliminaries

In this section we set the notation and definitions used throughout this chapter. First, we formally define MDPs. An MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}_s, \mathcal{P}, \mathrm{R}, T, \gamma)$ where $\mathcal{S}$ is the set of states, $\mathcal{A}_s$ is the set of available actions at state $s \in \mathcal{S}$, and $\mathrm{R} : \mathcal{S} \times \mathcal{A}_s \mapsto \mathbb{R}$ is the reward function that indicates the one-step reward for taking action a at state s. $\mathcal{P} := \{P(s,a) \in \Delta(|\mathcal{S}|) \mid (s,a) \in \mathcal{S} \times \mathcal{A}_s\}$ is the set of transition probabilities, where $\Delta(n)$ is the unit simplex in $\mathcal{R}^n$ and $|\mathcal{S}|$ is the cardinality of $\mathcal{S}$. Finally, T is the time horizon and γ is the discount factor.

A policy π is evaluated by its value function $V_t^\pi : \mathcal{S} \mapsto \mathbb{R}$, that is defined as

$$V_t^\pi(s) := \mathbf{E}\left[\sum_{i=t}^{T} \gamma^{i-t} r_i \,\middle|\, s_t = s\right].$$

The expectation is taken over the stochasticity of the policy π and transition probabilities $\mathcal{P}$. We study the problem of privacy-preserving policy synthesis, and in a synthesis problem, the goal is to find an optimal policy, in the sense that it achieves the highest value function beginning at initial state s_0.

We note that this chapter focuses on Markovian policies, i.e., the class of policies that only depend on the most recent state of the history. Markovian policies are shown to be optimal under some mild conditions (Puterman, 1994). We use the notation $\pi_t(a \mid s)$ to show the probability of taking action a at state s and stage t.

Algorithm 1: Privacy-Engineered Policy Synthesis Algorithm

Input: MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}_s, \mathcal{P}, \mathrm{R}, T, \gamma)$, Dirichlet Mechanism's concentration parameter k

Output: $\bar{\pi}$, $\bar{V}^{\bar{\pi}}$

1 Construct the set of privatized transition probabilities $\bar{\mathcal{P}} := \left\{\bar{P}(s,a) = \mathrm{Dir}_k\left(P(s,a)\right) \mid P(s,a) \in \mathcal{P}\right\}$.

2 Replace $\mathcal{M}$ with its privatized version $\bar{\mathcal{M}} := \left(\mathcal{S}, \mathcal{A}_s, r, \bar{\mathcal{P}}, T, \gamma\right)$

3 Synthesize policy $\bar{\pi}$ for $\bar{\mathcal{M}}$.

4 Compute the value function of $\bar{\pi}$, $\bar{V}^{\bar{\pi}}$.

2.4 Privacy-Engineered Policy Synthesis Algorithm

In this section, we present the proposed differentially private policy synthesis algorithm. The algorithm's pseudocode is provided in Algorithm 1, and what follows is its detailed description. The algorithm takes as input an MDP representation $\mathcal{M} = (\mathcal{S}, \mathcal{A}_s, \mathcal{P}, \mathrm{R}, T, \gamma)$ and the value of k that is the concentration parameter for the Dirichlet mechanism. The algorithm's outputs are a policy $\bar{\pi}$ and its associated value function $\bar{V}^{\bar{\pi}}$.

The algorithm comprises two main stages. In the first stage, it privatizes the MDP's transition probabilities by applying the Dirichlet mechanism independently on each transition probability vector in $\mathcal{P}$. We denote the set of transition probabilities after privatization by $\bar{\mathcal{P}} := \left\{\bar{P}(s,a) = \mathrm{Dir}_k\left(P(s,a)\right) \mid P(s,a) \in \mathcal{P}\right\}$. The second stage computes the optimal policy and the optimal value of the privatized MDP $\bar{\mathcal{M}} := \left(\mathcal{S}, \mathcal{A}_s, \bar{\mathcal{P}}, \mathrm{R}, T, \gamma\right)$.

An optimal policy is one that satisfies the Bellman condition of optimality, and the optimal value is the value of such policies (Puterman, 1994). In the case of a finite-horizon MDP, the optimal policy and value function pair, $(\bar{\pi}, \bar{V}_t)$, must satisfy the following two conditions: for all $t \in \{0, \ldots, T-1\}$ and all $s \in \mathcal{S}$,

$$\bar{V}_t(s) = \max_{\pi} \sum_{a \in \mathcal{A}_s} \pi(a \mid s) \left(\mathrm{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \bar{P}(s,a,s') \bar{V}_{t+1}(s') \right),$$

$$\bar{\pi}_t \in \underset{\pi}{\operatorname{argmax}} \sum_{a \in \mathcal{A}_s} \pi(a \mid s) \left(\mathrm{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \bar{P}(s,a,s') \bar{V}_{t+1}(s') \right),$$

where $\bar{P}(s,a,s')$ denotes the privatized probability that taking action a at state s takes the agent to state s'. We assume that the terminal values are given by a known function $\mathrm{R}_T : \mathcal{S} \mapsto \mathbb{R}$, *i.e.*, $\bar{V}_T(s) = \mathrm{R}_T(s)$, for all $s \in \mathcal{S}$.

For infinite-horizon and discounted MDPs, it can be shown that the optimal policy is a stationary policy, i.e., a policy that adopts the same decision rule at all stages (Puterman, 1994). Let $\bar{V}_\infty$ denote the optimal value of $\bar{\mathcal{M}}$. Then, for an infinite-horizon MDP, the second stage of Algorithm 1 computes $(\bar{\pi}, \bar{V}_\infty)$ such that for all $s \in \mathcal{S}$, they satisfy

$$\bar{V}_\infty(s) = \max_{\pi} \sum_{a \in \mathcal{A}_s} \pi(a \mid s) \left(\mathrm{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \bar{P}(s,a,s') \bar{V}_\infty(s') \right),$$

$$\bar{\pi} \in \underset{\pi}{\operatorname{argmax}} \sum_{a \in \mathcal{A}_s} \pi(a \mid s) \left(\mathrm{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \bar{P}(s,a,s') \bar{V}_\infty(s') \right).$$

There are various methods suggested to efficiently compute $\bar{\pi}$ and its value function, such as dynamic programming or linear programming (Puterman, 1994). The third and the fourth step of Algorithm 1 may adopt any of these methods to synthesize and evaluate an optimal policy for the privatized MDP $\bar{\mathcal{M}}$.

Algorithm 1 satisfies (ϵ, δ)-differential privacy due to differential privacy's immunity to post-processing, which the following lemma establishes.

Lemma 2.1. *[From (Dwork and Roth, 2014), Proposition 2.1] Let $M : \Delta(n) \mapsto \Delta(n)$ be a mechanism that is (ϵ, δ)-differentially private. Let $f : \Delta(n) \mapsto \mathbb{R}$ be an arbitrary mapping. Then, $f \circ M : \Delta(n) \mapsto \mathbb{R}$ is (ϵ, δ)-differentially private.*

Let $(\hat{\epsilon}, \hat{\delta})$ denote the level of the probabilistic differential privacy of the Dirichlet mechanism employed in Algorithm 1. By Lemma 2.1, the algorithm is $(\hat{\epsilon}, \hat{\delta})$-differentially private because the synthesis step is an instance of a post-processing mapping f.

2.5 Cost of Privacy

Recall that Algorithm 1 synthesizes a policy $\bar{\pi}$ based on privatized transition probabilities in $\bar{\mathcal{P}}$. It then computes the value function of $\bar{\pi}$, $\bar{V}_t^{\bar{\pi}}$, using $\bar{\mathcal{P}}$. Let $V_t^{\bar{\pi}} : \mathcal{S} \mapsto \mathbb{R}$ be the value function that the non-private transition probabilities in $\mathcal{P}$ assign to $\bar{\pi}$. The utility of Algorithm 1 is equal to $V^{\bar{\pi}}(s_0)$.

We assume that after the privatization stage, the algorithm loses access to the non-private transition probabilities in $\mathcal{P}$. The reason is that in many real-world applications, a central cloud is used to compute the policy, and agents submit their data to the cloud (Kong et al., 2017; Li et al., 2011). For agents to preserve their data privacy, they privatize their data prior to any submission to the cloud (Liu et al., 2018).

Had we had access to the non-private transition probabilities $\mathcal{P}$, $V^{\bar{\pi}}(s_0)$ could have been computed using an off-the-shelf policy evaluation algorithm. We start off the utility analysis of Algorithm 1 with introducing a concentration bound on the output of the Dirichlet mechanism.

Lemma 2.2. *Let* Dir_k *denote a Dirichlet mechanism with parameter* $k \in \mathbb{R}_+$*. Then, for all* $\beta > 0$*, and all* $p \in \Delta^\circ(n)$*,*

$$\mathbb{P}\left(\left\| \mathrm{Dir}_k(p) - p \right\|_\infty \geq \sqrt{\frac{\log\left(1/\beta\right)}{2(k+1)}} \right) \leq \beta.$$

Proof. Let $p \in \Delta^\circ(n)$ and $k > 0$. From the properties of the Dirichlet mechanism we have that $\mathbf{E}\left[\mathrm{Dir}_k(p)\right] = p$. Let $u \in \Delta(n)$ and $\lambda \in \mathbb{R}_+$. By Theorem 3.3 in (Marchal and Arbel, 2017), we have that

$$\mathbf{E}\left[\exp\left(\lambda u^T \left(\mathrm{Dir}_k(p) - \mathbf{E}\left[\mathrm{Dir}_k(p)\right]\right)\right)\right] \leq \prod_{i=1}^{n} \mathbf{E}\left[\exp\left(\lambda u_i \left(\left(\mathrm{Dir}_k(p)\right)_i - p_i\right)\right)\right].$$

Let $\mathrm{beta}(a, b)$ denote the beta distribution with parameter (a, b). It can be shown that each component of the Dirichlet mechanism satisfies

$$\left(\mathrm{Dir}_k(p)\right)_i \sim \mathrm{beta}(kp_i, k(1 - p_i)).$$

Recall that a random variable X is said to be R-subgaussian if it satisfies

$$\mathbf{E}\left[\exp\left(sX\right)\right] \leq \exp\left(\frac{R^2 s^2}{2}\right), \forall s \in \mathbb{R}.$$

A beta distribution with parameters (a, b) is $\sqrt{1/4(a+b+1)}$-subgaussian (Marchal and Arbel, 2017). Therefore,

$$\prod_{i=1}^{n} \mathbf{E}\left[\exp\left(\lambda u_i \left(\left(\mathrm{Dir}_k(p)\right)_i - p_i\right)\right)\right] \leq \prod_{i=1}^{d} \exp\left(\frac{\lambda^2 u_i^2 \sigma_i^2}{2}\right) = \exp\left(\frac{\lambda^2 \|u\|_2}{8(k+1)}\right).$$

Since u can be any vector in $\Delta(n)$, we consider an instance of u that puts weight 1 on the component of $\mathrm{Dir}_k(p) - p$ with maximum magnitude and zero elsewhere. Considering the case where $\lambda = 1$, we can write

$$\mathbf{E}\left[\exp\left\|\mathrm{Dir}_k(p) - p\right\|_\infty\right] \leq \exp\left(\frac{1}{8(k+1)}\right). \tag{2.1}$$

Finally, for all $\theta \geq 0$, we have that

$$\mathbb{P}\left(\left\|\mathrm{Dir}_k(p) - p\right\|_\infty \geq \alpha\right) = \mathbb{P}\left(\exp\left(\theta \left\|\mathrm{Dir}_k(p) - p\right\|_\infty\right) \geq \exp\left(\theta\alpha\right)\right).$$

By Markov's inequality,

$$\mathbb{P}\left(\exp\left(\theta \left\|\mathrm{Dir}_k(p) - p\right\|_\infty\right) \geq \exp\left(\theta\alpha\right)\right) \leq \mathbf{E}\left[\exp\left(\theta \left\|\mathrm{Dir}_k(p) - p\right\|_\infty\right)\right] \cdot \exp\left(-\theta\alpha\right).$$

Combining the above inequality with (2.1), we arrive at

$$\mathbb{P}\left(\left\|\mathrm{Dir}_k(p) - p\right\|_\infty \geq \alpha\right) \leq \exp\left(\frac{\theta^2}{8(k+1)} - \theta\alpha\right).$$

Taking $\theta = 4\alpha(k+1)$ and a proper change of variable from α to β concludes the lemma. □

The above lemma enables us to evaluate $V^{\bar{\pi}}(s_0)$, i.e., the conditional expectation of the value function without privacy, based on the privatized transition probabilities $\bar{\mathcal{P}}$ and k. In particular, for a finite-horizon MDP we provide an upper bound on $\left|\mathbf{E}\left[V_0^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k\right] - \bar{V}_0^{\bar{\pi}}(s_0)\right|$. For an infinite-horizon MDP, we upper bound $\left|\mathbf{E}\left[V_\infty^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k\right] - \bar{V}_\infty^{\bar{\pi}}(s_0)\right|$. We refer to both expressions as the "cost of privacy."

The bounds are based on the pessimistic and optimistic value functions that possible transition-probability vectors generate. Let $\alpha := \sqrt{\log(1/\beta)/2(k+1)}$, then Lemma 2.2 implies that for all $P(s,a) \in \mathcal{P}$ and the corresponding $\bar{P}(s,a) \in \bar{\mathcal{P}}$, $\mathbb{P}\left(||\bar{P}(s,a) - P(s,a)||_\infty \leq \alpha\right) \geq 1-\beta$. We define $\hat{\mathcal{P}}_{\alpha,\beta}$ as

$$\hat{\mathcal{P}}_{\alpha,\beta} = \{\beta P_1(s,a) + (1-\beta)P_2(s,a) \mid \|P_2(s,a) - \bar{P}(s,a)\|_\infty \leq \alpha, \\ P_1(s,a), P_2(s,a) \in \Delta(|\mathcal{S}|), (s,a) \in \mathcal{S} \times \mathcal{A}_s\} \quad (2.2)$$

We use the set $\hat{\mathcal{P}}_{\alpha,\beta}$ to compute a pessimistic and optimistic value function to bound the cost of privacy.

2.5.1 Finite-Horizon MDPs

We bound the cost of privacy for a finite-horizon MDP by establishing a common upper and lower bound for both $\mathbf{E}\left[V_0^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k\right]$ and $\bar{V}_0^{\bar{\pi}}(s_0)$. We first state a technical lemma that we later use to prove the main theorem of this section.

Lemma 2.3. *Fix k and a set of transition probabilities $\mathcal{P}$, and let*

$$\bar{\mathcal{P}} := \left\{\bar{P}(s,a) = \mathrm{Dir}_k\left(P(s,a)\right) \mid P(s,a) \in \mathcal{P}\right\}. \quad (2.3)$$

For any $\beta > 0$, let $\alpha := \sqrt{\log(1/\beta)/2(k+1)}$. Then, for all $\bar{P}(s,a) \in \bar{\mathcal{P}}$ and all $P(s,a) \in \mathcal{P}$,

$$\bar{P}(s,a) \in \hat{\mathcal{P}}_{\alpha,\beta} \quad \textit{and} \quad \mathbf{E}\left[P(s,a) \mid \bar{\mathcal{P}}, k\right] \in \hat{\mathcal{P}}_{\alpha,\beta}.$$

Proof. In the definition of $\hat{\mathcal{P}}_{\alpha,\beta}$ in (2.2), take $P_1(s,a) := \bar{P}(s,a)$, and $P_2(s,a) := \bar{P}(s,a)$. Then, $\beta P_1(s,a) + (1-\beta)P_2(s,a) = \bar{P}(s,a)$ is within the set $\hat{\mathcal{P}}_{\alpha,\beta}$, which concludes the first result.

For a given β, by Lemma 2.2, there exists $\beta' \geq \beta$, such that

$$\mathbb{P}\left(\|\mathrm{Dir}_k(p) - p\|_\infty \geq \sqrt{\frac{\log\left(1/\beta'\right)}{2(k+1)}}\right) = \beta.$$

Define $\alpha' := \sqrt{\log(1/\beta')/2(k+1)}$ and let

$$P_1(s,a) := \mathbf{E}\left[P(s,a) \,\middle|\, \bar{\mathcal{P}}, k, \|P(s,a) - \bar{P}(s,a)\|_\infty \geq \alpha'\right],$$

and

$$P_2(s,a) := \mathbf{E}\left[P(s,a) \,\middle|\, \bar{\mathcal{P}}, k, \|P(s,a) - \bar{P}(s,a)\|_\infty \leq \alpha'\right].$$

Since $\alpha' \leq \alpha$, $\|P_2(s,a) - \bar{P}(s,a)\|_\infty \leq \alpha$. Then, $\beta P_1(s,a) + (1-\beta)P_2(s,a) = \mathbf{E}\left[P(s,a) \mid \bar{\mathcal{P}}, k\right]$ is also within the set $\hat{\mathcal{P}}_{\alpha,\beta}$, which completes the proof. □

We now state the first main theorem of this section.

Theorem 2.4. *Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}_s, \mathcal{P}, \mathrm{R}, T, \gamma)$ and k be the input, and $(\bar{\pi}, \bar{V}_t^{\bar{\pi}})$ be the output of Algorithm 1, and let $T < \infty$. Fix $\beta > 0$, and let $\alpha := \sqrt{\log(1/\beta)/2(k+1)}$. Let $R_T : \mathcal{S} \mapsto \mathbb{R}$ denote the terminal value function of $\mathcal{M}$. Define $\underline{v}_t^{\bar{\pi}} : \mathcal{S} \mapsto \mathbb{R}$ and $\bar{v}_t^{\bar{\pi}} : \mathcal{S} \mapsto \mathbb{R}$ as follows. For all $s \in \mathcal{S}$, let $\bar{v}_T^{\bar{\pi}}(s) := R_T(s)$, $\underline{v}_T^{\bar{\pi}}(s) := R_T(s)$, and for all $t \in \{0, \ldots, T-1\}$, let*

$$\underline{v}_t^{\bar{\pi}}(s) := \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \min_{p \in \hat{\mathcal{P}}_{\alpha,\beta}} \sum_{s' \in \mathcal{S}} p(s,a,s') \underline{v}_{t+1}^{\bar{\pi}}(s') \right),$$

$$\bar{v}_t^{\bar{\pi}}(s) := \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \max_{p \in \hat{\mathcal{P}}_{\alpha,\beta}} \sum_{s' \in \mathcal{S}} p(s,a,s') \bar{v}_{t+1}^{\bar{\pi}}(s') \right).$$

Then, we have that

$$\left| \mathbf{E}\left[V_0^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k\right] - \bar{V}_0^{\bar{\pi}}(s_0) \right| \leq \bar{v}_0^{\bar{\pi}}(s_0) - \underline{v}_0^{\bar{\pi}}(s_0).$$

Proof. We first show by induction that for all stages $t \in \{0, 1, \ldots, T\}$, and all states $s \in \mathcal{S}$, $\underline{v}_t^{\bar{\pi}}(s)$ lower bounds $\bar{V}_t^{\bar{\pi}}(s)$. Since all terminal values are determined by R_T, we have that for all states, $\bar{V}_T^{\bar{\pi}}(s) = \underline{v}_T^{\bar{\pi}}(s) = R_T(s)$. Let $\tau \in \{1, \ldots, T-1\}$, and assume that for all states, we have that $\bar{V}_\tau^{\bar{\pi}}(s) \geq \underline{v}_\tau^{\bar{\pi}}(s)$. Then, for $t = \tau - 1$, and for all $s \in \mathcal{S}$,

we can write

$$\begin{aligned}
\bar{V}_{\tau-1}^{\bar{\pi}}(s) &= \sum_{a\in\mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \bar{P}(s,a,s') \bar{V}_{\tau}^{\bar{\pi}}(s') \right) \\
&\geq \sum_{a\in\mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \bar{P}(s,a,s') \underline{v}_{\tau}^{\bar{\pi}}(s') \right) \\
&\geq \sum_{a\in\mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \min_{p\in\hat{\mathcal{P}}_{\alpha,\beta}} \sum_{s'\in\mathcal{S}} p(s,a,s') \underline{v}_{\tau}^{\bar{\pi}}(s') \right) \qquad (2.4) \\
&= \underline{v}_{\tau-1}^{\bar{\pi}}(s).
\end{aligned}$$

The first inequality immediately results from the induction hypothesis. For the second inequality, note that by Lemma 2.3, $\bar{P}(s,a) \in \hat{\mathcal{P}}$; therefore, $\bar{P}(s,a)$ is a feasible solution of the minimization problem in (2.4).

By induction, we conclude that for all $t \in \{0,\ldots,T\}$, and all $s \in \mathcal{S}$, $\underline{v}_t^{\bar{\pi}}(s)$ lower bounds $\bar{V}_t^{\bar{\pi}}(s)$, which includes stage $t=0$ and state $s=s_0$, hence, $\bar{V}_0^{\bar{\pi}}(s_0) \geq \underline{v}_0^{\bar{\pi}}(s_0)$. An identical argument by reversing the direction of the inequalities and substituting the minimization in (2.4) with maximization, leads to $\bar{v}_t^{\bar{\pi}}(s)$ upper bounding $\bar{V}_t^{\bar{\pi}}(s)$, for all $s \in \mathcal{S}$ and all $t \in \{0,\ldots,T\}$. So far we have established that

$$\underline{v}_0^{\bar{\pi}}(s_0) \leq \bar{V}_0^{\bar{\pi}}(s_0) \leq \bar{v}_0^{\bar{\pi}}(s_0). \qquad (2.5)$$

We now consider the value of $\bar{\pi}$ based on the non-private transition probabilities in $\mathcal{P}$, that is denoted $V^{\bar{\pi}}$. Recall that for all stages $t \in \{1,\ldots,T\}$ and all states $s \in \mathcal{S}$, $V_t^{\bar{\pi}}(s)$ satisfies the following conditions: $V_T^{\bar{\pi}}(s) = R_T(s)$ and

$$V_{t-1}^{\bar{\pi}}(s) = \sum_{a\in\mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s,a,s') V_t^{\bar{\pi}}(s') \right).$$

Taking the expectation of both sides, we arrive at

$$\mathbf{E}\left[V_{t-1}^{\bar{\pi}}(s) \,\middle|\, \bar{\mathcal{P}}, k \right] = \sum_{a\in\mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbf{E}\left[P(s,a,s') V_t^{\bar{\pi}}(s') \,\middle|\, \bar{\mathcal{P}}, k \right] \right).$$

In order to establish an iterative relation between the values of $\mathbf{E}\left[V_{t-1}^{\bar{\pi}}(s) \,\middle|\, \bar{\mathcal{P}}, k\right]$ for different stages, we need to break the expectation on the right-hand side of the above equation. Consider the scenario wherein at each stage, the transition probabilities are independently privatized using the Dirichlet mechanism. Let $\tilde{V}^{\bar{\pi}}$ denote the value function that is associated with $\bar{\pi}$ according to the above scenario. Due to the independence assumption, we can write

$$\mathbf{E}\left[\tilde{V}_{t-1}^{\bar{\pi}}(s) \,\middle|\, \bar{\mathcal{P}}, k\right] = \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{E}\left[P(s,a,s') \,\middle|\, \bar{\mathcal{P}}, k\right] \mathbf{E}\left[\tilde{V}_{t}^{\bar{\pi}}(s') \,\middle|\, \bar{\mathcal{P}}, k\right] \right).$$

Assume that for an arbitrary $\tau \in \{1, \ldots, T-1\}$ and for all $s \in \mathcal{S}$, it holds that $\mathbf{E}\left[\tilde{V}_{\tau}^{\bar{\pi}}(s') \mid \bar{\mathcal{P}}, k\right] \geq \underline{v}_{\tau}^{\bar{\pi}}(s)$. Then, for $t = \tau - 1$, it holds that

$$\mathbf{E}\left[\tilde{V}_{\tau-1}^{\bar{\pi}}(s) \,\middle|\, \bar{\mathcal{P}}, k\right] \geq \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{E}\left[P(s,a,s') \,\middle|\, \bar{\mathcal{P}}, k\right] \underline{v}_{\tau}^{\bar{\pi}}(s') \right).$$

By Lemma 2.3, $\mathbf{E}\left[P(s,a) \mid \bar{\mathcal{P}}, k\right]$ belongs to the set $\hat{\mathcal{P}}_{\alpha,\beta}$. Therefore we can write

$$\mathbf{E}\left[\tilde{V}_{\tau-1}^{\bar{\pi}}(s) \mid \bar{\mathcal{P}}, k\right] \geq \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \min_{p \in \hat{\mathcal{P}}_{\alpha,\beta}} p(s,a,s') \underline{v}_{\tau}^{\bar{\pi}}(s') \right) = \underline{v}_{\tau-1}^{\bar{\pi}}(s).$$

Therefore, by induction, we have shown that for all stages and all states, $\underline{v}^{\bar{\pi}}$ is a lower bound to $\mathbf{E}\left[\tilde{V}^{\bar{\pi}} \mid \bar{\mathcal{P}}, k\right]$.

We now revisit the independence assumption. We introduce the worst-case expected value function that possible transition probabilities can generate as the lower bound to the value function of the generated policy without privacy. The independence assumption leads to independent minimization problems at each stage. The worst-case value function without the independence assumption has the following

restriction on the transition probabilities. Transition probabilities that appear as a minimization variable for the same state-action pair at different stages must take equal values. Since the feasible region of the case without independence is a subset of that of the case with independence, the lower bound for $\mathbf{E}\left[\tilde{V}^{\bar{\pi}} \mid \bar{\mathcal{P}}, k\right]$ also lower bounds $\mathbf{E}\left[V^{\bar{\pi}} \mid \bar{\mathcal{P}}, k\right]$.

Similarly, an identical argument by reversing the direction of the inequalities, and substituting the min operators with max operators leads to $\mathbf{E}\left[\tilde{V}^{\bar{\pi}} \mid \bar{\mathcal{P}}, k\right] \leq \bar{v}^{\bar{\pi}}$, for all stages and all states. Therefore,

$$\underline{v}_0^{\bar{\pi}}(s_0) \leq \mathbf{E}\left[V_0^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k\right] \leq \bar{v}_0^{\bar{\pi}}(s_0). \tag{2.6}$$

Combining (2.5) and (2.6) concludes the proof. □

The assumption of independence has also been adopted in (Nilim and Ghaoui, 2005; Iyengar, 2005; Xu and Mannor, 2012) wherein the problem of robust MDPs is studied. In the above works, the scenario wherein the independence assumption holds is called the dynamic model and the counterpart scenario is called the static model. See Section 2.2 of (Nilim and Ghaoui, 2005) for further details about the relationship between the static and the dynamic model.

2.5.2 Infinite-Horizon MDPs

In this section, we bound the cost of privacy for an infinite-horizon MDP. We first state a technical lemma, which we later use to bound the cost of privacy for infinite-horizon MDPs.

Lemma 2.5. *Fix k and an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}_s, \mathcal{P}, \mathrm{R}, T, \gamma)$, and let*

$$\bar{\mathcal{P}} := \left\{\bar{P}(s,a) = \mathrm{Dir}_k\left(P(s,a)\right) \mid P(s,a) \in \mathcal{P}\right\}.$$

For any $\beta > 0$, let $\alpha := \sqrt{\log(1/\beta)/2(k+1)}$. Define mappings $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3 : \mathbb{R}^{|\mathcal{S}|} \times \mathbb{R}^{|\mathcal{S}|}$

as

$$\mathcal{L}_1\mathbf{v} := \sum_{a\in\mathcal{A}_s}\bar{\pi}(a\mid s)\left(r(s,a)+\gamma\min_{p\in\hat{\mathcal{P}}_{\alpha,\beta}}\sum_{s'\in\mathcal{S}}p(s,a,s')\mathbf{v}(s')\right),$$

$$\mathcal{L}_2\mathbf{v} := \sum_{a\in\mathcal{A}_s}\bar{\pi}(a\mid s)\left(r(s,a)+\gamma\sum_{s'\in\mathcal{S}}\bar{P}(s,a,s')\mathbf{v}(s')\right),$$

$$\mathcal{L}_3\mathbf{v} := \sum_{a\in\mathcal{A}_s}\bar{\pi}(a\mid s)\left(r(s,a)+\gamma\sum_{s'\in\mathcal{S}}\mathbf{E}\left[P(s,a,s')\mathbf{v}(s')\,\middle|\,\bar{\mathcal{P}},k\right]\right).$$

Then, mappings $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ are γ-contraction mappings, i.e., for all $\mathbf{v_1},\mathbf{v_2}\in\mathbb{R}^{|\mathcal{S}|}$ and $i\in\{1,2,3\}$,

$$\left\|\mathcal{L}_i\mathbf{v_1}-\mathcal{L}_i\mathbf{v_2}\right\|_\infty \le \gamma\left\|\mathbf{v_1}-\mathbf{v_2}\right\|_\infty.$$

Proof. Let $U,V\in\mathbb{R}^{|\mathcal{S}|}$. Then, for all $s\in\mathcal{S}$, we can write

$$\mathcal{L}_1U(s)-\mathcal{L}_1V(s) = \gamma\sum_{a\in\mathcal{A}_s}\bar{\pi}(a\mid s)\cdot\left(\min_{p\in\hat{\mathcal{P}}_{\alpha,\beta}}\sum_{s'\in\mathcal{S}}p(s,a,s')U(s')-\min_{p\in\hat{\mathcal{P}}_{\alpha,\beta}}\sum_{s'\in\mathcal{S}}p(s,a,s')V(s')\right).$$

For all $(s,a)\in\mathcal{S}\times\mathcal{A}_s$, let $p^*(s,a)=\operatorname{argmin}_{p\in\hat{\mathcal{P}}_{\alpha,\beta}}\sum_{s'\in\mathcal{S}}p(s,a,s')V(s')$. Then,

$$\mathcal{L}_1U(s)-\mathcal{L}_1V(s)\le\gamma\sum_{a\in\mathcal{A}_s}\bar{\pi}(a\mid s)\cdot\sum_{s'\in\mathcal{S}}p^*(s,a,s')(U(s')-V(s'))\le \gamma\max_{s\in\mathcal{S}}\left|U(s)-V(s)\right|. \quad (2.7)$$

For mapping $\mathcal{L}_2$, we have that

$$\mathcal{L}_2U(s)-\mathcal{L}_2V(s)=\gamma\sum_{a\in\mathcal{A}_s}\bar{\pi}(a\mid s)\cdot\left(\sum_{s'\in\mathcal{S}}\bar{P}(s,a,s')\left(U(s')-V(s')\right)\right)\le \gamma\max_{s\in\mathcal{S}}\left|U(s)-V(s)\right|. \quad (2.8)$$

Finally, for $\mathcal{L}_3$, we write

$$\mathcal{L}_2U(s)-\mathcal{L}_2V(s)=\gamma\sum_{a\in\mathcal{A}_s}\bar{\pi}(a\mid s)\sum_{s'\in\mathcal{S}}\mathbf{E}\left[P(s,a,s')\,\middle|\,\bar{\mathcal{P}},k\right]\cdot(U(s')-V(s'))\le \gamma\max_{s\in\mathcal{S}}\left|U(s)-V(s)\right|. \quad (2.9)$$

Notice that $\|U - V\|_\infty = \max_{s \in \mathcal{S}} |U(s) - V(s)|$, and that (2.7), (2.8) and (2.9) hold for any $s \in \mathcal{S}$. Thus, $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ are γ-contraction mappings. $\square$

We now state the second main theorem of this section.

Theorem 2.6. *Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}_s, \mathcal{P}, \mathrm{R}, T, \gamma)$ and k be the input, and $(\bar{\pi}, \bar{V}_\infty^{\bar{\pi}})$ be the output of Algorithm 1, and let $T = \infty$. Fix $\beta > 0$, and let $\alpha := \sqrt{\log(1/\beta)/2(k+1)}$. For all $s \in \mathcal{S}$, let $\underline{v}_\infty^{\bar{\pi}} : \mathcal{S} \mapsto \mathbb{R}$ and $\bar{v}_\infty^{\bar{\pi}} : \mathcal{S} \mapsto \mathbb{R}$ satisfy*

$$\underline{v}_\infty^{\bar{\pi}}(s) = \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \min_{p \in \hat{\mathcal{P}}_{\alpha,\beta}} \sum_{s' \in \mathcal{S}} p(s,a,s') \underline{v}_\infty^{\bar{\pi}}(s') \right),$$

$$\bar{v}_\infty^{\bar{\pi}}(s) = \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \max_{p \in \hat{\mathcal{P}}_{\alpha,\beta}} \sum_{s' \in \mathcal{S}} p(s,a,s') \bar{v}_\infty^{\bar{\pi}}(s') \right).$$

Then, we have that

$$\left| \mathbf{E}\left[V_\infty^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k \right] - \bar{V}_\infty^{\bar{\pi}}(s_0) \right| \leq \bar{v}_\infty^{\bar{\pi}}(s_0) - \underline{v}_\infty^{\bar{\pi}}(s_0).$$

Proof. Consider the γ-contraction mappings $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ introduced in Lemma 2.5. Notice that $\underline{v}_\infty^{\bar{\pi}}$, $\bar{V}^{\bar{\pi}}\infty$ and $\mathbf{E}\left[V_\infty^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k \right]$ are the fixed points of mappings $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$, respectively. Contraction mappings are known to admit a fixed point by Banach fixed-point theorem. Furthermore, contractive mappings converge to their fixed points by applying the mapping repeatedly to an arbitrary initial $v \in \mathbb{R}^{|\mathcal{S}|}$. Let $v_i^{(k)}$ denote the k^{th} iteration corresponding to $\mathcal{L}_i$, and let all three mappings start from a common initial vector. Assume that at stage k, it holds that $v_1^{(k)} \leq v_2^{(k)}$, then for stage $k+1$, with a similar argument to the proof of Theorem 2.4, we can write

$$\begin{aligned}
v_2^{(k+1)}(s) &= \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \bar{P}(s,a,s') v_2^{(k)}(s') \right) \\
&\geq \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \bar{P}(s,a,s') v_1^{(k)}(s') \right) \\
&\geq \sum_{a \in \mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \min_{p \in \hat{\mathcal{P}}_{\alpha,\beta}} \sum_{s' \in \mathcal{S}} p(s,a,s') v_1^{(k)}(s') \right) \\
&= v_1^{(k+1)}(s).
\end{aligned}$$

As a result, we have that the limiting value of v_1^k and v_2^k when $k \to \infty$ also satisfy

$$\underline{v}_\infty^{\bar{\pi}} = \lim_{k\to\infty} v_1^{(k)} \leq \lim_{k\to\infty} v_2^{(k)} = \bar{V}_\infty^{\bar{\pi}}.$$

Using a similar argument, it can be shown that $\bar{V}_\infty^{\bar{\pi}}$ is upper bounded by $\bar{v}_\infty^{\bar{\pi}}$. Therefore we have that

$$\underline{v}_\infty^{\bar{\pi}} \leq \bar{V}_\infty^{\bar{\pi}} \leq \bar{v}_\infty^{\bar{\pi}}. \tag{2.10}$$

We now assume that at stage k, it holds that $v_1^{(k)} \leq v_3^{(k)}$. At stage $k+1$, we write

$$v_3^{(k+1)}(s) = \sum_{a\in\mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbf{E}\left[P(s,a,s') v_3^{(k)}(s') \,\middle|\, \bar{\mathcal{P}}, k \right] \right).$$

Notice that there is no stochasticity in $v_3^{(k)}$. Therefore, we have that

$$v_3^{(k+1)}(s) = \sum_{a\in\mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbf{E}\left[P(s,a,s') \,\middle|\, \bar{\mathcal{P}}, k \right] v_3^{(k)}(s') \right).$$

The above equation combined with the induction hypothesis implies that

$$v_3^{(k+1)}(s) \geq \sum_{a\in\mathcal{A}_s} \bar{\pi}(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbf{E}\left[P(s,a,s') \,\middle|\, \bar{\mathcal{P}}, k \right] v_1^{(k)}(s') \right).$$

By Lemma 2.3, we have that $v_3^{(k+1)} \geq v_3^{(k+1)}$. Therefore,

$$\underline{v}_\infty^{\bar{\pi}} = \lim_{k\to\infty} v_1^{(k)} \leq \lim_{k\to\infty} v_3^{(k)} = \mathbf{E}\left[V_\infty^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k \right]. \tag{2.11}$$

With a similar argument, it can be shown that $\bar{v}_\infty^{\bar{\pi}}$ upper bounds $\mathbf{E}\left[V_\infty^{\bar{\pi}}(s_0) \mid \bar{\mathcal{P}}, k \right]$, which combined by (2.10) and (2.11) concludes the proof. □

2.6 Computational Complexity

In the previous section, we introduced expressions that bound the cost of privacy for both finite- and infinite-horizon MDPs. The bounds do not take a closed form, and they are computed by iterative methods. In this section, we show that the cost of privacy for both cases can be computed efficiently. In particular we show that for both cases, the computational complexity is polynomial in problem parameters.

2.6.1 Finite-Horizon MDPs

Revisiting the definition of $\underline{v}_t^{\bar{\pi}}$ and $\bar{v}_t^{\bar{\pi}}$ in Theorem 2.4, the inner minimization or maximization problem must be solved for each of T stages and $|\mathcal{S}|$ states. We first consider computing the lower bound $\underline{v}_t^{\bar{\pi}}$. Fix $(s,a) \in \mathcal{S} \times \mathcal{A}_s$ and $t \in \{0,\dots,T-1\}$. Then the inner minimization problem can be recast as

$$
\begin{aligned}
\min_{P_1,P_2,p\in\mathbb{R}^{|\mathcal{S}|}} \quad & \sum_{s'\in\mathcal{S}} p(s,a,s')\underline{v}_t^{\bar{\pi}}(s') && && \text{(P)}\\
\text{subject to} \quad & \mathbf{1}^T P_1(s,a) = 1, P_1(s,a,s') \geq 0, && \forall s' \in \mathcal{S},\\
& \mathbf{1}^T P_2(s,a) = 1, P_2(s,a,s') \geq 0, && \forall s' \in \mathcal{S},\\
& \mathbf{1}^T p(s,a) = 1, p(s,a,s') \geq 0, && \forall s' \in \mathcal{S},\\
& P_2(s,a,s') - \bar{P}(s,a,s') \leq \alpha, && \forall s' \in \mathcal{S},\\
& P_2(s,a,s') - \bar{P}(s,a,s') \geq -\alpha, && \forall s' \in \mathcal{S},\\
& \beta P_1(s,a) + (1-\beta)P_2(s,a) = p(s,a).
\end{aligned}
$$

The above optimization problem is a linear program (LP) with $3|\mathcal{S}|$ variables and $6|\mathcal{S}|+3$ constraints. Similarly, the inner maximization problem in $\bar{v}_t^{\bar{\pi}}$ can be cast as an LP by negating the objective function in (P).

There exists numerous algorithms for solving an LP, each of which is associated with different computational complexities. We consider the interior-point method that is known to solve an LP in polynomial time, $\mathcal{O}(n^{3.5})$, where n is the number of variables (Karmarkar, 1984). Therefore, the computational complexity of computing the cost of privacy for a finite-horizon MDP is $\mathcal{O}\left(T|\mathcal{S}|^{4.5}|\mathcal{A}_s|\right)$.

2.6.2 Infinite-Horizon MDPs

in Lemma 2.5, we introduced $\mathcal{L}_1$ that is a γ-contraction mapping. Suppose there exists a constant $R \in \mathbb{R}_+$ such that the reward function of the underlying MDP satisfies $|r(s,a)| \leq R$, for all $(s,a) \in \mathcal{S}\times\mathcal{A}_s$. Then, all the value functions including the private, non-private, optimistic, and the pessimistic value function must be bounded

above by a constant $v_{\max}$. Let $v_1^{(k)}$ be the value of the k^{th} iteration corresponding to $\mathcal{L}_1$, and v_1^{∞} be the limiting value. We can write

$$\left\| v_1^{(k+1)} - v_1^{\infty} \right\|_{\infty} \leq \gamma \left\| v_1^{(k)} - v_1^{\infty} \right\|_{\infty}.$$

The above inequality is equivalent to

$$\left\| v_1^{(k)} - v_1^{\infty} \right\|_{\infty} \leq \frac{\gamma^k}{1-\gamma} \left\| v_1^{(1)} - v_1^{(0)} \right\|_{\infty} \leq 2v_{\max} \frac{\gamma^k}{1-\gamma}.$$

The above inequality indicates that in order to reach an ϵ-approximation of the limit, $\mathcal{O}(\log(1/\epsilon))$ iterations are required. The inner minimization problem is identical to the finite-horizon case, which we reformulated as an LP in (P). Combining the above arguments together, we conclude that the required number of iterations such that $\|v_1^{(k)} - \underline{v}_{\infty}^{\bar{\pi}}\|_{\infty} \leq \epsilon$, is $\mathcal{O}\left(|\mathcal{S}|^{4.5}|\mathcal{A}_s| \log\left(1/\epsilon\right)\right)$. The same computational complexity holds for the upper bound $\bar{v}_{\infty}^{\bar{\pi}}$. As a result, the cost of privacy for infinite-horizon MDPs can be computed in polynomial time as well.

2.7 Numerical Results

In this section, we empirically validate the developments of previous sections, wherein we introduced the expressions that compute the cost of privacy and their corresponding computational complexity. We apply Algorithm 1 to two example MDPs at a range of k values, which represent a range of privacy protection levels. The first is a small-sized MDP that represents a simple model for a corporation's investment planning. The second example is an MDP with a larger state and action space, which has transition probabilities, reward function, and terminal reward function generated randomly. For both examples, the algorithm is run 50 times, and Figure 2.1, which depicts the results, shows the mean values alongside their standard deviations that appear as error bars.

Example 1. Suppose a corporation has been tracking four startups, and it has to decide which startup to acquire. Assume that the corporation's model of each of the startup's probability of success is given by the MDP in Figure 2.2.

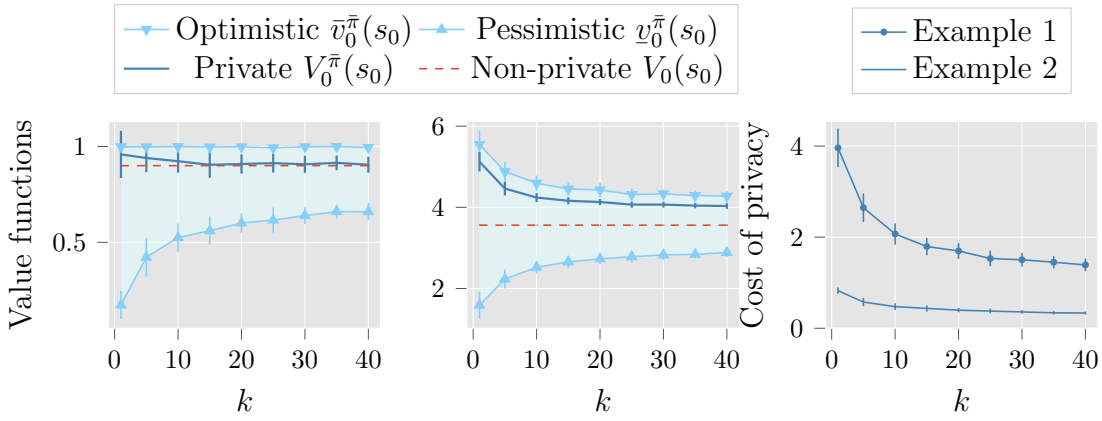

Figure 2.1: From left to right: the first two plots show all the value functions that are used to compute and validate the cost of privacy for Examples 1 and 2. The first plot corresponds to Example 1 and the second plot shows the results for Example 2. The third plot shows the cost of privacy itself for both examples.

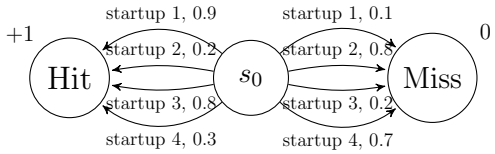

Figure 2.2: The corporation's investment model with four possible startups to acquire, given as an MDP.

The first empirical result of this section corresponds to applying the algorithm to the scenario described in Example 1, and is depicted in Figure 2.1. The results show that as k increases, the pessimistic and optimistic value functions provide better bounds for the private and non-private value functions. Therefore the cost of privacy decreases with k, which Figure 2.1 confirms.

Example 2. In this example, we apply the algorithm to a larger MDP in order to test its scalability. In particular, the MDP has 20 states, 5 actions available at each state, and a time horizon of 10.

Similar to the previous example, Figure 2.1 indicates that an increase in k improves the approximations of the private and non-private value functions by $\bar{v}_0^{\bar{\pi}}(s_0)$ and $\underline{v}_0^{\bar{\pi}}(s_0)$. As a result, the cost of privacy must decrease with k, which is confirmed

by Figure 2.1.

In Example 2, the computation of the cost of privacy for each instance of k took 4.88s on a desktop computer with a 3.5GHz CPU and 24GB of RAM. Doubling the time horizon of the MDP to 20 results in 9.34s/itr, and doubling the size of the action space to $|\mathcal{A}_s| = 10$ results in 9.68s/itr, which is consistent with the computational complexity introduced in Section 2.6.

For both examples, the negative correlation between k and the cost of privacy is consistent with the developments in Section 2.5. By Lemma 2.2, an increase in k results in a tighter concentration bound on the output of the Dirichlet mechanism, and it lowers α in Theorems 2.4 and 2.6. A smaller α further restricts the inner optimization problem in (P); thus, it helps the optimistic and the pessimistic value functions to provide better approximations, which leads to a lower cost of privacy.

2.8 Conclusion

We introduced a privacy-preserving policy synthesis algorithm that protects the privacy of the transition probabilities of its input MDP. The algorithm employs a Dirichlet mechanism to privatize the transition probabilities. We established a concentration bound on the output of the Dirichlet mechanism based on its scaling parameter k. We used the concentration bound to bound the cost of privacy imposed by privatizing the transition probabilities. We further showed that the cost of privacy can be computed efficiently by establishing that the computational complexity of the algorithm is polynomial in problem parameters. Finally, the simulation results validated the developments in both the soundness of the expressions we introduced for the cost of privacy and the computational complexity associated with computing them.

Chapter 3: Impact of Neural Network Architecture on Vulnerability to Membership Inference Attacks[1]

Abstract

In this chapter, we study the privacy implications of training recurrent neural networks (RNNs) with sensitive training datasets. Considering membership inference attacks (MIAs)—which aim to infer whether or not specific data records have been used in training a given machine learning model—we provide empirical evidence that a neural network's architecture impacts its vulnerability to MIAs. In particular, we demonstrate that RNNs are subject to a higher attack accuracy than feed-forward neural network (FFNN) counterparts.

3.1 Introduction

Machine learning models are frequently trained with personal, proprietary, operational, confidential, or otherwise sensitive datasets, which often raises privacy concerns. Even when these datasets are securely stored and safeguarded from unauthorized access, sharing the outputs of a machine learning model that has been trained with such sensitive data can lead to unintended information leakage (Mireshghallah et al., 2020; Rigaki and Garcia, 2020). Hence, it is imperative to foresee and preempt the privacy risks of training machine learning models with sensitive datasets.

We study the privacy risks of machine learning models that are powered by recurrent neural networks (RNNs) and compare them to their counterparts pow-

[1]The research presented in this chapter has been published in (Yang et al.). Parham Gohari formulated the problem, derived the theoretical results, designed the numerical experiments, and wrote the paper.

ered by feed-forward neural networks (FFNNs). As opposed to FFNNs, in which the nodes in every layer are only connected to subsequent-layer nodes, RNNs allow for backward connections in their architecture. RNNs are widely used in sequential machine learning tasks such as natural language processing (Wu et al., 2016), speech and handwriting recognition (Sak et al., 2014; Li and Wu, 2015; Graves et al., 2009), deep reinforcement learning (Li et al., 2015), and semantic segmentation of video sequences (Pfeuffer and Dietmayer, 2020). While the privacy risks of neural networks—irrespective of their architecture—have been subject to an active line of research, an account of whether or not the architecture of neural networks affects their privacy risks remains unknown.

We consider *membership inference attacks (MIAs)* as the underlying privacy threat. In an MIA, an adversary is allowed to query the output of a neural network with a collection of data records, and must subsequently infer whether or not those data records belong to the neural network's training dataset (Hu et al., 2022). Successful instances of these attacks with minimal access to the neural network can have significant privacy ramifications for the individuals who populate the training datasets with their data. For example, consider a machine learning model that has been trained with the data of individuals with certain characteristics such as a particular ethnic origin, religion, medical condition, gender, or sexuality. In this case, a successful MIA that asserts—or refutes—the membership of an individual's data may reveal those sensitive characteristics.

The main contributions of this chapter are twofold: the first contribution concerns how RNNs and FFNNs compare in their vulnerability to MIAs, and the second contribution concerns defending neural networks against MIAs.

In the first contribution, we design and conduct a series of experiments to compare vulnerability to MIAs in three representative machine learning tasks, namely image classification, machine translation, and deep reinforcement learning. In order to study the impacts of network architecture on vulnerability to MIAs, we are mindful

to separate other main factors that are known to impact vulnerability to MIAs such as overfitting (Salem et al., 2019; Shokri et al., 2017; Yeom et al., 2018; Bentley et al., 2020), number of trainable parameters (Nasr et al., 2018), diversity of the training data (Long et al., 2018), and number of prediction classes (Truex et al., 2021). Taking all of these factors into account, we observe that the MIAs consistently achieve a higher attack accuracy against the RNN models.

In order to investigate the root causes of the observed higher vulnerability in RNNs, we further study the behavior of the two architectures when they are queried with members of their training datasets and unseen data. We observe that when the uncertainty of the two models' predictions in terms of entropy is equal with respect to the validation dataset, the entropy with respect to the training data is lower in RNNs. Moreover, we demonstrate that the decisions of the MIAs resemble establishing a threshold for prediction entropy to distinguish member data from non-members. In such a threshold-based inference, a larger gap between the entropy of the predictions in training and validation—as observed in RNNs—increases attack accuracy. Moreover, we demonstrate that subsequent gradient updates in FFNNs can mask the membership of data used early in the history of training, whereas the MIAs remain relatively accurate even for such outdated data in RNNs.

In the second contribution, we shift the focus from vulnerability analysis to mitigation methods. A popular mitigation approach is to prevent overfitting as a root cause of vulnerability to MIAs—most prominently via weight regularization (Shokri et al., 2017; Salem et al., 2019). While weight regularization has been shown to be oftentimes highly effective for FFNNs to preempt MIAs, in the experiments, we demonstrate that the RNN models benefit from regularization only marginally. As a result, RNNs may be not only more vulnerable to MIAs but also harder to be defended against them.

Methods that leverage the promise of differential privacy are known to be the most effective in defending neural networks against MIAs (Hu et al., 2022). However,

the protection afforded by these methods typically comes at the expense of a reduction in utility in terms of model performance (Rahman et al., 2018). These methods impose an error margin on the inference power of MIAs and the error can be balanced against utility loss by adjusting the level of differential privacy (Yeom et al., 2018).

Existing methods that enforce differential privacy typically do so by obfuscating either of the following with noise: the objective function (Zhang et al., 2012) or the gradients calculated during training (Abadi et al., 2016a; McMahan et al., 2018), or the model's parameters post-training (Chaudhuri et al., 2011; Wu et al., 2016; Lu et al., 2022). As opposed to post-training methods in which the privacy level can be easily modified by modifying the level of noise that is added to the model's parameters, increasing the privacy level in the first two methods requires reversing training steps, which is challenging. As a result, post-training methods offer more flexibility in changing the privacy level. On the flip side, these methods may be less advantageous with respect to the privacy-utility trade-off that they face (Abadi et al., 2016a).

We compare RNNs and FFNNs in their utility loss due to differential privacy considering both approaches: the celebrated DP-SGD algorithm (Abadi et al., 2016a) which adds Gaussian noise to the gradients during training, and the Gaussian privacy module (GPM) which we introduce to add Gaussian noise to the trained parameters of a neural network post-training. For both methods, the experiment results indicate that adding the same level of noise degrades more utility in RNNs than FFNNs.

Although we develop the GPM solely to compare the utility loss of RNNs and FFNNs due to differential privacy, it may be of independent interest. We show that the GPM satisfies a relaxation of differential privacy called *random differential privacy* (Hall et al., 2013). In noise-additive differential privacy mechanisms, the noise is typically calibrated with the extent to which a single record of the training dataset can change the model's outcome, formally called *sensitivity*. Computing sensitivity analytically can be very challenging and one might have to resort to an upper bound for the sensitivity which can be too loose and subsequently cause too much noise to

be added (Chaudhuri et al., 2011; Wu et al., 2016; Lu et al., 2022). Alternatively, random differential privacy fixes a level of sensitivity with some confidence level and guarantees differential privacy for data records that give rise to that level of sensitivity.

We use the SensitivitySampler algorithm (Rubinstein and Aldà, 2017) to estimate the sensitivity of the models. We show that by utilizing these estimates, we are able to achieve an acceptable privacy-utility trade-off for the models in the experiments: reducing the MIAs' attack accuracy to roughly 50%—equivalent to random guessing—while trading off less than 10% utility. We further observe that the sensitivity estimates for the RNN and FFNN models in the experiments take similar values. As a result, adding the same level of noise to the two models using the DP-SGD algorithm and the proposed post-training mechanism, satisfies the same level of conventional and random differential privacy, respectively, yet it leads to more utility loss in RNNs than FFNNs. Since RNNs were consistently rendered more vulnerable to MIAs and more difficult to be defended, this chapter provides strong empirical evidence that the privacy risks of RNNs are more severe than FFNNs.

3.2 Preliminaries

In this section, we first review some background about the differences between RNNs and FFNNs. Then, we introduce the machine learning tasks that we consider in the experiments.

3.2.1 Recurrent vs. Feed-Forward Architecture

A neural network comprises a collection of nodes, each of which accepts an input and produces an output according to a fixed mapping called an activation function. The architecture of a neural network determines how the nodes of the networks are connected to one another. In a feed-forward architecture, the nodes can be stacked into an ordered sequence of layers from the network's input to its output such that the output of each node only affects the nodes in the subsequent layers.

Examples of FFNNs include multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), and more sophisticated designs such as transformers (Vaswani et al., 2017).

In a recurrent architecture, the connections between the nodes may form a cycle. The backward connection between an RNN's nodes can be unfolded into an infinite sequence of layers, each of which represents the node activations at different time steps. Therefore, an RNN's output depends on the entire history of its inputs, which results in exhibiting a temporally dynamic behavior. Such features make RNNs suitable for processing sequential data such as sentences, videos, and audio (Dupond, 2019).

FFNNs can also exhibit a temporally dynamic behavior through cascading MLPs or CNNs, or, more intelligently in transformers; however, the outputs in these methods only depend on a finite window in the history of inputs. Therefore, RNNs appear to be more expressive than FFNNs. On the other hand, it is easier to parallelize the training of FFNNs (Gehring et al., 2017). Furthermore, making predictions based on the entire history of inputs may be unnecessary as theoretically shown in (Sharan et al., 2018). As a result, there has been an increasing interest—with many successful instances—in replacing RNNs with FFNNs (Dauphin et al., 2017; Vaswani et al., 2017; Gehring et al., 2017).

3.2.2 RNN Applications Considered

In the experiments, we consider three representative machine learning tasks: image classification, machine translation, and deep reinforcement learning. In the sequel, we briefly introduce each of the tasks. Then, we state some of the possible privacy harms that MIAs may cause specific to these tasks.

In the image classification task, the model must label a given image using a fixed set of classes. The model's output is a probability vector that determines its confidence in assigning each of the labels to the given image. CNNs are dominantly

used in image classification (Gavrikov and Keuper, 2022; Graham, 2014); however, RNNs may also be used to process images as a sequence of pixels (Visin et al., 2015).

In image classification tasks, the models may be trained with labeled image datasets that contain sensitive information. For example, consider a healthcare provider who fine-tunes a medical image classifier to predict the risk factors pertaining to a certain disease for a certain minority population. In this case, an MIA with access to an aggregated list of patient records can infer whether or not some of the data subjects belong to the considered minority group.

In the machine translation task, the model must map a sequence of words, syllables, or otherwise tokens from a fixed input dictionary to a target dictionary. Both RNN and FFNN solutions use an *encoder-decoder* framework. The first half of the model—the encoder—computes an encoding of the input sequence through multiple encoder layers. Subsequently, the second half of the model—the decoder—uses the encoding and generates an output sequence through multiple decoder layers. The output of the model is a sequence of probability vectors over the target dictionary words. The dictionaries are appended with start and end tokens to signal the start and completion of sentences, respectively.

Recurrent architectures such as bi-LSTM (Wu et al., 2016) use a network of long-short term memory (LSTM) units to construct both the encoder and the decoder networks and can process arbitrary-length sequences. Feed-forward architectures such as transformers (Vaswani et al., 2017) fix a window of sequence lengths allowed and construct the encoder and the decoder networks using FFNNs. Both of the above network architectures are widely used in machine translation; however, since the debut of transformers in 2017, they have outperformed RNN-based models (Wolf et al., 2020).

As for the privacy risks that MIAs can pose to machine translation models, assume that some business analytics tool trains a model using internal meeting transcripts as training datasets. In this case, an MIA can infer whether or not a given

sentence has been discussed in the meetings.

Finally, in the deep reinforcement learning task that we consider, an agent must learn how to navigate through an unknown map and reach a target state under partial state observations. At every state observation, the agent must compute a probability vector over the available actions, which is called a policy. Under partial state observations, the optimal policy may require memory (Lusena, 2001) and RNNs can be integrated with deep reinforcement learning algorithms to capture long-term dependencies. Both MLPs and LSTMs are commonly used in deep reinforcement learning algorithms such as soft actor critic (Haarnoja et al., 2018), proximal policy optimization (PPO) (Schulman et al., 2017), etc.

In deep reinforcement learning tasks, an MIA can infer whether or not specific locations have been used to train the agent. For example, a new owner of an autonomous vehicle may be able to infer whether or not the previous owner has visited certain locations, thereby violating the previous owner's location privacy.

3.3 Methods

In this section, we describe the threat model that we consider for MIAs. Then, we lay out the methodology that we use to design the MIAs in the experiments and compare the MIA layouts with existing MIAs in the literature.

3.3.1 Threat Model and Assumptions

There exist two parties in the threat model that we consider: a victim and an attacker. The victim aims to train a neural network for a given machine learning task and a given training dataset. For example, the victim may train an image classifier using a dataset of labeled images. In order to train the neural network, the victim must choose a training algorithm alongside its hyperparameters, loss function, and neural network specifications—including the number of hidden layers, number of nodes per layer, architecture, activation functions, etc. Once the victim's neural network is

fully trained, the victim proceeds with generating predictions for outsider inquiries, i.e., the victim receives a data record and subsequently responds by publishing its predictions for the received data.

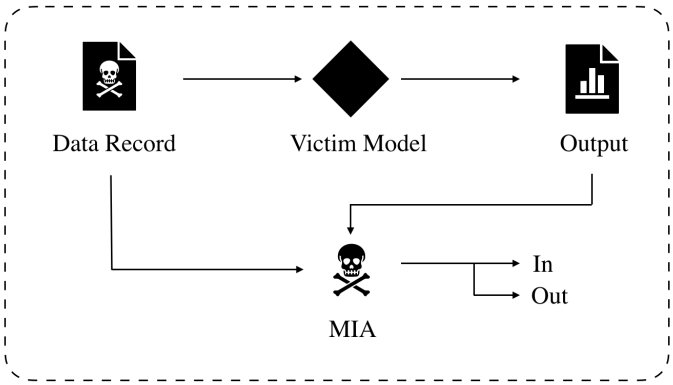

Figure 3.1: Schematic of the execution of an MIA.

The attacker in the considered threat model conducts an MIA; that is, it submits an inquiry to the victim using some data record and must infer whether or not the data record belongs to the victim's training dataset as depicted in Figure 3.1. MIAs are typically categorized into two groups: black-box and white-box attacks (Hu et al., 2022). The former group assumes that the attacker can only access the input and the output of the victim's neural network. White-box attackers may have access to the value of the weights and the output of the nodes anywhere in the victim's neural network. Additionally, white-box attacks may also probe the victim's loss function and its gradient for the queried data record.

In terms of the side-information that is available to the attacker, the survey in (Hu et al., 2022) assumes that a black-box MIA's side-information is limited to the distribution of the training data—implying that the attacker can obtain a training dataset that is similar to that of the victim. The survey considers any additional assumption on the available side-information as an indicator of white-box attacking. However, such a distinction about side-information is not uniformly followed in the literature; for example, the work in (Shokri et al., 2017) assumes that the attacker

knows the training algorithm of the victim, and the MIAs in (Sablayrolles et al., 2019) are provided with side-information about the victim's training algorithm, hyperparameters, and the network specifications, yet both works consider their MIAs as black-box attacks. Access to the value of the loss function and its gradient is the key enabler that enhances the attack accuracy in white-box attacks as empirically demonstrated by Nasr et al. (2019). We therefore draw the line between black-box and white-box attacks based on the access granted to the attacker and not the side-information.

We now state our assumptions on the attacker's access limits and side information. In terms of access limitations, we assume that the attacker has black-box access to the input and output layers of the victim's neural network. The attacker is able to access the input layer by submitting an unlimited number of inquiries and is able to observe the output layer by evaluating the confidence scores with which the victim responds to an inquiry. In terms of the side-information that is available to the attacker, we assume that the attacker knows the victim's task, the training algorithm alongside its hyperparameters, and the network specifications.

Sablayrolles et al. (2019) show that an optimal MIA—under mild assumptions on the distribution of the neural network's parameters—utilizes the victim's full confidence scores. Our goal in the experiments is to investigate whether or not the architecture of a neural network affects its vulnerability to MIAs. As a result, we assume full confidence-score observability in the threat model in an effort to maximize the accuracy of the MIAs and control the variables that have the potential of affecting the accuracy of MIAs besides architecture.

3.3.2 Designing MIAs

We follow the framework of *shadow models* (Shokri et al., 2017) in designing the MIAs in this chapter. Intuitively, a shadow model must mimic the victim's behavior without having access to its training dataset. Following the assumption

that the attacker knows the victim's training data distribution, we assume that there exists a data source from which the attacker can obtain a similar training dataset to train a shadow model—see Figure 3.2. In order to increase accuracy, MIAs often obtain multiple training datasets from the data source and subsequently train multiple shadow models to better mimic the victim's behavior.

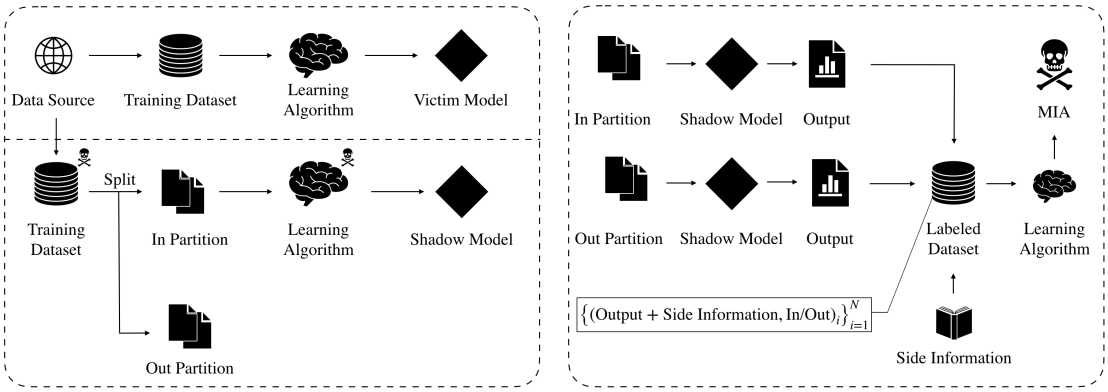

Figure 3.2: Left: a schematic of training shadow models. The skull above the shadow model's training dataset and learning algorithm indicates that the two components may differ between the victim and the attacker's side. Right: training the MIA using the output of the shadow model and side-information.

For the image classification and machine translation tasks, we allocate two disjoint partitions of a large dataset to the victim and the attacker, separately. Analogously for the deep reinforcement learning task, we use two disjoint sets of environment maps for the victim and the attacker to train their models.

In the next step, the attacker splits its training dataset into two partitions. The first partition will be used to train the shadow models following the side-information that the attacker has regarding the victim's training algorithm and network specifications. Once the shadow models are trained, the attacker is provided with a proxy to the victim's neural network. The attacker knows which data records it has used to train the shadow models and it knows that the second partition has not been used to train any of the shadow models. Therefore, the attacker can train a binary classifier to distinguish between the outputs that correspond to member data and those

corresponding to non-member data. The attacker can then use the binary classifier against the victim to execute the MIA as depicted in Figure 3.1.

Shadow-model-based MIAs typically use a 3-tuple format for the entries of the binary classifier's training dataset as shown in the bottom box in Figure 3.2. Each entry corresponds to a query that is made from a shadow model and the query may originate from either partition of the attacker's training dataset. The first element contains the shadow model's output, the second element indicates what the shadow model's optimal output must have been, and the third element indicates whether the query was made using a member data record or a non-member data record.

For the first element, we use the shadow model's full confidence score in vector format. If the shadow model's output is a sequence of predictions—as the case in machine translation and deep reinforcement learning—we concatenate the confidence scores into one vector.

For the second element, we use the query's corresponding label in the training dataset as a one-hot vector or a concatenation of a sequence of one-hot vectors—such labels are readily available in the image classification and machine translation tasks. In the deep reinforcement learning task, such a labeled training dataset does not exist; we, therefore, train a "labeling agent" to generate these labels. The labeling agent simply learns a reward-maximizing policy in the environment in which a shadow model's policy is queried. For each state observation at which the shadow model's policy is queried, the labeling agent provides its policy as a reference label.

Once the binary classifier's training dataset is fully populated, the attacker uses the first two elements as features and uses the third element as binary labels—member or non-member—and concludes the design of the MIA by training a binary classifier that distinguishes between member and non-member queries. The attacker then uses the trained binary classifier against the victim to execute the MIA as shown in Figure 3.1.

3.3.3 Connection with the Existing MIAs

We now state how the MIAs implemented in this chapter compare with the existing MIAs in the literature. Regarding the image classification task, our MIA design is identical to the design in (Shokri et al., 2017). However, for the machine translation and deep reinforcement learning experiments, the existing MIAs have minor incompatibilities with this chapter's threat model which we address by modifying them.

Song and Shmatikov (2019); Hisamoto et al. (2020) study developing MIAs specifically for machine translation models. However, neither of the two existing works assume full confidence-score observability because they both aim to design practical MIAs with minimal side-information assumptions. In particular, Song and Shmatikov (2019) develop an MIA that is intended to be used by individuals who wish to *audit* a natural language processing model—a process by which the individuals investigate whether or not their data has been used to train a natural language processing model. In this scenario, full confidence-score observability is not realistic and the authors feed a redacted list of the output word rankings to the MIAs, instead. We use the same MIA design as (Song and Shmatikov, 2019) except that we use full confidence scores instead of word rankings.

Hisamoto et al. (2020) use a similar design, but they take a further step towards developing practical MIAs and drop the assumption that the MIA knows the underlying distribution of the training data. However, the authors find that the resulting MIAs are not effective as their accuracy does not exceed random guessing by much.

We now review the existing MIAs in deep reinforcement learning tasks. The work in (Pan et al., 2019)—which we follow closely in our MIA design for deep reinforcement learning—is the first to consider a privacy attack against reinforcement learning agents that resembles MIAs. However, instead of modeling the MIA as a binary classifier, the privacy attack uses a multi-class classifier. As a result, the

attacker must train a labeling agent for every possible environment map prior to the execution of the attack. By using a binary classifier, we train labeling agents only for the environment map with which the MIA is faced. In another work, Gomrokchi et al. (2021) develop an MIA that infers the membership of a batch-constrained deep Q-learning agent's roll-out trajectories stored in its replay buffer. We do not follow the above work's methodology because we do not restrict the algorithm that is used to train the reinforcement learning agents.

3.4 Vulnerability to Privacy Threats

In this section, we report and analyze the results of a series of experiments by which we compare the vulnerability of RNNs and FFNNs to MIAs. In order to perform a meaningful comparison, we must control factors that affect vulnerability to MIAs other than network architecture. We review these vulnerability factors and discuss how we take them into account in our experimental setup. Finally, we report and analyze the numerical results.

3.4.1 Vulnerability Factors

Overfitting has been extensively studied as the main source of the vulnerability of machine learning models to MIAs (Hu et al., 2022). Overfitting refers to the condition in which a machine learning model performs poorly when queried with data records outside its training dataset. There exist mounting empirical evidence that MIAs are more successful against models that overfit their training data (Salem et al., 2019; Shokri et al., 2017). However, there also exist successful instances of MIAs used against models with relatively low overfitting (Long et al., 2018). In these instances, the underlying distribution of the training data as well the size of the training datasets may leave some data records more vulnerable than others. By identifying such data records, MIAs may still maintain a high attack accuracy for models with low overfitting (Hu et al., 2022).

A theoretical account of the connection between overfitting and MIA accuracy remained unknown until the work in (Yeom et al., 2018). The said work characterizes overfitting by *average generalization error* defined as

$$R_{\text{gen}} = \mathop{\mathrm{E}}_{\substack{S\sim\mathcal{D}^n \\ z\sim\mathcal{D}}}[\ell_S(z)] - \mathop{\mathrm{E}}_{\substack{S\sim\mathcal{D}^n \\ z\sim\mathcal{S}}}[\ell_S(z)], \tag{3.1}$$

where $\mathcal{D}$ is the underlying distribution of the training data; S is the victim's training dataset comprising n samples drawn from $\mathcal{D}$; ℓ is a fixed loss function; and $\ell_S(\cdot)$ is the value of the model's loss function after being trained with S.

Under the assumption that the victim's loss function is bounded above and its value is accessible to the attacker, Yeom et al. establish that a higher average generalization error is a sufficient condition—but not necessary—for a higher attack accuracy (Yeom et al., 2018). The authors further provide empirical evidence that the sufficiency relationship holds when the assumptions are relaxed to black-box MIAs. Later, a theoretical account of the relationship between the generalization gap—training accuracy minus validation accuracy—and the accuracy of black-box MIAs was provided in (Bentley et al., 2020).

In light of the established relationship between overfitting and attack accuracy, we are mindful to consider RNNs and FFNNs with similar training and validation performance levels. In order to control the effect of the size and the distribution of the victim's training dataset on vulnerability to MIAs, we use the same training dataset for both of the RNN and FFNN models. As the vulnerability to MIAs may not be evenly distributed across a collection of data records (Long et al., 2018), we evaluate the MIAs against the RNN and FFNN models using the same dataset. Finally, we consider RNNs and FFNNs whose number of parameters are close because it has been empirically demonstrated that a higher number of parameters increases vulnerability to MIAs (Nasr et al., 2019).

3.4.2 Experimental Setup

We consider three representative machine learning tasks for the experiments of this section: image classification, machine translation, and deep reinforcement learning. In image classification, consistent with the threat model, we assume that there exists a data source that generates labeled image samples and use the `CIFAR10` dataset (Krizhevsky, 2009) as $50,000$ samples drawn from the data source. We split these samples evenly into two partitions: one used by the victim and the other used by the attacker for the training of the shadow models—we train 5 shadow models.

With the victim's portion of the training samples, we separately train an FFNN model and an RNN model. The FFNN model is an instance of ResNet101 (He et al., 2016) implemented in the Keras library and specified as follows: 101 convolutional layers followed by one max-pooling layer, one fully connected linear layer, and an output layer with softmax activation. For the RNN model, we use ResNet (Visin et al., 2015) implemented by PyTorch under default parameters, which has the following specifications: 4 bi-directional LSTMs, 2 fully connected layers with ReLU activation, and an output layer with softmax activation. The former model contains $42,678,666$ trainable parameters and the latter has $42,569,590$ trainable parameters. Both models use the categorical cross-entropy loss function as their learning's objective function and use the Adam optimizer. The learning rates used are 0.001 and 0.01 for the former and the latter model, respectively. The shadow model of the MIAs uses the same specifications as the victims for their training.

For the machine translation experiments, we choose a translation from French to English. Similar to image classification, we assume there exists a data source from which translated pairs of English and French sentences can be sampled. We take the `Multi30K` dataset (Elliott et al., 2016) as $30,000$ samples from the data source and split the samples evenly between the victim and the attacker.

For the RNN model, we use a bi-directional LSTM with a dot-product attention mechanism developed in (Luong et al., 2015). For the FFNN model, we use the

standard transformer network specified in (Vaswani et al., 2017). The RNN model and the FFNN model have $3,213,191$ and $3,225,714$ trainable parameters, respectively. Both networks use the negative log-likelihood function as their learning algorithm's loss function and use Adam optimizer with a learning rate of 0.001.

Finally, for the deep reinforcement learning task, we use the *MiniGrid Multi-Room N4 v0* environment from the MiniGrid toolkit (Chevalier-Boisvert et al., 2018). The victim's goal is to train a deep reinforcement learning agent that can navigate its way through four rooms with closed doors and reach the green tile as shown in Figure 3.3. The victim is provided with a limited number of floor-maps for training and must generalize to unseen floor-maps. The attacker's goal, on the other hand, is to infer the membership of floor-maps. In this experiment, the MiniGrid toolkit serves as the data source and the attacker is able to obtain an arbitrary number of floor-maps by feeding a randomly generated seed number to the toolkit's simulator.

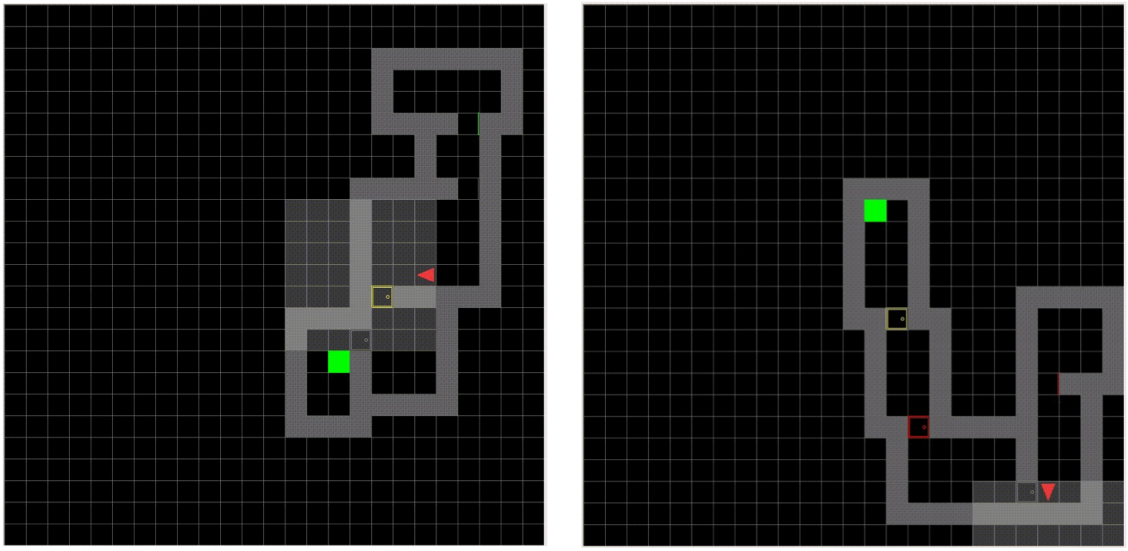

Figure 3.3: The MiniGrid-MultiRoom-N4-v0 environment. The agent (red triangle) must find the goal state (green square) while observing the highlighted box that surrounds it. The environment consists of multiple floor maps, two of which are shown above.

For the FFNN agent, we use an MLP network with $5,335$ trainable parame-

ters and the following specifications: the actor network has two hidden layers with dimension 74, and the critic network has 2 hidden layers with dimension 64. Both networks use a softmax output layer and tanh as their activation function. The RNN agent uses an MLP with some additional LSTM cells. The RNN has $5,216$ trainable parameters and its specifications are as follows: both the actor and the critic networks have 2 linear layers with hidden dimension 32, 4 single-directional LSTM cells, and a softmax output layer. We use the PPO algorithm (Schulman et al., 2017) implemented by the RL-Starter-Files library (Willems, 2018) with default parameters to train the victim agents and their respective shadow models and labeling models.

3.4.3 Numerical Results

Following the experimental setup above, we train each of the described RNN and FFNN models for a range of epoch numbers and plot the training and validation performance of the models. For the image classification experiment, we use the percentage of the correct predictions—or prediction accuracy—as the performance measure; for machine translation, we measure performance using the bilingual evaluation understudy (BLEU) score (Papineni et al., 2002), which captures how a model's translation correlates to that of a human; and for deep reinforcement learning, we use the total episodic reward as the performance measure. The reward at time-step t is

$$r(t) = \begin{cases} 1 - \gamma \frac{t}{T}, & \text{goal reached,} \\ 0, & \text{otherwise,} \end{cases} \tag{3.2}$$

where T is the episode length—set to 200 in the experiments— and γ is the discount factor—set to 0.9.

At every epoch number tested, we train a separate MIA whose shadow models are trained for the same number of epochs as that of the victim. We evaluate the performance of the MIAs by measuring the percentage of correct inferences which we refer to as attack accuracy. In all instances, the MIAs' validation datasets have an equal number of members and non-member records; hence, random guessing in this case achieves 50% attack accuracy.

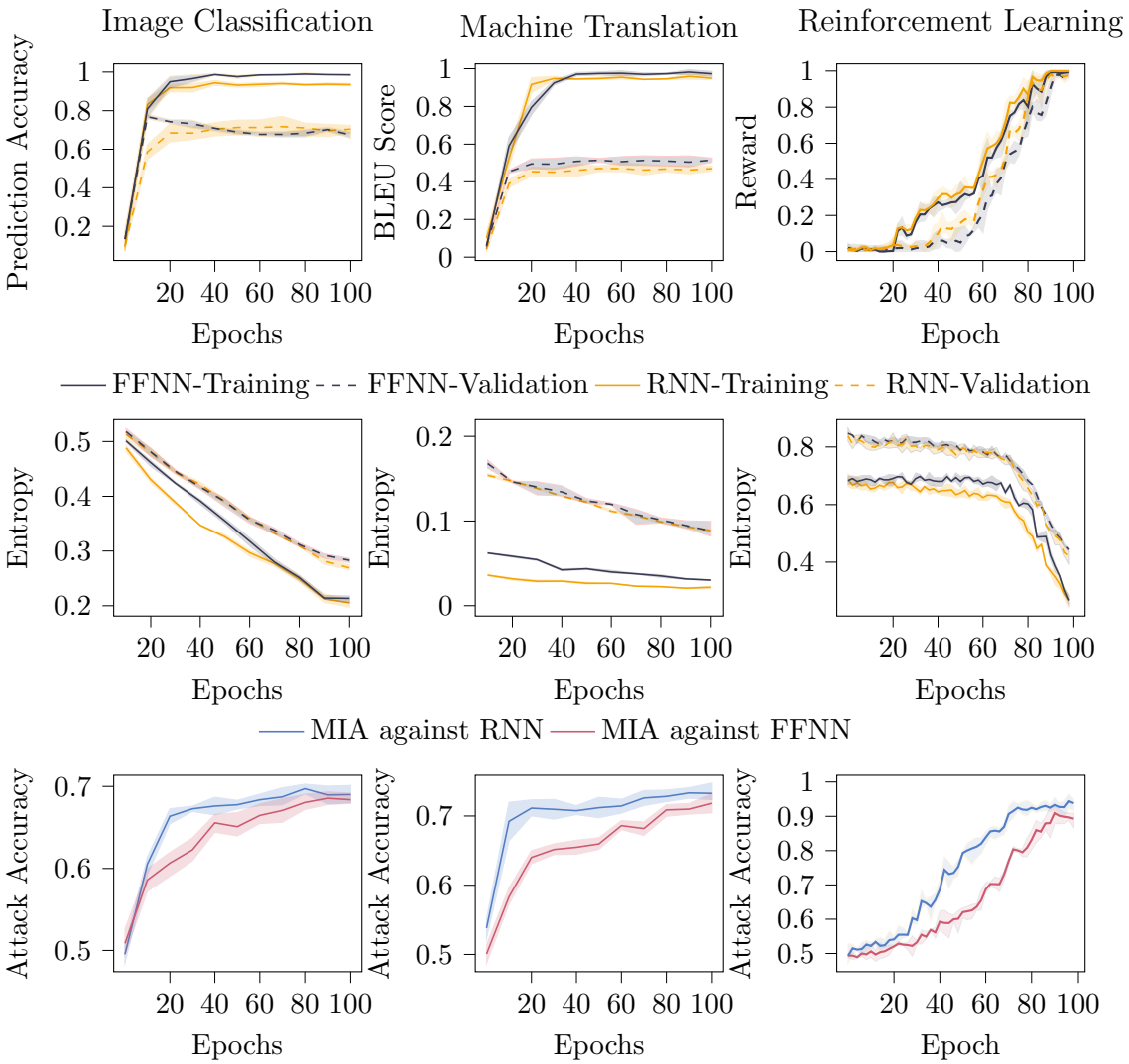

Figure 3.4: Comparing the vulnerability of RNNs and FFNNs to MIAs in three representative machine learning tasks: image classification (left), machine translation (center), and deep reinforcement learning (right). The first row plots training and validation performance vs. number of epochs; the second row plots the average prediction entropy vs. number of epochs; and the third row plots attack accuracy vs. number of epochs.

In Figure 3.4, it can be observed that the attack accuracy against the RNN models is consistently higher than it is against FFNN models. In particular, The RNNs are more vulnerable before and after the performance level of the victims

converges; however, the gap between the attack accuracy of the two MIAs narrows as the models train for higher epoch numbers.

In the image classification experiment, the validation performance of the RNN and FFNN models are approximately equal. The FFNN model has a higher generalization gap than the RNN model upon convergence, and it has slightly more trainable parameters, yet surprisingly, the attack accuracy against the FFNN model is lower than the RNN model. In the machine translation experiment, the two models have approximately equal performance levels both in training and validation but the MIA against the RNN achieves a higher attack accuracy. Finally, in the deep reinforcement learning experiment, the two models appear to have a zero generalization gap upon convergence, yet the MIA against the RNN model is more accurate than it is against the FFNN model.

Prediction Entropy as a Vulnerability Factor: In order to further investigate the reasons behind the excessive vulnerability of RNNs to MIAs, we measure the uncertainty of the models' outputs in terms of *average prediction entropy*, which we define as follows: let m be the number of prediction categories and $Y = \left\{\left(y^{(i)}, p^{(i)}\right)\right\}_{i=1}^{L}$ be a sequence of L pairs of prediction outcomes and confidence scores, respectively. Then, the corresponding average prediction entropy is

$$H_{\text{APE}}(Y) = -\frac{1}{L}\sum_{i=1}^{L}\sum_{j=1}^{m} p_j^{(i)} \log\left(p_j^{(i)}\right). \tag{3.3}$$

We report the average prediction entropy of the RNN and the FFNN models in Figure 3.4. The results show that, while the prediction entropy of the two models is approximately equal over the validation dataset, their prediction entropy with respect to their training data differs noticeably—at least in the early stages of training. In the initial stages, the entropy gap between the validation and the training dataset in RNNs is larger than FFNNs. As the training prediction entropy of the FFNN model approaches that of the RNN model, the gap between the attack accuracy of

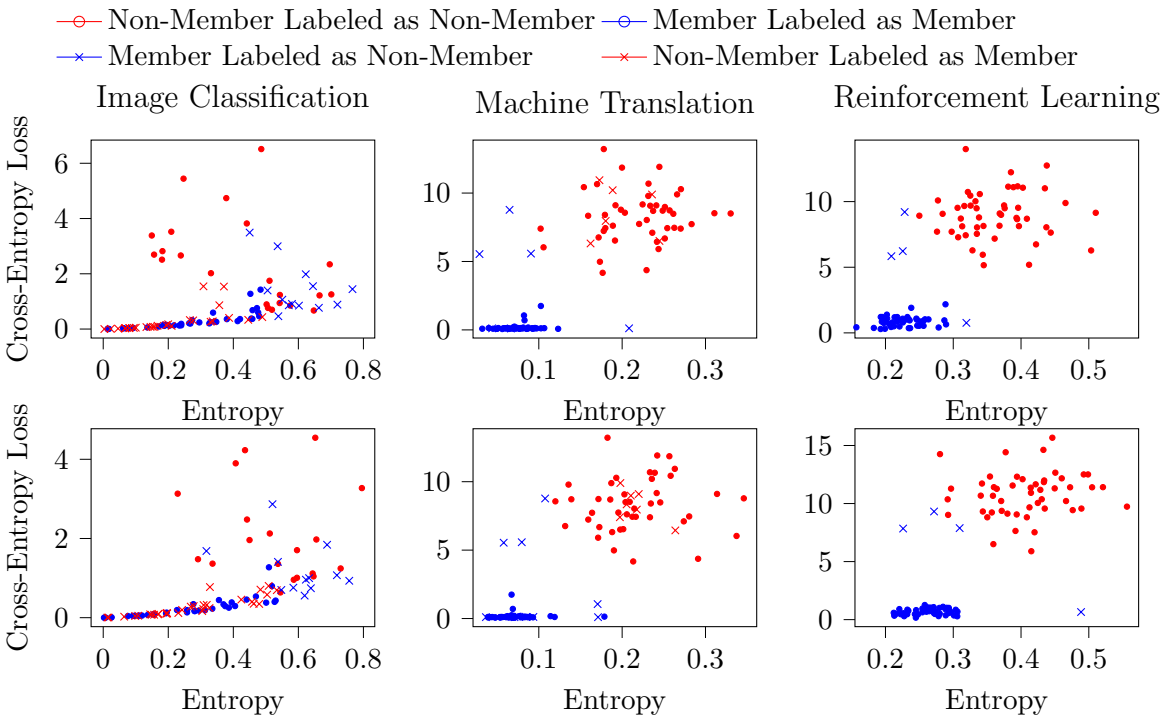

Figure 3.5: An illustration of the decision boundaries of MIAs with respect to cross-entropy loss and entropy of the confidence scores. The top row shows the results for RNNs and the bottom row shows the results for FFNNs.

the two MIAs narrows. As a result, the ability of RNNs to maintain a lower prediction entropy than FFNNs vis-á-vis member data records may render them more vulnerable to MIAs.

In the next experiment, we demonstrate that the MIAs are indeed sensitive to the entropy of the victim's predictions. To illustrate this, for each inference made by the MIA, we measure the victim's cross-entropy loss—as a performance measure—and we measure the prediction entropy of the victim. Then, we generate a scatter plot in which the y-axis represents cross-entropy and the x-axis measures prediction entropy. We use two colors to distinguish between member and non-member inferences made by the MIA and use a distinct marker to represent erroneous inferences. The results in Figure 3.5 suggest that the MIAs divide the scatter area into 4 quadrants and label

data records with cross-entropy loss below a certain threshold and entropy below a certain threshold as member data.

Model Memorization as a Vulnerability Factor: In the last experiment, we demonstrate that the RNN and FFNN models also differ in the way they retain their performance with respect to member data post-training. If a model responds to a post-training query using member data with the same accuracy as it previously held while being trained with that member data, we say that the model has *memorized* its training data. If the model's accuracy for such a query decreases after training, we say that the model has *forgotten* the training data.

Model memorization, if not associated with overfitting, is favorable from a performance-maximizing perspective. For example, in the deep reinforcement learning experiment, both the RNN and the FFNN agents reach the goal state within roughly 35 time steps when validated in unseen floor maps. However, when the RNN agent is queried in a member floor map, it reaches the goal in approximately 20 time steps, whereas the FFNN agent still reaches the goal in 35 time steps. As a result, the RNN agent appears to memorize the floor maps whereas the FFNN agent seems to forget. We note that the reward function used in the training of the agents is relatively insensitive to the number of steps taken to reach the goal. Instead, it is more sensitive to whether or not the agent reaches the goal at all in an episode. In particular, a 75% increase in the number of steps from 20 to 35 decreases the total reward only by 7.42% according to (3.3). Hence, the RNN agent appears to memorize the floor maps even though it was not specifically incentivized by the reward system to do so.

From a privacy perspective, such discrepancies between a model's performance with respect to seen and unseen data are harmful as they can be exploited by an adversary via MIAs. To illustrate this, we partition the training datasets into a collection of disjoint batches of data and assign an order to each batch arbitrarily at random. We then use these batches sequentially to train the RNN and FFNN models.

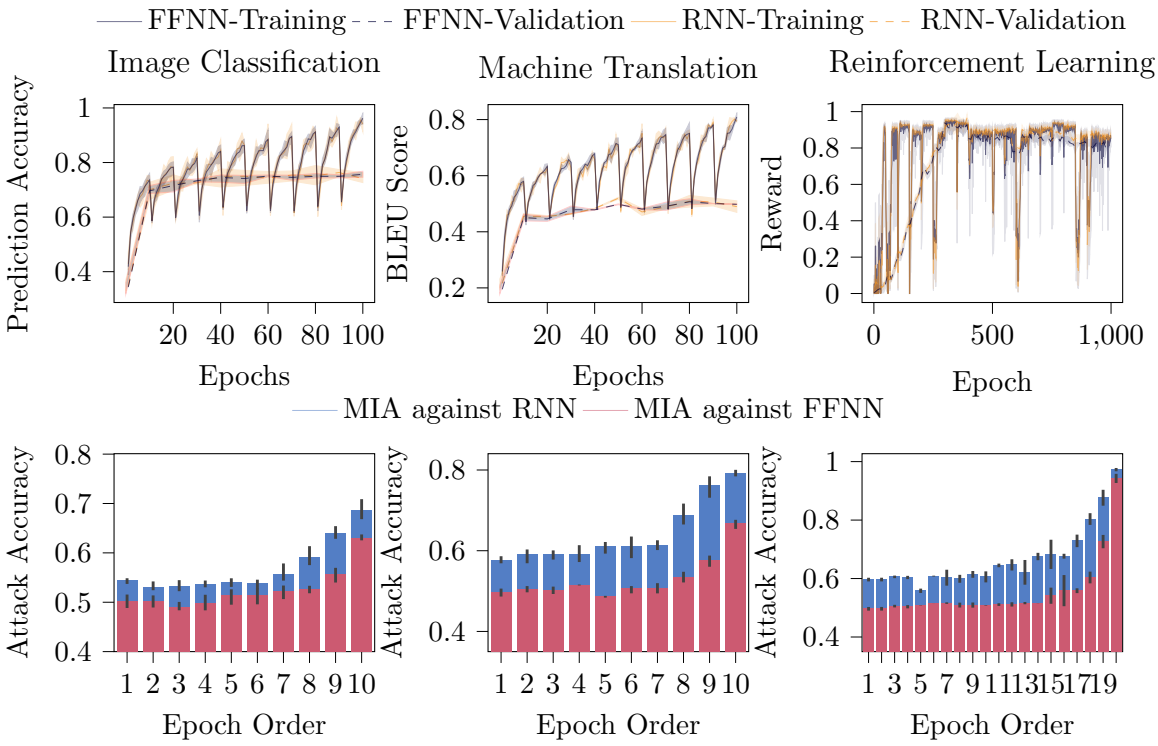

Figure 3.6: Comparing model memorization in RNNs and FFNNs. The top row plots the training and validation performance of the models when trained sequentially with a collection of ordered batches of training data. The saw-tooth pattern in the training performance is common in sequential training of machine learning models and is due to the *catastrophic interference* phenomenon (McCloskey and Cohen, 1989). The validation lines are smoothed out by measuring performance at the end of every 10 epochs. The bottom row plots the attack accuracy of the MIAs with respect to the individual epochs in the order in which they were introduced to the victims during training.

Once the two models are trained, we report the accuracy of the MIAs with respect to the percentage of correct inferences vis-á-vis each batch. The results in Figure 3.6 indicate that the MIAs' accuracy for older batches of data in FFNNs quickly diminishes to 50%, whereas in RNNs, the MIAs maintain non-trivial accuracy even for the early batches of data. As a result, we posit that model memorization is another factor that renders RNNs more vulnerable to MIAs than FFNNs.

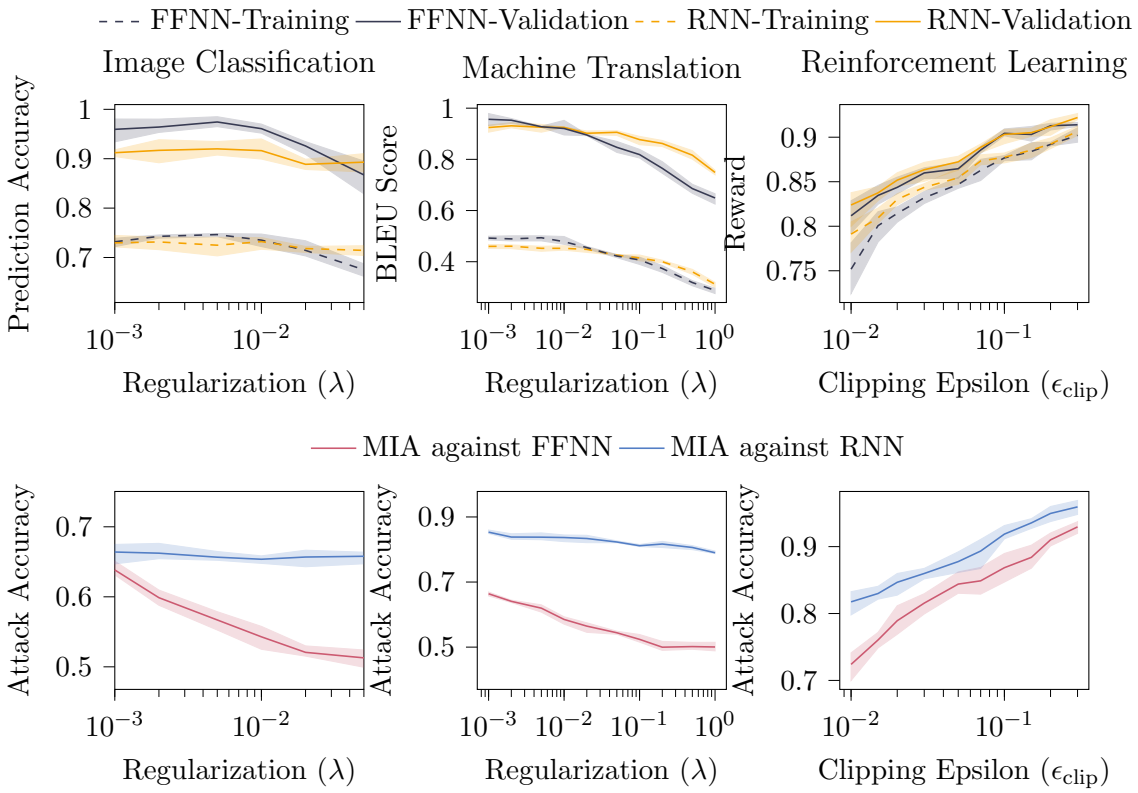

Figure 3.7: Effects of regularization on attack accuracy. From the left to the right column: image classification, machine translation, and deep reinforcement learning. The MIAs are trained separately for each regularization value tested and the shadow models use the same parameters as their victims.

3.5 Preempting Privacy Threats

In this section, we shift the focus from studying vulnerability to studying defense methods against MIAs. We first briefly discuss regularization methods, then, we study methods that leverage the promise of differential privacy.

3.5.1 Defense via Regularization

We now investigate the effects of overtraining and regularization in the considered machine learning tasks. Increasing the training time of machine learning algorithms often results in overfitting. For example, the validation performance of

the FFNN model in the image classification task decreases after training for 10 epochs in Figure 3.4, whereas its training performance keeps increasing. On the other hand, training machine learning models for an extended number of epochs may not always lead to overfitting. Such a phenomenon in RNNs was first reported in (Song and Shmatikov, 2019) for natural language processing models which is consistent with our results in Figures 3.4.

Regularization methods such as ℓ_2-regularization are effective in preventing overfitting and they have been shown to be effective in reducing the vulnerability of FFNN image-classification models to MIAs (Shokri et al., 2017; Salem et al., 2019). However, regularization may add bias to the converging performance levels because they alter the objective function. In particular, these methods compute the ℓ_2-norm of the node activations as a penalty term, which is subsequently multiplied with a regularization coefficient λ and added to the model's loss function. In Figure 3.7, we observe that regularization affects the FFNN and RNN models in the image classification and machine translation experiments differently. In particular, the MIA accuracy in the FFNN models is highly sensitive to the regularization coefficient λ, whereas the MIA accuracy against RNN models is impacted by regularization only marginally.

For the deep reinforcement learning agents, we test a different method of regularization. It is common in deep reinforcement learning algorithms such as the PPO and trust-region policy optimization (TRPO) (Schulman et al., 2017) to regularize the Kullback-Leibler divergence between the policy updates in order to increase model stability (Liu et al., 2019). In the PPO algorithm, which we use to train the RNN and FFNN agents in the deep reinforcement learning experiment, a parameter called the clipping epsilon ϵ_{clip} controls the policy updates as follows: a small value of ϵ_{clip} prevents the agent from taking large gradient steps whereas a large epsilon does not restrict the agent as much. In this case, the validation performance of both the RNN and FFNN agents is sensitive to regularization. However, the RNN agent remains

more vulnerable to the MIA than the FFNN agent, and its respective MIA accuracy is relatively less sensitive to regularization based on the corresponding line slopes.

3.5.2 Defense via Differential Privacy

Recall from Chapter 1 that differential privacy is a characteristic of an algorithm and provides a quantitative definition of data privacy. A differentially private algorithm makes it hard for any observer to link the algorithm's outputs to the individual entries of the dataset that contributed to generating that output. It is best justified to use differential privacy when the purpose of the algorithm is to compute some aggregate information about a dataset whose entries contain sensitive information. For example, the US Census Bureau uses differential privacy to protect the data subjects in its publications (Abowd, 2018). We now revisit the formal definition of (ϵ, δ)-differential privacy (Dwork and Roth, 2014).

Definition 3.1. Let $f : \mathcal{D} \mapsto \mathcal{R}$ be a query function from an input domain $\mathcal{D}$ to an output domain $\mathcal{R}$. Define two datasets D and D'—both in $\mathcal{D}$—adjacent if the number of the entries in which the two datasets hold different values is at most one. Let $(\Omega, \mathcal{F}, \mu)$ be a probability space and $\mathcal{M}$ be a σ-algebra such that $(\mathcal{R}, \mathcal{M})$ is measurable. For a given $\epsilon \geq 0$ and $\delta \in [01]$, a mechanism $M : \mathcal{R} \times \Omega \mapsto \mathcal{R}$ satisfies (ϵ, δ)-differenetial privacy if, for all $R \subseteq \mathcal{R}$ and all adjacent D and D',

$$\mathrm{P}\left[M\left(f(D)\right) \in R\right] \leq \exp(\epsilon) \cdot \mathrm{P}\left[M\left(f\left(D'\right)\right) \in R\right] + \delta. \tag{3.4}$$

If a mechanism satisfies (3.4) with $\delta = 0$, it satisfies pure ϵ-differential privacy. Intuitively, the parameter ϵ captures the strength of privacy protections and δ captures the probability that pure ϵ-differential privacy fails. Privacy failure could happen due to two reasons: either (3.4) holds for a larger ϵ or no finite ϵ ever satisfies pure differential privacy. It is customary to choose single-digit values for ϵ and choose δ to be $\mathcal{O}\left(|D|^{-1}\right)$, where $|D|$ is the size of the dataset that we wish to protect (Dwork and Roth, 2014). However, in some applications, even large values for ϵ may still provide a strong privacy shield (Bhowmick et al., 2018).

Differential privacy is immune to post-processing, meaning that post-hoc computations on the output of a differentially private mechanism do not affect the level of differential privacy. Subsequent queries from the output of a differentially private mechanism may weaken privacy, however. In general, the overall privacy level of a sequence of k queries from an (ϵ, δ)-differentially private mechanism results in $(k\epsilon, k\delta)$-differential privacy according to the Composition Theorem (Dwork and Roth, 2014). The overall privacy level is often referred to as the *privacy budget*. In applications wherein multiple queries are made from some sensitive dataset, one must be mindful of the total privacy budget expended.

3.5.2.1 Enforcing Differential Privacy

The methods that we use in this section to enforce differential privacy utilize the Gaussian mechanism for differential privacy. The mechanism adds a zero-mean Gaussian noise to the output of a query function with a sensitive input dataset. The mechanism calibrates the variance of the noise based on the *sensitivity* of the query function, defined as follows:

Definition 3.2. Let $f : \mathcal{D} \mapsto \mathcal{R}$ be a query function that maps from a dataset domain $\mathcal{D}$ to a normed space $\mathcal{R}$. The sensitivity of f, denoted $\mathrm{S}(f)$, is

$$\mathrm{S}(f) := \sup_{D,D'} \|f(D) - f(D')\|, \tag{3.5}$$

where $\|\cdot\|$ is the norm operator and D and D' are any two adjacent datasets under the definition of adjacency established in Definition 3.1.

The following theorem from (Dong et al., 2019), see Section 2.4 therei, establishes the (ϵ, δ)-differential privacy of the Gaussian mechanism.

Theorem 3.1. *Let f be a query function with sensitivity $\mathrm{S}(f)$. Fix $\sigma > 0$ and define the Gaussian mechanism as $M_G(f(D); \sigma) = f(D) + \xi$ such that $\xi \sim \mathcal{N}(0, \sigma^2\mathbf{I})$. For all $\epsilon > 0$, let*

$$\delta = \Phi\left(-\frac{\epsilon}{\mu} + \frac{\mu}{2}\right) - \exp(\epsilon)\Phi\left(-\frac{\epsilon}{\mu} - \frac{\mu}{2}\right), \tag{3.6}$$

where $\mu = \frac{\mathrm{S}(f)}{\sigma}$ *and* Φ *is the cumulative distribution function of the standard normal distribution. Then, the Gaussian mechanism satisfies* (ϵ, δ)*-differential privacy.*

Later in the experiments of this section, we deploy the Gaussian mechanism in two algorithms: the DP-SGD algorithm (Abadi et al., 2016a) in which the Gaussian mechanism is used to privatize the gradients during the training of a neural network and a post-training privacy mechanism in which we deploy the Gaussian mechanism to privatize the trained parameters of a neural network.

DP-SGD modifies the stochastic gradient descent (SGD) algorithm such that the training algorithm itself satisfies differential privacy. In particular, the mechanism M in Definition 3.1 is the training algorithm that maps a training dataset to a set of network parameters. The mechanism M repeatedly performs the following at every update step: clips the gradients computed over a batch of training data; it computes the average of the clipped gradients; invokes the Gaussian mechanism to privatize the gradients; and finally, performs an SGD update with the privatized gradient. In other words, DP-SGD repeatedly applies the following update rule:

$$\text{DP-SGD:} \quad \theta_{i+1} = \theta_i - \eta \frac{1}{|X_i|} M_{\mathrm{G}} \left(\sum_{x \in X_i} \mathrm{CL}(g_x(\theta_i), C); C\sigma \right), \tag{3.7}$$

where i is the current iteration number and θ_i is the neural network's trainable parameters at iteration i; η is the learning rate; X_i is a minibatch of training data; and with ℓ the loss function and C a fixed scalar,

$$g_x(\theta_i) := \frac{\partial \ell(\theta, x)}{\partial \theta}\Big|_{\theta=\theta_i} \quad \text{and} \quad \mathrm{CL}(g_x, C) := g_x \cdot \min\left(1, \frac{C}{\|g_x\|}\right) \tag{3.8}$$

are the calculated gradient and the clipping function, respectively.

DP-SGD comprises a *moments accountant* subroutine that tracks the total privacy budget expended during training. The predictions that the resulting neural network subsequently generates post training preserve differential privacy with the same privacy budget because (i) differential privacy is immune to post-processing and

Algorithm 2: GAUSSIAN PRIVACY MODULE

Input: hyperparameters $\mathcal{H}$ and training algorithm $\mathcal{A}_{\mathcal{H}}$, training dataset D, Gaussian mechanism variance σ, query set X

Output: $\tilde{Y}$

$\theta \leftarrow \mathcal{A}_{\mathcal{H}}(D)$; /* Train NN_θ*/

$\tilde{\theta} \leftarrow M_{\mathrm{G}}(\theta; \sigma)$; /* Invoke the Gaussian mechanism*/

for x_i *in* X **do**

  $\tilde{y}_i \leftarrow \mathrm{NN}_{\tilde{\theta}}(x_i)$; /* Respond to each query*/

$\tilde{Y} \leftarrow \{y_i, i = 1 \dots |X|\}$

(ii) the privatized gradients fully characterize the trained neural network given a fixed initialization θ_0.

DP-SGD invokes the Gaussian mechanism at every gradient update step; therefore, subsequent gradient updates can mitigate the negative impacts of injecting noise on the model's utility. However, some queries may require higher privacy—thus less precision—than others. In order to increase the level of privacy in DP-SGD, some weight updates must be reversed, which is challenging. As a result, the DP-SGD algorithm may only be suitable for applications in which the underlying privacy interests necessitate *limiting* the flow of information about the training data, as opposed to those necessitating a discretionary *control* over the flow of such information.

A post-training privacy mechanism that mounts on a fully trained model as an external module can offer the flexibility required for controlling the flow of information. In this case, instead of having to retrain the model, one can apply changes to the privacy module. In Algorithm 2, we introduce the Gaussian privacy module (GPM) which is our proposed post-training privacy mechanism. By using the GPM, adjusting the level of privacy becomes as simple as a one-time adjustment of the variance of the Gaussian mechanism.

We now reconcile Algorithm 2 and Theorem 3.1 to compute the privacy budget that the GPM consumes. The first step of the algorithm—where the weights are calculated by the training algorithm $\mathcal{A}_{\mathcal{H}}$—characterizes the query function f in The-

orem 3.1. In order to compute the privacy parameters according to (3.6), one must know the sensitivity of the query function, $\mathrm{S}(f)$, a priori. The training algorithm maps a training dataset to a set of network parameters and its sensitivity captures the extent to which adjacent training datasets generate different parameters. Without any restricting measures, the sensitivity can be arbitrarily large. The DP-SGD algorithm faces the same issue of unbounded sensitivity and uses gradient clipping to limit sensitivity. Inspired by the gradient-clipping trick to bound sensitivity in DP-SGD, by the following theorem, we establish an upper bound on the sensitivity of Algorithm 2 for training algorithms whose loss function are L-Lipschitz and use SGD with gradient clipping and smoothing.

Theorem 3.2. *With $\mathcal{H}$ a fixed set of hyperparameters, including a fixed initialization and a fixed seed for generating random numbers, let $\mathcal{A}_{\mathcal{H}}$ be an* SGD *algorithm modified with gradient clipping and loss-function smoothing; that is, at every iteration i,*

$$\theta_{i+1} = \theta_i - \eta \frac{1}{|X_i|} \sum_{x \in X_i} \mathrm{CL}\left(\mathrm{E}_{Z \sim \mathcal{N}(0, \sigma_s^2 \mathbf{I})}\left[g_x(\theta_i + Z)\right], C\right), \tag{3.9}$$

where σ_s^2 is the smoothing variance. Let the loss function ℓ be L-Lipschitz, m be the minibatch size, and $\beta = L/\sigma_s$. Then, after training for T iterations, it holds that

$$\mathrm{S}(\mathcal{A}_{\mathcal{H}}) \leq 2 \frac{(1+\eta\beta)^T - 1}{(m-1)\beta} C. \tag{3.10}$$

Proof. Let $\bar{\ell}(\theta_i, x) = \mathrm{E}_{Z \sim \mathcal{N}(0, \sigma_s^2 \mathbf{I})}\left[\ell(\theta_i + Z, x)\right]$. Such an operation is known as randomized smoothing which transforms the L-Lipschitz loss function ℓ into L/σ_{s}-smooth $\bar{\ell}$ (Scaman et al., 2020); that is,

$$\left\| \frac{\partial \bar{\ell}(\theta, x)}{\partial \theta}\Big|_{\theta=a} - \frac{\partial \bar{\ell}(\theta, x)}{\partial \theta}\Big|_{\theta=b} \right\| \leq \frac{L}{\sigma_{\mathrm{s}}} \|a - b\|. \tag{3.11}$$

We also have that

$$\frac{\partial \bar{\ell}(\theta, x)}{\partial \theta}\Big|_{\theta=\theta_i} = \mathrm{E}_{Z \sim \mathcal{N}(0, \sigma_s^2 \mathbf{I})}\left[g_x(\theta_i + Z)\right]. \tag{3.12}$$

Considering SGD's update rule with clipped gradients and randomized smoothing, we have that, for two adjacent datasets D and D' and their respective minibatches at stage 0, X_0 and X_0',

$$\theta_1 = \theta_0 - \eta \frac{1}{|X_0|} \sum_{x \in X_0} \mathrm{CL}\left(\frac{\partial \bar{\ell}(\theta, x)}{\partial \theta}\Big|_{\theta_0}, C \right) \tag{3.13}$$

and

$$\theta_1' = \theta_0 - \eta \frac{1}{|X_0'|} \sum_{x' \in X_0'} \mathrm{CL}\left(\frac{\partial \bar{\ell}(\theta, x')}{\partial \theta}\Big|_{\theta_0}, C \right). \tag{3.14}$$

The two minibatches can only differ in one data record and fixing the random seeds ensures that the same data indices will be chosen for both X_0 and X_0'. As a result,

$$\begin{aligned} \|\theta_1 - \theta_1'\| = \\ \eta \left\| \sum_{x \in X_0 \setminus X_0'} \mathrm{CL}\left(\frac{\partial \bar{\ell}(\theta, x)}{\partial \theta}\Big|_{\theta_0}, C \right) - \sum_{x' \in X_0' \setminus X_0} \mathrm{CL}\left(\frac{\partial \bar{\ell}(\theta, x')}{\partial \theta}\Big|_{\theta_0}, C \right) \right\| \\ \leq 2\eta \frac{C}{m}. \end{aligned} \tag{3.15}$$

For the next SGD update, we write

$$\theta_2 = \theta_1 - \eta \frac{1}{|X_1|} \sum_{x \in X_1} \mathrm{CL}\left(\frac{\partial \bar{\ell}(\theta, x)}{\partial \theta}\Big|_{\theta_1}, C \right) \tag{3.16}$$

and

$$\theta_2' = \theta_1' - \eta \frac{1}{|X_1'|} \sum_{x' \in X_1'} \mathrm{CL}\left(\frac{\partial \bar{\ell}(\theta, x')}{\partial \theta}\Big|_{\theta_1'}, C \right). \tag{3.17}$$

Due to the smoothness of $\bar{\ell}$, we have that

$$\left\| \mathrm{CL}\left(\frac{\partial \bar{\ell}(\theta, x)}{\partial \theta}\Big|_{\theta=a}, C \right) - \mathrm{CL}\left(\frac{\partial \bar{\ell}(\theta, x)}{\partial \theta}\Big|_{\theta=b}, C \right) \right\| \leq \min\left(2C, \frac{L}{\sigma_\mathrm{s}} \|a - b\| \right). \tag{3.18}$$

With $\beta = \frac{L}{\sigma_\mathrm{s}}$, we can write

$$\|\theta_2 - \theta_2'\| \leq \|\theta_1 - \theta_1'\| + \eta \left(1 - \frac{1}{m} \right) \|\theta_1 - \theta_1'\| \beta + 2\eta \frac{C}{m}. \tag{3.19}$$

The reason that (3.19) holds is that X_1 and X_1' are obtained from two adjacent datasets and because of the fixed-seed assumption, they hold equal entries except for one; for the equal entries, the second term on the right-hand side of (3.19) can be used and for the non-equal entry, the third term can be used as an upper bound. Analogously, for every stage $i \geq 2$, we have

$$\|\theta_i - \theta_i'\| \leq 2\eta\frac{C}{m} + \left[1 + \eta\left(1 - \frac{1}{m}\right)\beta\right]\|\theta_{i-1} - \theta_{i-1}'\|, \tag{3.20}$$

or

$$\|\theta_i - \theta_i'\| \leq 2\frac{(1+\eta\beta)^i - 1}{(m-1)\beta}C, \tag{3.21}$$

which concludes the proof. □

The established sensitivity bound established under the assumptions of Theorem 3.2 immediately implies the differential privacy of the GPM due to Theorem 3.1. However, the upper bound in (3.10) grows exponentially with the training horizon T. It is often the case that upper bounds for sensitivity are too loose and empirical measurements of the sensitivity take much smaller values. The SensitivitySampler algorithm (Rubinstein and Aldà, 2017) in combination with the notion of *random differential privacy* (Hall et al., 2013) addresses such an issue. The former is an algorithm that estimates sensitivity and the latter is a relaxation of (ϵ, δ)-differential privacy.

Definition 3.3. The mechanism M in Definition 3.1 satisfies (ϵ, δ)-random differential privacy with confidence $\gamma \in (0, 1)$ if, for all adjacent datasets D and D' drawn from a fixed data source DS,

$$\Pr_{D\sim\mathrm{DS}D'\sim\mathrm{DS}}\left[\forall R \subset \mathcal{R}, \Pr_{y\sim M(f(D))}[y \in R] \leq \exp(\epsilon)\cdot\Pr_{y'\sim M(f(D'))}[y' \in R] + \delta\right] \geq 1 - \gamma. \tag{3.22}$$

Compared to (ϵ, δ)-differential privacy wherein δ captures the probability of privacy failure due to unlikely *outputs*, random differential privacy considers γ as the

Algorithm 3: SensitivitySampler

Input: training algorithm $\mathcal{A}_{\mathcal{H}}$ with hyperparameters $\mathcal{H}$, sample size n, data source DS, training dataset size N,

Output: $\bar{S}$

for $i = 1$ *to* n **do**
  for $j = 1$ *to* $N + 1$ **do**
    $d_j \sim \text{DS}$; `/* Sample from the data source*/`
  $D_1 \leftarrow \cup_{j=1}^{N} d_j$;
  $D_2 \leftarrow \left(\cup_{j=1}^{N+1} d_j\right) \setminus \{d_N\}$;
  $\theta \leftarrow \mathcal{A}_{\mathcal{H}}(D_1)$; `/* Train Model 1*/`
  $\theta' \leftarrow \mathcal{A}_{\mathcal{H}}(D_2)$; `/* Train Model 2*/`
  $\bar{S}^{(i)} \leftarrow \|\theta - \theta'\|_2$;
$\bar{S} \leftarrow \max_i \bar{S}^{(i)}$;

probability that (ϵ, δ)-differential privacy fails due to unlikely *input* datasets (Rubinstein and Aldà, 2017).

We use the SensitivitySampler algorithm in the context of training a neural network for machine learning as described in Algorithm 3. The algorithm repeatedly samples two adjacent training datasets from a fixed data source, invokes the training algorithm for both of the sampled training datasets, and estimates the sensitivity of the training algorithm based on the maximum 2-norm difference between the observed network parameters. The following theorem, which is an immediate result of Corollary 20 of (Rubinstein and Aldà, 2017), establishes the random differential privacy of the GPM.

Theorem 3.3. *Fix a set of hyperparameters $\mathcal{H}$ and training algorithm $\mathcal{A}_{\mathcal{H}}$. Let $\bar{S}$ be the output of Algorithm 3 run with n samples. Further, let*

$$\rho = \exp\left(\frac{1}{2}W_{-1}\left(-\frac{1}{4n}\right)\right) \quad \textit{and} \quad \gamma = \rho + \sqrt{\frac{\log(1/\rho)}{2n}}, \tag{3.23}$$

where W_{-1} is the Lambert W function defined as the inverse relation of the function $f(z) = z\exp(z)$. With σ the variance of the Gaussian mechanism in Algorithm 2, for

all $\epsilon > 0$ *and*

$$\delta = \Phi\left(-\frac{\epsilon}{\mu} + \frac{\mu}{2}\right) - \exp(\epsilon)\Phi\left(-\frac{\epsilon}{\mu} - \frac{\mu}{2}\right), \tag{3.24}$$

where $\mu = \bar{S}/\sigma$*, Algorithm 2 satisfies* (ϵ, δ)*-random differential privacy with confidence* γ*.*

With the theoretical preliminaries set in this subsection, we now move on to the experiments.

3.5.2.2 Experiments

Similar to Section 3.4 in which we compared vulnerability to MIAs, we consider RNN and FFNN models in three representative machine learning tasks, namely image classification, machine translation, and deep reinforcement learning. However, for the machine translation task, we fine-tune a pre-trained model, BERT (Devlin et al., 2019), with a subset of training samples from the `WMT14` English-French training dataset (Bojar et al., 2014) instead of training a model from scratch using the `Multi30K` dataset. `WMT14` contains substantially more samples than `Multi30K` and is therefore more suitable for the SensitivitySampler algorithm.

In the first experiment, we use the DP-SGD algorithm to enforce differential privacy using a range of values for noise variance. Then, we measure the cost of privacy in terms of utility loss, which we formally define as follows:

Definition 3.4. Let $\mathbb{M}$ be an evaluation metric that takes as input a set of predictions Y alongside their ground-truth labels Y_{GT}, and returns a numerical value that indicates the quality of the predictions. Then, the utility loss is

$$\mathcal{L}_{util} = 1 - \frac{\mathbb{M}(\tilde{Y}, Y_{\text{GT}})}{\mathbb{M}(Y, Y_{\text{GT}})}. \tag{3.25}$$

We now report the results. The top row of Figure 3.8 indicates that the RNN models consistently trade-off more utility than the FFNN models at every noise variance tested. The same level of noise translates to the same level of (ϵ, δ)-differential

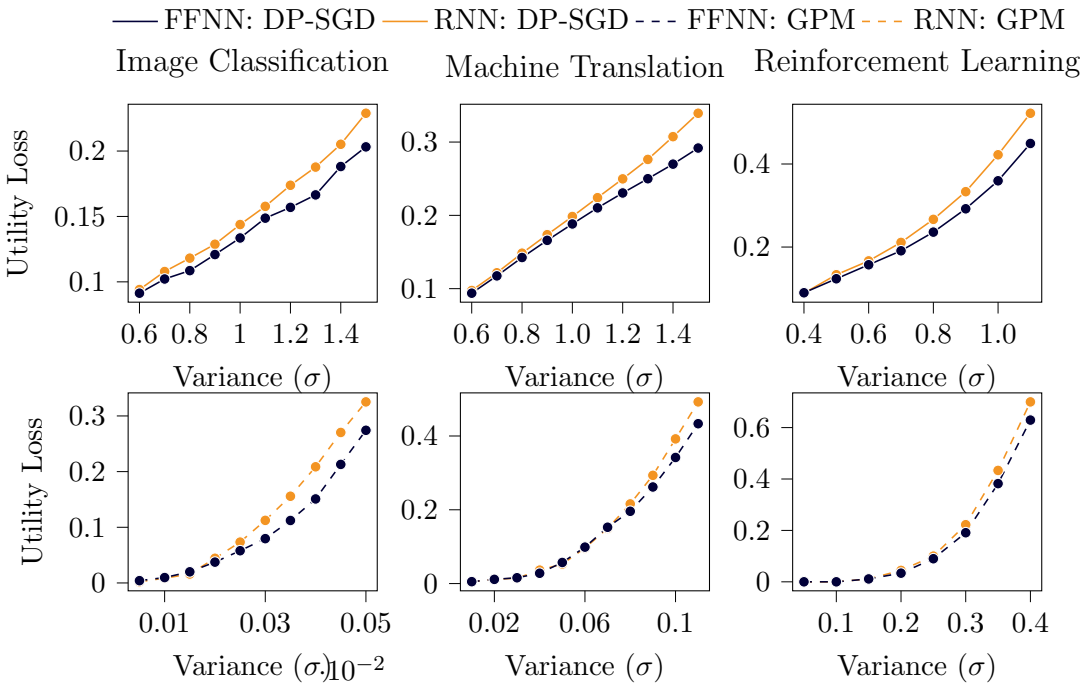

Figure 3.8: The privacy-utility trade-off of DP-SGD and GPM. The top row corresponds to the utility loss caused by DP-SGD and the second row corresponds to the utility loss caused by the GPM. The RNNs consistently trade off more utility than FFNNs for both DP-SGD and GPM.

privacy in DP-SGD; as a result, enforcing the same level of (ϵ, δ)-differential privacy is more costly in RNNs than FFNNs with respect to utility loss.

A similar observation can be made when the GPM enforces random differential privacy for the RNN and FFNN models. In this experiment, we fine-tune the hyperparameters of the training algorithms such that: (i) the two models achieve similar validation performance levels before the GPM is deployed and (ii) Algorithm 3 estimates the same level of sensitivity for the two models as reported in Table 3.1. We refer to these estimates as *empirical sensitivity*. The empirical sensitivities in Table 3.1 correspond to $n = 500$ samples which translates to confidence $\gamma < 0.08$ established by (3.23) in Theorem 3.3.

In Figure 3.8, where we plot utility loss vs. noise variance, it can be observed

Table 3.1: Empirical sensitivity estimated by Algorithm 3.

Task	$\bar{S}_{\text{RNN}}$	$\bar{S}_{\text{FFNN}}$
Image Classification	0.013209	0.013518
Machine Translation	0.11678	0.11845
Reinforcement Learning	0.093682	0.093429

that deploying the GPM consistently trades off more utility in RNNs than FFNNs. The results in Figure 3.8 also illustrate that the RNNs trade-off more utility for the same level of random differential privacy because the sensitivities of the two models are approximately equal.

3.6 Conclusion

In this chapter, we provided empirical evidence that MIAs can achieve higher accuracy when they attack RNNs compared with their FFNN counterparts. We showed that RNNs maintain a larger entropy gap between the predictions corresponding to member data and those corresponding to unseen data as a key vulnerability factor that is more elevated in RNNs than FFNNs. We also found that RNNs memorize their training data in a way that an MIA can maintain a non-trivial attack accuracy over the entire history of their training, whereas the corresponding attack accuracy for the FFNNs quickly drops to 50% as we move back in the training history.

In the second part of the study, we considered two prominent mitigation methods: weight regularization and differential privacy. Then, we showed that regularization was less effective in protecting RNNs compared to FFNNs. Moreover, we showed that enforcing differential privacy in RNNs can be more costly than FFNNs in terms of the privacy-utility trade-off.

We conclude this chapter with the observation that the privacy risks of deploying RNNs in machine learning are higher than FFNNs with the same level of performance. Alongside the existing computational drawbacks of training RNNs, our results provide further incentives to replace RNNs with FFNNs.

Chapter 4: Privacy Engineering Co-MARL Algorithms[1]

Abstract

In this chapter, we study the privacy engineering of value-based Co-MARL algorithms. In 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. In particular, the technical privacy policy that we consider in this chapter prohibits agents from sharing their environment interaction data. We demonstrate that state-of-the-art Co-MARL algorithms such as VDN, QMIX, and QTRAN rely on sharing environment interaction data. Consequently, we develop privacy-engineered counterparts for each of the three named algorithms.

4.1 Introduction

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 (Yang and Wang, 2020). Co-MARL algorithms can extend the capabilities of single-agent reinforcement learning algorithms to complete complex tasks that involve multiple

[1]The research presented in this chapter will appear in (Gohari et al., 2023). Parham Gohari formulated the problem, derived the theoretical results, designed and implemented the numerical experiments, and wrote the paper.

agents; for instance, in self-driving vehicles where reinforcement learning is a popular approach for autonomous driving (Aradi, 2022), Co-MARL may enable a fleet of autonomous vehicles to cooperate and reduce traffic congestion (Fax and Murray, 2004).

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 (Li et al., 2022).

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 (Sharma et al., 2021) are major examples in which a central optimizer 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 sacrificing multi-agent coordination?

We show that it is indeed possible to design such algorithms and propose three privacy-engineered Co-MARL algorithms based on three well-studied value-based algorithms for Co-MARL, namely VDN (Sunehag et al., 2018), QMIX (Rashid

et al., 2020), and QTRAN (Son et al., 2019). During training, the vanilla versions of these algorithms require a central optimizer that collects the environment interactions of the agents and consolidates the received data into a joint replay buffer. The joint replay buffer is then used to train a centralized action-value function approximator that estimates the long-term rewards associated with state observation and action pairs and supports the team's decision-making. These algorithms use distinct function approximator structures for the centralized action-value function such that it can be easily decomposed to a set of local action-value functions for each of the agents at test time. In all three algorithms, the actions that are optimal with respect to the local action-value functions amount to the optimal actions for the team; therefore, the agents do not need to share data during test time.

We incorporate three privacy-engineering techniques to modify the information flows of the three vanilla algorithms that we study and protect the confidentiality of the agents' privacy-sensitive environment interaction data. First, we design a distributed optimization for the training of the team's action-value function. In the vanilla algorithms, the central optimizer trains the entirety of the neural network that parameterizes the team's action-value function using the contents of the joint replay buffer. In contrast, we distribute the parameters of the team's action-value function between the agents. Then, we design a communication protocol that allows the agents to jointly train their share of the team's action-value neural network parameters without sharing environment interaction data and without bias compared to the vanilla algorithm.

In the second measure, we integrate secure multi-party computation techniques with the proposed distributed optimization schemes to ensure that the agents' communications do not undermine the confidentiality of their environment interaction data. In these communications, the agents often share the final or intermediate values of their local neural networks which may correlate with their underlying environment interaction data. However, the agents only require these values aggregated over the entire team, and secure multi-party computation techniques allow the agents

to compute these aggregate values while hiding their individual contributions from one another. We utilize additive secret sharing (Xiong et al., 2020) to obfuscate the individual contributions with correlated random numbers that act as encryption keys while allowing them to accurately compute the aggregated values.

In the third and last privacy measure, we use differential privacy to avoid unintended information disclosures about the agents' environment interaction data through inference attacks, such as those studied in Chapter 3. The agents choose their actions based on their internal neural network's predictions and unintentional information leaks can occur when neural networks release such predictions. For example, membership and attribute inference attacks discussed in the previous chapter, and inversion attacks are known to make accurate inferences about the training data of a neural network by mere input and output observations (Zhang et al., 2023). Since the agents' local neural networks are trained with the contents of their replay buffers, we use differential privacy to protect the confidentiality of the agents' past environment interactions. In particular, we use the DP-SGD (Abadi et al., 2016a) algorithm to achieve differential privacy. When the agents train their internal neural networks with DP-SGD, the plausible deniability guarantees provided by differential privacy can reduce the risks of indirect information leakage about the agents' past environment interaction data.

The DP-SGD algorithm can be readily integrated with our last two privacy-engineering techniques; however, its differential privacy analysis does not 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 (Abadi et al., 2016b) and Maximum-Overlap Parallel Composition (Smith et al., 2022) to compute the differential privacy level of DP-SGD when applied to deep reinforcement learning.

We implement a Python library for all three privacy-engineered algorithms and test it in the StarCraft Multi-Agent Challenge (SMAC) suite (Samvelyan et al., 2019). In the numerical results, we dissect the different privacy-engineering components and study the trade-offs between privacy, precision, and performance.

4.2 Preliminaries

In this section, we cover some technical background on Co-MARL's problem formulation and review Vanilla VDN. Then, we describe our privacy objectives for Co-MARL in the format of a technical privacy policy.

4.2.1 Dec-POMDP

Co-MARL algorithms typically model multi-agent cooperation using decentralized partially observable Markov decision processes (Dec-POMDPs) (Guicheng and Yang, 2022). A Dec-POMDP $\mathcal{G}$ is a tuple $\mathcal{G} = (\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{Z}, \mathcal{P}, \mathcal{R}, \gamma)$ 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 $\boldsymbol{\tau}_t = ((\boldsymbol{o}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{o}_{t-1}, \boldsymbol{a}_{t-1}), \boldsymbol{o}_t)$ 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.,

$$V^{\boldsymbol{\pi}}(\boldsymbol{s}) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \mid \boldsymbol{s}_1 = \boldsymbol{s}\right], \tag{4.1}$$

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

$$Q^{\boldsymbol{\pi}}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}\left[\mathcal{R}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V^{\boldsymbol{\pi}}(\boldsymbol{s}') \mid \boldsymbol{s}' \sim \mathcal{P}(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})\right]. \tag{4.2}$$

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.

4.2.2 Deep Q-Learning

Deep Q-Learning (DQN) (Mnih et al., 2015) 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 $(s_t, a_t, s_{t+1}, \mathcal{R}(s_t, a_t, s_{t+1}))$, or in short (s, a, s', r). 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 (Dao and Lee, 2019).

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

$$\ell(E; \theta) = \sum_{(s,a,s',r) \in E} \left(r + \gamma \max_{a'} Q^{\pi}\left(s', a'; \theta^{-}\right) - Q^{\pi}(s, a; \theta)\right)^2, \tag{4.3}$$

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

4.2.3 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 $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}, r)$ is used instead of (s, a, s', r). We also replace all state values s with histories τ to support partial observability. The resulting loss function is

$$\ell(E;\theta) = \sum_{(\boldsymbol{\tau},\boldsymbol{a},\boldsymbol{\tau'},r)\in E} \left(r + \gamma \max_{\boldsymbol{a'}} Q^{\boldsymbol{\pi}} \left(\boldsymbol{\tau'}, \boldsymbol{a'}; \theta^{-} \right) - Q^{\pi}(\boldsymbol{\tau}, \boldsymbol{a}; \theta) \right)^2 . \tag{4.4}$$

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 (Yang and Wang, 2020).

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 4.1. 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 $i \in \mathcal{N}$,

$$\left(\arg\max_{\boldsymbol{a}} Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{a})\right)_i = \arg\max_{a_i} Q_i(\tau_i, a_i). \tag{4.5}$$

All three algorithms considered in this study for privacy engineering are instances of CTDE, and in the next section, we develop their privacy-engineered counterparts.

4.3 Engineering VDN, QMIX, and QTRAN for Privacy

In this section, we introduce the privacy-engineered versions of the VDN, QMIX, and QTRAN algorithms. For each algorithm, we first review the vanilla version and then, we develop the privacy-engineered versions.

4.3.1 VDN

In this section, we first review the Vanilla VDN algorithm and explain its information flows. Then, we propose the privacy-engineered version, which we call PE-VDN.

4.3.1.1 Vanilla VDN

The Vanilla VDN algorithm achieves decentralizability by approximating the team's central Q-function with the summation of a set of Q_i-functions, each of which

is dedicated to a specific agent. That is, the algorithm's function approximator for the team's Q-function is structured as follows

$$Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{a}) \approx Q_{\text{tot}}\left(\boldsymbol{\tau}, \boldsymbol{a}; [\theta_{Q_i}]_{i\in\mathcal{N}}\right) = \sum_{i\in\mathcal{N}} Q_i\left(\tau_i, a_i; \theta_{Q_i}\right). \tag{4.6}$$

This additivity structure implies decentralizability in (4.5) because actions that are optimal with respect to each local Q_i-function are optimal for Q_{tot} as well; however, the team's optimal Q-function in decentralizable tasks may not always be decomposable as (4.6).

In Vanilla VDN, each Q_i-function is approximated by a neural network whose parameters are denoted θ_{Q_i}. However, these neural networks are not accessed individually during training. Instead, the centralized training algorithm handles the Q_{tot}-function which is simply the summation of the Q_i-functions, each of which is parameterized by its dedicated parameters, θ_{Q_i}. In the sequal, we often refer to θ_{Q_i} as the i^{th} branch of Q_{tot}.

In the centralized training phase of Vanilla VDN, the agents must send their environment interaction data to the central optimizer to create a consolidated replay buffer and train the team's Q-function, Q_{tot}. The optimizer first draws a minibatch of the consolidated replay buffer, denoted E. Let, $e = (\boldsymbol{\tau}, \boldsymbol{a}, \boldsymbol{\tau'}, r) \in E$ 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. Vanilla VDN's loss function is as follows:

$$\begin{aligned}\ell_{\text{VDN}}\left(e; [\theta_{Q_i}]_{i\in\mathcal{N}}\right) &= \left(r + \gamma \max_{\boldsymbol{a'}} Q_{\text{tot}}\left(\boldsymbol{\tau'}, \boldsymbol{a'}\right) - Q_{\text{tot}}\left(\boldsymbol{\tau}, \boldsymbol{a}\right)\right)^2 \\ &= \left(r + \sum_{i\in\mathcal{N}} \left(\gamma \max_{a'} Q_i\left(\tau'_i, a'; \theta_i^-\right) - Q_i\left(\tau_i, a_i; \theta_{Q_i}\right)\right)\right)^2.\end{aligned} \tag{4.7}$$

Figure 4.1 depicts the computational graph of Vanilla VDN's forward pass with two agents.

The trainable parameters in Vanilla VDN are the parameters of the local Q_i-functions. Once the computational graph for the forward pass of the loss function's

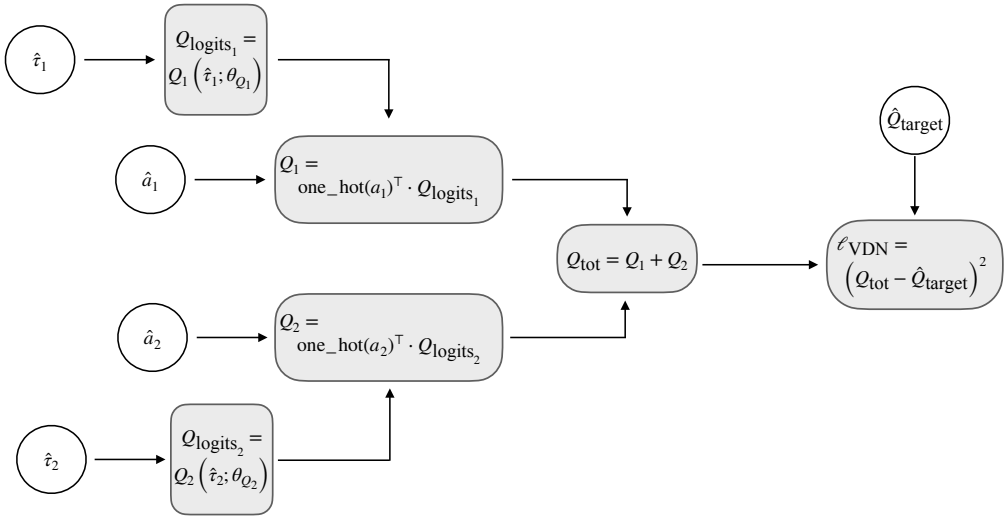

Figure 4.1: The computational graph of the forward pass in Vanilla VDN with two agents. Parameters annotated with hats do not carry gradient attributes. Q_{logits} denotes a vector that contains the action values for all available actions. one_hot$(\cdot)$ converts an integer index to a one-hot indicator vector. The computational graph of $Q_{\text{target}} = r + \gamma \sum_{i \in \mathcal{N}} \max_a Q_i\left(\tau_i', a; \theta_{Q_i}^-\right)$ is omitted because its gradients are detached.

computation is formed, auto-grad libraries such as PyTorch can compute the gradient of the trainable parameters. In the earlier example of two agents, the backward pass is depicted in Figure 4.2. Once centralized training concludes every agent uses its corresponding $Q_i\left(\cdot, \cdot; \theta_{Q_i}\right)$ function for decentralized execution.

4.3.1.2 Privacy-Engineered VDN (PE-VDN)

In Vanilla VDN, the central optimizer computes the loss function's gradients with respect to the parameters of all branches within Q_{tot}. Recall that these branches are dedicated to specific agents for decentralized execution. Evaluating the gradient of the loss function in (4.7), for every branch θ_{Q_i} and minibatch sample e, we can

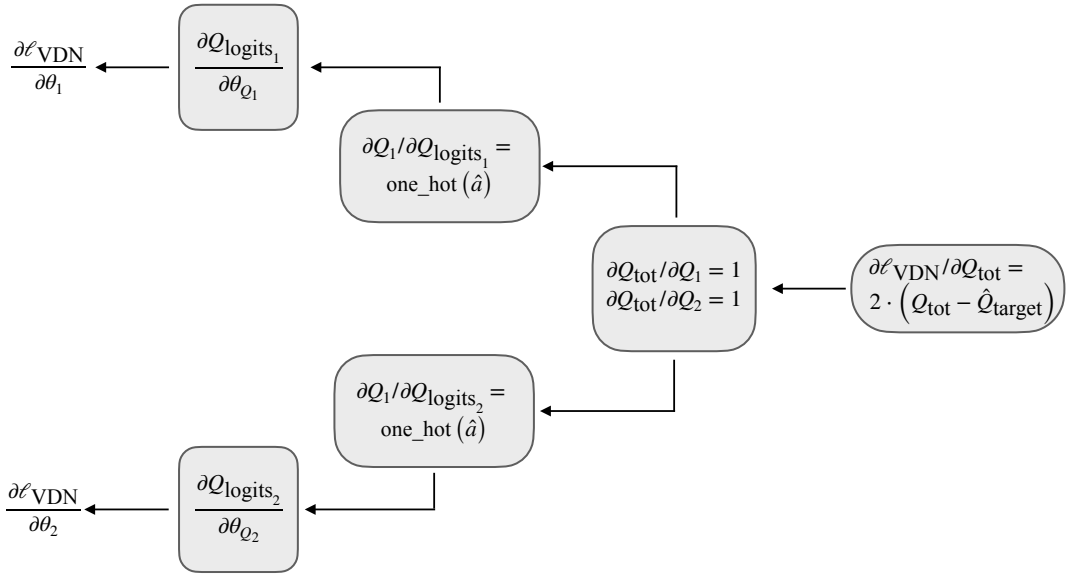

Figure 4.2: The computational graph of the backward pass in Vanilla VDN with two agents.

write

$$\frac{\partial \ell_{\text{VDN}}(e; [\theta_{Q_i}]_{i \in \mathcal{N}})}{\partial \theta_{Q_i}} = \underbrace{-2 \left(r + \sum_{i \in \mathcal{N}} \left(\gamma \max_{a'} Q_i \left(\tau_i', a'; \theta_i^- \right) - Q_i \left(\tau_i, a_i; \theta_{Q_i} \right) \right) \right)}_{A} \cdot \underbrace{\frac{\partial Q_i(\tau_i, a_i; \theta_{Q_i})}{\partial \theta_{Q_i}}}_{B}. \quad (4.8)$$

From the gradient expression and its separation into terms A and B, it can be observed that the A term is the only factor that couples the gradients of the different branches within Q_{tot} and is a function of all of the agents' environment interaction data. The B term, however, is the gradient of the i^{th} branch, and computing it only requires the parameters of that branch and its corresponding agent's environment interaction data. Therefore, if the agents somehow cooperate and jointly compute the coupling term, they may be able to compute the team's loss function as well as the gradients of their dedicated branches on their own and without sharing environment

interaction data with a central optimizer.

We grant each agent the ownership of its dedicated branch and design a communication protocol that allows every agent to create a local copy of the computational graph of ℓ_{VDN} in both forward and backward passes. The communications must ensure that the agent's local forward and backward passes yield the same loss value and gradients vis-à-vis Vanilla VDN. To achieve this, we require every agent to broadcast the following message at every iteration of the training:

$$m_i = \left[Q_i\left(\tau_i, a_i; \theta_{Q_i}\right), \max_a Q_i\left(\tau_i', a; \theta_i^-\right) \right]. \tag{4.9}$$

These messages can be computed using the corresponding agent's environment interaction data, and its local Q_i-function's current and target parameters. By computing the summation of these messages, every agent will be able to create an equivalent of Vanilla VDN's computational graph in both forward and backward directions. In particular, the forward pass involves plugging the summation of the messages in (4.7), and the backward pass involves plugging the summations in (4.8). Figures 4.3 and 4.4 depict the forward and backward passes of the agents' local computational graphs, respectively.

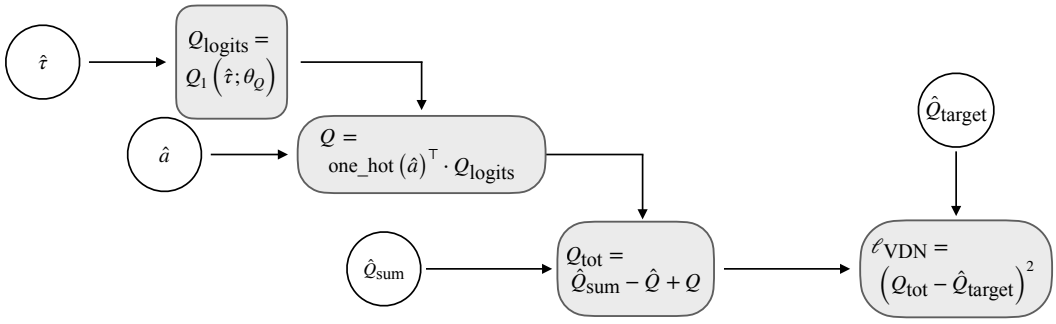

Figure 4.3: The forward pass of the agents in PE-VDN. $Q_{\text{sum}} = \sum_{i \in \mathcal{N}} Q_i\left(\tau_i, a_i; \theta_{Q_i}\right)$ and $Q_{\text{target}} = r + \gamma \sum_{i_\in \mathcal{N}} \max_a Q_i\left(\tau_i', a; \theta_{Q_i}^-\right)$, which use the first and the second part of the m_i-messages, respectively.

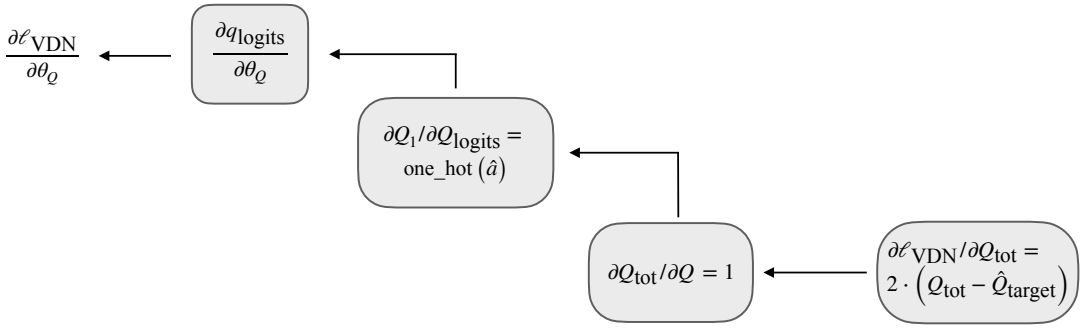

Figure 4.4: The backward pass of the agents in PE-VDN. Each agent's backward pass in PE-VDN is equivalent to the backward pass of its corresponding branch in Vanilla VDN.

The proposed distributed optimization scheme eliminates the need for sharing environment interaction data with a central node. However, there is still an opportunity for further enhancing the privacy guarantees pertaining to the confidentiality of the agents' environment interaction data. Although the distributed optimization scheme can be easily implemented by requiring every agent to broadcast their m_i messages, doing so may lead to unintentional information leakage; the m_i messages are functions of their corresponding agent's privacy-sensitive environment interaction data and correlate with them. In the sequel, we demonstrate how the agents can compute the coupling terms while hiding their m_i values from one another.

We use a secure multi-party computation technique called secret sharing to compute the summation in 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. Specifically, a (k, n)-secret sharing of s guarantees that observers with access to up to $k - 1$ shares can learn no information about s (Shamir, 1979).

We use additive secret sharing which is an efficient (n, n)-secret sharing scheme

(Xiong et al., 2020). In particular, let s be the secret value over a finite filed $\mathbb{Z}_p := \{0, 1, \ldots, p-1\}$. Then, additive secret sharing splits the secret $s \in \mathbb{Z}_p$ into n shares $S(s) = (r_1, r_2, \ldots, r_n)$ such that $r_1, \ldots, r_{n-1}$ are chosen uniformly at random from $\mathbb{Z}_p$, and $r_n = p - \left(\sum_{i=1}^{n-1} r_i \mod p\right) + s$. Additive secret sharing ensures that a complete set of n shares accurately reconstruct the secret via $S^{-1}(r_1, \ldots, r_n) := (\sum_{i=1}^{n} r_i) \mod p$. Otherwise, for any given set of shares with less than n shares, every element of $\mathbb{Z}_p$ is equally likely to generate those shares. Additive secret sharing is *additively homomorphic*, i.e., for secrets $s_1, \ldots, s_m$, all in $\mathbb{Z}_p$, we have that

$$\sum_{i=1}^{m} s_i = S^{-1}\left(\sum_{i=1}^{m} S(s_i)\right), \tag{4.10}$$

where the right-hand side of (4.10) is the element-wise summation of the secret shares.

The additive homomorphism of additive secret sharing ensures the accuracy of the following three-step secure n-party summation protocol: first, each party invokes additive secret sharing on its secret value and sends each of the other agents a piece of the shares. Then, the agents locally compute the sum of the shares that they receive from other agents and broadcast the results. Finally, the agents recover the summation of their secret values by computing the sum of the broadcast values.

Returning to the problem of computing the A term in the gradient expression in (4.8), the secure summation protocol based on additive secret sharing can be deployed with one intermediate step. Additive secret sharing applies to finite-field integers and we need to encode the m_i values in that format. We use the encoding

$$\text{enc}(x) = \text{int}\left(10^{\texttt{PRECISION}} \cdot x\right) \mod p, \tag{4.11}$$

in which `PRECISION` denotes the number of decimal places that the encoding preserves. To reverse the encoding back to floating point, we use the decoding function

$$\text{dec}(x) = \begin{cases} x/10^{\texttt{PRECISION}} & \text{if } x \leq p/2, \\ (x-p)/10^{\texttt{PRECISION}} & \text{if } x > p/2. \end{cases} \tag{4.12}$$

To summarize, in order to compute the summation in the A term, the agents send the secret shares $S(\text{enc}(m_i))$ to one another via peer-to-peer messaging. Then, each agent computes the summation of its received shares and broadcasts the result. Finally, each agent reconstructs a copy of the A term with the summation of the broadcast values. Integrating additive secret sharing with the distributed computation of the gradients ensures that inter-agent communications in the decentralized training of PE-VDN do not undermine the confidentiality of the agents' environment interaction data.

4.3.2 QMIX

We now move on to the privacy engineering of the second Co-MARL algorithm. First, we review Vanilla QMIX and describe its information flows. Then, we introduce its privacy-engineered version, PE-QMIX.

4.3.2.1 Vanilla QMIX

Vanilla QMIX achieves decentralizability through *monotonicity*. That is, the algorithm approximates the team's Q-function as

$$Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{a}) \approx Q_{\text{tot}}\left(\boldsymbol{\tau}, \boldsymbol{a}; [\theta_{Q_i}]_{i\in\mathcal{N}}\right) = f\left([Q_i(\tau_i, a_i; \theta_{Q_i})]_{i\in\mathcal{N}}\right), \tag{4.13}$$

where f is a mixing function that satisfies the following: for all joint histories $\boldsymbol{\tau}$ and joint actions $\boldsymbol{a}$,

$$\frac{\partial f}{\partial Q_i(\tau_i, a_i; \theta_{Q_i})} \geq 0, \quad \forall i \in \mathcal{N}. \tag{4.14}$$

The additivity assumption in Vanilla VDN is a special case of monotonicity. However, similar to additivity, not all decentralizable tasks satisfy monotonicity.

Vanilla QMIX parameterizes the mixing function with so-called *hypernetworks*. These hypernetworks are neural networks that map the global state of the environment to a set of weights and biases for the mixing function. In the Vanilla QMIX algorithm, it is assumed that the agents have access to the global state of the environment during training, possibly through a simulator. However, the Q_i-functions

still handle partial observations because it is assumed that the agents lose access to the global state at test time.

Let $z_f^{[k]}$, $g^{[k]}$, and $a_k^{[k]}$ denote the pre-activation values, activation function, and the value of the mixing network's k^{th} layer, respectively. Furthermore, let $W_f^{[k]}$ be the weights and $b_f^{[k]}$ be the biases of that layer. Then, for $k = 1$, we write

$$\left(z_f^{[1]}\right)_j = \sum_{i \in \mathcal{N}} Q_i(\tau_i, a_i) \cdot \left(W_f^{[1]}\right)_{i,j} + \left(b_f^{[1]}\right)_j \quad \text{and} \quad a^{[1]} = g^{[1]}\left(z^{[1]}\right). \tag{4.15}$$

Subsequently for $k > 1$, we write

$$\left(z_f^{[k]}\right)_j = \sum_{i=1}^{n_{k-1}} \left(a^{[k-1]}\right)_i \cdot \left(W_f^{[k]}\right)_{i,j} + \left(b_f^{[k]}\right)_j \quad \text{and} \quad a^{[k]} = g^{[k]}\left(z^{[k]}\right), \tag{4.16}$$

where n_{k-1} is the number of nodes in layer $k-1$. In particular, for the last layer K, we have

$$Q_{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{a}) = \sum_{i=1}^{n_{K-1}} \left(a^{[K-1]}\right)_i \cdot \left(W_f^{[K]}\right)_{i,1} + \left(b_f^{[K]}\right)_1. \tag{4.17}$$

We denote the hypernetwork that generates the mixing network's weights in layer k by $h_W^{[k]}$, and $h_b^{[k]}$ denotes the hypernetwork that generates the biases in that layer. In order to satisfy monotonicity, Vanilla QMIX uses the absolute value of the hypernetwork outputs that generate the weights of the mixing function f. That is, for all node indices $j = 1, \ldots n_k$ and $i = 1, \ldots, n_{k-1}$, we have

$$\left(W_f^{[k]}\right)_{i,j} = \left|\left(h_W^{[k]}(\boldsymbol{s})\right)_{i,j}\right| \quad \text{and} \quad \left(b_f^{[k]}\right)_j = \left(h_b^{[k]}(\boldsymbol{s})\right)_j, \tag{4.18}$$

With the established notation, we are now ready to review Vanilla QMIX's loss function. The loss function includes the parameters of the Q_i-functions, $[\theta_{Q_i}]_{i \in \mathcal{N}}$, as well as the hypernetworks' weights and biases as trainable parameters. Specifically, the loss function is

$$\begin{aligned}
&\ell_{\text{QMIX}}\left(e; [\theta_{Q_i}]_{i\in\mathcal{N}}, \left[\theta_{h_W}^{[k]}\right]_{k\in[K]}, \left[\theta_{h_b}^{[k]}\right]_{k\in[K]}\right) \\
&\qquad = \left(r + \gamma \max_{\boldsymbol{a}'} Q_{\text{tot}}\left(\boldsymbol{\tau}', \boldsymbol{a}'\right) - Q_{\text{tot}}\left(\boldsymbol{\tau}, \boldsymbol{a}\right)\right)^2 \\
&\qquad = \left(r + \gamma f\left(\left[\max_{a'} Q_i\left(\tau_i', a'; \theta_i^-\right)\right]_{i\in\mathcal{N}}\right) - f\left(\left[Q_i(\tau_i, a_i; \theta_{Q_i})\right]_{i\in\mathcal{N}}\right)\right)^2.
\end{aligned} \tag{4.19}$$

Figure 4.5 summarizes the computational graph of Vanilla QMIX's forward pass of the loss function in a two-agent example. Once training concludes, the mixing network and the underlying hypernetworks are discarded, and the Q_i-functions support the decision-making of the agents at test time.

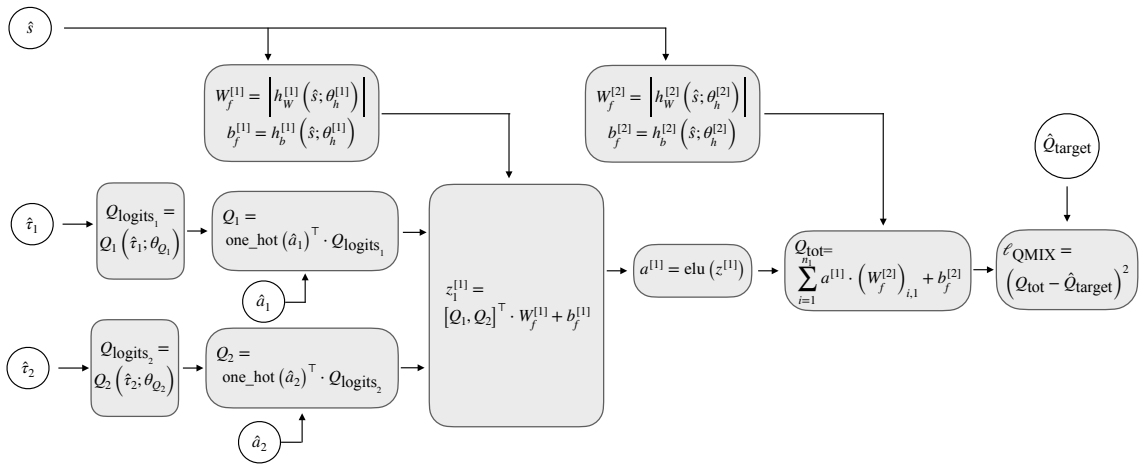

Figure 4.5: A two-agent example of the computational graph of Vanilla QMIX's forward pass of the loss function. In this example, the mixing function consists of two layers, each of which has its upstream hypernetworks with parameters $\theta_h^{[1]}$ and $\theta_h^{[2]}$. The first layer of the mixing function has the exponential linear unit (elu) as its activation function. The computational path to $Q_{\text{target}} = r + \max_{\boldsymbol{a}'} Q_{\text{tot}}(\boldsymbol{\tau}', \boldsymbol{a}')$ is similar to that of Q_{tot} but not depicted due to its lack of contribution to the gradients.

4.3.2.2 Privacy-Engineered QMIX (PE-QMIX)

The central optimizer in the Vanilla QMIX algorithm consolidates the agents' replay buffers to train each of the agent's Q_i-functions and the hypernetworks. In this section, we develop an alternative distributed optimization framework for the agents to train these neural networks without relying on a central optimizer.

In Vanilla QMIX, the hypernetworks process the global state of the environment. We avoid using the global state as the input of the hypernetworks because the global state characterizes all of the agents' environment surroundings; assuming access to such information makes the technical privacy policy irrelevant. Instead,

we equip each agent with an individual encoder that processes partial environment observations for the hypernetworks. Figure 4.6 illustrates how we incorporate these encoders to avoid using state information in Vanilla QMIX.

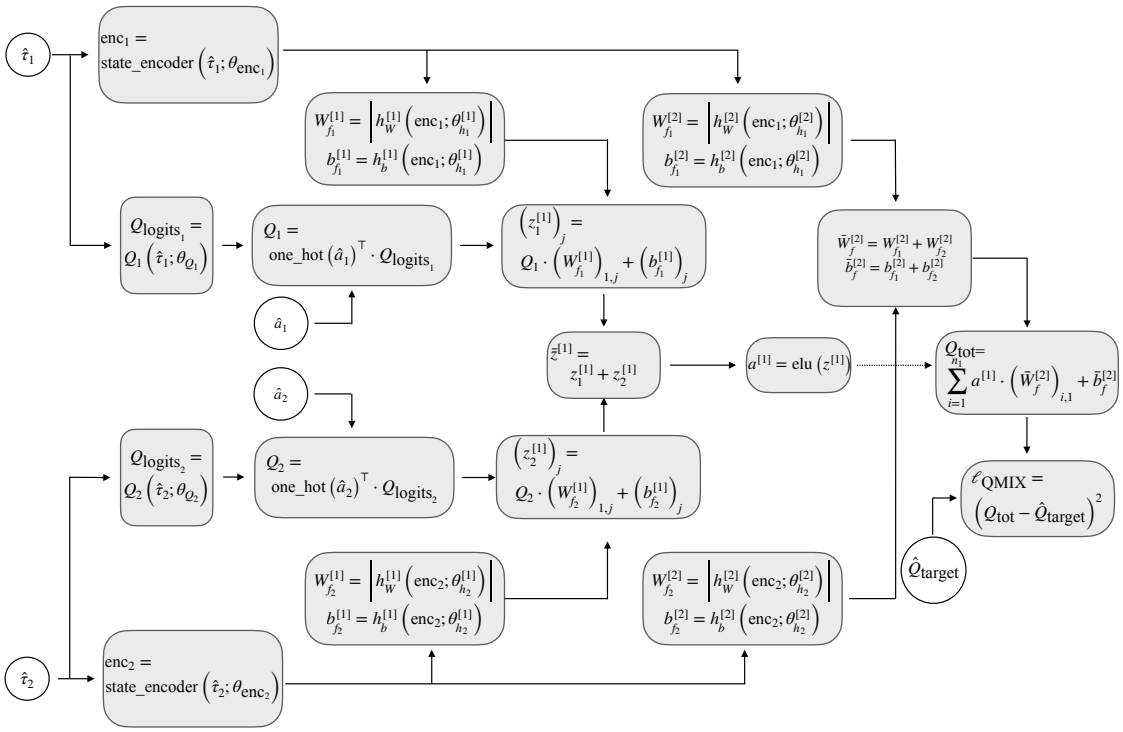

Figure 4.6: Incorporating encoders to process partial observations for the input of hypernetworks instead of processing global state information.

Similar to the approach that we adopted for the distributed optimization scheme in PE-VDN, we grant the agents the ownership of the parameters of their Q_i-functions. However, in the Vanilla QMIX, these are not the only trainbale parameters of the team's action-value function Q_{tot}. Distributing the trainable parameters of the mixing function adds an additional layer of complexity compared to PE-VDN. We address this issue by having each agent maintain a local share of the mixing network. These local shares have a common architecture across the agents—including the number of layers, the number of nodes within each layer, and the connection graph between the nodes; however, the values of the weights and biases may be different.

Similar to Vanilla QMIX, the weights and the biases of each agent's share of the mixing network are calculated by hypernetworks. However, we avoid using the global state as the input of the hypernetworks because the global state characterizes all of the agents' environment surroundings; assuming access to such information makes the technical privacy policy irrelevant. Instead, we equip each agent with an individual encoder that processes partial environment observations for the hypernetworks.

During training, for each layer in the mixing network, the distributed optimization framework dedicates a separate round of peer-to-peer messaging for the agents to aggregate and reconcile their shares of the mixing network. Specifically, let $W_{f_i}^{[k]}$ and $b_{f_i}^{[k]}$ be the weights and biases of the k^{th} layer of agent i's share of the mixing network. Note that these weights and biases are functions of their corresponding agent's hypernetwork parameters, encoder parameters, and partial observations. In the first round of peer-to-peer messaging, the agents jointly compute the following summation:

$$\left(z^{[1]}\right)_j = \sum_{i \in \mathcal{N}} \left(Q_i(\tau_i, a_i) \cdot \left(W_{f_i}^{[1]}\right)_{i,j} + \left(b_{f_i}^{[1]}\right)_j \right). \tag{4.20}$$

Subsequent to the first round, the agents advance in their mixing networks one layer at a time and jointly compute the summations

$$\left(\bar{W}_f^{[k]}, \bar{b}_f^{[k]}\right) = \left(\sum_{i \in \mathcal{N}} W_{f_i}^{[k]}, \sum_{i \in \mathcal{N}} b_{f_i}^{[k]}\right), \qquad k = 2, \ldots, K, \tag{4.21}$$

where K is the mixing network's last layer. Since the summation of the non-negative weights that every agent's weight hypernetworks produce remains non-negative, the aggregate mixing network still satisfies monotonicity. These communication rounds allow the agents to reconcile their shares of the mixing network and arrive at a common Q_{tot}. Specifically, the agents start with the jointly computed $z^{[1]}$ and sequentially use $\left(\bar{W}_f^{[k]}, \bar{b}_f^{[k]}\right)$ to compute Q_{tot} as follows:

$$\left(z^{[k]}\right)_j = \sum_{i=1}^{n_{k-1}} (g^{[k-1]}\left(\left(z^{[k-1]}\right)_i\right) \cdot \left(\bar{W}_f^{[k]}\right)_{i,j} + \left(\bar{b}_f^{[k]}\right)_j \quad k = 2, \ldots, K-1, \tag{4.22}$$

and

$$Q_{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{a}) = z^{[K]} = \sum_{i=1}^{n_{K-1}} g^{[K-1]}\left(\left(z^{[K-1]}\right)_i\right) \cdot \left(\bar{W}_f^{[K]}\right)_{i,1} + \bar{b}_f^{[K]}. \tag{4.23}$$

After the agents jointly compute a forward pass of the mixing function, every agent will be able to compute the loss function in (4.19), except that this loss functions only uses the parameters of the corresponding agent's Q_i-function and hypernetworks as the trainable parameters. This forward pass of the loss function is depicted in Figure 4.7.

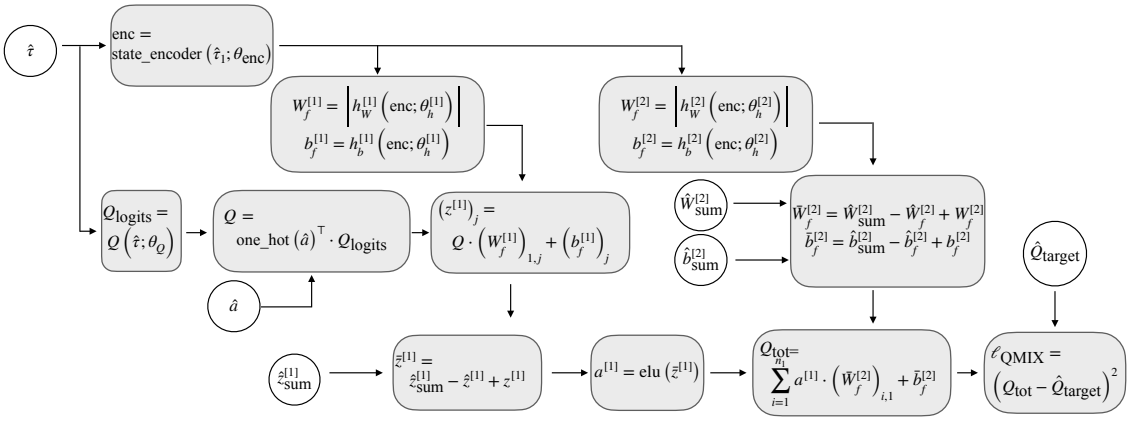

Figure 4.7: An example of the computational graph that each agent creates for the forward pass of the loss function in PE-QMIX. In this example, the mixing function comprises two layers and the first layer is activated by the elu function.

In the proposed peer-to-peer messaging scheme, even though the agents compute the loss with incomplete computational graphs compared with Vanilla QMIX, they can still compute the gradients of their dedicated trainable parameters without any bias. However, arbitrary distributions of the mixing network shares do not always lead to unbiased gradients. For example, if the agents estimate the global state of the environment with the summation of their state encoders and follow the rest of the communication protocol as described above, the gradients of the values that represent the summation of the encoder outputs will be biased. In this example, the

bias may be corrected with the summation of the gradients at that layer across all agents.

We briefly note that all of the peer-to-peer communication rounds involved in the proposed distributed optimization scheme compute summations. Hence, similar to PE-VDN, these summations can be seamlessly integrated with additive secret sharing. By using additive secret sharing, the agents can compute a forward pass of their loss function without revealing their individual contributions to the intermediate values that they jointly compute according to (4.20) and (4.21). Off-the-shelf secure multi-party computation libraries such as MP-SPDZ (Keller, 2020) can compute the forward pass of the loss function while hiding even intermediate values; nonetheless, we deliberately compute these intermediate values and reveal them to the agents so they can maintain an accurate computation graph for the backward pass.

4.3.3 QTRAN

In this section, we study the privacy engineering of the last Co-MARL algorithm, QTRAN. We first review the vanilla version and describe its information flows. Then, we introduce the privacy-engineered version.

4.3.3.1 Vanilla QTRAN

As opposed to Vanilla VDN and QMIX, Vanilla QTRAN does not rely on any additional assumption such as additivity or monotonicity to achieve decentralizability. Instead, the QTRAN algorithm offers a decomposition method that is free from structural assumptions on the team's action-value function. In particular, the QTRAN algorithm aims to satisfy a so-called *factorization* condition that verifies whether or not a candidate team action-value function Q_{tot} is correctly decomposed by a set of action-value functions $[Q_i]_{i\in\mathcal{N}}$.

Theorem 4.1 (Theorem 1 from (Son et al., 2019))**.** *For a given team action-value function Q_{tot} and given $[Q_i]_{i\in\mathcal{N}}$, let $\bar{\boldsymbol{a}} = [\arg\max_a Q_i(\tau_i, a)]_{i\in\mathcal{N}}$. Then, Q_{tot} and*

$(Q_i)_{i\in\mathcal{N}}$ *satisfy decentralizability if* Q_{tot} *is factorized by* $[Q_i]_{i\in\mathcal{N}}$*, i.e., with* $V(\boldsymbol{\tau}) = \max_{\boldsymbol{a}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) - \sum_{i\in\mathcal{N}} Q_i(\tau_i, \bar{a}_i)$*,*

$$\sum_{i\in\mathcal{N}} Q_i(\tau_i, a_i) - Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) + V(\boldsymbol{\tau}) = \begin{cases} 0, & \boldsymbol{a} = \bar{\boldsymbol{a}}, \\ \geq 0, & \boldsymbol{a} \neq \bar{\boldsymbol{a}}. \end{cases} \tag{4.24}$$

Vanilla QTRAN does not rely on any structural assumptions for Q_{tot} such as additivity and monotonicity. That being said, Son et al. (2019) use a particular structure for the Q_{tot} in their implementations. In their approach, each of the Q_i-functions comprise a state-action encoder, denoted $h_{Q_i}(\tau_i, a_i)$, and a state encoder that is denoted $h_{V_i}(\tau_i)$. The summation of these encoder values—$h_Q(\boldsymbol{\tau}, \boldsymbol{a}) = \sum_{i\in\mathcal{N}} h_{Q_i}(\tau_i, a_i)$ and $h_V(\boldsymbol{\tau}) = \sum_{i\in\mathcal{N}} h_{V_i}(\tau_i, a_i)$—are then fed to two separate neural networks to obtain $Q_{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{a})$ and $V(\boldsymbol{\tau})$. We denote the parameters of Q_{tot} by ρ and the parameters of V by υ.

Vanilla QTRAN optimizes a linear combination of three loss functions. The first loss function, ℓ_{td}, aims to minimize the Bellman error and is similar to the loss functions in Vanilla VDN and QMIX. Specifically, this loss function is

$$\begin{aligned} &\ell_{\text{td}}\left(e; [\theta_{Q_i}]_{i\in\mathcal{N}}, \rho, \upsilon\right) = \\ &\qquad \left(r + \gamma Q_{\text{tot}}\left(h_Q(\boldsymbol{\tau'}, \bar{\boldsymbol{a}}'; [\theta_i^-]_{i\in\mathcal{N}}); \rho^-\right) - Q_{\text{tot}}\left(h_Q(\boldsymbol{\tau}, \boldsymbol{a}; [\theta_{Q_i}]_{i\in\mathcal{N}}); \rho\right)\right)^2, \end{aligned} \tag{4.25}$$

where $\bar{\boldsymbol{a}}' = \left[\arg\max_a Q_i\left(\tau_i', a; \theta_{Q_i}^-\right)\right]_{i\in\mathcal{N}}$. The next two loss functions aim to satisfy the factorization condition. These two loss functions are:

$$\begin{aligned} &\ell_{\text{opt}}\left(e; [\theta_{Q_i}]_{i\in\mathcal{N}}, \rho, \upsilon\right) = \\ &\left(\sum_{i\in\mathcal{N}} Q_i\left(\tau_i, \bar{a}_i; [\theta_{Q_i}]_{i\in\mathcal{N}}\right) - \hat{Q}_{\text{tot}}\left(h_Q(\boldsymbol{\tau}, \bar{\boldsymbol{a}}; [\theta_{Q_i}]_{i\in\mathcal{N}}); \rho\right) + V\left(h_V(\boldsymbol{\tau}; [\theta_{Q_i}]_{i\in\mathcal{N}}); \upsilon\right)\right)^2, \end{aligned} \tag{4.26}$$

and

$$\ell_{\text{nopt}}\left(e;[\theta_{Q_i}]_{i\in\mathcal{N}},\rho,\upsilon\right)=$$
$$\left(\left\{\sum_{i\in\mathcal{N}}Q_i\left(\tau_i,a_i;[\theta_{Q_i}]_{i\in\mathcal{N}}\right)-\hat{Q}_{\text{tot}}\left(h_Q(\boldsymbol{\tau},\boldsymbol{a};[\theta_{Q_i}]_{i\in\mathcal{N}});\rho\right)+V\left(h_V(\boldsymbol{\tau};[\theta_{Q_i}]_{i\in\mathcal{N}});\upsilon\right)\right\}_+\right)^2, \tag{4.27}$$

where $\{z\}_+ := \min(z,0)$. In ℓ_{opt} and ℓ_{nopt} expressions, $\hat{Q}_{\text{tot}}$ denotes the stabilized team's action-value function, which means that its gradients will not be taken into account in the backward pass. The final loss function of Vanilla QTRAN is a linear combination of ℓ_{td}, ℓ_{opt}, and ℓ_{nopt} with coefficients λ_{opt} and λ_{nopt} as follows:

$$\ell_{\text{QTRAN}}(e;[\theta_{Q_i}]_{i\in\mathcal{N}},\rho,\upsilon)=\ell_{\text{td}}+\lambda_{\text{opt}}\ell_{\text{opt}}+\lambda_{\text{nopt}}\ell_{\text{nopt}} \tag{4.28}$$

Once training concludes, all neural networks except for Q_i-functions are discarded since they are not parts of the decentralized execution.

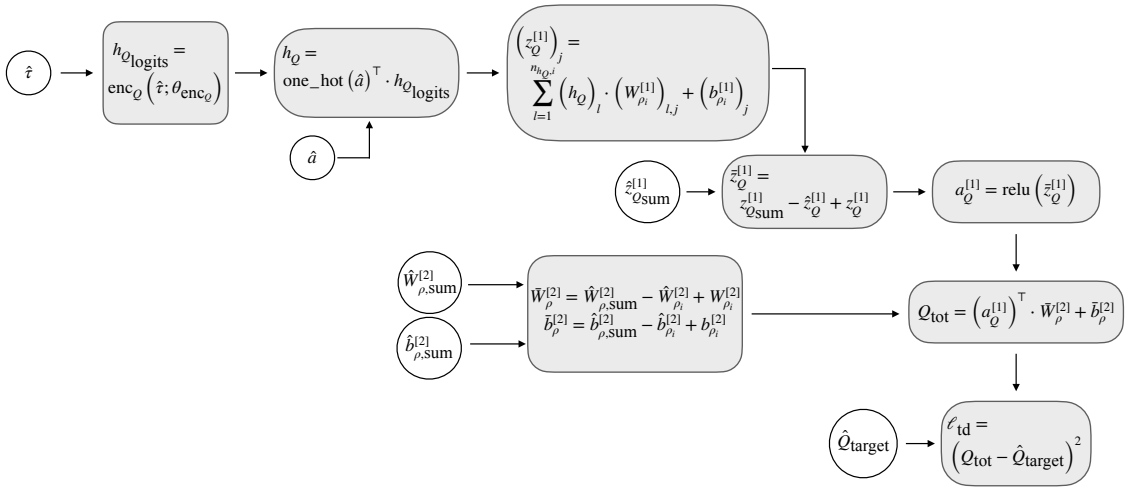

Figure 4.8: The computational graph of the forward pass of ℓ_{td} in PE-VDN.

4.3.3.2 Privacy-Engineered QTRAN (PE-QTRAN)

In PE-QTRAN, each agent trains its encoder and $[\theta_{Q_i}]_{i\in\mathcal{N}}$ parameters, and maintains and trains a local share of the Q_{tot}- and V-functions. In order to compute

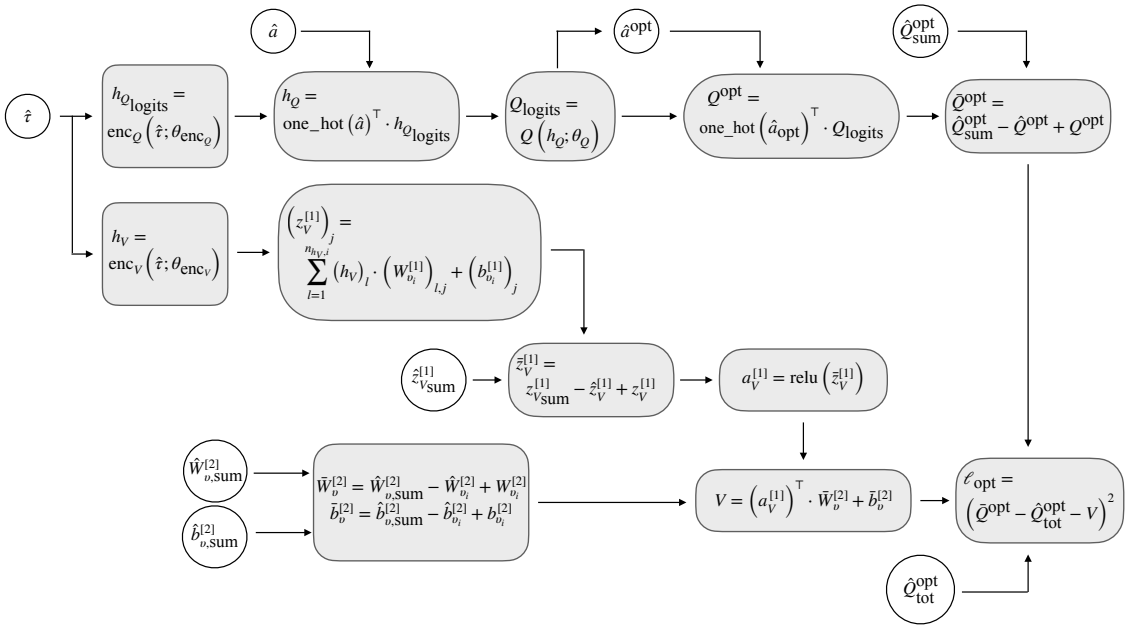

Figure 4.9: The computational graph of the forward pass of ℓ_{opt} in PE-QTRAN.

a forward pass of ℓ_{QTRAN}, similar to PE-QMIX, the agents reconcile their shares of Q_{tot} and V one layer at a time. In particular, for agent i, let $n_{h_Q,i}$ and $n_{h_V,i}$ be the sizes of its h_Q and h_V encodings, respectively. Then, the agents jointly compute the first layer of the Q_{tot}- and V-functions as follows:

$$\left(z_{Q_{\text{tot}}}^{[1]}\right)_j = \sum_{i\in\mathcal{N}} \left(\sum_{l=1}^{n_{h_Q,i}} \left(h_{Q_i}(\tau_i, a_i)\right)_l \cdot \left(W_{\rho_i}^{[1]}\right)_{l,j} + \left(b_{\rho_i}^{[1]}\right)_j \right), \tag{4.29}$$

$$\left(z_V^{[1]}\right)_j = \sum_{i\in\mathcal{N}} \left(\sum_{l=1}^{n_{h_V,i}} \left(h_{V_i}(\tau_i)\right)_l \cdot \left(W_{\upsilon_i}^{[1]}\right)_{l,j} + \left(b_{\upsilon_i}^{[1]}\right)_j \right), \tag{4.30}$$

where $W_{\rho_i}^{[1]}$ and $W_{\upsilon_i}^{[1]}$ are the weights of the first layer of Q_{tot} and V, respectively, and $b_{\rho_i}^{[1]}$ and $b_{\upsilon_i}^{[1]}$ are the biases. For the subsequent layers $k \geq 2$, the agents compute the summations

$$\left(\bar{W}_\rho^{[k]}, \bar{b}_\rho^{[k]}\right) = \left(\sum_{i\in\mathcal{N}} W_{\rho_i}^{[k]}, \sum_{i\in\mathcal{N}} b_{\rho_i}^{[k]}\right) \quad \text{and} \quad \left(\bar{W}_\upsilon^{[k]}, \bar{b}_\upsilon^{[k]}\right) = \left(\sum_{i\in\mathcal{N}} W_\upsilon^{[k]}, \sum_{i\in\mathcal{N}} b_\upsilon^{[k]}\right), \tag{4.31}$$

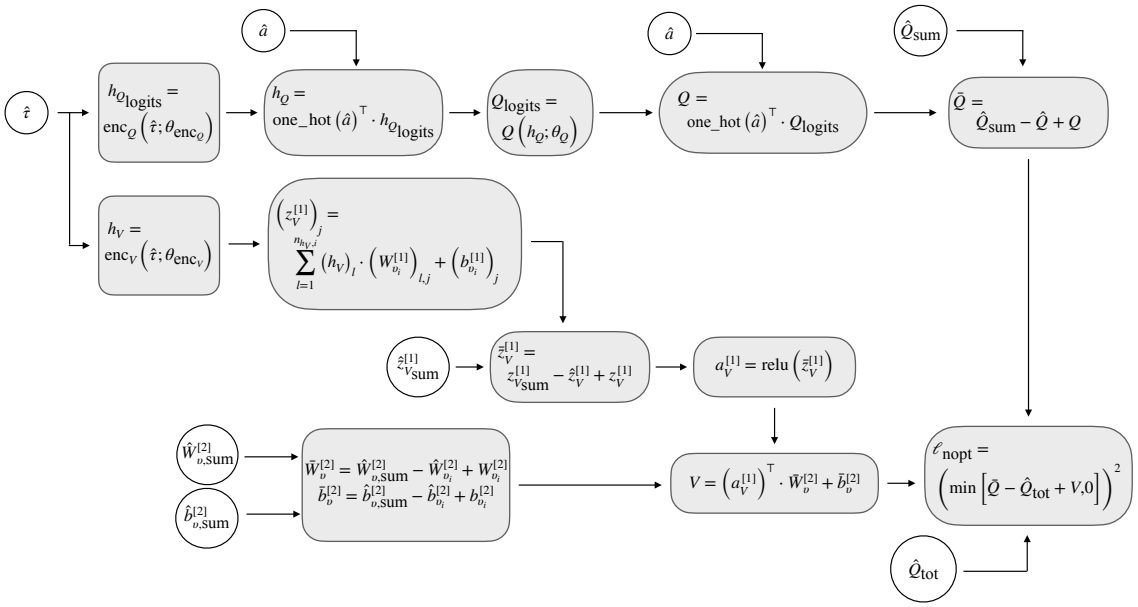

Figure 4.10: The computational graph of the forward pass of ℓ_{nopt} in PE-QTRAN.

and compute the subsequent layers of the Q_{tot} and V neural networks similar to PE-QMIX in (4.22). The summation of the Q_i-values can be jointly computed analogously to PE-VDN. At this point, every agent is able to compute a forward pass of ℓ_{QTRAN}. Figures 4.8, 4.9, and 4.4 provide further details about the computational graph of the ℓ_{td}, ℓ_{opt}, and ℓ_{nopt} loss functions in PE-QTRAN, respectively. We briefly note that all of the required communication protocols involve computing summations which can be readily integrated with additive secret sharing for secure multi-party computation.

4.4 Enforcing Differential Privacy

As we discussed in Chapter 3, the predictions of neural networks could reveal specific relationships in their training dataset that were not intended to be exposed. When the agents make decisions based on their jointly trained action-value functions, they may expose themselves to privacy attacks that aim to make inferences about their past environment interactions. Privacy-enhancing techniques based on differential privacy can provide theoretical privacy guarantees against such privacy threats

(Zhang et al., 2021). 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 (Abadi et al., 2016b) is a differentially private supervised learning algorithm for neural networks and enforces differential privacy by clipping the gradients and repeatedly injecting calibrated Gaussian noise into the clipped gradients. The algorithm's so-called Moments Accountant method tracks the differential privacy level—ϵ and δ—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 $\text{Clip}_C(g_x) := g_x \cdot \max\left(1, \|g_x\|_2/C\right)^{-1}$. These steps can be integrated with the proposed PE-VDN, PE-QMIX, and PE-QTRAN algorithms in the previous section. It is important that the indices selected through Poisson sampling be the same across the agents because the entries of the agents' replay buffers are coupled with team rewards. The agents must first reach a consensus over the Poisson indices, and then choose those indices from their respective replay buffers. For example, one of the agents may choose Poisson samples and broadcast them to all other agents. Once the samples are determined, the agents can follow the DP-SGD algorithm by clipping the computed gradients and adding Gaussian noise to the clipped gradients.

We now study the differential privacy level of the DP-SGD algorithm when applied to PE-VDN, PE-QMIX, and PE-QTRAN. The Moments Accountant method computes the level of differential privacy for static datasets, whereas, in PE-VDN,

PE-QMIX, and PE-QTRAN, the replay buffers are continuously updated with the newest sample replacing the oldest. The following theorem demonstrates how the Moments Accountant method can be reconciled with dynamic replay buffers.

Theorem 4.2. *Fix a Poisson sampling rate q, noise variance σ^2, and δ 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, PE-QMIX, or PE-QTRAN 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

$$
\begin{aligned}
&D_1 = \{d_1, d_2, \ldots, d_B\} \\
&D_2 = \{d_2, d_3, \ldots, d_{B+1}\} \\
&\vdots \\
&D_B = \{d_B, d_{B+1}, \ldots, d_{2B-1}\} \\
&D_{B+1} = \{d_{B+1}, d_{B+2}, \ldots, d_{2B}\} \\
&\vdots
\end{aligned}
$$

At the i^{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 (Smith et al., 2022) can leverage such special set overlaps to provide better differential privacy bounds. Consider the overlap set

$$
X := \left\{ I \subseteq \{1, \ldots, T\} \mid \bigcap_{i \in I} D_i \neq \varnothing \right\}. \tag{4.32}
$$

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 (Smith et al., 2022), 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. □

4.5 Numerical Results

We implement a Python library for PE-VDN, PE-QMIX, and PE-QTRAN[2]. We use the StarCraft Multi-Agent Competition (SMAC) suite (Samvelyan et al., 2019), 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.

Figure 4.11: A screenshot of SMAC's `3m` environment.

In the first set of experiments, we compare the performance of the three proposed algorithms, namely PE-VDN, PE-QMIX, PE-QTRAN, with independent Q-learning (IQL). In Figures 4.12, 4.13, and 4.14, we observe that all three proposed algorithms outperform IQL in terms of stability of the learning curve and convergence

[2]All codes and instructions are provided at the github repository `https://github.com/parhamgohari/PrivacyCoMARL.git`.

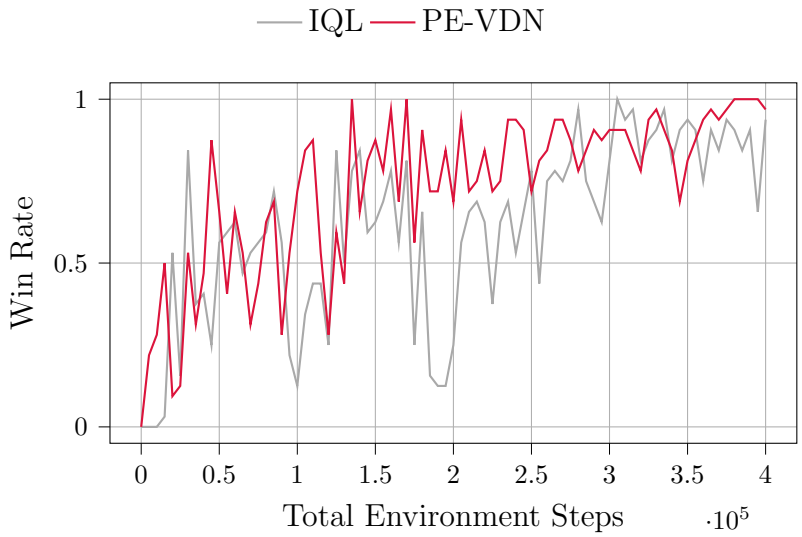

Figure 4.12: Comparison of the team's win rate in IQL and PE-VDN.

rate; therefore, the agents indeed benefit from communicating with one another. The learning curve of the PE-QTRAN algorithm appears to show better quality in terms of both stability and quality and convergence rate. However, in terms of neural network sizes and computational complexity, training with PE-VDN was approximately 1.5 times faster.

In the second 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 (Ponomareva et al., 2023). In particular, hyperparameter tuning may play a key role in model accuracy, training stability, and sample complexity (Ponomareva et al., 2023). Clipping gradients may add bias, perturbing the gradients may deflect the parameters away from local minima and destabilize training, and achieving meaningful ϵ and δ values—$\epsilon < 3$ and $\delta < 1/n^{1.1}$, where n is the training dataset's size (Ponomareva et al., 2023)—may require larger

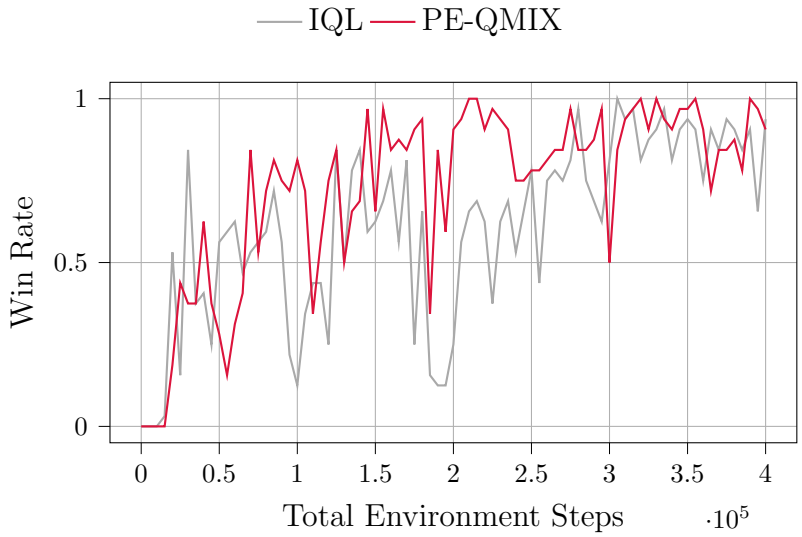

Figure 4.13: Comparison of the team's win rate in IQL and PE-QMIX.

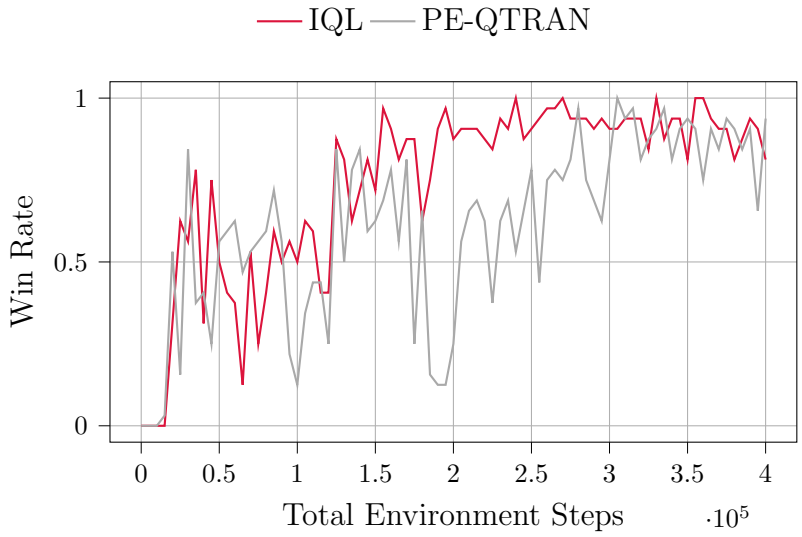

Figure 4.14: Comparison of the team's win rate in IQL and PE-QTRAN.

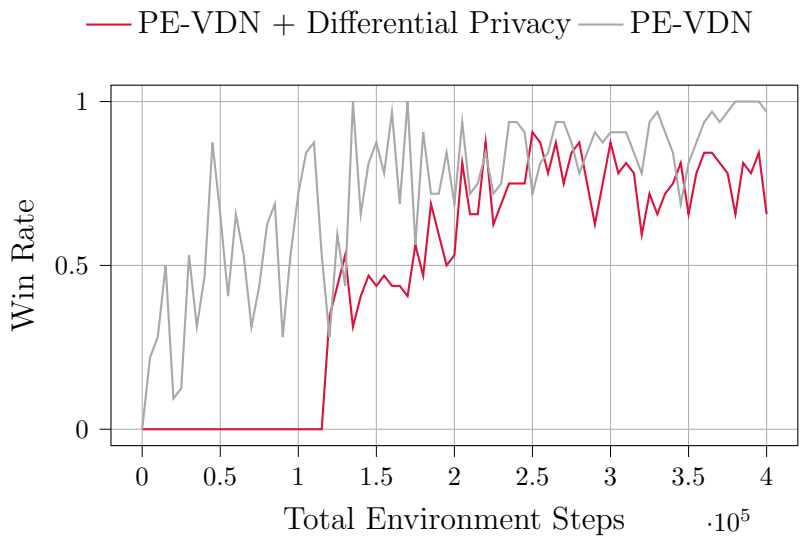

Figure 4.15: Comparison of the team's win rate in PE-VDN under $(2.90, 4.9 \cdot 10^{-4})$-differential privacy and PE-VDN without differential privacy protections. The win rates correspond to the anchor models.

training datasets.

In our implementations, we use the Opacus (Yousefpour et al., 2021) library for DP-SGD. We use Opacus' built-in Moments Accountant method to track differential privacy levels as stated in Theorem 4.2. Without differential privacy, we used the Adam optimizer with a learning rate of $5 \cdot 10^{-4}$. In light of the guidelines in (Ponomareva et al., 2023), we found that using the SGD optimizer with weight decay 0.01, momentum 0.9, and a significantly higher learning rate $1 \cdot 10^{-3}$ performs better for DP-SGD. We were able to achieve approximately 80% win rate in PE-VDN and PE-QMIX under $(2.90, 4.9 \cdot 10^{-4})$-differential privacy. Figures 4.15 and 4.16 compare the learning curve of the two algorithms with and without differential privacy. In these two plots, we use an "anchoring" technique that simply consists of saving the model with the best win rate during periodic evaluations as the anchor model.

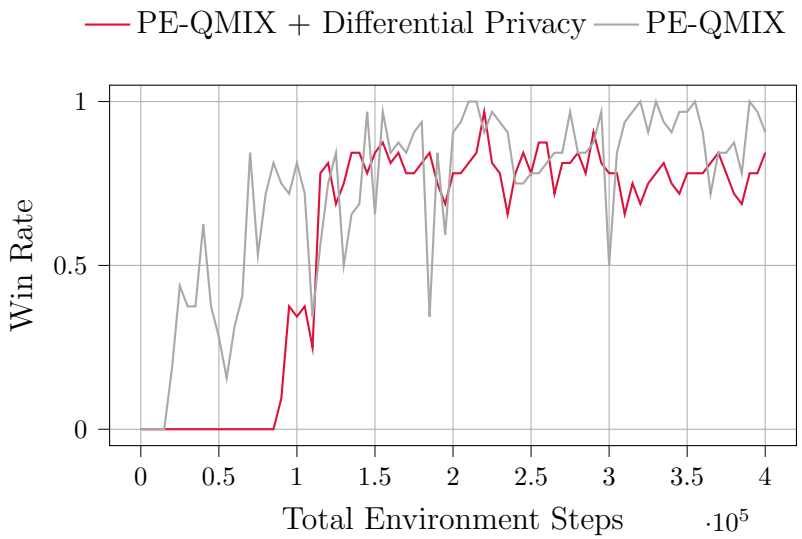

Figure 4.16: Comparison of the team's win rate in PE-QMIX under $(2.90, 4.9 \cdot 10^{-4})$-differential privacy and PE-QMIX without differential privacy protections. The win rates correspond to the anchor models.

4.6 Conclusion

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

Chapter 5: Conclusions and Future Research Opportunities

This dissertation demonstrated that it is indeed possible to address privacy concerns in AI systems through privacy engineering. In particular, the results highlight the effectiveness of the privacy analysis methodology that was adopted in this research based on technical privacy policies. We showcased how forming technical privacy policies can reduce the ambiguity often present in the social concept of privacy and lead to well-defined engineering problems which is within the same wave length of AI algorithms.

We studied privacy engineering AI systems for two sequential decision-making problems, namely policy synthesis in MDPs and Co-MARL. We set the privacy priority of the former to guarantee that the MDP's transition probabilities will not be revealed through the agents' policies, and set the privacy priority in the latter to protect the confidentiality of the agents' individual interactions with the environment. We developed a privacy engineered a policy synthesis algorithm using a specialized differential privacy mechanism that we designed for simplex-valued data. Furthermore, we privacy engineered three value-based Co-MARL algorithms that conformed to CTDE, namely VDN, QMIX, and QTRAN. The proposed algorithms allow the agents to coordinate their training for higher team rewards while maintaining the secrecy of their environment interaction data.

Several future research opportunities may follow the contributions of this dissertation. First and foremost, the introduced privacy engineering methodology can be extended to AI applications other than sequential decision-making problems. Moreover, there is always an opportunity to improve the proposed privacy-engineered algorithms' performance and the technical privacy guarantees. Furthermore, using technical privacy policies may be an effective way for bridging the gaps between legal and technical privacy mechanisms. For example, a technical privacy policy may

be an appropriate means to finding a common vocabulary between privacy laws and technical privacy definitions such as differential privacy and secure multi-party computation. This dissertation may motivate a subsequent social study of the potential benefits and drawbacks of using technical privacy policies in consumer-facing technologies and services.

Works Cited

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*. ACM, 2016a. doi: 10.1145/2976749.2978318.

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*. ACM, 2016b. doi: 10.1145/2976749.2978318.

John M. Abowd. The U.S. census bureau adopts differential privacy. In Yike Guo and Faisal Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD8*. ACM, 2018. doi: 10.1145/3219819.3226070.

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.

Jason W. Bentley, Daniel Gibney, Gary Hoppenworth, and Sumit Kumar Jha. Quantifying membership inference vulnerability via generalization gap and other model metrics. 2020.

Abhishek Bhowmick, John C. Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning. 2018.

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve

Saint-Amand, Radu Soricut, Lucia Specia, and Ales Tamchyna. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation, WMT@ACL*. The Association for Computer Linguistics, 2014. doi: 10.3115/v1/w14-3302.

Luca Bonomi, Li Xiong, Rui Chen, and Benjamin C. M. Fung. Privacy preserving record linkage via grams projections. abs/1208.2773, 2012.

Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014. ISBN 978-0-521-83378-3. doi: 10.1017/CBO9780511804441.

Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic mdp-behavior planning for cars. In *14th International IEEE Conference on Intelligent Transportation Systems, ITSC*. IEEE, 2011a. doi: 10.1109/ITSC.2011.6082928.

Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic mdp-behavior planning for cars. In *14th International IEEE Conference on Intelligent Transportation Systems, ITSC*. IEEE, 2011b. doi: 10.1109/ITSC.2011.6082928.

Sylvain Chatel, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Privacy and integrity preserving computations with CRISP. In *30th USENIX Security Symposium, USENIX Security*. USENIX Association, 2021.

Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 2011. doi: 10.5555/1953048.2021036.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym, 2018. URL `https://github.com/maximecb/gym-minigrid`.

Jorge Cortés, Geir E. Dullerud, Shuo Han, Jerome Le Ny, Sayan Mitra, and George J. Pappas. Differential privacy in control and network systems. In *55th IEEE Conference on Decision and Control, CDC*. IEEE, 2016. doi: 10.1109/CDC.2016.7798915.

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.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, Proceedings of Machine Learning Research. PMLR, 2017. URL `http://proceedings.mlr.press/v70/dauphin17a.html`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423.

Jinshuo Dong, Aaron Roth, and Weijie J. Su. Gaussian differential privacy. 2019.

Samuel Dupond. A thorough review on the current advance of neural network structures. *Annual Reviews in Control*, 2019. doi: 10.1146/annurev-statistics-040220-112019.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends Theoretical Computer Science*, 2014. doi: 10.1561/0400000042.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC*, Lecture Notes in Computer Science. Springer, 2006. doi: 10.1007/11681878_14.

Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language, hosted by the 54th Annual Meeting of the Association for Computational Linguistics, VL@ACL*. The Association for Computer Linguistics, 2016. doi: 10.18653/v1/w16-3210.

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.

Paul Gavrikov and Janis Keuper. CNN filter DB: an empirical investigation of trained convolutional filters. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01848.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, Proceedings of Machine Learning Research. PMLR, 2017. URL `http://proceedings.mlr.press/v70/gehring17a.html`.

Dorothy J Glancy. Privacy in autonomous vehicles. *Santa Clara Law Review*, 2012. ISSN 0146-0315.

Parham Gohari, Matthew T. Hale, and Ufuk Topcu. Privacy-preserving policy synthesis in markov decision processes. In *59th IEEE Conference on Decision and Control, CDC*. IEEE, 2020a. doi: 10.1109/CDC42340.2020.9304015.

Parham Gohari, Bo Wu, Matthew T. Hale, and Ufuk Topcu. The dirichlet mechanism for differential privacy on the unit simplex. In *2020 American Control Conference, ACC*. IEEE, 2020b. doi: 10.23919/ACC45564.2020.9147447.

Parham Gohari, Bo Wu, Calvin Hawkins, Matthew T. Hale, and Ufuk Topcu. Differential privacy on the unit simplex via the dirichlet mechanism. *IEEE Transactions Information Forensics and Security*, 2021. doi: 10.1109/TIFS.2021.3052356.

Parham Gohari, Matthew T. Hale, and Ufuk Topcu. Privacy-engineered value decomposition networks for cooperative multi-agent reinforcement learning. In *62nd IEEE Conference on Decision and Control, CDC*. IEEE, 2023.

Maziar Gomrokchi, Susan Amin, Hossein Aboutalebi, Alexander Wong, and Doina Precup. Where did you learn that from? surprising effectiveness of membership inference attacks against temporally correlated data in deep reinforcement learning. 2021.

Michaela Götz, Ashwin Machanavajjhala, Guozhang Wang, Xiaokui Xiao, and Johannes Gehrke. Publishing search logs - A comparative study of privacy guarantees. *IEEE Transactions on Knowledge and Data Engineering*, 2012. URL `https://doi.org/10.1109/TKDE.2011.26`.

Benjamin Graham. Spatially-sparse convolutional neural networks. 2014. URL `http://arxiv.org/abs/1409.6070`.

Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009. URL `https://doi.org/10.1109/TPAMI.2008.137`.

Zhitao Guan, Guanlin Si, Xiaosong Zhang, Longfei Wu, Nadra Guizani, Xiaojiang Du, and Yinglong Ma. Privacy-preserving and efficient aggregation based on blockchain for power grid communications in smart communities. *IEEE Communications Magazine*, 2018. doi: 10.1109/MCOM.2018.1700401.

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.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. 2018.

Matthew T. Hale, Austin Jones, and Kevin Leahy. Privacy in feedback: The differentially private LQG. In *2018 Annual American Control Conference, ACC*. IEEE, 2018. doi: 10.23919/ACC.2018.8431397.

Robert J. Hall, Larry A. Wasserman, and Alessandro Rinaldo. Random differential privacy. *Journal of Privacy and Confidentiality*, 2013. doi: 10.29012/jpc.v4i2.621.

Shuo Han and George J. Pappas. Privacy in control and dynamical systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018. doi: 10.1146/annurev-control-060117-105018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90.

Sorami Hisamoto, Matt Post, and Kevin Duh. Membership inference attacks on sequence-to-sequence models: Is my data in your machine translation system? *Transactions of the Association for Computional Linguistics*, 2020. doi: 10.1162/tacl_a_00299.

Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, and Aaron Roth. Differential privacy: An economic method for choosing epsilon. In *IEEE 27th Computer Security Foundations Symposium, CSF*. IEEE Computer Society, 2014. doi: 10.1109/CSF.2014.35.

Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys*, 2022. doi: 10.1145/3523273.

Garud N. Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 2005. doi: 10.1287/moor.1040.0129.

Austin Jones, Kevin Leahy, and Matthew T. Hale. Towards differential privacy for symbolic systems. In *2019 American Control Conference, ACC 2019*. IEEE, 2019. doi: 10.23919/ACC.2019.8814723.

Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 1984. doi: 10.1007/BF02579150.

Dmitrii Borisovich Karp. Normalized incomplete beta function: Log-concavity in parameters and other properties. *Journal of Mathematical Sciences*, 2016. doi: 10.1007/s10958-016-2958-z.

Shiva Prasad Kasiviswanathan and Adam D. Smith. On the 'semantics' of differential privacy: A bayesian formulation. *Journal of Privacy and Confidentiality*, 2014. doi: 10.29012/jpc.v6i1.634.

Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020. doi: 10.1145/3372297.3417872.

Linghe Kong, Muhammad Khurram Khan, Fan Wu, Guihai Chen, and Peng Zeng. Millimeter-wave wireless communications for iot-cloud supported autonomous vehicles: Overview, design, and challenges. *IEEE Communications Magazine*, 2017. doi: 10.1109/MCOM.2017.1600422CM.

Samuel Kotz, Narayanaswamy Balakrishnan, and Norman L. Johnson. Continuous multivariate distributions: Models and applications, volume 1, second edition. *Wiley Series in Probability and Statistics*, 2000. doi: 10.1002/0471722065.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL `http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Xiangang Li and Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*. IEEE, 2015. doi: 10.1109/ICASSP.2015.7178826.

Xiujun Li, Lihong Li, Jianfeng Gao, Xiaodong He, Jianshu Chen, Li Deng, and Ji He. Recurrent reinforcement learning: A hybrid approach. 2015. URL `http://arxiv.org/abs/1509.03044`.

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.

Zhenjiang Li, Cheng Chen, and Kai Wang. Cloud computing for agent-based urban transportation systems. *IEEE Intelligent Systems*, 2011. doi: 10.1109/MIS.2011.10.

Boyi Liu, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural trust region/proximal policy optimization attains globally optimal policy. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/227e072d131ba77451d8f27ab9afdfb7-Abstract.html`.

Jianqing Liu, Chi Zhang, and Yuguang Fang. EPIC: A differential privacy framework to defend smart homes against internet traffic analysis. *IEEE Internet Things Journal*, 2018. doi: 10.1109/JIOT.2018.2799820.

Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A. Gunter, and Kai Chen. Understanding membership inferences on well-generalized learning models. 2018.

Zhigang Lu, Hassan Jameel Asghar, Mohamed Ali Kâafar, Darren Webb, and Peter Dickinson. A differentially private framework for deep learning with convexified loss functions. *IEEE Transactions of Information Forensics and Security*, 2022. doi: 10.1109/TIFS.2022.3169911.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP*. The Association for Computational Linguistics, 2015. URL `https://doi.org/10.18653/v1/d15-1166`.

Christopher Lusena. Finite memory policies for partially observable markov decision proesses. *University of Kentucky Doctoral Dissertations*, 2001. URL `https://uknowledge.uky.edu/gradschool_diss/323`.

Ashwin Machanavajjhala, Daniel Kifer, John M. Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *Proceedings of the 24th International Conference on Data Engineering, ICDE*. IEEE Computer Society, 2008. doi: 10.1109/ICDE.2008.4497436.

Olivier Marchal and Julyan Arbel. On the sub-gaussianity of the beta and dirichlet distributions. *Electronic Communications in Probability*, 2017. doi: 10.1214/17-ECP92.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 1989. doi: 10.1016/S0079-7421(08)60536-8.

H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *6th International Conference on Learning Representations, ICLR*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=BJ0hF1Z0b`.

Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science FOCS*. IEEE Computer Society, 2007. doi: 10.1109/FOCS.2007.41.

Fatemehsadat Mireshghallah, Mohammadkazem Taram, Praneeth Vepakomma, Abhishek Singh, Ramesh Raskar, and Hadi Esmaeilzadeh. Privacy in deep learning: A survey. abs/2004.12254, 2020.

Sudip Misra, Ayan Mondal, Shukla Banik, Manas Khatua, Samaresh Bera, and Mohammad S. Obaidat. Residential energy management in smart grid: A markov decision process-based approach. In *IEEE International Conference on Green Computing and Communications (GreenCom) and IEEE Internet of Things (iThings) and IEEE Cyber, Physical and Social Computing (CPSCom)*. IEEE, 2013a. doi: 10.1109/GreenCom-iThings-CPSCom.2013.200.

Sudip Misra, Ayan Mondal, Shukla Banik, Manas Khatua, Samaresh Bera, and Mohammad S. Obaidat. Residential energy management in smart grid: A markov decision process-based approach. In *IEEE International Conference on Green Computing and Communications (GreenCom) and IEEE Internet of Things (iThings) and IEEE Cyber, Physical and Social Computing (CPSCom)*. IEEE, 2013b. doi: 10.1109/GreenCom-iThings-CPSCom.2013.200.

Volodymyr Mnih, 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`.

Milad Nasr, Reza Shokri, and Amir Houmansadr. Machine learning with membership privacy using adversarial regularization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM, 2018. doi: 10.1145/3243734.3243855.

Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy, SP*. IEEE, 2019. doi: 10.1109/SP.2019.00065.

Arnab Nilim and Laurent El Ghaoui. *Robust Control of Markov Decision Processes with Uncertain Transition Matrices*. PhD thesis, 2005.

Kobbi Nissim and Alexandra Wood. Is privacy privacy? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2018. doi: 10.1098/rsta.2017.0358.

Erfan Nozari, Pavankumar Tallapragada, and Jorge Cortés. Differentially private distributed convex optimization via functional perturbation. *IEEE Trans. Control. Netw. Syst.*, 2018. doi: 10.1109/TCNS.2016.2614100.

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`.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. ACL, 2002. doi: 10.3115/1073083.1073135.

Andreas Pfeuffer and Klaus Dietmayer. Robust semantic segmentation in adverse weather conditions by means of fast video-sequence segmentation. In *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC*. IEEE, 2020. doi: 10.1109/ITSC45102.2020.9294554.

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. doi: 10.48550/arXiv.2303.00654.

András Prékopa. Logarithmic concave measures with application to stochastic programming. *Acta Scientiarum Mathematicarum*, 1971. URL `http://rutcor.rutgers.edu/~prekopa/SCIENT1.pdf`.

András Prékopa. On logarithmic concave measures and functions. *Acta Scientiarum Mathematicarum*, 1973. URL `http://rutcor.rutgers.edu/~prekopa/SCIENT2.pdf`.

John E. Prescott, Ajay K. Kohli, and N. Venkatraman. The market share-profitability relationship: An empirical assessment of major assertions and contradictions. *Strategic Management Journal*, 1986. URL https://www.jstor.org/stable/2486069.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. ISBN 978-0-47161977-2.

Md. Atiqur Rahman, Tanzila Rahman, Robert Laganière, and Noman Mohammed. Membership inference attack against differentially private deep learning model. *Transactions om Data Privacy*, 2018. URL http://www.tdp.cat/issues16/tdp.a289a17.pdf.

JS Rao and Milton Sobel. Incomplete dirichlet integrals with applications to ordered uniform spacings. *Journal of Multivariate Analysis*, 1980. doi: 10.1016/0047-259X(80)90073-1.

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.

Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning. abs/2007.07646, 2020.

Benjamin I. P. Rubinstein and Francesco Aldà. Pain-free random differential privacy with sensitivity sampling. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, Proceedings of Machine Learning Research. PMLR, 2017. URL http://proceedings.mlr.press/v70/rubinstein17a.html.

Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, Yann Ollivier, and Hervé Jégou. White-box vs black-box: Bayes optimal strategies for membership inference. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, Proceedings of Machine Learning Research. PMLR, 2019. URL http://proceedings.mlr.press/v97/sablayrolles19a.html.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*. ISCA, 2014. doi: 10.21437/Interspeech.2014-80.

Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *26th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2019. URL https://www.ndss-symposium.org/ndss-paper/ml-leaks-model-and-data-independent-membership-inference-attacks-and-defenses-on-machine-learning-models/.

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.

Kevin Scaman, Ludovic Dos Santos, Merwan Barlier, and Igor Colin. A simple and efficient smoothing method for faster optimization and local exploration. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/481d462e46c2ab976294271a175b8929-Abstract.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.

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

Vatsal Sharan, Sham M. Kakade, Percy Liang, and Gregory Valiant. Prediction with a short memory. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*. ACM, 2018. doi: 10.1145/3188745.3188954.

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.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy, SP*. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.41.

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.

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`.

Congzheng Song and Vitaly Shmatikov. Auditing data provenance in text-generation models. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM*

SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD. ACM, 2019. URL https://doi.org/10.1145/3292500.3330885.

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 Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *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.

David M. Szymanski, Sundar G. Bharadwaj, and P. Rajan Varadarajan. An analysis of the market share-profitability relationship. *Journal of marketing*, 1993. doi: 10.2307/1251851.

Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. Demystifying membership inference attacks in machine learning as a service. *IEEE Transactions on Services Computing*, 2021. doi: 10.1109/TSC.2019.2897554.

Salil P. Vadhan. The complexity of differential privacy. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*. Springer International Publishing, 2017. doi: 10.1007/978-3-319-57048-8_7.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Parv Venkitasubramaniam. Privacy in stochastic control: A markov decision process perspective. In *51st Annual Allerton Conference on Communication, Control, and Computing*. IEEE, 2013. doi: 10.1109/Allerton.2013.6736549.

Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron C. Courville, and Yoshua Bengio. Renet: A recurrent neural network based alternative to convolutional networks. 2015.

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.

Yu Wang, Zhenqi Huang, Sayan Mitra, and Geir E. Dullerud. Differential privacy in linear distributed control systems: Entropy minimizing mechanisms and performance tradeoffs. *IEEE Trans. Control. Netw. Syst.*, 2017. doi: 10.1109/TCNS.2017.2658190.

Lucas Willems. Rl-starter-files, 2018. URL `https://github.com/lcswillems/rl-starter-files`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP*. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-demos.6.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey,

Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. 2016.

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

Huan Xu and Shie Mannor. Distributionally robust markov decision processes. 2012. doi: 10.1287/moor.1120.0540.

Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. 2020.

Yunhao Yang, Parham Gohari, and Ufuk Topcu. On the privacy risks of deploying recurrent neural networks in machine learning models. *Proceedings of Privacy Enhancing Technologies*. doi: 10.56553/popets-2023-0005.

Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *31st IEEE Computer Security Foundations Symposium, CSF*. IEEE Computer Society, 2018. doi: 10.1109/CSF.2018.00027.

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.

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.

Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett. Functional mechanism: Regression analysis under differential privacy. *Proceedings of the VLDB Endowment*, 2012. doi: 10.14778/2350229.2350253.

Mengyuan Zhang, Jiming Chen, Lei Yang, and Junshan Zhang. Dynamic pricing for privacy-preserving mobile crowdsensing: A reinforcement learning approach. *IEEE Network*, 2019. doi: 10.1109/MNET.2018.1700468.

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.