UNDERSTANDING AND MITIGATING PRIVACY VULNERABILITIES IN DEEP LEARNING

A Dissertation
Presented to
The Academic Faculty

By

Sanjay Kariyappa

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering
Department of Electrical and Computer Engineering

Georgia Institute of Technology

Dec 2022

UNDERSTANDING AND MITIGATING PRIVACY VULNERABILITIES IN DEEP LEARNING

Thesis committee:

Prof. Moinuddin K Qureshi
School of Computer Science
Georgia Institute of Technology

Prof. Ghassan AlRegib
School of Electrical and Computer Engineering
Georgia Institute of Technology

Prof. Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Prof. Atul Prakash
School of Electrical and Computer Engineering
University of Michigan

Prof. G. Edward Suh
School of Electrical and Computer Engineering
Cornell University

Date approved: October 31, 2022

Imagination is more important than knowledge.

Albert Einstein

To my parents

ACKNOWLEDGMENTS

This PhD has been a transformative adventure, filled with challenging obstacles and exhilarating new highs. Completing it would not have been possible without the love, support, and guidance of my family, advisor, collaborators, and friends, whose contributions I'd like to acknowledge.

First, I thank my loving parents, Gowri KG and Kariyappa B. Despite their humble beginnings, they've worked hard to ensure that I had a vastly better childhood than their own. Pursuing higher education in the US would not have been possible without their support and encouragement. They've taught me the value of hard work and have always pushed me to follow my dreams. The opportunities I have in life are owed to their sacrifices, for which I'm eternally grateful. I dedicate this thesis to them.

The overwhelmingly positive experience of my PhD is attributable primarily to my advisor Prof. Moinuddin Qureshi. Most of what I know about doing research, academic writing, and presenting, I've learned directly from him or by trying to mimic him. Moin has been a guiding light through the darkest parts of my PhD. His kindness and empathy, especially in challenging times, have made me feel well-supported. I'm also extremely grateful for the freedom he has given me to pursue topics I'm genuinely passionate about.

My colleagues at the Memory Systems Lab have been an influential part of my PhD. I want to thank Vinson Young for his mentorship when I started my PhD. I've greatly enjoyed the company of my lab mates- Swamit Tannu, Gururaj Saileshwar, and Poulami Das. I appreciate the feedback and guidance from them over the years. I also want to thank Mohammad, Ramin, Narges, Aditya, Anish, Suhas, and Saurav for their company. I'm also grateful for the mentorship of Sean Lee, Prof. Ed Suh, and Beliz Gokkaya at Meta AI, and Geoff Burr at IBM Research. I've had the opportunity to collaborate with and learn from a fantastic set of researchers who have been instrumental in shaping my thesis. I thank Prof. Atul Prakash, Sidney Tsai, Stefano Ambrogio, Ousmane Dia, Chuan Guo, Kiwan Maeng, Wenjie Xiong, and Neal Mangaokar for this.

Lastly, I want to acknowledge the excellent set of friends I've made at Georgia Tech, who have been a second family to me. I feel lucky to have the friendship of Rakshith, whose emotional support and delicious cooking have made it much easier to get through bad days. I'm also thankful to Suraj, Soup, Yashas, Deepak, Shreya, Pradyumna, and Girish for their friendship. I've loved our time together celebrating festivals, going to salsa socials, and playing late-night Catan games. These are some of the best memories I've made during my PhD! I thank Alisha for her support and kindness, which has been uplifting in times of self-doubt and uncertainty. I'd also like to thank my younger brother Savinay and my friends Atul, Brighton, Suhas, Prabhava, and Shaista for their encouragement and support throughout my PhD. Finally, I express my gratitude towards Georgia Tech - a place that has provided me access to fantastic teachers, exceptional researchers, and exciting new opportunities. I'm going to forever cherish the amazing memories I've made here.

SUMMARY

Advancements in Deep Learning (DL) have enabled leveraging large-scale datasets to train models that perform challenging tasks at a level that mimics human intelligence. In several real-world scenarios, the data used for training, the trained model, and the data used for inference can be private and distributed across multiple distrusting parties. The distributed and privacy-sensitive nature of such data can pose a challenge for training DL models and using these models to provide inference services.

Several frameworks for training and inference have been developed to prevent the leakage of sensitive data. For instance, frameworks like federated learning and split learning have been proposed to train a model collaboratively on distributed data without explicitly sharing the private data to protect training data privacy. To protect model privacy during inference, the model owners have adopted a client-server architecture to provide inference services, wherein the end-users are only allowed black-box access to the predictions of the model for their input queries.

While these frameworks have the appearance of keeping the data and model private, the information exchanged during training/inference has the potential to compromise the privacy of the parties involved by leaking sensitive data. Indeed, recent attacks have demonstrated the vulnerabilities of these frameworks by carrying out gradient inversion attacks to leak private data in FL and model stealing attacks to leak private models in remote inference. However, these attacks are limited in applicability and often require specialized setups that provide an advantage to the attacker, which may not exist for real-world applications. For instance, prior works on gradient inversion in FL have been successful only for small batch sizes and require access to in-distribution data to scale to high-dimensional inputs. Due to the technical limitations of prior attacks, these frameworks continue to provide the appearance of privacy without any theoretically backed reasoning.

The goal of this thesis is to provide a better understanding of the privacy properties of

the DL frameworks used for privacy-preserving training and inference. We aim to understand if these frameworks are truly capable of preventing the leakage of model and training data in realistic settings. In this pursuit, we discover new vulnerabilities that can be exploited to design powerful attacks that can overcome the limitations of prior works and break the illusion of privacy. Our findings highlight the limitations of these frameworks and underscore the importance of principled techniques to protect privacy. Furthermore, we leverage our improved understanding to design better defenses that can significantly deter the efficacy of an attack. The research contributions of this thesis are organized along three key areas of thrust as described below.

1. Input Privacy in Federated Learning (FL): FL aims to protect input privacy by requiring data-owners (clients) to perform training locally using their private data and share aggregate gradients with a central server to train a model collaboratively. We develop a gradient inversion attack called *Cocktail Party Attack (CPA)* to leak the private inputs from the aggregate gradients from fully connected layers. CPA frames the attack as a blind source separation problem and adapts independent component analysis to carry out the attack and leak the private inputs. CPA significantly outperforms prior works, scales to high-dimensional inputs like ImageNet, and works for large batch sizes of up to 1024 without requiring any specialized models or prior information. The efficacy of our attack demonstrates that aggregation alone does not provide any privacy guarantees, and the poor scalability of prior works is due to their technical limitations and not because of any inherent privacy properties of FL.

2. Label Privacy in Split Learning (SL): SL is a privacy-preserving training technique for vertically partitioned data that proposes to split a model across multiple users and train it end-to-end by exchanging the embedding and gradient information. We show that gradients transmitted during SL can compromise privacy by leaking private label information. To this end, we develop *ExPLoit*– a label leakage attack that replaces the unknown labels and model parameters with learnable parameters and uses the gradient data obtained during SL

to optimize the parameters to uncover the private labels with high accuracy.

3. Model Privacy in Remote Inference: To protect the privacy of DNN models, model owners offer inference as a remote service, where a client can obtain the predictions of the model by submitting an input query to a server that hosts the model. Under this setting, we investigate if the predictions can leak information about the model. To this end, we develop a model stealing attack MAZE that can create a functionally equivalent clone model using the model's black-box predictions under a data-free setting. Our attack demonstrates that lack of access to in-distribution data does not provide a sufficient defense to prevent an adversary from carrying out model stealing attacks. Lastly, we propose the *Adaptive Misinformation (AM)* defense to protect the model from model stealing attacks. AM prevents a data-limited attacker from accessing the true predictions of the model by servicing out-of-distribution queries with misinformation. Such a mechanism impedes an adversary from carrying out model stealing attacks while maintaining high utility for benign clients.

In summary, this thesis demonstrates that the information exchanged during training and inference can compromise data and model privacy by discovering the vulnerabilities of frameworks used for privacy-preserving training and inference. We leverage these vulnerabilities to develop new attacks that overcome the limitations of prior works and achieve state-of-the-art attack performance, breaking the illusion of privacy and emphasizing the need for principled defenses. Furthermore, our insights have led to the development of defenses that safeguard model privacy with minimal impact on performance, enabling the practical deployment of models for remote inference while protecting privacy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Privacy Challenges in Machine Learning

The rapid advancement in deep learning over the past decade has enabled the development of high-performance neural networks that perform several challenging tasks at levels that approach or even surpass human intelligence. This breakthrough performance has led to the rising adoption of deep learning models for applications such as self-driving cars, voice assistants, product recommendation, text generation, and image synthesis. The training of DL models and their use in inference can involve multiple stakeholders, each with their own privacy requirements, as shown in Figure 1.1. To explain, we consider training data, model, and inference data privacy individually and describe how information exchanged between stakeholders during training and inference has the potential to compromise privacy.

1.1.1 Training Data Privacy

The amount of training data available is vital in deep learning and directly dictates the performance of the trained models. The growing reliance on the internet for various aspects of our life has led to the collection of vast amounts of data, which can be potentially used to train high-performance models. However, the data collected from apps and services such as online shopping, social media, communication, healthcare, finance, etc., tend to be distributed across multiple stakeholders who run these services. While aggregating the data would enable the development of powerful models, doing so can be challenging due to user privacy requirements and privacy constraints imposed by the law. E.g., laws like HIPAA require hospitals to keep medical records private. For finance and internet companies, user agreements and privacy regulations prevents them from sharing data.

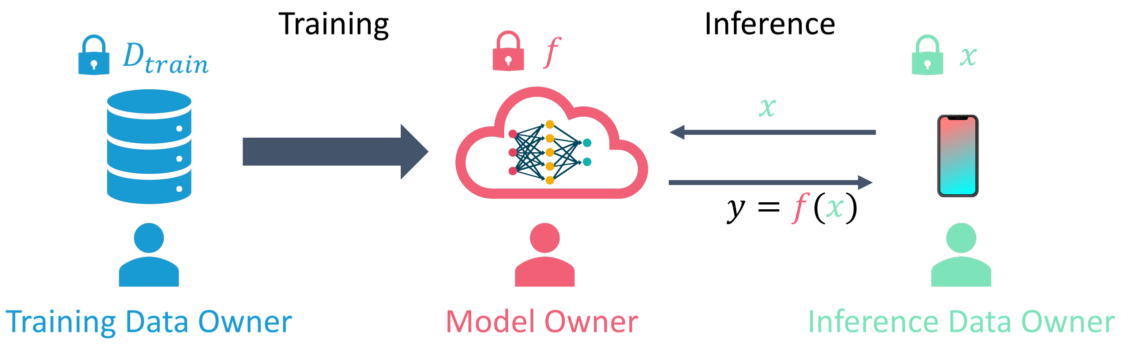


Figure 1.1: Training and inference in machine learning can involve communication between multiple stakeholders: the training data owner, model owner and the inference data owner. Each stakeholder wants to keep their respective data (i.e. training data, model and inference data) private.

To address this problem, federated [1, 2] and split learning [3, 4] frameworks have been developed to perform privacy-preserving training on distributed data. These methods avoid the direct sharing of data and require the data owners to perform training locally and share the gradients (i.e., model updates) to train a model collaboratively. The key design principle behind these approaches is the belief that sharing model gradients is more private compared to sharing raw data.

While the theory indicates that such indirect data sharing still has the potential to encode private information, just how much information is leaked by these frameworks is unclear. Indeed, several recent attacks have proposed attacks that succeed in recovering private inputs from gradients. However, these attacks have only been demonstrated in restricted settings. For instance, in FL, attacks typically only work when aggregation is performed for small batch sizes and do not scale to high-dimensional inputs. In SL, label recovery has only been demonstrated when there is a high class-imbalance for a binary classification task. The difficulty of scaling up prior works to more challenging real-world settings has led to a sense of security and has strengthened the belief that such sharing of gradients might provide sufficient protection against attacks without any principled reason. This motivates us to study the vulnerabilities in FL and SL to better understand their privacy properties.

1.1.2 Model Privacy

Training a high-performance DNN model for challenging real-world tasks can be a costly endeavor. Companies invest significant amounts of money and engineering resources for data curation, model design, and training. The models trained from these efforts are thus prized intellectual properties of the companies that own them. Theft/leakage of the model could have severe adverse consequences to the company for several reasons. First, the stolen model can be used by an adversary to offer competing services that can be detrimental to the business of the model owner. Second, stolen models can be used to craft adversarial examples to cause misclassifications in the model's predictions, which can be problematic for safety critical applications like self-driving cars. Third, stolen models can be used to leak sensitive training information using model inversion attacks, compromising training data privacy. For these reasons, safeguarding the model against theft is of prime importance to the company.

To protect models from theft, the parameters and the architecture of the model are kept secret, and customers are only allowed to access the black-box predictions of the model using a client-server model. Unfortunately, prior works [5, 6] have demonstrated that a functionally equivalent clone model can be trained using the black-box predictions of the model on distributionally similar data. However, such attacks are only possible if the adversary has access to a large number of examples sampled from a dataset that is distributionally similar to the target dataset. This limits the applicability of prior works and led to the belief that black-box access might be secure in cases where the adversary does not have access to data with significant distributional similarity. These limitations of prior works motivate us to investigate the possibility of model stealing in settings where the adversary is severely data-limited.

Furthermore, protecting the model against theft is a challenging task that needs to be addressed to safely deploy DNN models. While prior works [7, 8] have proposed to mount a defense by perturbing the predictions of the model, such defenses often incur significant degradation in performance, which hurts the utility of the model. This motivates the need for better defenses that can secure the model against model stealing attacks without significant degradation in performance.

1.1.3 Inference Data Privacy

Most customer-facing products that leverage deep learning perform inference with a client-server architecture. In such cases, the user is required to send their data to the server, which performs inference on the data and returns the predictions to the user. Since customer data is readily accessible to the service provider, remote inference offers no privacy to the end-user, which can be problematic if the customer data contains sensitive private information. Several solutions like noise injection [9, 10], homomorphic encryption, and trusted execution environments [11, 12] have been developed to address this challenge. However, their adoption remains challenging due to the loss of utility and computational overheads associated with these methods. Developing practical solutions that enable privacy-preserving remote inference is crucial for its adoption in domains such as healthcare and finance, where data privacy is essential. Motivated by this need, our research has developed a solution to address these challenges [13]. However, in the interest of space, we restrict the scope of this thesis to focus on training data and model privacy.

1.1.4 Goals of the Thesis

Protecting the privacy of the data used during training and the model used during inference are two key challenges that need to be addressed to make the training and deployment of deep learning models more viable in commercial applications. The objective of this thesis is to study and address the privacy vulnerabilities in the frameworks that are used for privacy-preserving training and inference. To this end, we set the following specific goals:

- Understand the limits of aggregation-based privacy in federated learning by developing gradient inversion attacks to leak private inputs that are used for training.

- Investigate if the gradients shared during two-party split learning can leak private labels by developing a new label leakage attack.
- Understand if access to distributionally similar data is a prerequisite for model stealing by developing a novel data-free model stealing attack.
- Safeguard models from theft by developing new defenses that offer protection against model stealing attack with negligible loss in utility.

1.2 Thesis Statement

Frameworks used for privacy-preserving DNN training and inference are vulnerable to attacks that can compromise the privacy of the training data and the model used for inference. Identifying these vulnerabilities is crucial to understanding the limitations of these frameworks and can aid in designing practical defenses that improve privacy.

1.3 Thesis Contributions

We are interested in investigating the vulnerabilities in privacy-preserving training and inference frameworks that can be exploited to leak private data and models. To this end, we start by studying federated learning and split learning, and design powerful new attacks that overcome the limitations of prior works. Our attacks break the illusion of privacy associated with these privacy-preserving training frameworks and emphasize the need for principled defenses. Next, we study model privacy in the context of remote ML inference. We show that models are susceptible to theft even when the adversary is severely data limited by designing a data-free model stealing attack. Finally, to secure remote ML inference against the threat of model stealing, we design a practical defense that protects against MS attack with negligible degradation in utility in the data-limited setting. We explain each of these contributions in detail below.

1.3.1 Cocktail Party Attack: Breaking aggregation based privacy in FL using ICA

Federated learning (FL) aims to perform privacy-preserving machine learning on distributed data held by multiple data owners. To this end, FL requires the data owners to perform training locally and share the gradient updates (instead of the private inputs) with the central server, which are then securely aggregated over multiple data owners. Aggregation alone does not offer provable privacy protection for the training data, as prior works have demonstrated the recovery of private inputs from the aggregate gradients. However, these attacks have only been demonstrated on small batch sizes, suggesting that aggregation could offer sufficient protection against such attacks if the batch size is sufficiently large.

This thesis proposes the Cocktail Party Attack (CPA) that, contrary to the prior belief, can recover the private inputs from gradients aggregated over a very large batch. CPA leverages the crucial insight that aggregate gradients from a fully connected layer are a linear combination of its inputs, which allows us to frame gradient inversion as a blind source separation (BSS) problem (informally called the cocktail party problem). We adapt independent component analysis (ICA)—a classic solution to the BSS problem—to recover private inputs for fully-connected and convolutional networks and show that CPA significantly outperforms prior gradient inversion attacks, scales to ImageNet-sized inputs, and works on large batch sizes of up to 1024.

1.3.2 ExPLoit: Extracting private labels in split-learning

Split learning performs privacy-preserving training on vertically partitioned data by jointly training a model on the private input and label data held by two parties. To preserve the privacy of the input and label data, this technique uses a split model trained end-to-end by exchanging the intermediate representations (IR) of the inputs and gradients of the IR between the two parties. While the theory indicates that gradients shared during SL should encode the private labels, prior works on label leakage attacks only work in restricted settings like an imbalanced binary classification problem.

We overcome the limitations of prior works by proposing *ExPLoit* – a label-leakage attack that allows an adversarial input-owner to extract the private labels of the label-owner during split-learning. ExPLoit frames the attack as a supervised learning problem by using a novel loss function that combines gradient-matching and several regularization terms developed using key properties of the dataset and models. Our evaluations on a binary conversion prediction task and several multi-class image classification tasks show that ExPLoit can uncover the private labels with near-perfect accuracy of up to 99.53%, demonstrating that split learning provides negligible privacy benefits to the label owner.

1.3.3 MAZE: Data-Free Model Stealing Attack

Model Stealing (MS) attacks allow an adversary with black-box access to a DL model to replicate its functionality by training a clone model using the predictions of the target model for different inputs, presenting a threat to model privacy in remote inference. However, the best available existing MS attacks fail to produce a high-accuracy clone without access to the target dataset or a representative dataset necessary to query the target model, providing a false sense of security.

We show that preventing access to the target dataset is not an adequate defense to protect a model. We propose *MAZE* – a data-free model stealing attack using zeroth-order gradient estimation that produces high-accuracy clones. In contrast to prior works, MAZE uses only synthetic data created using a generative model to perform model stealing. MAZE trains the generator using zeroth order gradient estimates that can be computed from the black box predictions of the target model. Our attack enables high accuracy model stealing on several image classification models without requiring any data, providing a normalized clone accuracy in the range of $0.90\times$ to $0.99\times$, and even outperforms prior works that rely on partial data. The efficacy of our attack highlights the need for defenses that protect against model stealing attacks.

1.3.4 Defending against Model Stealing with Adaptive Misinformation

The threat of model stealing attacks motivates us to study practical defenses that can protect against such attacks during remote inference. We observe that in the data-limited setting, the attacker does not have access to in-distribution examples and instead uses synthetic or surrogate data to query the target model. We leverage this constraint of the attacker to design a practical defense against model stealing attacks.

We propose the *Adaptive Misinformation Defense* to defend against such model stealing attacks in the data-limited setting. We identify that all existing model stealing attacks invariably query the target model with Out-Of-Distribution (OOD) inputs. By selectively sending incorrect predictions for OOD queries, our defense substantially degrades the accuracy of the attacker's clone model (by up to 40%), while minimally impacting the accuracy ($< 0.5\%$) for benign users. Compared to existing defenses, our defense has a significantly better security vs accuracy trade-off and incurs minimal computational overheads.

CHAPTER 2
BACKGROUND

This thesis investigates privacy vulnerabilities in three ML frameworks: federated learning (FL), split learning (SL), and remote inference. This chapter provides background on these frameworks and discusses their privacy risks that can leak model and training data. We also mention the key limitations of prior works that serve as the basis for the research questions addressed by this thesis.

2.1 Federated Learning

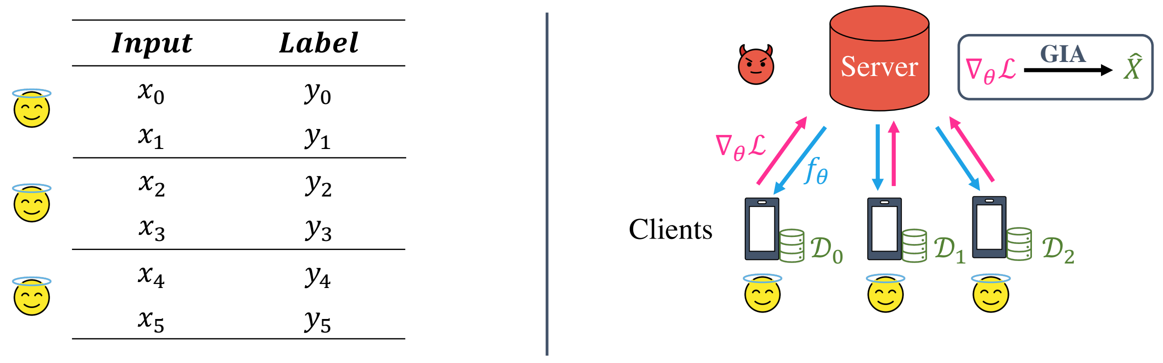
FL is a framework for performing privacy-preserving training on horizontally partitioned data. We first provide background on horizontally partitioned data, explain how FL can be used to perform training on such datasets, and describe the privacy risks involved in FL.

2.1.1 Horizontally Partitioned Data

A dataset is said to be horizontally partitioned if it is divided into subsets that are held by multiple data-owners. To explain, let $\mathcal{D} = \{x_i, y_i\}$ denote a dataset consisting of input, label pairs. Let $\mathcal{D}$ be divided between three data-owners $\mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2$ who each own a subset of data: $\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2$, such that $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_1 \cup \mathcal{D}_2$ as shown in Figure 2.1a. $\mathcal{D}$ is said to be horizontally partitioned between $\mathcal{U}_0, \mathcal{U}_1$ and $\mathcal{U}_2$. The goal of a privacy preserving training framework is to train a model f_θ on this distributed dataset $\mathcal{D}$ while protecting the privacy of the data held by each data-owner.

2.1.2 Using Federated Learning to Train on Horizontally Partitioned Data

FL is a privacy-preserving training framework that can be used to collaboratively train a model on horizontally partitioned data held by multiple clients. FL involves a central server



(a) Horizontally Partitioned Data **(b) Federated learning**

Figure 2.1: (a) A dataset is considered to be horizontally split if each data owner has a different subsets of the points in the dataset. (b) Federated learning is a privacy preserving machine learning framework to train a model collaboratively on a distributed dataset that is horizontally split between multiple clients. FL requires the clients to train the model locally on their private data and only share the model updates (gradients) with a central server.

and multiple clients who hold private subsets of the data $\mathcal{D}_i$ as shown in Figure 2.1b. To train a model f_θ, the server starts by distributing the model to the clients. Each client uses the batch of private data to compute the aggregate gradients of the loss with respect to the model parameters θ and sends the gradient $\nabla_\theta \mathcal{L}$ to the central server. The server collects and aggregates the gradients and uses the aggregated gradient to update the model parameters. This process is repeated until the model converges. FL has been adopted in several commercial applications such as smart keyboards [14] and voice assistants [15] to train models on distributed private data.

2.1.3 Data Privacy Risks

While FL avoids the direct sharing of private data, the aggregate gradient updates $\nabla_\theta \mathcal{L}$ transmitted by the clients during FL could be used to leak private data. This is because the gradients are computed as a function of the inputs and thus are capable of encoding the inputs in some form. A curious central server can use these gradients to potentially recover the private inputs. The process of recovering the private inputs from the aggregate gradient updates shared during FL is termed as *gradient inversion*.

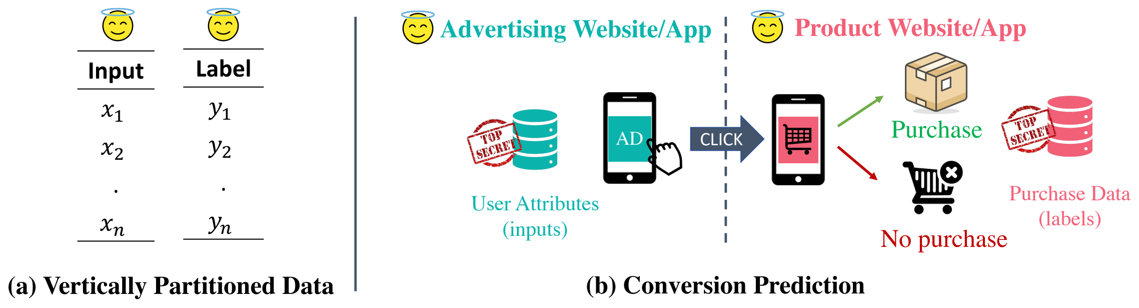


Figure 2.2: (a) A dataset is considered to be vertically partitioned if its features are divided across multiple data-owners. (b) Conversion prediction estimates the likelihood of a purchase when a user clicks on an ad. The training data is vertically partitioned, with the user attributes (inputs) held by the ad company and the purchase data (outputs) held by the product company.

2.1.4 Limitations of Prior Works

Zhu et al. [16] was the first work that demonstrated that it is possible to carry out gradient inversion to leak private data in FL using a *gradient matching* objective. Several follow on works have built on the insight of Zhu et al. to improve upon their results. However, these attacks have primarily been demonstrated on datasets with low-dimensional inputs and for small batch sizes. The efficacy of these attacks deteriorates significantly when gradient aggregation is done with a large number of samples, especially with high-dimensional inputs, limiting their applicability.

2.1.5 Key Research Question

The difficulty of scaling up prior works to large batch sizes has led to the belief that FL might be secure if the gradients are aggregated over a large enough batch size. This limitation of prior works motivates us to ask the following key research question: *Is aggregation alone sufficient to guarantee privacy in federated learning?* Answering this question would improve our understanding of the privacy properties of FL.

2.2 Split Learning

Split learning is a framework used for privacy-preserving training on vertically partitioned data. We first describe vertically partitioned data and the need to train on such data using the motivating example of the conversion prediction task. Next, we explain the framework of split learning and the privacy risks associated with split learning. Finally, we briefly discuss the limitations of prior attacks and the key research question that we set out to answer in our research.

2.2.1 Vertically Partitioned Data

Vertically partitioned data consists of a dataset whose features are partitioned across multiple data owners. In the simplest case, the input features x_i and the corresponding labels y_i can be held by an input-owner and a label-owner respectively as shown in Figure 2.2a. The goal is to train a model in this partitioned dataset $\mathcal{D} = x_i, y_i$ in a privacy-preserving manner. Note that the input features can be subdivided between multiple data-owners, in which case training involves more than two parties. However, we limit the scope of this thesis to the simpler setting where all the input features are held by a single input-owner.

Conversion Prediction: Conversion prediction is a strong motivating example to demonstrate the need for training a model on such vertically split data. Given the attributes of a user and an ad, conversion prediction estimates the likelihood of a user purchasing the product. Conversion prediction is an essential component of ad-ranking algorithms, as ads with a high likelihood of conversion are more relevant to the user and need to be ranked higher. The data required to train the model are split between the advertising and product websites, as depicted in Figure 2.2b. The user attributes, which serve as the inputs, are stored with the advertising company, while the purchase data, which serve as the labels, are held with the product company. The companies are interested in training a model to predict the conversion likelihood while keeping their datasets private.

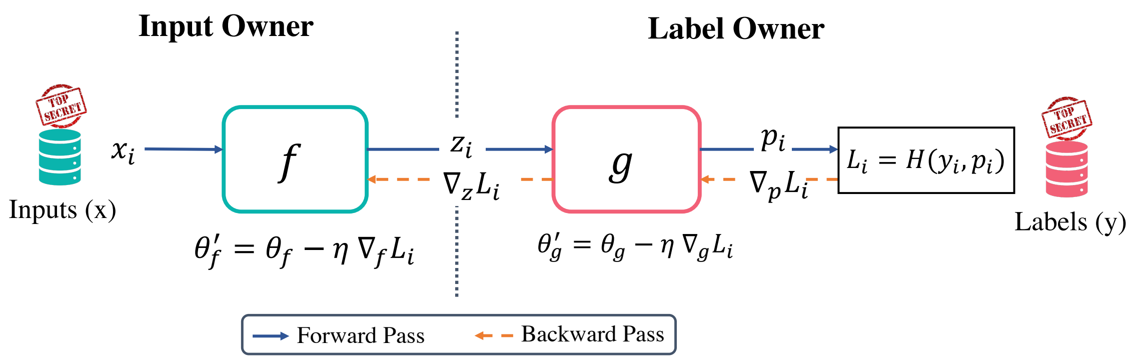


Figure 2.3: Split learning can be used for vertical federated learning by training a composition model $g \circ f$, split between the input and model owner. Split learning aims to preserve the privacy of the input and label owners by exchanging embedding z_i and the gradient $\nabla_z L_i$ data instead of the private input x_i and labels y_i

2.2.2 Using Two-Party Split Learning to train on Vertically Partitioned Data

Two-party split learning can be used to train on data that is vertically partitioned between the input and label owner, without having to share the private data. The input owner owns the inputs $\mathcal{D}_{inp} = \{x_i\}$ and the label owner owns the labels $\mathcal{D}_{label} = \{y_i\}$, corresponding to each input. Split learning uses a composition model $g \circ f$ that is distributed between the two parties to learn on the vertically partitioned data. A single training iteration involves a forward and a backward pass (as shown in Figure 2.3), which proceeds as follows:

- *Forward pass:* The input owner samples a batch of inputs $\{x\}_{batch} \sim \mathcal{D}_{inp}$ and performs forward propagation through $f : \mathcal{X} \rightarrow \mathcal{Z}$ and produces the corresponding embeddings $\{z\}_{batch}$. These embeddings, along with the corresponding *inputIDs* are sent to the label owner. The label owner feeds the embeddings to $g : \mathcal{Z} \rightarrow \mathcal{Y}$ to produce the predictions $\{p\}_{batch}$, which along with the labels $\{y\}_{batch}$ are used to compute the model's loss $L = \mathbb{E}[H(y,p)]$.

- *Backward pass:* The label owner initiates backpropagation and returns the loss gradient $\{\nabla_z L\}_{batch}$ to the input owner. Both the label and input owner compute the gradient of the loss with respect to the model parameters and update model parameters using gradient descent as shown below:

$$\theta_g^{t+1} = \theta_g^t - \eta \nabla_{\theta_g} L; \;\; \theta_f^{t+1} = \theta_f^t - \eta \nabla_{\theta_f} L. \tag{2.1}$$

2.2.3 Data Privacy Risks

The gradients and embeddings shared during SL have the potential to encode private training data. Specifically, SL carries the risk of leaking training data in the following ways:

1. *Input Leakage:* A curious label-owner can potentially infer the input owner's private inputs $\{x_i\}$ from the gradients shared during SL, compromising input privacy.

2. *Label Leakage:* A curious input-owner can potentially infer the label owner's private labels $\{y_i\}$ from the embeddings shared during SL, compromising label privacy.

We restrict the scope of this thesis to focus on label privacy. We term an attack that leaks private labels from the gradients obtained during SL as a *label leakage attack*

2.2.4 Limitations of Prior Works

Label privacy in split learning has received limited attention. A recent work [17] proposed a label leakage attack specifically for the conversion prediction task, where there is a high class imbalance, by leveraging the correlation between the gradient norms and the class label. However, their attack only works for binary classification problems with high class imbalance and is not applicable to the general setting of multi-class classification problems with balanced class distribution.

2.2.5 Key Research Question

The limitations of prior works motivate us ask the following question: *Can high-accuracy label leakage be carried out for multi-class classification problems with a balanced class* *distribution?* Answering this question will help us understand if split learning can protect against label leakage in the general setting of a multi-class classification problem.

2.3 Remote Inference

The remote inference framework uses a client-server architecture to offer ML inference services to end-users. To explain, consider a model owner with a private model f_θ that is hosted as a remote inference service to perform a classification task. To avail the service, the end-user sends an input x to the model owner. The model owner uses this input to perform inference and returns the model's prediction $p = f_\theta(x)$ to the end-user. Depending on the service agreement, the returned prediction can be either soft-label (softmax probabilities for all classes) or hard-label (top-1 class label). The black-box nature of remote inference prevents the end user from directly accessing the model architecture and parameters, ostensibly providing privacy to the model owner.

2.3.1 Model Privacy Risks

Several recent works [6, 5] have proposed *model stealing attacks* to create a functionally equivalent clone model, just using the black-box predictions of a target model hosted as a remote inference service. Such attacks are carried out by querying the model with a *surrogate dataset* that is *semantically similar* to the one that was used to train the target model. The predictions from these queries are then used to perform knowledge distillation to train a functionally equivalent clone model $f'_{\theta'}$. The success of these attacks demonstrates that the predictions of the model on semantically similar data are capable of encoding enough information to replicate the functionality of the target model. Thus, remote inference is incapable of protecting model privacy under the presence of a semantically similar dataset.

2.3.2 Limitations of Prior Works

The availability of a semantically similar surrogate dataset plays a vital role in the efficacy of model stealing attacks. However, such a dataset may not always be available to the attacker. For instance, it is hard to procure data in several domains like finance and medicine, where data is considered sensitive and is heavily guarded. Under such settings, the efficacy of model stealing attacks degrades significantly if the surrogate dataset is not sufficiently similar to the dataset used to train the target model.

Additionally, in settings where a semantically similar dataset is available, the susceptibility of remote inference to model stealing emphasizes the need for a defense to protect model privacy. Prior works [8, 7] have proposed to defend against model stealing by perturbing the model predictions. However, indiscriminately perturbing the predictions causes the model's utility to degrade significantly, making these defenses impractical to adopt.

2.3.3 Key Research Questions

The limitations of prior works motivate us to ask the following research questions:

1. *Does remote inference protect model privacy when a semantically similar dataset is unavailable to an attacker?*

2. *Is it possible to defend against model stealing attacks, while preserving the utility of the model for benign users?*

Answering these questions will help us understand the limitations of remote inference and design defenses that can improve model privacy.

CHAPTER 3
COCKTAIL PARTY ATTACK: BREAKING AGGREGATION BASED PRIVACY IN FEDERATED LEARNING USING INDEPENDENT COMPONENT ANALYSIS

In this chapter, we investigate the privacy vulnerabilities in federated learning that can be used to carry out gradient inversion attacks to leak private training data from aggregate gradients. Specifically, we develop a gradient inversion attack that can recover private inputs from aggregate gradients, proving that even with large batch sizes, gradient aggregation alone does not provide privacy and is insufficient to protect against gradient inversion.

3.1 Introduction

Federated learning (FL) is a powerful and flexible framework for privacy-preserving machine learning (ML) model training on distributed data. The FL framework typically consists of a central server and multiple clients that hold private training data. The protocol involves the server distributing the model parameters θ to the clients, and then the clients use this model to compute gradient update $\nabla_\theta \mathcal{L}$ using their private data as shown in Figure 2.1. The gradient updates are aggregated and shared with the server who updates the model parameters using the aggregated gradients and this process is repeated until convergence. While FL avoids the direct sharing of data, this in itself does not guarantee privacy as the gradient update shared with the server can contain information about the private training data. For instance, Zhu et al. [16] showed concretely that these gradient updates can be *inverted* to recover their associated private data in a process now called *gradient inversion*. In realistic settings, however, gradient inversion in FL is considered to be hard [16], as gradients are aggregated across a large number of inputs (e.g., 1024), which thwarts most existing gradient inversion attacks.

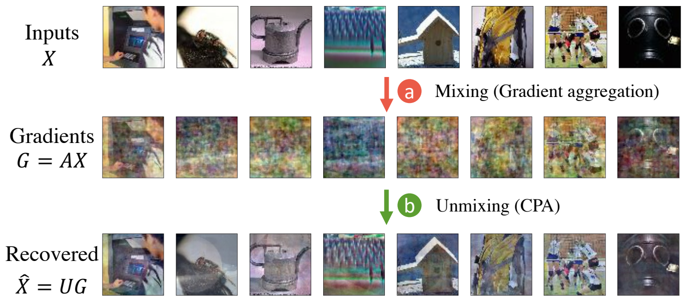


Figure 3.1: (a) Aggregate gradients from an FC layer are linear combinations of its inputs, as demonstrated in the visualization of the gradients in the second row. (b) *Cocktail party attack* uses this observation to frame gradient inversion as a blind source separation problem and recovers the inputs from the gradients by optimizing an unmixing matrix U using independent component analysis.

Our goal is to show that this hardness is not an inherent privacy property of FL, but rather a technical limitation of the existing attack algorithms. To this end, we propose the *Cocktail Party Attack (CPA)*—a gradient inversion attack that is able to recover the inputs to a fully-connected (FC) layer from its aggregated gradient. This is made possible by a novel insight that the aggregated gradient for an FC layer is a linear combination of its inputs, which allows us to frame the recovery of these inputs as a *blind source separation* (BSS) problem and solve it using *independent component analysis*.

When applied to a fully-connected network (or any network where the first layer is an FC layer), CPA can readily perform gradient inversion to recover input images. Figure 3.1 shows an illustration of CPA on a fully-connected network, where a batch of training images is faithfully recovered from its aggregated gradient. We further extend CPA to perform gradient inversion on convolutional networks by first recovering the per-sample embeddings to an FC layer of the network and then inverting these embeddings using feature inversion to recover the input images. Empirically, we show that CPA has the following advantages over prior work:

- We evaluate CPA by inverting the gradients from an FC model trained on CIFAR-10 and Tiny-Imagenet, and a VGG-16 model trained on ImageNet to show that CPA can perform high-quality recovery of private inputs even with batch size as large as 1024.

- Compared to prior work based on gradient matching, CPA can recover inputs with better quality and scales to datasets with larger input sizes (e.g., ImageNet). Furthermore, we show that gradient matching can be combined with CPA to further improve attack performance.

- CPA only uses simple image priors such as smoothness and does not require knowledge of the input data distribution or changes to the model parameters, and hence is more versatile and applicable to real world settings.

The efficacy of CPA shows that aggregation alone does not provide meaningful privacy guarantees and defenses like differential privacy are truly necessary to prevent gradients from leaking private data in FL.

3.2 Background

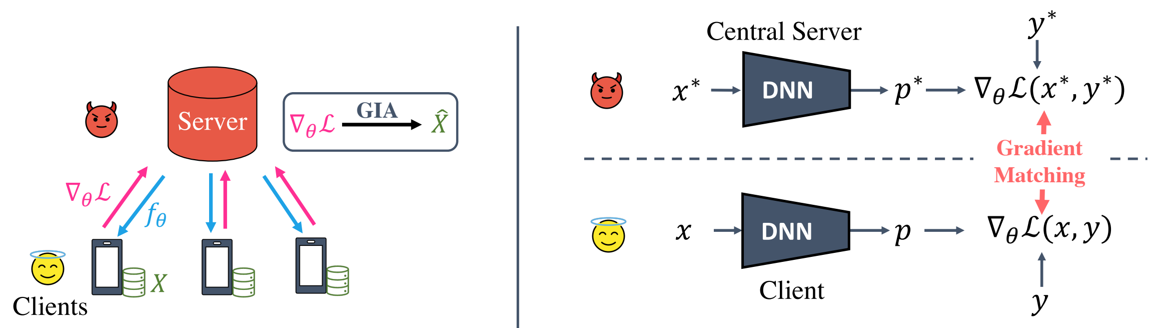
This section provides background on gradient inversion attacks (GIA) in FL. We also provide an overview of prior works on GIA and describe their limitations, which serves as the motivation for our attack.

3.2.1 Gradient Inversion Attack

The goal of a gradient inversion attack is to recover the inputs from the aggregate gradients produced during training. Such attacks can be used by a server to leak the private inputs of the clients in FL. We explain the attack objective and constraints below.

Attack Objective: Let $\mathcal{A}$ denote the gradient inversion attack. The goal of $\mathcal{A}$ is to estimate the batch of inputs $\hat{X}$ from the aggregate gradient $\nabla_\theta \mathcal{L}(X, Y)$, such that the estimated inputs $\hat{X}$ are semantically similar to the original inputs X as shown in Equation 3.1.

$$\min d(\hat{X}, X), \text{where } \hat{X} = \mathcal{A}(\nabla_\theta \mathcal{L}(X, Y)) \tag{3.1}$$


Server
$\nabla_\theta \mathcal{L}$ $\xrightarrow{\text{GIA}}$ $\hat{X}$
$\nabla_\theta \mathcal{L}$
f_θ
X
Clients
Central Server
y^*
x^*
DNN
p^*
$\nabla_\theta \mathcal{L}(x^*, y^*)$
Gradient Matching
x
DNN
p
$\nabla_\theta \mathcal{L}(x, y)$
Client
y

(a) Federated learning

(b) Gradient inversion attack using gradient matching (prior work)

Figure 3.2: (a) FL aims to perform privacy preserving ML on distributed data by requiring the clients to perform training locally on their private data X and send the weight updates/gradients $\nabla_\theta \mathcal{L}$ to a central server. (b) Gradient inversion attacks (GIA) break privacy by recovering the private data from the gradients. Prior works carry out GIA by optimizing a set of dummy parameters (x^*, y^*) with a gradient matching objective.

Attack Constraints: We assume an honest-but-curious adversary, meaning that the central server is not allowed to modify the FL protocol or insert malicious weights [18, 19] to achieve the attack objective.

3.2.2 Related Work

Most prior works on gradient inversion primarily rely on the *gradient matching* objective to carry out the attack. We start by describing gradient matching, followed by attacks that build on this objective using various input priors like total variation prior, batch-norm statistics, and generative image priors to improve attack performance. We also discuss a recent line of work that uses malicious model parameters to carry out GIA. We describe the limitations of these prior works and also discuss the key challenge in scaling gradient matching based attacks to datasets with large inputs.

Gradient Matching: Gradient matching [16] performs gradient inversion by optimizing a batch of dummy inputs and labels (x^*, y^*) to produce a gradient that matches the one received by the server during FL as shown in Figure 3.2b. This can be done by minimizing the distance between the gradient produced by the dummy variables $\mathcal{L}_\theta(x^*, y^*)$ and the gradient received during FL $\mathcal{L}_\theta(x, y)$ as shown in Equation 3.2. This method was shown to work well on small datasets like CIFAR-10 [20] up to a batch size of 8.

$$\hat{x}, \hat{y} = \arg\min_{x^*, y^*} d(\mathcal{L}_\theta(x^*, y^*), \mathcal{L}_\theta(x, y)) \tag{3.2}$$

Here, d denotes a distance metric between vectors like cosine similarity or L_2 norm. A subsequent work [21] proposed a method to infer the ground truth labels by examining the gradients of the last layer. Knowing the labels can help improve the quality of gradient inversion. However, this method only works when no two inputs in the batch belong to the same output class, limiting its applicability.

Total Variation (TV) Prior: Geiping et al. [22] proposed to use the TV prior [23] as a regularization term along with the gradient matching objective. The TV prior (Equation 3.3) penalizes high-frequency components in the input and encourages the optimization to find natural-looking images. With the TV prior, gradient matching can be scaled to work on a batch size of up to 100 for CIFAR-10 images and up to a small number of inputs for ImageNet.

$$\mathcal{R}_{TV}(x^*) = \mathbb{E}\big[|x^*_{i+1,j} - x^*_{ij}|\big] + \mathbb{E}\big[|x^*_{i,j+1} - x^*_{ij}|\big] \tag{3.3}$$

Batch Norm Statistics: Several works [24, 25] have proposed to use the mean and variance of the activations captured by the batch norm (BN) layers as a prior to improve gradient inversion using the regularization term in Equation 3.4. Here, $\mu_l(x^*)$ and $\sigma_l^2(x^*)$ represent the mean and variance of the activation produced by the input x^* at the l-th BN layer. This regularization term encourages the optimization to find inputs that produce activations whose distribution matches the BN statistics. Evaluations from this work show that the BN prior can enable gradient inversion on ImageNet with a batch size of up to 48 examples.

$$\mathcal{R}_{BN}(x^*) = \mathbb{E}_l\Big[||\mu_l(x^*) - BN_l(mean)||_2 + ||\sigma_l^2(x^*) - BN_l(var)||_2\Big] \tag{3.4}$$

A key limitation of this work is that it can only be used for models that use BN layers. In a real-world FL, the model might not contain BN layers because BN layers often degrades accuracy with a non-IID data [26], which is common in FL [27]. Furthermore, BN statistics can be used to perform model inversion to leak the training data directly from the model parameters [28] (without the need for gradients), posing a concern about the premise of using networks with BN layers to train on private data.

Generative Image Prior: Instead of performing optimization in the space of inputs, a recent work [29] proposes to move the optimization to the smaller latent space of a generative model (G) to find an input $x^* = G(z^*)$ that satisfies the gradient matching objective as shown in Equation 3.5. The reduced space of optimization and the prior induced by the generative model helps the attack scale to imagenet-scale datasets with a small batch size.

$$\hat{z}, \hat{y} = \underset{z^* \in \mathbb{R}^k, y^*}{\arg\min}\, d(\mathcal{L}_\theta(G(z^*), y^*), \mathcal{L}_\theta(x, y)) \tag{3.5}$$

Note that this method requires the adversary to have access to in-distribution data or a generative model that is trained on in-distribution data. This may not be realistic in several settings (e.g. medicine and finance), where in-distribution data/generative model may not be available. Additionally, such methods may not work well under dataset shift.

Malicious Parameters: Several recent works [18, 19, 30] have proposed methods to leak private inputs by using malicious model parameters under the assumption of a dishonest central server. Such methods use weights that cause the aggregate gradient or the difference between two aggregate gradients to be predominantly influenced by a single input. However, under our threat model of an honest central server, malicious modifications to model parameters are not allowed. Thus we do not consider such attacks in our work.

3.2.3 Challenges for Scaling to Large Input Sizes

The optimization complexity of gradient matching poses a fundamental limitation to its scalability. Most prior works (with the exception of generative image prior) perform optimization directly in the input-space. The size of this optimization problem is $\mathcal{O}(n \times d_{in})$, where n is the batch size and d_{in} denotes the dimensionality of the input. The difficulty of performing this optimization scales linearly with the dimensionality of the input, preventing gradient-matching from scaling to high-dimensional inputs.

In contrast, our proposed attack (CPA) has an optimization complexity of $\mathcal{O}(n \times n)$, which is independent of the input-dimensionality. This unique feature keeps the size of the optimization problem small and enables CPA to scale to ImageNet-sized datasets even with large batch sizes.

3.3 Cocktail Party Attack

We propose CPA – a gradient-inversion attack specifically for full-connected (FC) layers. CPA frames gradient inversion as a blind source separation (BSS) problem and uses independent component analysis (ICA) to recover the private inputs from aggregate gradients. We start by providing background on the BSS problem and describe how gradient inversion in FC layers can be viewed as a BSS problem. Next, we describe ICA–a classic signal processing technique– that can be used to solve the the BSS problem. Lastly, we consider the problem of performing gradient inversion for FC models trained on image data, and describe how CPA can be used directly with simple image prior to perform gradient inversion and leak the private inputs. While FC models are typically not used for image classification, it represents the simplest setting to describe our attack. In the next section, we extend our attack to work on CNN models.

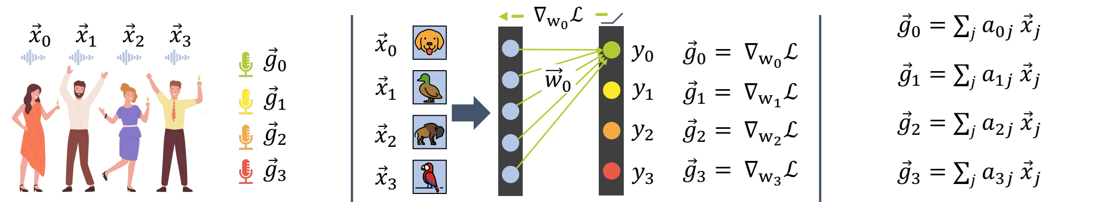

$\vec{x}_0$ $\vec{x}_1$ $\vec{x}_2$ $\vec{x}_3$
$\vec{g}_0$
$\vec{g}_1$
$\vec{g}_2$
$\vec{g}_3$
$\nabla_{w_0}\mathcal{L}$
$\vec{w}_0$
y_0 $\vec{g}_0 = \nabla_{w_0}\mathcal{L}$
y_1 $\vec{g}_1 = \nabla_{w_1}\mathcal{L}$
y_2 $\vec{g}_2 = \nabla_{w_2}\mathcal{L}$
y_3 $\vec{g}_3 = \nabla_{w_3}\mathcal{L}$
$\vec{g}_0 = \sum_j a_{0j}\,\vec{x}_j$
$\vec{g}_1 = \sum_j a_{1j}\,\vec{x}_j$
$\vec{g}_2 = \sum_j a_{2j}\,\vec{x}_j$
$\vec{g}_3 = \sum_j a_{3j}\,\vec{x}_j$
(a) Microphone recordings in a cocktail party (b) Gradients from a fully connected layer (c) Linear combinations of inputs

Figure 3.3: The microphone recordings in the cocktail party problem and the gradients from a fully connected layer can both be represented as linear combination of inputs. Recovering the inputs in both cases can be viewed as a blind source separation problem.

3.3.1 Framing Gradient Inversion as a Blind Source Separation Problem

CPA frames gradient inversion as a BSS problem. We explain the BSS problem using the motivating example of the cocktail party problem and show that gradient inversion for an FC layer can also be viewed as a BSS problem.

Cocktail Party Problem: Consider a cocktail party where there are a group of four people talking simultaneously as depicted in Figure 3.3a. A microphone placed near this group picks up an audio recording consisting of an overlapping set of voices from the four speakers. We can model this recording by viewing it as a linear combination of the voices of the four speakers. More formally, if $\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x}_3$ denote the speech vectors of the four speakers and $\vec{g}_0, \vec{g}_1, \vec{g}_2, \vec{g}_3$ denotes the recordings of four microphones places at different points, the recording of the i-th microphone is given by:

$$\vec{g}_i = a_{i0}\vec{x}_0 + a_{i1}\vec{x}_1 + a_{i2}\vec{x}_2 + a_{i3}\vec{x}_3 \tag{3.6}$$

Here, a_{ij} denote the unknown mixing coefficients. The BSS problem can be stated as follows: given the mixed signals $\{\vec{g}_i\}$, recover the individual source signals $\{\vec{x}_i\}$.

Gradient Inversion for FC Layer: Much like the cocktail party problem, the aggregate gradients from an FC layer can be viewed as linear combinations of the inputs used to generate them. To demonstrate this, consider an FC layer with four hidden neurons y_0, y_1, y_2, y_3 as shown in Figure 3.3b. Let $\vec{w}_0, \vec{w}_1, \vec{w}_2, \vec{w}_3$ represent the weight vectors associated with each output neuron. Let $\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x}_3$ be a batch of four inputs used to perform a single iteration of training. $\vec{g}_i = \nabla_{w_i}\mathcal{L}$ is the aggregate gradient of the loss with respect to $\vec{w}_i$, which is computed by taking the mean of the individual gradients $\nabla_{w_i}\mathcal{L}^j$ associated with each input $\vec{x}_j$. This aggregate gradient $\vec{g}_i$ can further be expressed as a linear combination of the inputs $\vec{x}_j$ as follows:

$$\vec{g}_i = \frac{1}{4}\sum_j \nabla_{w_0}\mathcal{L}^j = \frac{1}{4}\sum_j \frac{\partial\mathcal{L}}{\partial y_i^j}\frac{\partial y_i^j}{\partial w_i} = \frac{1}{4}\sum_j \frac{\partial\mathcal{L}}{\partial y_i^j}\vec{x}_j \tag{3.7}$$

Notice that the aggregate gradients here have a 1-to-1 correspondence with the cocktail party problem. The inputs here are similar to the speakers and gradients are similar to the recordings of the microphones. Recovering the inputs $\{\vec{x}_i\}$ (source signals) from a set of aggregate gradients $\{\vec{g}_i\}$ (mixed signals) can thus be viewed as a BSS problem.

Solving BSS with an Unmixing Matrix: Eqn. Equation 3.7 can be expressed as a matrix multiplicaion operation as follows: $G = AX$. The rows of $X \in \mathbb{R}^{n\times d}$ denote the inputs (source signals), rows of the $A \in \mathbb{R}^{n\times n}$ represent the coefficients of linear combination and rows of the $G \in \mathbb{R}^{n\times d}$ denote the gradients (aggregate signals). We can estimate the source matrix $\hat{X}$ from G by using an *unmixing matrix* U as follows: $\hat{X} = UG$. Each row of $\hat{X}$ represents a single recovered source signal $\hat{x}_i$. Note that X can be recovered perfectly if $U = A^{-1}$ i.e. $\hat{X} \to X$ as $U \to A^{-1}$. Estimating $\hat{X}$ can thus be reduced to finding the unmixing matrix U.

3.3.2 Gradient Inversion using ICA

Independent component analysis (ICA) is a classic signal processing technique that can be used to solve the BSS problem by estimating the unmixing matrix U. To do this, ICA starts with a randomly initialized unmixing matrix U^* and optimizes it to enforce certain properties on the recovered source signals. To explain, let $x_i^* = u_i^* G$ denote the i-th source signal recovered from multiplying the i-th row of U^* with G. Note that the source signals represent images in our case (Figure 3.3b). ICA optimizes U^* so that the recovered source signals $\{x_i^*\}$ satisfy the following key properties:

- *Non-Gaussianity:* Values of real-world signals such as images and speech typically do not follow a Gaussian distribution. We can measure non-Gaussianity using the negentropy metric, which can be estimated using Eqn. Equation 3.8 [31]. A high value of negentropy indicates a high degree of non-Gaussianity.

$$J(x^*) = \mathbb{E}\Big[\frac{1}{a^2}\log\cosh^2(ax_i^*)\Big]. \tag{3.8}$$

- *Mutual Independence (MI):* We assume that the source signals are independently chosen and thus their values are uncorrelated. Since each row u_i^* of the unmixing matrix corresponds to a recovered source signal x_i^*, we can enforce MI by minimizing the absolute pairwise cosine similarity between the rows of U^*.

$$\mathcal{R}_{MI} = \mathop{\mathbb{E}}_{i\neq j} exp\Big(T|CS(u_i^*, u_j^*)|\Big). \tag{3.9}$$

- *Source Prior:* Any prior information about the source signals can be included in our optimization in the form of an additional regularization term $\mathcal{R}_P$.

We find the unmixing matrix U by solving an optimization problem that combines the above properties.[1]

$$U = \arg\max_{U^*} \mathop{\mathbb{E}}_{i}\Big[J(u_i^*G) - \lambda_P\mathcal{R}_P(u_i^*G))\Big] - \lambda_{MI}\mathcal{R}_{MI} \tag{3.10}$$

The U matrix obtained from solving Eqn. Equation 3.10 can be used to estimate the source matrix as follows: $\hat{X} = UG$.

[1]We whiten and center the gradients before using it in our optimization. These are standard pre-processing steps used in ICA.

3.3.3 CPA for FC Models

For an FC model trained on image data, where the inputs are directly fed to an FC layer, we can recover the inputs directly by inverting the gradients of the first FC layer using CPA. Since the source signals are images, we can use TV regularization $\mathcal{R}_{TV}$ (Eqn. Equation 3.3) as the source prior in Eqn. Equation 3.10. One caveat of ICA is that it does not recover the sign of the input (i.e. $\hat{x}_i$ can be a sign-inverted version of x_i). However, this can be easily resolved by selecting between $\hat{x}_i$ and $-\hat{x}_i$ through a visual comparison.

3.4 Extending Cocktail Party Attack to CNNs

Since our attack is only applicable to FC layers, CPA cannot be used directly on CNN models, where the input is first passed through convolutional layers before being fed to a FC layer. However, we can use CPA in composition with a feature inversion attack (FIA) to recover the inputs (as shown in Figure 3.4) with the following two-step procedure:

1. Use CPA on the FC layer to recover the embeddings from the gradients.

2. Use FIA to estimate the inputs from the embeddings.

We describe these two steps in greater detail below.

3.4.1 Leaking Private Embeddings using CPA

The gradients from the FC layer can be viewed as linear combinations of the embeddings z that act as the input to the FC layer. We can use CPA to invert the gradients from the FC layer (G) and recover an estimate of the embeddings $\hat{z} = UG$. However, we can no longer use TV prior in our optimization (Eqn. Equation 3.6) to find U since the signal being recovered (z) is not an image. Instead, we use the following properties of embeddings produced with a ReLU non-linearity to design regularization terms for our optimization:

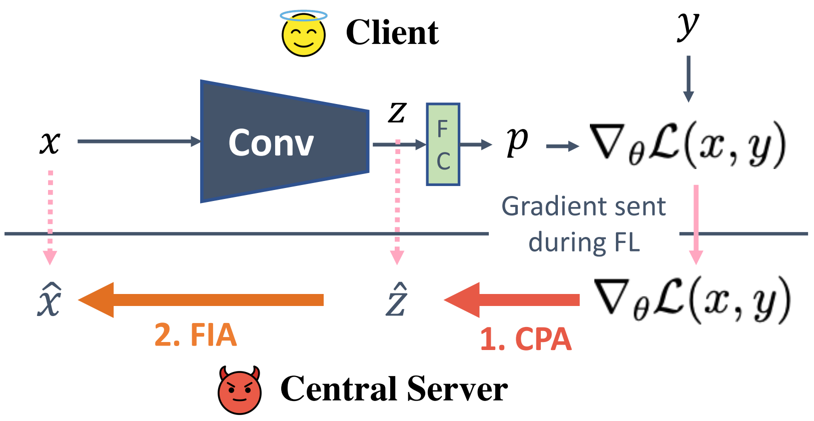


Figure 3.4: We attack CNN models by first recovering the private embedding from the FC layer using CPA and then using a feature inversion attack (FIA) to recover the input from these embeddings.

- *z is sparse:* Embeddings produced by networks that use ReLU non-linearity are sparse, as ReLU squashes negative activations to 0. We can use the L_1-norm: $|z^*|_1$ in our optimization to encourage sparsity of the estimated embeddings.

- *z is a non-negative vector:* Values of the embedding vector are non-negative as ReLU truncates negative inputs to 0. We can minimize the following regularization function $\mathcal{R}_{NN}(z^*) = ReLU(-z^*)$ to encourage z^* to be non-negative. However, the embedding recovered by ICA can be sign inverted. In this sign-inverted setting, the embedding can be non-positive vector. To allow for sign inverted recovery, we propose the *sign regularization* (SR) function: $\mathcal{R}_{SR} = \min(ReLU(z^*), ReLU(-z^*))$. Minimizing $\mathcal{R}_{SR}$ ensures that z^* is either non-negative or non-positive.

We combine the above regularization terms with the non-Gaussianity and mutual indepencence assumptions to derive the final optimization objective to estimate the unmixing matrix U as follows:

$$\begin{aligned} U = \arg\max_{U^*} \mathop{\mathbb{E}}_{i} \Big[& J(u_i^* G) - \lambda_{SP} |u_i^* G|_1 \\ & - \lambda_{SR} \mathcal{R}_{SR}(u_i^* G) \Big] - \lambda_{MI} \mathcal{R}_{MI}. \end{aligned} \tag{3.11}$$

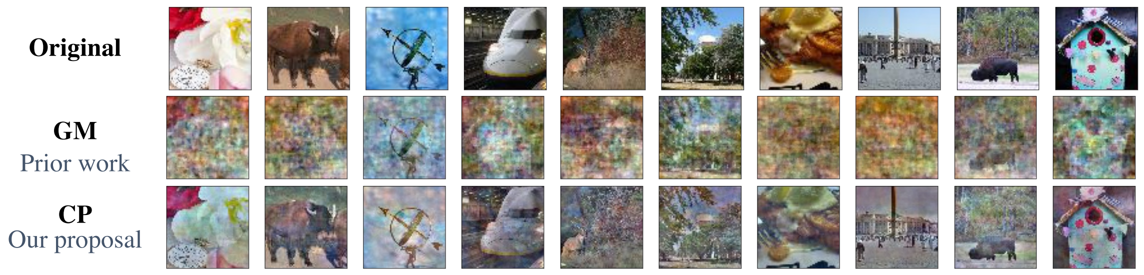


Figure 3.5: Comparison of a subset of images recovered from gradient matching (GM) and cocktail party (CP) attacks by inverting the gradients from the FC-2 network with a batch of 64 Tiny-ImageNet inputs. The quality of images recovered by CP is significantly better than the GM attack.

The unmixing matrix can be used to recover the private embeddings $\hat{Z}$ from the gradient G as follows: $\hat{Z} = UG$. We can use these private embeddings to recover the input using a feature inversion attack.

3.4.2 Feature Inversion Attack

FIA inverts the embedding produced by a neural network to recover the input. Formally, given a network $f : X \to Z$, and an embedding x, FIA recovers an estimate of the input $\hat{x}$ from z. We do this by solving the following optimization problem:

$$\hat{x} = \arg\max_{x^*} CS(f(x^*), z) - \lambda_{TV}\mathcal{R}_{TV}(x^*) \tag{3.12}$$

The first term maximizes the cosine similarity between the embedding from the dummy input $z^* = f(x^*)$ and the true embedding z. The second term is TV regularization, which suppresses high-frequency components. Solving this optimization problem allows us to estimate the private inputs $\{\hat{x}_i\}$ from the embedding $\{\hat{z}_i\}$ recovered by CPA, which completes the gradient inversion attack. Additionally, we can also use the gradient information to improve FIA by including the gradient matching objective in the optimization as follows:

$$\begin{aligned}\hat{x} = \arg\max_{x^*} CS(f(x^*), z) &+ \lambda_{GM} CS(\nabla_\theta \mathcal{L}(x^*), \nabla_\theta \mathcal{L}(x)) \\ &- \lambda_{TV}\mathcal{R}_{TV}(x^*).\end{aligned} \tag{3.13}$$

3.5 Experiments

To demonstrate the efficacy of our attack, we evaluate our proposed attack on FC and CNN models trained on image classification tasks. While FC networks are typically not used for image classification, they allow us to demonstrate the efficacy of our attack in its simplest form. Our evaluations on the CNN model (VGG-16) demonstrates the utility of our attack in a more realistic problem setting.

3.5.1 Setup

Model and Datasets: For our experiments on the FC model, we use a simple 2-layer network (FC-2), with the following network architecture: $[Linear(256) - ReLU() - Linear(k)]$ for a k-class classification problem. We train FC-2 on the CIFAR-10 [20] and Tiny-ImageNet datasets for 20 epochs using the Adam [32] optimizer with a learning rate of 0.001. We perform our CNN experiments with a pre-trained VGG-16 network and the ImageNet dataset.

Evaluation Methodology: We evaluate gradient inversion attacks with the following batch sizes: $[8, 16, 32, 64, 128, 256]$ for the FC-2 model and $[32, 64, 128, 256, 512, 1024]$ for VGG-16. We perform evaluations by first sampling a batch of inputs $\{x_i\}$ from an unseen test set to generate the aggregate gradient $\nabla_\theta \mathcal{L}$. We then use different gradient inversion attacks to recover an estimate of the inputs $\{\hat{x}_i\}$ from the aggregate gradients and compare their performance. Since our experiments are on image data, we use the LPIPS score [33] to quantify the perceptual similarity between the original and recovered images, and use this to evaluate the attacks. We repeat the attack on 5 batches of data and report the average LPIPS scores in our results. We set the number of optimization rounds for all attacks to 25,000.

Hyperparameter Tuning: For all the hyperparameters ($\lambda_{TV}, \lambda_{MI}, T, \lambda_{SP}, \lambda_{SR}$), we sweep their values in the range $[0.00001, 10]$ using a single batch of inputs and pick the set

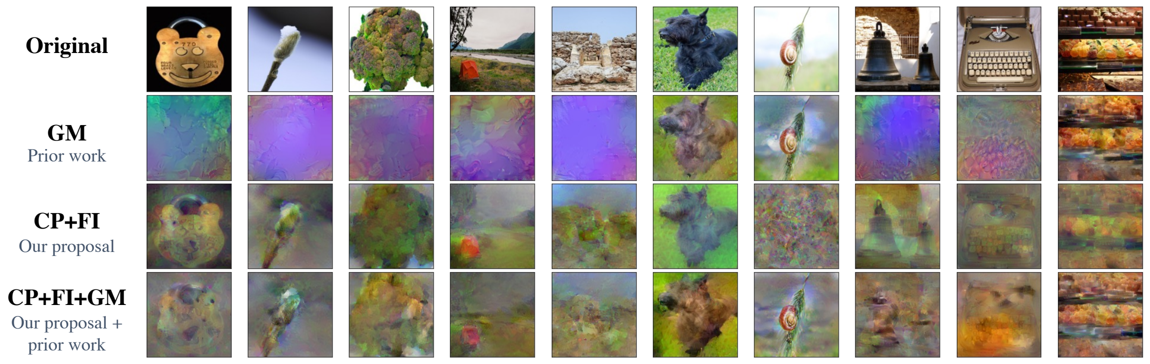


Figure 3.6: Comparison of a subset of images recovered from gradient matching (GM), cocktail party + feature inversion (CP+FI) and cocktail party + feature inversion + gradient matching (CP+FI+GM) by inverting the gradients from a VGG-16 network with a batch of 256 ImageNet inputs. CP+FI (our proposal) can recover more images compared to GM (prior work). CP+FI+GM improves the quality of recovered images by combining the benefits of our proposal and prior work.

of values that yield the best LPIPS score to carry out our attack. Note that the inputs used in the hyperparameter sweep are separate from the ones used to report our results. **Prior Work for Baseline:** Our threat model assumes an honest-but-curious attacker who does not have access to in-distribution examples[2]. The Geiping et al. [22] attack (which uses the gradient matching objective and TV prior) represents the stongest prior work under this setting[3]. We use this prior work (with the best choice of hyperparameters) as the baseline in our evaluations.

3.5.2 Results for FC-2

We first present the results from our experiments on the FC-2 models trained on the CIFAR-10 and Tiny-ImageNet datasets. Figure 3.5 shows qualitative results comparing CPA and GMA for Tiny-ImageNet with a batch size of 64. The images recovered by CPA have better quality and higher perceptual similarity with the original images, compared to the images recovered by GMA. Table 3.1 shows quantitative results (LPIPS scores) comparing CPA and GMA with various batch sizes. A lower LPIPS value indicates better perceptual

[2]Except a single batch of inputs used to tune the hyperparameters

[3]Attacks that use generative models [29] assume access to in-distribution data and cannot be used under our threat model

Table 3.1: $LPIPS \downarrow$ scores comparing the performance of cocktail party (CP) and gradient matching (GM) attacks on FC-2 trained on CIFAR-10 and Tiny-ImageNet. CP (our proposal) significantly outperforms GM (prior work) across all batch sizes.

Attack	**Batch Size**					
	8	16	32	64	128	256
	CIFAR-10					
GM	0.283	0.390	0.491	0.569	0.610	0.614
CP	**0.101**	**0.160**	**0.197**	**0.352**	**0.521**	**0.610**
	Tiny-ImageNet					
GM	0.182	0.234	0.368	0.620	0.687	0.720
CP	**0.082**	**0.143**	**0.164**	**0.217**	**0.232**	**0.388**

similarity and thus a better attack performance. Our result can be interpreted by considering the size of the optimization problem being solved by CPA and GMA.

- *CPA* has an optimization complexity $\mathcal{O}(n \times n)$, as it is optimizing over V^*, which is an $n \times n$ matrix.
- *GMA* has an optimization complexity $\mathcal{O}(n \times d)$ as it is optimizing directly in the input space.

Here, n denotes batch size and d denotes the dimensionality of the input ($d = 3072$ for CIFAR-10 and $d = 12288$ for Tiny-ImageNet). With this in mind, we make the following key observations from our results:

- *Comparison with prior work:* CPA significantly outperforms GMA across all batches sizes since the size of the optimization problem is much smaller for CPA compared to GMA. E.g. for n=64 with Tiny-ImageNet (d=12288), the size of the optimization is 4096 for CPA and 786432 for GMA.
- *Sensitivity to batch size:* The size of the optimization problem increases with an increase in batch size for both CPA and GMA causing their performance to degrade for larger batch sizes.

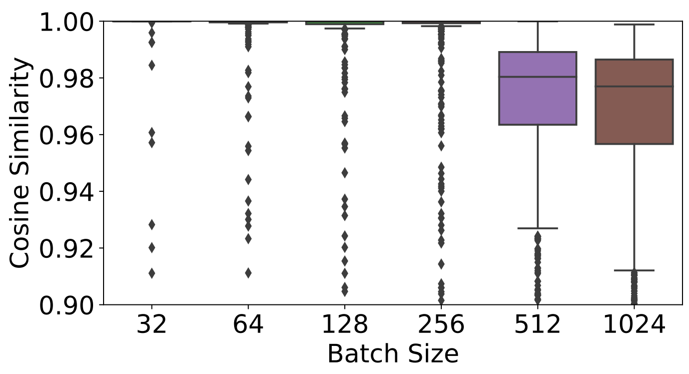


Figure 3.7: Distribution of cosine similarity (CS) values computed between the private embeddings z and the embeddings recovered by CPA $\hat{z}$. Most of the CS values are close to the ideal value of 1.

- *Sensitivity to input dimensionality:* CPA's optimization is independent of the input dimensionality d. Consequently, CPA performs significantly better for datasets with larger inputs (Tiny-ImageNet) compared to GMA, especially for larger batch sizes.

3.5.3 Results for VGG-16

Next, we present the results from our experiments with the VGG-16 network trained on the ImageNet dataset. Our proposed attack uses a 2-step process that combines CPA and FIA (Figure 3.4) to perform gradient inversion.

Embedding recovery: Our attack starts by recovering the private embeddings $\hat{z}$ from the gradients of the FC layer. We evaluate the quality of these recovered embeddings by computing its cosine similarity (CS) with the original embedding z. Figure 3.7 shows the distribution of the CS values for various batch sizes. Our results show that CPA allows near-perfect recovery of embeddings in most cases, with the CS values degrading slightly for larger batch sizes.

Gradient Inversion: We use the embeddings recovered from CPA to estimate the inputs with a feature inversion attack. Table 3.2 shows the $LPIPS$ scores comparing gradient matching attack (prior work), cocktail party + feature inversion attack (our proposal) and cocktail party + feature inversion + gradient matching attack (our proposal + prior work). We make the following key observations:

Table 3.2: $LPIPS \downarrow$ scores of images recovered using GM (prior work), CP+FI (our proposal) and CP+FI+GM (prior work + our proposal) attacks, with VGG-16 network trained on ImageNet.

Attack	**Batch Size**					
	32	64	128	256	512	1024
GM	0.536	0.594	0.609	0.652	OOM	OOM
CP+FI	0.483	0.493	0.479	0.495	0.507	0.509
CP+FI+GM	**0.392**	**0.430**	**0.423**	**0.469**	OOM	OOM

- *Comparison with prior work:* CP+FI has better average $LPIPS$ score as it can recover more images compared to GM. CP+FI+GM improves the number of images recovered by combining the benefits of our proposal (CP+FI) and prior work (GM).

- *Sensitivity to batch size:* The performance of GM degrades significantly with larger batch sizes. In contrast, CP+FI shows a smaller degradation and shows better scalability to larger batch sizes.

- *Memory Footprint:* The optimization for gradient matching is $\mathcal{O}(n \times d)$. The memory footprint of this optimization can exceed the available GPU memory when the input dimensionality (d) is large. For the experiments with ImageNet, we found that an 8-GPU machine cannot handle batch sizes in excess of 256 causing out of memory (OOM) errors. In contrast, the optimization for CPA is independent of d and can scale to a batch size of 1024.

3.6 Limitation and Defenses

Batch Size: ICA requires the number of aggregate gradients from neurons (mixed signals) to be greater than or equal to the number of inputs (source signals). Thus choosing a very large batch size that exceeds the number of neurons in the FC layer can prevent our attack.

Embedding size: The efficacy of feature inversion attack depends on the size of the embedding. For a CNN that produces a smaller sized embedding, FIA might be harder to carry out. However, this limitation can be overcome if the attacker knows the input data distribution, which can be used to aid feature inversion.

Differential Privacy (DP) Defense: Table 3.3 shows the $LPIPS$ scores from CP and GM evaluated under DP noise [34, 35]. We use the FC2 model with Tiny-ImageNet dataset and a batch size of 8 for these experiments. We scale the gradients to have unit norm and perturb the gradients with different amounts of Gaussian noise. We also show the ϵ values for (ϵ, δ) DP with $\delta = 0.00001$ corresponding to different amounts of noise. Our evaluations show that DP noise provides an effective defense against our attack.

Table 3.3: $LPIPS \downarrow$ scores of recovered images from CP and GM attacks under varying magnitudes of DP noise.

σ	0	0.0001	0.001	0.01
ϵ	∞	6056.00	606.60	60.56
GM	0.182	0.426	0.728	0.701
CP	0.0082	0.474	0.721	0.723

3.7 Extensions to CPA

Our evaluations assume that the the network does not use batchnorm layers and that the attacker does not have access to the input data distribution. When this information is available, it can be combined with our attack to further improve performance. Additionally, the private embeddings leaked from CPA can also be used to infer additional attributes about the input [36]. Lastly, our work can also be extended to language models and recommendation systems where it is common for the input to be fed directly to a FC layer. We also believe that our attack is more amenable to modern neural network architectures like transformers [37] due to the same reason. We leave extending our attack to other tasks and DNN architectures as part of our future work.

3.8 Summary

We propose *Cocktail Party Attack (CPA)* – a gradient inversion attack that can recover private inputs from aggregate gradients in FL. Our work is based on the key insight that gradients from an FC layer are linear combinations of its inputs. CPA uses this insight to frame gradient inversion for an FC layer as a blind source separation problem and uses independent component analysis to recover the inputs. CPA can be used directly on FC models to recover the inputs. It can also by extended to CNN models by first recovering the embeddings from an FC layer and then using a feature inversion attack to recover the inputs from the embeddings. Our evaluations on several image classification tasks show that CPA can perform high-quality gradient inversion, scales to ImageNet-sized inputs, and works with a batch size as large as 1024. CPA is orthogonal to prior works that use gradient matching and can be combined with gradient matching based approaches to further improve gradient inversion. Our work demonstrates that that aggregation alone is not sufficient to ensure privacy and methods like differential privacy are truly necessary to provide meaningful privacy guarantees in FL.

CHAPTER 4
EXPLOIT: EXTRACTING PRIVATE LABELS IN SPLIT-LEARNING

This chapter examines the privacy vulnerabilities in split learning that can be used to leak private training data. Our goal is to show that the gradient information transmitted by the label owner during two-party split learning contains enough information to leak private labels. To this end, we develop a novel attack – ExPLoit – that can leak private labels with high accuracy and overcome prior work's limitations.

4.1 Introduction

Split learning [3, 4] is a framework that allows distributed training of a model on vertically partitioned data, where the inputs and the corresponding labels are held by two different parties. Split learning uses a composition of two models $f : \mathcal{X} \to \mathcal{Z}$ and $g : \mathcal{Z} \to \mathcal{Y}$ that is split between the input and label owners. The composition network $g \circ f$ is trained end-to-end by requiring the input owner to transmit the embedding z_i (intermediate representation) to the label owner in the forward pass and the label owner returning the gradient $\nabla_z L_i$ to the input owner during the backward pass. This allows the split model to be trained on the distributed data while keeping the sensitive data with their respective owners.

Unfortunately, split learning does not have formal privacy guarantees, and it is unclear if it allows the input and label owners to hide their private data from each other. In this chapter, we set out to understand if an adversarial input owner can break label privacy by extracting the private labels in two-party split learning. To this end, we propose *ExPLoit*– a label-leakage attack that frames the problem of learning the private labels as a supervised learning task, by leveraging the gradient information ($\nabla_z L_i$) obtained during split learning. ExPLoit "replays" split learning by replacing the label owner's model g and labels $\{y_i\}$ with a randomly initialized surrogate model g' (with parameters $\theta_{g'}$) and surrogate labels

$\{y'_i\}$ respectively. We aim to learn the private labels by training these surrogate parameters using the following key objectives:

1. *Gradient Matching Objective:* The gradient computed using the surrogate model and labels during *replay split learning* should match the gradients received from the label owner during the original split learning process.

2. *Label Prior Objective:* The distribution of surrogate labels must match the expected label prior distribution. For instance, if the classification problem in consideration has a uniform label prior, the surrogate labels must also have a uniform distribution.

3. *Label Entropy Objective:* Since we consider datasets with hard labels, each individual surrogate label y'_i must have low entropy.

4. *Accuracy Objective:* The predictions made by the surrogate model must be close to the surrogate labels, achieving high accuracy.

Combining the above objectives yields a loss function that can be used to train the surrogate parameters and labels. By minimizing this loss function over all the embedding, gradient pairs $\{z_i, \nabla_z L_i\}$ received during split learning, the surrogate labels $\{y'_i\}$ can be trained to match the private labels of the label owner $\{y_i\}$, allowing an adversarial input owner to carry out a label leakage attack with high accuracy.

4.2 Background

We formally define the threat model by laying out the objectives of an attacker seeking to carry out a label leakage attack. We also define the objectives of a defender who is trying to protect against a label leakage attack. Additionally, we describe the limitations of existing label leakage attacks, which serves as the motivation for our research.

4.2.1 Label Leakage Attack Objective

In this chapter, we develop a label leakage attack, where an adversarial input owner tries to learn the label owner's private labels $\mathcal{D}_{label} = \{y_i\}$. We consider an honest-but-curious adversary, where the input owner tries to infer the private labels while honestly following the split learning protocol. During the training process of split learning, for each input x_i, the adversarial input owner transmits the embeddings z_i and receives the gradient $\nabla_z L_i$ from the label owner. The adversary uses an algorithm $\mathcal{A}$ to estimate the private labels y_i' for each input using these $\{z_i, \nabla_z L_i\}$ pairs obtained during split learning as shown below:

$$\mathcal{A}(\mathcal{D}_{grad}) \rightarrow \mathcal{D}_{y'}, \text{ where } \mathcal{D}_{grad} = \{z_i, \nabla_z L_i\},\ \mathcal{D}_{y'} = \{y_i'\}. \tag{4.1}$$

The attack objective is to maximize the accuracy of estimated labels as follows:

$$\max \mathop{\mathbb{E}}_{y_i \sim \mathcal{D}_{label}} [Acc(y_i, y_i')]. \tag{4.2}$$

Since the private labels $\{y_i\}$ are unavailable to the input owner, evaluating Equation 4.2 is not possible. Instead, our proposed attack uses a surrogate objective that can be optimized to uncover the private labels with high accuracy.

Attack Constraints: We assume that the parameters and architecture of the label-owner's model g is unknown to the attacker. The number of output classes and the class prior distribution are assumed to be known.

4.2.2 Label Leakage Defense Objective

A defense that protects against label leakage attack needs to balance the following utility and privacy objectives:

Utility Objective: Train the composition model $g \circ f$ to have high classification accuracy on an unseen validation set associated with the classification task (Equation 4.3).

$$\max \mathop{\mathbb{E}}_{x_i, y_i \sim \mathcal{D}_{val}} [Acc(y_i, g(f(x_i))] \tag{4.3}$$

Privacy Objective: Minimize the accuracy of the estimated labels $\{y_i'\}$ that can be recovered from the gradient information used in split learning (Equation 4.4).

$$\min \mathop{\mathbb{E}}_{y_i \sim \mathcal{D}_{label}} [Acc(y_i, y_i')] \tag{4.4}$$

4.2.3 Limitations of Existing Attacks

We provide an overview of recent related works on label-leakage attacks in split learning and describe their limitations.

Norm-based attack [38] is a label-leakage attack proposed specifically for the conversion prediction problem. This attack leverages the observation that only a small fraction of ad clicks result in a purchase, resulting in a high class imbalance in the training dataset of the conversion prediction task. As a result of this imbalance, the magnitude of the gradients is higher for the infrequent class. This attack proposes to use the magnitude of loss gradient ($\|\nabla_z L_i\|_2$) to infer the private labels. A key limitation of this attack is that it only works on binary classification problems with high class imbalances and is not generally applicable to multi-class classification settings with balanced class distribution.

UnSplit[39] proposes to learn the private labels in split learning using a gradient-matching objective [16] by minimizing the mean squared error (MSE) between the surrogate and true gradients using the following objective: $\min_{\theta_g', \{y_i'\}} MSE(\nabla_z L_i', \nabla_z L_i)$. Here, θ_g' are the parameters of the surrogate model and $\{y_i'\}$ are the surrogate labels. Results from this work show that UnSplit only works well when the label-owner's model g is one-layer deep, limiting its applicability to toy models.

Model Completion Attack [40] uses unlabeled embeddings $\mathcal{D} = \{z_i\}$ and a small number of labeled embeddings $\mathcal{D}^l = \{z_i^l, y_i^l\}$ to train a surrogate model $g' : \mathcal{Z} \to \mathcal{Y}$ using semi-supervised learning. Since g' functionally approximates the label-owner's model g, it can be used to predict the labels for the input embeddings $y_i' = g'(z_i)$, allowing the input owner to guess the private labels. This proposal suffers from two key drawbacks. First, it requires the adversary to have access to labeled examples, which may not always be available. E.g., in case of conversion prediction, labels for the input data cannot be gathered even using human annotators, as it is not readily apparent from the data. Second, the efficacy of this attack is limited by the accuracy of the model trained by the adversary.

4.3 ExPLoit

We propose *ExPLoit*– a label leakage attack that can be used by a malicious input-owner to learn the private labels in split learning. Our key insight is that the label leakage attack can be framed as a supervised learning problem by replacing the unknown parameters of the label owner with learnable surrogate parameters. This allows the adversarial input owner to "replay" the split learning process with surrogate parameters. We train these surrogate parameters by using a novel loss function that uses gradient-matching and several regularization terms that leverage key properties of the model and training data. By optimizing the surrogate parameters to minimize this loss function, we can recover the label owner's private labels. The rest of this section describes our proposed attack in greater detail.

4.3.1 Surrogate Parameter Substitution

From the input owner's point of view, the split learning process has two sets of unknown parameters associated with the label-owner: the label owner's model g and the private labels $\{y_i\}$ (see Figure 4.1a). Our goal is to uncover these unknown values by treating them as learnable parameters. To do so, we start by substituting these unknowns with randomly initialized surrogate parameters, as shown in Figure 4.1b. We replace g with a surrogate model g' (with parameters $\theta_{g'}$), and $\{y_i\}$ with a set of surrogate labels $\{y_i'\}$. We want y_i'

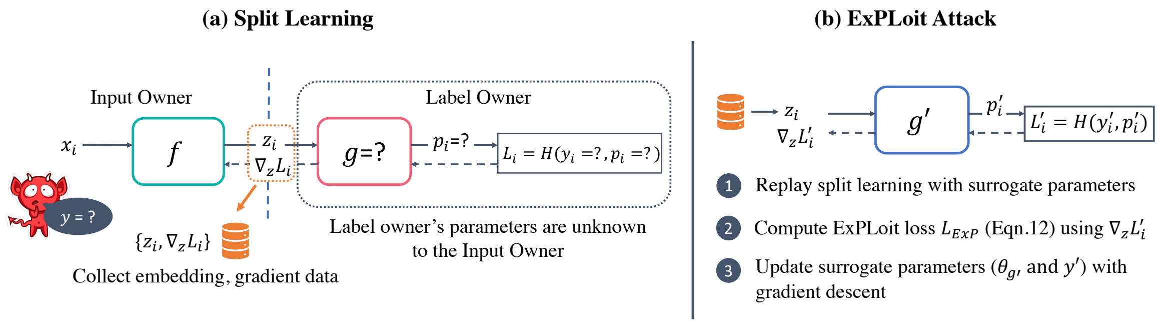

(a) Split Learning
Input Owner
Label Owner
x_i
f
z_i
$\nabla_z L_i$
g=?
p_i=?
$L_i = H(y_i =?, p_i =?)$
y = ?
$\{z_i, \nabla_z L_i\}$
Collect embedding, gradient data
Label owner's parameters are unknown to the Input Owner
(b) ExPLoit Attack
z_i
$\nabla_z L'_i$
g'
p'_i
$L'_i = H(y'_i, p'_i)$
1 Replay split learning with surrogate parameters
2 Compute ExPLoit loss L_{ExP} (Eqn.12) using $\nabla_z L'_i$
3 Update surrogate parameters ($\theta_{g'}$ and y') with gradient descent

Figure 4.1: (a) *Split Learning:* The adversarial input owner collects the embedding and gradient data $\{z_i, \nabla_z L_i\}$ when performing split learning with the label owner. (b) *ExPLoit Attack:* The embedding and gradient data is used to train surrogate model and label parameters (g' and $\{y'_i\}$) and uncover the private labels.

to be a point on an $n-1$ dimensional probability simplex for an n-class classification problem. To enforce this property, we set $y'_i = Softmax(\hat{y}_i)$, where $\hat{y}_i \in \mathcal{R}^n$. With the surrogate parameters in place, the goal of our attack is to learn the surrogate labels $\{y'_i\}$ (or equivalently to learn $\{\hat{y}_i\}$).

4.3.2 Replay Split Learning

To train the surrogate parameters, we first "replay" the split learning process using surrogate parameters as shown in Figure 4.1b. First, in the forward pass, the embedding z_i (collected during split learning) is fed into g' to get the prediction p'_i, which along with the surrogate labels y'_i can be used to compute the loss $L'_i = H(y'_i, p'_i)$. Next, we perform back-propagation through g' and compute the gradient of the loss with respect to the embedding $\nabla_z L'_i$. We use this gradient data as part of our loss function to learn the private labels of the label-owner, as described below.

4.3.3 ExPLoit Loss

To train the surrogate parameters $\theta_{g'}$ and $\hat{y}$, we formulate a loss function using four objectives that leverage key properties of the model and datasets:

1. *Gradient Objective*: The gradient $\nabla_z L'_i$, obtained during *replay split learning*, must

match the original gradients $\nabla_z L_i$, obtained during the original split learning process. This can be achieved by minimizing the l^2 distance between $\nabla_z L'_i$ and $\nabla_z L_i$ as shown below:

$$\min_{\theta'_g, \{\hat{y}_i\}} \mathbb{E} \left\| \nabla_z L'_i - \nabla_z L_i \right\|_2 . \tag{4.5}$$

2. *Label Prior Objective*: The distribution of surrogate labels must match the label prior P_y of the dataset. The probability distribution of the surrogate labels can be computed by taking the expectation of the surrogate labels[1] $P_{y'} = \mathbb{E}(y'_i)$. We perform the following optimization to match the distributions of the original and surrogate labels:

$$\min_{\theta'_g, \{y'_i\}} \mathcal{D}_{KL}(P_y \| P_{y'}). \tag{4.6}$$

3. *Label Entropy Objective*: Each individual surrogate label y'_i must have low entropy as the datasets we consider have zero entropy one-hot labels.

4. *Accuracy Objective*: The surrogate model must have high prediction accuracy with respect to the surrogate labels. In other words, the predictions of the surrogate model p'_i must be close to the surrogate labels y'_i.

To achieve the label entropy and accuracy objectives, we can minimize the normalized cross-entropy loss between p'_i and y'_i as follows:

$$\min_{\theta'_g, \{y'_i\}_i} \frac{\mathbb{E}[H(y'_i, p'_i)]}{H(P_y)}. \tag{4.7}$$

Note that the cross-entropy term $H(y'_i, p'_i)$ in Equation 4.7 can be expressed as a sum of the label entropy and KL divergence between the surrogate label and prediction: $H(y'_i, p'_i) = H(y'_i) + \mathcal{D}_{KL}(y'_i \| p'_i)$. Thus, by minimizing cross-entropy, we can minimize the entropy of surrogate labels (label entropy objective) and match the model's predictions with the surrogate labels (accuracy objective). We normalize cross-entropy with the entropy of the

[1]Each surrogate label y'_i is a n-dimensional probability vector that represents the probability distribution over the n output classes.

label prior $H(P_y)$ to ensure that the metric is insensitive to the number of label classes and the label priors [41]. We combine all the learning objectives described above to derive the final loss function as shown below:

$$\begin{aligned} L_{ExP} = {} & \mathbb{E}\Big[\, \|\nabla_z L_i - \nabla_z L'_i\|_2 \Big] + \\ & \lambda_{ce} \cdot \mathbb{E}\Big[H(y'_i, p'_i)/H(P_y) \Big] + \lambda_p \cdot \mathcal{D}_{KL}(P_y \| P_{y'}). \end{aligned} \tag{4.8}$$

Here, λ_{ce} and λ_p dictate the relative importance of the cross-entropy and label prior terms compared to the gradient loss term (first term in Equation 4.8). By optimizing the surrogate model and label parameters using this loss function, we can recover the private labels of the label owner with high accuracy. We consider the gradient loss term to be the primary optimization objective of our loss function. The cross-entropy and label prior terms act as regularizers and help us achieve a better label leakage accuracy.

4.3.4 Putting It All Together

The individual components described thus far can be combined to carry out our label leakage attack. Our attack starts with the input owner performing split learning process with the label owner, as shown in Figure 4.1a. During this process, the input owner collects the embedding z_i and the corresponding loss gradient $\nabla_z L_i$ for each input. Using this data, the input owner can use the ***ExPLoit*** attack to leak the private labels. Our attack is described in algorithm 1. In the outer loop, we pick values for λ_p, λ_{ce} and the learning rates $\eta_{g'}, \eta_{\hat{y}}$ using a Bayesian hyperparameter optimization algorithm. The surrogate parameters $\{\hat{y}_i\}_i$ and $\theta_{g'}$ are randomly initialized, and each inner loop of the attack proceeds as follows:

1. Replay split learning with surrogate parameters with the following steps:
    (a) Sample a batch of embeddings, gradients and surrogate labels: $\{z, \nabla_z L, \hat{y}\}_{batch}$.
    (b) Perform forward pass and compute loss $\{L'\}$ using predictions $\{p'\}$ and surrogate labels $\{y'\}$.

    (c) Perform backpropagation to compute the loss gradients $\{\nabla_z L'\}$.

2. Compute the ExPLoit loss: L_{ExP} (Equation 4.8).

3. Update surrogate parameters $\theta_{g'}$ and $\{\hat{y}_i\}$ to minimize L_{ExP}.

We repeat the above steps until the values of the surrogate parameters converge.

Hyperparameter Optimization: We learn a set of surrogate labels $\{y_i'\}$ in each outer loop for different selections of the hyperparameters. Unfortunately, evaluating the accuracy of the surrogate labels produced in each iteration is not possible since the input owner is unaware of any of the true labels. Consequently, we cannot use accuracy to guide the hyperparameter search. Instead, we evaluate the gradient loss term $\mathbb{E}[\|\nabla_z L_i' - \nabla_z L_i\|_2]$ after completing each outer iteration and use this as our objective function to be minimized by tuning the hyperparameters. We report the accuracy of the surrogate labels obtained for the best set of hyperparameters that minimizes this objective.

4.4 Experiments

We evaluate ExPLoit with multiple datasets and model architectures to show that it can leak private labels with high accuracy, across different settings. We describe our experimental setup followed by the results in this section.

4.4.1 Experimental Setup

The datasets and the corresponding split-models ($f \circ g$) used in our evaluations are shown in Table 4.1.

Datasets: FashionMNIST, CIFAR-10, CIFAR-100, and Tiny-ImageNet are computer vision datasets used to perform multi-class image classification. The Criteo dataset consists of conversion logs for online ad-clicks, with each entry consisting of 3 continuous, and 17 categorical features, along with a binary label indicating if the ad-click resulted in a purchase (conversion). Note that the Criteo dataset has a large class imbalance (90% of the labels are 0's, and the rest are 1's).

Models: We use a 4-layer convolutional neural network for FashionMNIST and a 21-layer ResNet model for CIFAR-10, CIFAR-100 and Tiny-ImageNet. The model for the conversion prediction task (Criteo) consists of a learnable embedding layer (to handle categorical features) followed by four fully connected (FC) layers. All the models are split into two sub-models f (input-owner's model) and g (label-owner's model), which are jointly trained using split learning. The layer at which the model is split is referred to as the *cut-layer*. To test the sensitivity of our attack to the cut-layer, we perform experiments with two different configurations: Config-1 and Config-2, which splits the model at different points as shown in Table 4.1. The label-owner's model g only consists of FC layers in Config-1. In contrast, the g model in Config-2 is larger and consists of both convolutional (Conv) and FC layers. The vision models are trained for 10 epochs and the CVR model for 5 epochs using the Adam optimizer with a learning rate of 0.001. We evaluate the efficacy of label leakage attacks at different points during the split-model training by carrying out the attack after each training epoch.

Table 4.1: Datasets and the corresponding split-models used in our experiments.

Dataset	Config-1		Config-2		g'
	f	g	f	g	
FashionMNIST	$Conv \times 4$	$FC \times 2$	$Conv \times 3$	$Conv - FC \times 2$	$FC \times 3$
CIFAR-10	$Conv - Res \times 3$	$FC \times 2$	$Conv - Res \times 2$	$Res - FC \times 2$	$FC \times 3$
CIFAR-100	$Conv - Res \times 3$	$FC \times 2$	$Conv - Res \times 2$	$Res - FC \times 2$	$FC \times 3$
Tiny-ImageNet	$Conv - Res \times 3$	$FC \times 2$	$Conv - Res \times 2$	$Res - FC \times 2$	$FC \times 3$
Criteo	$Emb - FC \times 2$	$FC \times 2$	$Emb - FC$	$FC \times 3$	$FC \times 4$

Attack parameters: We assume that the architecture of the label owner's model g is not known to the input owner. Thus, we use a 3-layer fully connected DNN (FC[128-64-10] for image classification and FC[32-32-10] for Criteo) as the surrogate model g'. We set $N_{iter} = 500$, and the learning rate range to $[10^{-5}, 10^{-4}]$ for $\eta_{g'}$ and $[10^{-2}, 10^{-1}]$ for $\eta_{\hat{y}}$. The range for λ_{ce} and λ_p is set to $[0.1, 3]$. We carry out the ExPLoit attack for each training

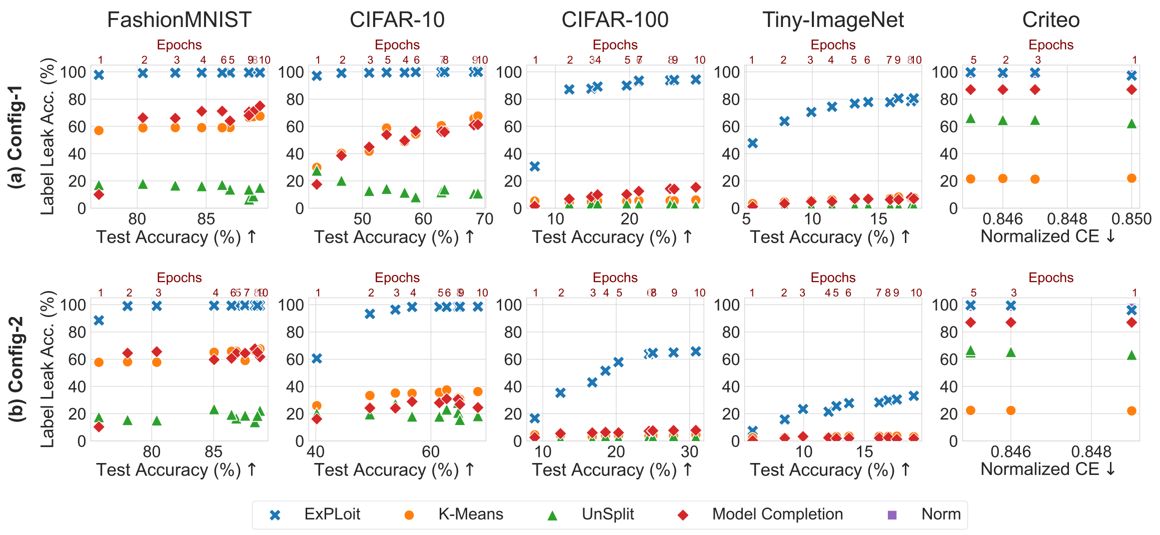


Figure 4.2: Results comparing *ExPLoit* with the K-means baseline and prior works (UnSplit, Model Completion and Norm-based attacks) for two configurations of the split model: (a) Config-1 and (b) Config-2 (see Table 4.1). ExPLoit significantly outperforms prior works and can leak private labels with a high accuracy (up to 99.53%) for most datasets.

epoch of split learning.

Evaluation Metric: By learning the surrogate labels associated with each input, ExPLoit groups the inputs that belong to the same class together. However, the true class labels corresponding to each input group remains to be determined. For an unbalanced dataset, the label prior information can be used to infer the label by considering the size of each input group. For a balanced dataset, the adversary can use inputs with known labels (one for each output class) to determine the label associated with each input group. We report the clustering accuracy [42] obtained with the best hyperparameters (corresponding to the lowest gradient loss) in our results below.

4.4.2 Results

We plot label leakage accuracy for ExPLoit and prior works, and compare it with the test accuracy/normalized cross entropy[2] of the split-model, across various datasets and model

[2]We use normalized cross-entropy (NCE) instead of test accuracy to measure the model performance for the Criteo dataset as it has a high class imbalance. A lower value of NCE indicates better performance.

configurations in Figure 4.2. ExPLoit achieves near-perfect label leakage accuracy for most datasets (99.53% for CIFAR-10) and significantly outperforms all prior works. We explain the attack sensitivity to various parameters like dataset, split-model training epoch, cut layer, and compare our results with prior works below. We use the accuracy numbers from Config-1 (Figure 4.2a), Epoch-10 to discuss the results, unless specified otherwise.

Sensitivity to Split-model Training Epoch: Our results show that the efficacy of ExPLoit improves when attacking the later epochs of the split-model training. E.g., the label leakage accuracy of ExPLoit for CIFAR-100 is just 30.65% for Epoch-1 and improves to 94.4% for Epoch-10. The reason for this trend is because our attack approximates the label owner's model g using a fixed surrogate model g'. However, in reality, g is not fixed, and changes as training progresses. The rate of this change is smaller for the latter epochs. Consequently, our ability to approximate g with a fixed surrogate model improves for the later epochs, which improves our attack's efficacy. While ExPLoit already achieves near perfect accuracy for most datasets with just 10 epochs of split-model training, we expect this accuracy to improve further as the split-model is trained for a greater number of epochs.

Sensitivity to Datasets: ExPLoit is more effective for datasets with lower input dimensionality and fewer classes. For instance, ExPLoit has a label leakage accuracy of 99.5% for FashionMNIST, whereas the accuracy drops to 94.38% for CIFAR-100 and 80.61% for Tiny-ImageNet. This is because our attack has a higher number of surrogate parameters for CIFAR-100 and Tiny-ImageNet compared to FashionMNIST (due to larger input size and number of output classes), which increases the difficulty of the learning problem.

Sensitivity to Cut layer and Model Architecture: We evaluate our attack on two split network configurations: Config-1 and Config-2, which represent two different choices of the cut layer. ExPLoit achieves a higher label-leakage accuracy for Config-1 compared to Config-2. For instance, in the case of CIFAR-100, our attack produces a label leakage accuracy of 94.38% for Config-1 and 65.74% for Config-2. The reason for this discrepancy is two-fold. First, g is larger for Config-2, compared to Config-1, which makes it harder

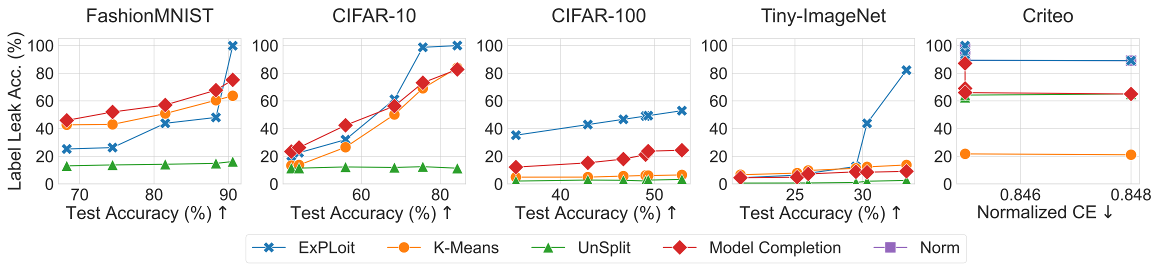


Figure 4.3: Results showing the utility-privacy trade-off for various attacks under the gradient noise defense. A larger gradient noise reduces the efficacy of label leakage attacks at the cost of degradation in model accuracy.

to approximate with a surrogate model g' for Config-2. Second, our g' model is architecturally similar to the g model in Config-1 as both these models use fully connected layers, while the g models in Conv-2 uses FC and Conv layers. Thus, the efficacy of our attack reduces when g is larger and has a dissimilar model architecture compared to g'. **Comparisons with Baseline and Prior Work:** We compare the performance of ExPLoit against an unsupervised learning baseline (K-Means Clustering) and three recent attacks: UnSplit, Model Completion and Norm based attack.

K-Means Attack: Through the split-learning process, the label-owner is able to generate embeddings $z_i = f(x_i)$, for each input x_i. One way to estimate the private labels is by using unsupervised learning to group these embeddings. We perform K-Means clustering using the embeddings $\{z_i\}$ and report the resulting label leakage accuracy in Figure 4.2. The efficacy of the attack improves with the quality of embeddings. Consequently, the attack performs well for simpler datasets like FashionMNIST, where it is easier to learn good embeddings with relatively few training epochs. The attack accuracy accuracy also improves for later epochs as the model f learns better embeddings as training progresses.

UnSplit Attack [39]: The UnSplit attack uses the gradient matching loss to learn the private labels. The authors of [39] showed that this technique is effective only when g is a single layer network and does not work for multi-layer networks. Consistent with their results, our experiments with the UnSplit attack also showed very low efficacy (10.99% attack accuracy for CIFAR-10, which is comparable to a random guess).

Model Completion Attack[40]: This attack proposes to train a surrogate model g' using semi-supervised learning to predict the private labels corresponding to the inputs. Similar to the K-Means attack, the efficacy of this attack depends on the quality of emebddings produced by f. Consequently, this attack works well for simpler datasets, providing an accuracy of 75.06% for FashionMNIST, while the accuracy degrades for more complex datasets like CIFAR-10 (61.3% accuracy), CIFAR-100 (15.3% accuracy) and Tiny-ImageNet (6.875% accuracy).

Norm-based attack[38] : The norm-based attack uses gradient norm to predict the labels in imbalanced binary classification problems. Evaluations on the Criteo dataset shows that this attack can achieve high accuracy (comparable with ExPLoit). The leakage accuracy also improves for the later epochs of the split-model training as the difference in gradient norms becomes more pronounced.

ExPLoit significantly outperforms all prior works, providing a near-perfect label leakage accuracy for most datasets. E.g. ExPLoit achieves a label-leakage accuracy of 99.53% for CIFAR-10, which is 32.38% higher than the next best attack. This difference is even more pronounced when the attack is carried out at an earlier training epoch (67.2% higher accuracy compared to next best attack at Epoch-1 for CIFAR-10). This is because, unlike prior works, the efficacy of ExPLoit does not depend on the quality of embeddings produced by f. The efficacy of our attack demonstrates that split learning offers negligible privacy benefits for the label owner.

4.5 Gradient Noise Defense

ExPLoit uses the gradient information obtained from the label owner during split learning to leak the private labels. A common method that can be used to deter information leakage is by perturbing the loss gradients with noise. We carry out experiments to test if gradient noise can be used to defend against ExPLoit.

4.5.1 Setup

We propose to perturb the gradients with Gaussian noise as shown in Equation 4.9.

$$\nabla_z \hat{L}_i = \nabla_z L_i + \eta, \ \ where\, \eta \sim \mathcal{N}(0, \sigma) \tag{4.9}$$

The label owner can transmit these noisy gradients $\nabla_z \hat{L}_i$ to the input owner to perform split learning. Adding noise prevents the input owner from having reliable access to the true gradients. This reduces the efficacy of ExPLoit, providing better privacy to the label owner. On the other hand, noisy gradients are detrimental to training the split model and results in lower accuracy, thus impacting utility. To evaluate the utility-privacy trade-off offered by this defense, we perform split learning with different amounts of gradient noise by sweeping σ in Equation 4.9. For each value of σ, we train the split-model till convergence and report the test and label leakage accuracy with ExPLoit. Since the gradients obtained during split learning are noisy, optimizing the hyperparameters of our attack using the gradient loss objective is not optimal. Instead we tune the hyperparameters using $L_{ExP}(\lambda_{ce} = 1, \lambda_p = 1)$ as the optimization objective.

4.5.2 Results

The utility-privacy trade-off with gradient noise defense for ExPLoit is shown in Figure 4.3. Against the ExPLoit attack, this defense provides a better utility-privacy trade-off for lower dimensional datasets like Criteo and FashionMNIST. For instance, gradient noise degrades the label leakage accuracy for FashionMNIST by 51.8% with only a 2.26% reduction of test accuracy. In contrast, for CIFAR-10, a 79% reduction in label leakage accuracy incurs a 38% reduction in test accuracy. This discrepancy is because adding gradient noise hampers the quality of the input owner's model f. Simpler datasets are more resilient to this degradation, whereas more complex datasets like CIFAR-10, CIFAR-100 and Tiny-ImageNet are impacted more if f is not trained properly. Thus, while gradient noise might be suitable for simpler datasets, it may not be practical for more complex datasets.

Additionally, we evaluate the gradient noise defense against other attacks. While the gradients are not directly used by the K-Means and model completion attacks, gradient noise reduces the quality of embeddings learnt by f, which degrades the efficacy of these attacks. We find that these attacks are more resilient to gradient noise for simpler datasets (FashionMNIST) compared to ExPLoit as the quality of the embeddings does not degrade significantly. However, ExpLoit performs better than these two attacks for more complex datasets like CIFAR-100 and Tiny-ImageNet. The UnSplit attack continues to provide no benefit under gradient noise.

4.6 Summary

Split learning has been proposed as a method for privacy-preserving training on vertically partitioned data. However, the information exchanged during split learning carries the risk of leaking private data. To demonstrate this vulnerability, we propose *ExPLoit* – a label-leakage attack that allows an adversarial input owner to learn the label owner's private labels during two-party split learning. Our key insight is that the attack can be framed as a learning problem by substituting the unknown parameters of the label owner with leranable surrogate parameters. We use the gradient data collected during split learning and a novel loss function to train these surrogate parameters. Our evaluations on several image-classification tasks and a conversion prediction task show that ExPLoit can leak private labels with near-perfect accuracy of up to 99.53%. ExPLoit also outperforms recent prior works, offering up to 67.2% improvement in label leakage accuracy. We also evaluate gradient noise as a defense to improve label privacy. While this provides a reasonable defense for simpler datasets, we find that the utility-privacy tradeoff of this technique is unfavorable for more complex datasets. Our findings demonstrate that split learning provides negligible privacy benefits to the label owner and highlights the need for techniques that provide principled privacy guarantees to train on vertically partitioned data.

Algorithm 1: ExPLoit Attack

```
Input: $\{z_i\}, \{\nabla_z L_i\}, P_y, N_{iter}$
Output: $\{y_i^*\}$
$\mathcal{D}_{train} = \{z_i, \nabla_z L_i, y_i'\}$
for $i \leftarrow 0$ to $N_{iter}$ do
    $\lambda_p, \lambda_{ce}, \eta_{g'}, \eta_{\hat{y}} \leftarrow BayesOpt()$
    Initialize $\{\hat{y}_i\}, g'(\cdot; \theta_{g'})$
    repeat
        for $\{z, \nabla_z L, \hat{y}\}_{batch}$ in $\mathcal{D}_{train}$ do
            $\{y_i'\} = \{Softmax(\hat{y}_i)\}$
            $P_{y'} = \mathbb{E}(y_i')$

            // 1. Replay Split Learning
            for $\{z, \nabla_z L, \hat{y}\}_i$ in $\{z, \nabla_z L, \hat{y}\}_{batch}$ do
                $p_i' = g'(z_i; \theta_{g'})$
                $L_i' = \mathcal{D}_{KL}(y_i' \| p_i')$
                Compute $\nabla_z L_i'$
            end

            // 2. Compute ExPLoit loss
            $L_{ExP} =$
              $\mathbb{E}[\|\nabla_z L_i' - \nabla_z L_i\|_2] + \lambda_{ce} \cdot \mathbb{E}[H(y_i', p_i')/H(P_y)] + \lambda_p \cdot \mathcal{D}_{KL}(P_y \| P_{y'})$
            // 3. Update surrogate model, label parameters
            $\theta_{g'} \leftarrow \theta_{g'} - \eta_{g'} \cdot \nabla_{\theta_{g'}} L_{ExP}$
            $\hat{y} \leftarrow \hat{y} - \eta_{\hat{y}} \cdot \nabla_{\hat{y}} L_{ExP}$
        end
    until Convergence;
    $NewBest = UpdateBayesOpt(\mathbb{E}[\|\nabla_z L_i' - \nabla_z L_i\|_2])$
    if $NewBest$ then
        $\{y_i^*\} \leftarrow \{Softmax(\hat{y}_i)\}$
    end
end
```

CHAPTER 5
MAZE: DATA-FREE MODEL STEALING ATTACK

Model stealing attacks pose a threat to the privacy of models in remote inference. However, a key limitation of existing attacks is that they require distributionally similar data to carry out model stealing with high accuracy. In this chapter, we investigate if model stealing is possible in a data-free setting, where the adversary does not have access to any dataset. To this end, we develop a novel attack – *MAZE*– that carries out data-free model stealing with high accuracy, demonstrating that the availablity of data is not a fundamental requirement for model stealing, but is rather a technical limitation of existing attacks.

5.1 Introduction

The ability of Deep Neural Networks (DNNs) to achieve state of the art performances in a wide variety of challenging computer-vision tasks has spurred the wide-spread adoption of these models by companies to enable various products and services such as self-driving cars, license plate reading, disease diagnosis from medical images, activity classification from images and video, and smart cameras. As the performance of ML models scales with the training data [43], companies invest significantly in collecting vast amounts of data to train high-performance ML models. Protecting the confidentiality of these models is vital for companies to maintain a competitive advantage and to prevent the stolen model from being misused by an adversary to compromise security and privacy. For example, an adversary can use the stolen model to craft adversarial examples [44, 45, 46], compromise user membership privacy through membership inference attacks [47, 36, 48], and leak sensitive user data used to train the model through model inversion attacks [49, 28, 50]. Thus, ML models are considered valuable intellectual properties of the owner and are closely guarded against theft and data leaks.

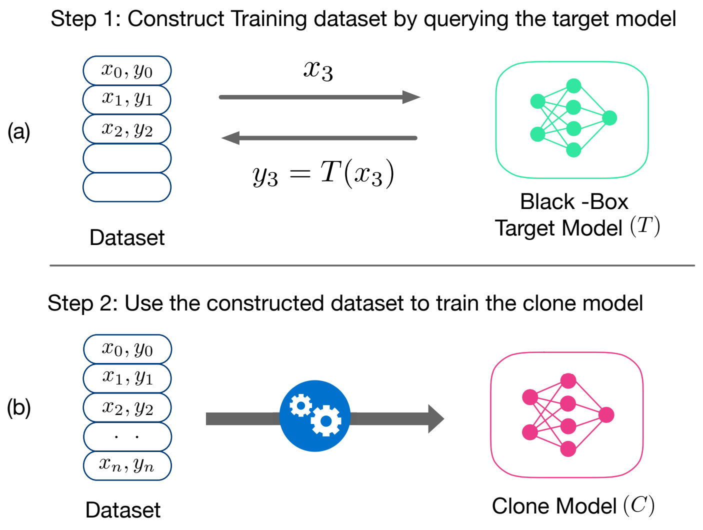


Figure 5.1: Model stealing attacks: The target model is queried using a set of inputs $\{x_i\}_{i=1}^n$ to obtain a labeled training dataset $\{x_i, y_i\}_{i=1}^n$, which is used to train the clone model.

Model functionality stealing attacks compromise the confidentiality of ML models by allowing an adversary to train a *clone model* that closely mimics the predictions of the target model, effectively copying its functionality. These attacks only require black-box access to the target model where the adversary can access the predictions of the model for any given input. Figure 5.1 illustrates the steps involved in carrying out a MS attack. The adversary first queries the target model T with various inputs $\{x_i\}_{i=1}^n$ and uses the predictions of the target model $y_i = T(x_i)$ to construct a labeled dataset $\mathcal{D} = \{x_i, y_i\}$. This dataset is then used to train a clone model C to match the predictions of T.

Key Limitation of Existing Attacks: In the current state of the art methods (e.g., [6, 5]), the availability of in-distribution or similar surrogate data to query the target model plays a key role in the ability of the attacker to train high accuracy clone models. However, in most real-world scenarios, the training data is not readily available to the attacker as companies typically train their models using proprietary datasets. To carry out MS in such a data-limited setting, existing attacks either assume partial availability of the target dataset or the availability of a *surrogate* dataset that is semantically similar to the target dataset (e.g., using CIFAR-100 to attack a CIFAR-10 model). For example, *Jacobian-Based Dataset*

Augmentation (JBDA) [6] is an attack that uses a subset of the training data to create additional synthetic data, which is used to query the target model. *KnockoffNets* [5] is another MS attack that uses a *surrogate* dataset to query the target model. These attacks become ineffective without access to the target dataset or a representative surrogate dataset. [1]

Our goal: In this chapter, we want to show that a highly accurate MS attack is feasible without relying on any access to the target dataset or even a surrogate dataset. This would demonstrate that the lack of access to data does not pose a fundamental limitation for an adversary to carry out model stealing.

Key Contributions: We make the following key contributions in this chapter:

- We propose *MAZE*– the first data-free model stealing attack capable of training high-accuracy clone models across multiple image classification datasets and complex DNN target models. In contrast to existing attacks that require some form of data to query the target, MAZE uses synthetic data created using a generative model to carry out MS attack. Our evaluations across DNNs trained on various image classification tasks show that MAZE provides a normalized clone accuracy of $0.90\times$ to $0.99\times$ (normalized clone accuracy is the accuracy of the clone model expressed as a fraction of the target-model accuracy). Despite not using any data, MAZE outperforms recent attacks that rely on partial data (JBDA, clone accuracy of $0.13\times$ to $0.69\times$) or surrogate data (KnockoffNets, clone accuracy of $0.52\times$ to $0.97\times$).

- Our key insight is to draw inspiration from data-free knowledge distillation (KD) and zeroth-order gradient estimation to train the generative model used to produce synthetic data in MAZE. Similar to data-free KD, the generator is trained on a disagreement objective, which encourages it to produce synthetic inputs that maximize the disagreement between the predictions of the target (teacher) and the clone (student) models. By training the clone model on such synthetic examples we can improve the

[1]We refer the interested readers to Section 6.1 of the KnockoffNets paper [5] for a discussion on the importance of using semantically similar datasets to carry out the attack.

alignment of the clone model's decision boundary with that of the target, resulting in a high-accuracy clone model.

In data-free KD, training the generator on the disagreement objective is possible since white-box access to the teacher model is available. But, unlike in data-free KD, MAZE operates in a black-box setting. We therefore leverage *zeroth-order gradient estimation (ZO)* [51, 52] to approximate the gradient of the black-box target model and use this to train the generator. Unfortunately, we found a direct application of ZO gradient estimation to be impractical on real-world image classification models since the dimensionality of the generator's parameters can be in the order of millions. We propose a way to overcome the dimensionality problem by estimating gradients with respect to the significantly lower-dimensional synthetic input and show that our method can be successfully used to train a generator in a query-efficient manner.

- In some cases, partial datasets may be available. Recognizing that, we propose an extension of MAZE, called *MAZE-PD*, for scenarios where a small partial dataset (e.g., 100 examples) is available to the attacker. MAZE-PD leverages the available data to produce queries that are closer to the training distribution than in MAZE by using generative adversarial training. Our evaluations show that MAZE-PD provides near-perfect clone accuracy ($0.97\times$ to $1.0\times$), while reducing the number of queries by $2\times$-$24\times$ compared to MAZE.

In summary, our key finding is that an attacker only requires black-box access to the target model and no in-distribution data to create high-accuracy clone models in the image classification domain. If even a very limited amount of in-distribution data is available, near-perfect clone accuracy is feasible. This raises questions on how machine learning models can be better protected from competitors and bad actors in this domain.

5.2 Related Work

Several types of MS attacks have been proposed in recent literature. Depending on the goal of the attack, MS attacks can be categorized into: (1) parameter stealing (2) hyper-parameter stealing (3) functionality stealing attacks. Parameter stealing attacks [53, 54] focus on stealing the exact model parameters, while hyper-parameter stealing attacks [55, 56] aim to determine the hyper-parameters used in the model architecture or the training algorithm of the target model. Our work, MAZE and MAZE-PD, are designed to carry out a *functionality stealing* attack, where the goal is to replicate the functionality of a blackbox target model by training the clone model on the predictions of the target. As the attacker typically does not have access to the dataset used to train the target model, attacks need alternate forms of data to query the target model and perform model stealing. Depending on the availability of data, functionality stealing attacks can be classified as using (1) partial-data, (2) surrogate-data, or (3) data-free, i.e., synthetic data. We discuss prior works in each of these three settings and also briefly discuss relationship between model stealing and knowledge distillation.

5.2.1 Model Stealing with Partial Data

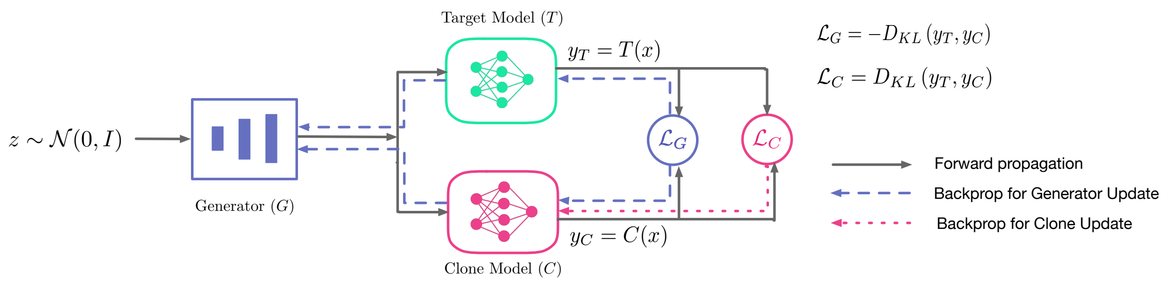
In the partial-data setting, the attacker has access to a subset of the data used to train the target model. While this in itself may be insufficient to carry out model stealing, it allows the attacker to craft synthetic examples using the available data. *Jacobian Based Dataset Augmentation (JBDA)* [6] is an example of one such attack that assumes that the adversary has access to a small set of *seed* examples from the target data distribution. The attack works by first training a clone model C using the seed examples and then progressively adding synthetic examples to the training dataset. JBDA uses a perturbation based heuristic to generate new synthetic inputs from existing labeled inputs. E.g., from an input-label pair (x, y), a synthetic input x' is generated by using the jacobian of the clone model's loss function $\nabla_x \mathcal{L}\left(C\left(x;\theta_c\right), y\right)$ as shown in Equation 5.1.

$$x' = x + \lambda sign\left(\nabla_x \mathcal{L}\left(C\left(x;\theta_c\right), y\right)\right) \tag{5.1}$$

The dataset of synthetic examples $\{x'_i\}$ generated this way are labeled by using the predictions of the target model $y'_i = T(x'_i)$ and the labeled examples $\{x'_i, y'_i\}$ are added to the pool of labeled examples that can be used to train the clone model C. In addition to requiring a set of seed examples from the target distribution, a key limitation of JBDA is that, while it works well for simpler datasets like MNIST, it tends to produce clone models with lower classification accuracy for more complex datasets. For example, our evaluations in Section section 5.5 show that JBDA provides a normalized clone accuracy of only $0.13\times$ (GTSRB dataset) and $0.18\times$ (SVHN dataset).

5.2.2 Model Stealing with Surrogate Data

In the surrogate data setting, the attacker has access to alternate datasets that can be used to query the target model. KnockoffNets [5] is an example of a MS attack that is designed to operate in such a setting. With a suitable surrogate dataset, KnockoffNets can produce clone models with up to $0.97\times$ the accuracy of the target model. However, the efficacy of such attacks is dictated by the availability of a suitable surrogate dataset. For instance, if we use the MNIST dataset to perform MS on a FashionMNIST model, it only produces a clone model with $0.41\times$ the accuracy of the target model (See Table 5.1 for full results). This is because the surrogate dataset is not representative of the target dataset, which reduces the effectiveness of the attack.



Target Model (T)
$y_T = T(x)$
$\mathcal{L}_G = -D_{KL}(y_T, y_C)$
$\mathcal{L}_C = D_{KL}(y_T, y_C)$
$z \sim \mathcal{N}(0, I)$
$\mathcal{L}_G$
$\mathcal{L}_C$
Generator (G)
Forward propagation
Backprop for Generator Update
Backprop for Clone Update
$y_C = C(x)$
Clone Model (C)

Figure 5.2: MAZE Attack Setup: MAZE uses a generative model G to produce the synthetic input queries $\{x\}$ to perform Model Stealing. The clone model C is trained to match the predictions of the target model T. G is trained to produce queries that maximize the dissimilarity between y_T and y_C. Optimizing $\mathcal{L}_G$ requires backpropagation through T to update G. However, we only have black-box access to T, therefore we use zeroth-order gradient estimation to perform gradient descent on $\mathcal{L}_G$.

5.2.3 Data-Free Model Stealing

In the data-free setting, the adversary does not have access to any data. This represents the hardest setting to carry out MS as the attacker has no knowledge of the data distribution used to train the target model. A recent work by Roberts et al. [57] studies the use of inputs derived from various noise distributions to carry out MS attack in the data-free setting. While this attack works well for simple datasets like MNIST, our evaluations show that such attacks do not scale to more complex datasets such as CIFAR-10 (we obtained relative clone accuracy of only $0.11\times$), limiting their applicability (See Table 5.1 for full results).

5.2.4 Knowledge distillation

Model stealing is related to knowledge-distillation (KD) [58], but in KD, unlike in model stealing, the target model is available to the attacker and is simply being summarized into a simpler architecture.

5.3 Preliminaries

The goal of this chapter is to develop a model functionality stealing attack in the data-free setting, which can be used to train a high-accuracy clone model only using black-box access to the target model. We formally state the objective and constraints of our proposed data-free model stealing attack.

Attack Objective: Consider a target model T that performs a classification task with high accuracy. Our goal is to train a clone model C that replicates the functionality of the target model by maximizing the accuracy on a test set $\mathcal{D}_{test}$ as shown in Equation 5.2.

$$\max_{\theta_C} \underset{x,y\sim\mathcal{D}_{test}}{\mathbb{E}} [Acc(C(x;\theta_C), y)] \tag{5.2}$$

Attack Constraints: We assume that the adversary does not know any details about the Target model's architecture or the model parameters θ_T. The adversary is only allowed black-box access to the target model. We assume the *soft-label* setting where the adversary can query the target model with any input x and observe its output probabilities $\vec{y} = T(x;\theta_T)$. We consider model stealing attacks under two settings based on the availability of data:

1. *Data-free setting (Primary goal):* The adversary does not have access to the dataset $\mathcal{D}_T$ used to train the target model or a good way to sample from the target data distribution $\mathbb{P}_T$. (Section section 5.4)

2. *Partial-data setting (Secondary goal):* The adversary has access to a small subset (e.g., 100) of training examples randomly sampled from the training dataset of the target model. (Section section 5.6)

For both of these settings we assume the availability of a test set $\mathcal{D}_{test}$, which is used to report the test accuracies of the clone models produced by our attack.

5.4 MAZE: Data-Free Model Stealing

We propose *MAZE*[2], a data-free model stealing attack using zeroth order gradient estimation. Unlike existing attacks, MAZE does not require access to the target or a surrogate dataset and instead uses a generative model to produce the synthetic queries for launching the attack. Figure 5.2 shows an overview of MAZE. In this section, we first describe the training objectives of the clone and the generator model. We then motivate the need for gradient estimation to update G in the black-box setting of MS attack and show how zeroth-order gradient estimation can be used to optimize the parameters of G. Finally, we discuss our algorithm to carry out model stealing with MAZE.

5.4.1 Training the Clone Model

The clone model is trained using the input queries produced by the generator. The generator G takes in a low dimensional latent vector z, sampled from a random normal distribution, and produces an input query $x \in \mathbb{R}^d$ that matches the input dimension of the target classifier (Equation 5.3). We use x to obtain the output probabilities of the target model $\vec{y_T}$ and clone model $\vec{y_C}$ on x as shown in Equation 5.4.

$$x = G(z; \theta_G); \;\; z \sim \mathcal{N}(0, \boldsymbol{I}) \tag{5.3}$$

$$\vec{y_T} = T(x; \theta_T); \; \vec{y_C} = C(x; \theta_C) \tag{5.4}$$

Where θ_T, θ_C and θ_G represent the parameters of the target, clone, and generator models, respectively. The clone model is trained using the loss function in Equation 5.5 to minimize the KL divergence between $\vec{y_C}$ and $\vec{y_T}$.

$$\mathcal{L}_C = D_{KL}(\vec{y_T} \| \vec{y_C}) \tag{5.5}$$

[2]Code: https://github.com/sanjaykariyappa/MAZE

5.4.2 Training the Generator Model

The generator model G synthesises the queries necessary to perform model stealing. Similar to recent works in data-free KD [59, 28, 60], MAZE trains the generator to produce queries that maximize the disagreement between the predictions of the teacher and the student by maximizing the KL-divergence between $\vec{y_T}$ and $\vec{y_C}$. The loss function that is used to train the generator model is described below in Equation 5.6. We refer to this as the *disagreement objective*.

$$\mathcal{L}_G = -D_{KL}(\vec{y_T} \| \vec{y_C}) \tag{5.6}$$

Training G on this loss function maximizes the disagreement between the predictions of the target and the clone model. Since C and G have opposing objectives, training both models together results in a two-player game, similar to *Generative Adversarial Networks* [61], resulting in the generation of inputs that maximize the learning of the clone model. By training C to match the predictions of T on the queries generated by G, we can perform knowledge distillation and obtain a highly accurate clone model.

Training G using the loss function in Equation 5.6 requires backpropagating through the predictions of the target model T, as shown by the dashed lines in Figure 5.2. Unfortunately, as we only have black-box access to T, we cannot perform back-propagation directly, preventing us from training G and carrying out the attack. To solve this problem, our insight is to use *zeroth-order gradient estimation* to approximate the gradient of the loss function $\mathcal{L}_G$. The number of black-box queries necessary for ZO gradient estimation scales with the dimensionality of the parameters being optimized. Estimating the gradients of $\mathcal{L}_G$ with respect to the generator parameters θ_G directly is expensive as the generator has on the order of millions of parameters. Instead, we choose to estimate the gradients with respect to the synthetic input x produced by the generator, which has a much lower dimensionality (3072 for CIFAR-10), and use this estimate to back propagate through G.

This modification allows us to compute a gradient estimates in a query efficient manner to update the generator model. The following section describes how we efficiently apply zeroth-order gradient estimation to train the generator model.

5.4.3 Train via Zeroth-Order Gradient Estimate

Zeroth-order gradient estimation [51, 52] is a popular technique to perform optimization in the black-box setting. We use this technique to train our generator model G. Recall that our objective is to update the generator model parameters θ_G using gradient descent to minimize the loss function $\mathcal{L}_G$ as shown in Equation 5.7.

$$\theta_G^{t+1} = \theta_G^t - \eta \nabla_{\theta_G} \mathcal{L}_G \tag{5.7}$$

Updating θ_G in this way requires us to compute the derivative of the loss function $\nabla_{\theta_G} \mathcal{L}_G$. By the use of chain-rule, $\nabla_{\theta_G} \mathcal{L}_G$ can be decomposed into two components as shown in Equation 5.8.

$$\nabla_{\theta_G} \mathcal{L}_G = \frac{\partial \mathcal{L}_G}{\partial \theta_G} = \frac{\partial \mathcal{L}_G}{\partial x} \times \frac{\partial x}{\partial \theta_G} \tag{5.8}$$

We can compute the second term $\frac{\partial x}{\partial \theta_G}$ in Equation 5.8 by performing backpropagation through G. Computing the first term $\frac{\partial \mathcal{L}_G}{\partial x}$ however requires access to the model parameters of the target model (θ_T). Since T is a black-box model from the perspective of the attacker, we do not have access to θ_T, which prevents us from computing $\frac{\partial \mathcal{L}_G}{\partial x}$ through backpropagation. Instead, we propose to use an approximation of the gradient by leveraging zeroth-order gradient estimation. To explain how the gradient estimate is computed, consider an input vector $x \in \mathbb{R}^d$ generated by G that is used to query T. We can estimate $\frac{\partial \mathcal{L}_G}{\partial x}$ by using the method of forward differences [62] as shown in Equation 5.9.

$$\hat{\nabla}_x \mathcal{L}_G(x; u_i) = \frac{d \cdot (\mathcal{L}_G(x + \epsilon u_i) - \mathcal{L}_G(x))}{\epsilon} u_i \tag{5.9}$$

Where u_i is a random variable drawn from a d dimensional unit sphere with uniform probability and ϵ is a small positive constant called the *smoothing factor*. The random gradient estimate, shown in Equation 5.9, tends to have a high variance. To reduce the variance, we use an averaged version of the random gradient estimate [63, 64] by computing the forward difference using m random directions $\{u_1, u_2, ..u_m\}$, as shown below in Equation 5.10.

$$\hat{\nabla}_x \mathcal{L}_G(x) = \frac{1}{m} \sum_{i=1}^{m} \hat{\nabla}_x \mathcal{L}_G(x; u_i) \tag{5.10}$$

Where $\hat{\nabla}_x \mathcal{L}_G$ is an estimate of the true gradient $\nabla_x \mathcal{L}_G$. By substituting $\hat{\nabla}_x \mathcal{L}_G$ into Equation 5.8, we can compute an approximation for the gradient of the loss function of the generator: $\hat{\nabla}_{\theta_G} \mathcal{L}_G$. The gradient estimate $\hat{\nabla}_{\theta_G} \mathcal{L}_G$ computed this way can be used to perform gradient descent by updating the parameters of the generator model θ_G according to Equation 5.7. By updating θ_G, we can train G to produce the synthetic examples required to perform model stealing.

5.4.4 MAZE Algorithm for Model Stealing Attack

We outline the algorithm of MAZE in algorithm 2 by putting together the individual training algorithms of the generator and clone models. We start by fixing a query budget Q, which dictates the maximum number of queries we are allowed to make to the target model T. ϵ is the smoothing parameter and m is the number of random directions used to estimate the gradient. We set the value of ϵ to 0.001 in our experiments. N_G, N_C represent the number of training iterations and η_G, η_C represent the learning rates of the generator and clone model, respectively. N_R denotes the number of iterations for experience replay.

Algorithm 2: MAZE Algorithm for Model Stealing Attack

```
Input: $T, Q, \epsilon, m, N_G, N_C, N_G, \eta_G, \eta_C$
Output: Clone model $C(\cdot; \theta_C)$
Initialize $G(\cdot; \theta_G), C(\cdot; \theta_C), q \leftarrow 0, \mathcal{D} \leftarrow \{\}$
while $q < Q$ do
    // Generator Training
    for $i \leftarrow 0$ to $N_G$ do
        $x = G(z) : z \sim \mathcal{N}(0, I)$
        $\mathcal{L}_G = -D_{KL}\left(T(x) \| C(x)\right)$
        $\hat{\nabla}_{\theta_G} \mathcal{L}_G \leftarrow ZO\_grad\_est(G, T, C, x, \epsilon, m)$
        $\theta_G \leftarrow \theta_G - \eta_G \hat{\nabla}_{\theta_G} \mathcal{L}_G$
    end
    // Clone Training
    for $i \leftarrow 0$ to $N_C$ do
        $x = G(z) : z \sim \mathcal{N}(0, I)$
        $\mathcal{L}_C = D_{KL}\left(T(x) \| C(x)\right)$
        $\theta_C \leftarrow \theta_C - \eta_C \nabla_{\theta_C} \mathcal{L}_C$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, T(x))\}$
    end
    // Experience Replay
    for $i \leftarrow 0$ to $N_T$ do
        $(x, y_T) \sim \mathcal{D}$
        $\mathcal{L}_C = D_{KL}\left(y_T \| C(x)\right)$
        $\theta_C \leftarrow \theta_C - \eta_C \nabla_{\theta_C} \mathcal{L}_C$
    end
    $q \leftarrow update(q)$
end
```

The outermost loop of the attack repeats till we exhaust our query budget Q. The attack algorithm involves three phases: 1. *Generator Training* 2. *Clone Training* and 3. *Experience Replay*. In the *Generator Training* phase, we perform N_G rounds of gradient descent for G, which is trained to produce inputs x that maximize the KL-divergence between the predictions of the target and clone model. θ_G is updated by using zeroth-order gradient estimates as described in Section subsection 5.4.3. This is followed by the *Clone Training* phase, where we perform N_C rounds of gradient descent for C. In each round, we generate a batch of inputs $x = G(z)$ and use these inputs to query the target model. The clone model is trained to match the predictions of the target model by minimizing $D_{KL}(T(x) \| C(x))$. The input, prediction pair: $(x, T(x))$ generated in each round is stored in dataset $\mathcal{D}$. Finally, we perform *Experience Replay*, where we train the clone on previously seen inputs that are stored in $\mathcal{D}$. Retraining on previously seen queries reduces *catastrophic forgetting* [65] and ensures that the clone model continues to classify old examples seen during the earlier part of the training process correctly.

5.4.5 Computing the Query Cost

The target model needs to be queried in order to update both the generator and the clone models. Considering a batch size of 1, one training iteration of G requires $m + 1$ queries to T for the zeroth-order gradient estimation and each training loop of C requires 1 query. Experience replay, on the other hand, does not require any additional queries to T. Thus, with a batch size of B, the query cost of each iteration is described by Equation 5.11

$$\text{Query cost per iteration} = B(N_G(m + 1) + N_C) \tag{5.11}$$

We use $B = 128$, $N_G = 1, N_C = 5, N_R = 10$ and $m = 10$ in our experiments, unless stated otherwise. Thus, each iteration of the attack requires 2048 queries. We use a query budget of $5M$ for FashionMNIST and SVHN and a query budget of $30M$ for GTSRB and CIFAR-10 datasets to report our results.

5.5 Experimental Evaluation

We validate our attack by performing model stealing on various target models and provide experimental evidence to show that our attack can produce high accuracy clone models without using any data. We compare our results against two prior works– *KnockoffNets* and *Jacobian Based Dataset Augmentation* (JBDA)– and show that the clone models produced by our attack have comparable or better accuracy than the ones produced by these prior works, despite not using any data.

Table 5.1: Comparison of clone accuracies obtained from various attacks. Numbers in the bracket express the accuracy as a multiple of the target model accuracy. MAZE obtains high accuracy ($0.90\times$ to $0.99\times$), despite not using any data.

Dataset	**Target Accuracy (%)**	**MAZE (data-free)**	**KnockoffNets (surrogate data)**	**JBDA (partial-data)**	**Noise (data-free)**
FashionMNIST	91.04	**81.9 (0.90×)**	47.26 (0.52×)	62.65 (0.69×)	62.91 (0.69×)
SVHN	95.25	**93.85(0.99×)**	92.77 (0.97×)	17.16(0.18×)	51.86 (0.54×)
GTSRB	97.43	88.31 (0.91×)	**89.86 (0.92×)**	12.80(0.13×)	38.38 (0.39×)
CIFAR-10	92.26	**89.85 (0.97×)**	82.56 (0.89×)	25.11 (0.27×)	10.17 (0.11×)

5.5.1 Setup: Dataset and Architecture

We perform our evaluations by attacking DNN models that are trained on various image-classification tasks. The datasets and target model accuracies used in our experiments are mentioned in Table 5.1. We use a LeNet for the FashionMNIST and ResNet-20 for the other datasets as the target model. Our attack assumes no knowledge of the target model and uses a randomly initialized 22-layer WideResNet [66] as the clone model for all the datasets. In general, any sufficiently complex DNN can be used as the clone model. We use an SGD optimizer with an initial learning rate of 0.1 to train our clone model. For G, we use a generative model with 3 convolutional layers. Each convolutional layer in G is followed by a batchnorm layer and the activations are upsampled to ensure that the outputs generated by G are of the correct dimensionality corresponding to the dataset being attacked. We use an SGD optimizer with an initial learning rate of 0.0001 to train G. The learning rates for both the clone and generator models are decayed using cosine annealing.

5.5.2 Configuration of Existing Attacks

Existing MS attacks either use surrogate data or synthetic datasets derived from partial access to the target dataset. We compare MAZE with the following attacks:

1. KnockoffNets [5] attack uses a surrogate dataset to query the target model to construct a labeled dataset using the predictions of the target model. This labeled dataset is used to train the clone model. We use MNIST, CIFAR10, CIFAR100, and CIFAR10 as the surrogate datasets for FashionMNIST, SVHN, CIFAR10, and GTSRB models, respectively. In each case, we query the target model with the training examples of the surrogate dataset. We then use the dataset constructed from these queries to train the clone model for 100 epochs using an SGD optimizer with a learning rate of 0.1 with cosine annealing scheduler.

2. JBDA [6]: attack performs MS by using synthetic examples to query the target model. These synthetic examples are generated by adding perturbations to a set of *seed* examples, which are obtained from the data distribution of the target model. The perturbations are computed using the Jacobian of the clone model's loss function (Equation 5.1). We start with an initial dataset of 100 seed examples and perform 6 rounds[3] of synthetic data augmentation with the clone model being trained for 10 epochs between each round. λ in Equation 5.1 dictates the magnitude of the perturbation. We set this to a value of 0.1. We use Adam optimizer with a learning rate of 0.001 to train the clone model. *3. Noise:* To test if inputs sampled from noise can be used to carry out MS attack, we design a *Noise* attack. We follow the proposal by Roberts et al. [57] and use random samples from an Ising prior model to query the target model. This attack serves as a baseline data-free MS attack to compare with our proposal.

5.5.3 Key Result: Normalized Clone Accuracy

Table 5.1 shows the clone-accuracy obtained by attacking various target models using MAZE. The numbers in brackets express the clone accuracy normalized to the accuracy of the target model being attacked. We also compare MAZE with existing MS attacks and highlight the best clone accuracy for each dataset in bold. Our results show that MAZE produces high accuracy clone models with a normalized accuracy greater than $0.90\times$ for all the target models under attack. In contrast, the baseline *Noise* attack fails to produce high accuracy clone models for most of the datasets.

Furthermore, the results from our attack also compare favorably against *KnockoffNets*

[3]We found that the accuracy of the JBDA attack stagnates beyond 6 augmentation rounds. This is in line with the observations made by Juuti et al. [67].

and *JBDA*, both of which require access to some data. We find that the effectiveness of KnockoffNets is highly dependent on the surrogate data being used to query the target model. For example, using MNIST to attack FashionMNIST dataset results in a low accuracy clone model ($0.52\times$ target accuracy) as these datasets as visually dissimilar. However, using CIFAR-100 to query CIFAR-10 results in a high accuracy clone model ($0.89\times$ target accuracy) due to the similarities in the two datasets. JBDA seems to be effective for attacking simpler datasets like $FashionMNIST$, but the accuracy reduces when attacking more complex datasets. This is in part because JBDA produces queries that are highly correlated to the initial set of "seed" examples, which sometimes results in worse performance even compared to noise (e.g. SVHN). By using the disagreement objective to train the generator, MAZE can generate queries that are more useful in training the clone model and result in higher accuracy of clones ($0.91\times$-$0.99\times$) compared to other attacks like JBDA ($0.13\times$-$0.69\times$) that use synthetic data.

5.6 MAZE-PD: MAZE with Partial-Data

The accuracy and speed of our attack can be improved if a few examples from the training-data distribution of the target model are available to the adversary. In this section we develop *MAZE-PD*, an extension of MAZE to the partial-data setting. In the data-free setting of MAZE, G is trained on a *disagreement objective* to produce inputs that maximize the disagreement between the target and the clone model. In the presence of a limited amount of data, we can additionally train the generator to produce inputs that are closer to the target distribution by using the *Waserstein Generative Adversarial Networks (WGANs)* [68] training objective. We observe that even a small amount of data from the target distribution (100 examples) can enable the generator to produce synthetic inputs that are closer to the target distribution. By improving the quality of the generated queries, MAZE-PD not only improves the effectiveness of the attack but also allows the attack

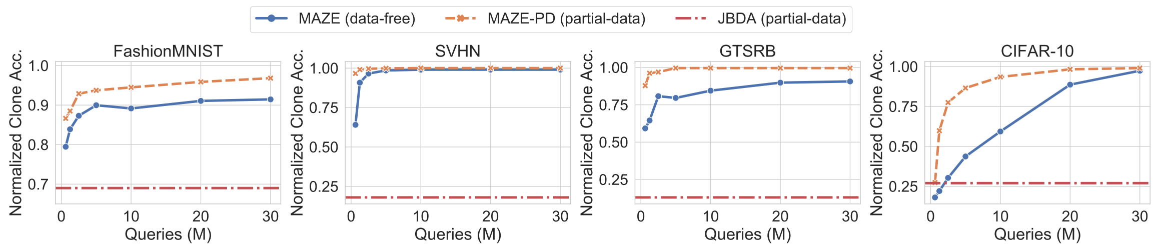


Figure 5.3: Normalized clone accuracy of MAZE (data-free), MAZE-PD (partial-data), and JBDA (partial-data) as the query budget is varied. Our results show that for a given query budget, MAZE-PD can train a clone model with higher accuracy than MAZE. The accuracy of MAZE-PD is also significantly better than the JBDA attack.

to succeed with far fewer queries compared to MAZE. In this section, we describe how WGANs can be incorporated into the training of the generator model to develop MAZE-PD. We also provide empirical evidence to show that MAZE-PD improves clone accuracy and reduces query cost significantly compared to MAZE.

5.6.1 Incorporating WGAN in MAZE

We describe the modifications to the training algorithm of MAZE (Algorithm algorithm 2) to incorporate WGAN training in the partial-data setting. In addition to the generator (G) and clone (C) models, we define a *critic model* D, which estimates the Wasserstein distance between the target data distribution $\mathbb{P}_T$ and the synthetic data distribution of the generator $\mathbb{P}_G$ using the function described in Equation 5.12.

$$W\left(\mathbb{P}_T, \mathbb{P}_G\right) = \max_{\theta_D} \mathbb{E}_{x\sim\mathbb{P}_T}\left[D(x)\right] - \mathbb{E}_{z\sim\mathcal{N}(0,I)}\left[D\left(G(z)\right)\right] \tag{5.12}$$

The generator model aims to produce examples closer to the target distribution by minimizing the Wasserstein distance estimated by the critic model. To incorporate the WGAN objective in the training of the generator, we modify the original loss function of G (Equation 5.6) with an additional term as shown in Equation 5.13.

$$x = G(z); z \sim \mathcal{N}(0, I)$$
$$\mathcal{L}_G = -D_{KL}(T(x) \| C(x)) - \lambda D(x) \tag{5.13}$$

The first term in Equation 5.13 represents the disagreement loss from MAZE and the second term is the WGAN loss. The hyper-parameter λ balances the relative importance between these two losses. To train the *critic model* D, we add an extra training phase (described by Algorithm algorithm 3) to our original training algorithm. We also include a gradient penalty term $GP = (\|\nabla_x D(x)\|_2 - 1)^2$ in $\mathcal{L}_D$ to ensure that D is 1-Lipschitz continuous.

Algorithm 3: Critic Training

```
// Critic Training
for $i \leftarrow 0$ to $N_d$ do
    $z \sim \mathcal{N}(0, I); x \sim \{x_i\}_{i=1}^{n}$
    $\mathcal{L}_D = D(G(z)) - D(x) + GP$
    $\theta_D \leftarrow \theta_D - \eta_D \nabla_{\theta_D} \mathcal{L}_D$
end
```

The training loops for the *clone training* and *experience replay* in Algorithm algorithm 2 remain unchanged. Using these modifications we can train a generator model that produces inputs closer to the target distribution $\mathbb{P}_T$.

5.6.2 Results: Clone Accuracy with Partial Data

We repeat the MS attack using MAZE-PD in the partial data setting. We assume that the attacker now has access to 100 random examples from the training data of the target model, which is roughly 0.2% of the total training data used to train the target model. We use $\lambda = 10$ in Equation 5.13 and $N_d = 10$ in Algorithm algorithm 3. Note that critic training does not require extra queries to the target model. The rest of the parameters are kept the same as before. Fig Figure 5.3 shows our results comparing the normalized clone accuracy obtained with MAZE-PD and MAZE (data-free) for various query budgets. For a given query budget, MAZE-PD obtains a higher clone accuracy compared to MAZE and achieves near-perfect clone accuracy ($0.97\times$-$1.0\times$) for all the datasets. Additionally, MAZE-PD offers a reduction of $2\times$ to $24\times$ in the query budget compared to MAZE for a given clone accuracy.

Comparison with JBDA: We compare the performance of MAZE-PD with JBDA, which also operates in the partial-data setting. JBDA produces low clone accuracies for most datasets (less than $0.30\times$ for SVHN, GTSRB, and CIFAR-10). In contrast, MAZE-PD obtains highly accurate clone models ($0.97\times$-$1.0\times$) across all four datasets.

5.7 Discussion

5.7.1 Follow-on Works

Our proposal MAZE is the first data free model stealing attack. Since the publication of our paper, several follow-on works have emerged that build and improve upon our insights. A line of work has looked at finding ways to save on query budget by avoiding the gradient computation through the target model. [69] is one such work that proposes to use the gradient of the clone model as a proxy for the gradient of the target model. [70] proposes a heuristic to generate synthetic data, without requiring backpropagation through the target model. Data-free model stealing has also been extended to other domains beyond image classification. Li et al. [71] extend data-free model stealing to the setting of image retrieval and Yue et al. [72] apply data-free model stealing to recommender systems. Overall, our work has spurred a lot of interest in the community to improve the efficacy, reduce the cost and extend the reach of data-free model stealing attacks.

5.7.2 Potential Defenses

MAZE requires access to the prediction probabilities of T in order to estimate gradient information. Thus, a natural way to defend against it would be to limit access to the predictions of the model to the end-user by restricting the output of the model only to provide hard-labels. Such a defense would make it harder, although not necessarily impossible, for an adversary to estimate the gradient by using numerical methods [73, 74, 75]. Unfortunately, such a defense may limit benign users from using the prediction probabilities from the service for downstream processing tasks.

Another potential method to defend against MAZE is to prevent an adversary from accessing the true predictions of the model by perturbing the output probabilities with some noise. Several defenses have been proposed along these lines [8, 76, 77]. Unfortunately, a key shortcoming of perturbation-based defense is that it can destroy information contained in the class probabilities that can be important for a benign user of the service for downstream processing tasks. Furthermore, such a defense can reduce classification accuracy for benign users, which is undesirable.

Yet another way to reduce the effectiveness of our attack may be to limit the number of queries that each user can make to the service. However, an adversary could circumvent such a defense by launching a distributed attack where the task of attacking the model is split across multiple users. Furthermore, limiting the number of queries may also constrain some of the legitimate users of the service from making benign queries, which may also be undesirable.

5.8 Summary

We propose *MAZE*, a high-accuracy MS attack that requires no input data. MAZE is the first data-free MS attack that works effectively for complex DNN models trained across multiple image-classification tasks. MAZE uses a generator trained with zeroth-order optimization to craft synthetic inputs, which are then used to copy the functionality of the target model to the clone model. Our evaluations show that MAZE produces clone models with high classification accuracy ($0.90\times$ to $0.99\times$). Despite not using any data, MAZE outperforms recent attacks that rely on partial-data or surrogate-data. Our work presents an important step towards developing highly accurate data-free MS attacks.

In addition, we propose *MAZE-PD* to extend MAZE to the partial-data setting, where the adversary has access to a small number of examples from the target distribution. MAZE-PD uses generative adversarial training to produce inputs that are closer to the target distribution. This further improves accuracy ($0.97\times$ to $1.0\times$) and yields a significant reduction in the number of queries ($2\times$ to $24\times$) necessary to carry out the attack compared to MAZE.

The effectiveness of our data-free attack shows that having access to semantically similar dataset is not a prerequisite for carrying out high-accuracy model stealing attacks, highlighting the threat to model privacy in remote inference. Our work also motivates the need for defenses that can prevent such model stealing attack from being carried out.

CHAPTER 6
DEFENDING AGAINST MODEL STEALING WITH ADAPTIVE MISINFORMATION

Model stealing poses a serious threat to model privacy in remote inference settings. Several attacks have demonstrated that it is possible to carry out model stealing in data-limited settings using semantically similar datasets. Furthermore, our work in the previous chapter shows that it is also possible to carry out high-accuracy model stealing in the data-free setting. To protect model privacy, we require practical defenses that can protect against the threat of model stealing from adversaries, while providing high utility to benign users during remote inference. In this chapter, we propose the *Adaptive Misinformation* defense[1], which significantly deters the effectiveness of model stealing attacks while having a minimal degradation in the model's utility for benign users.

6.1 Introduction

Several companies employ DNN models to offer classification as a service to end users who may not have the resources to train their own models. Such services are typically hosted with a remote inference framework, where user is allowed to interact with the model in a black-box fashion to obtain the classification outputs for the user's inputs. The model parameters and the architecture are kept hidden from the end-user to protect model privacy.

Unfortunately, recent attacks [5, 6, 78] have shown that it is possible for an adversary to carry out model stealing and train a clone model that achieves a classification accuracy that is remarkably close to the accuracy of the target model (up to $0.99\times$). Moreover, these attacks can be performed even when the adversary is constrained in the following ways:

1. The adversary only has *black-box* query access to the model i.e. the attacker can

[1]Code: https://github.com/sanjaykariyappa/adaptive_misinformation

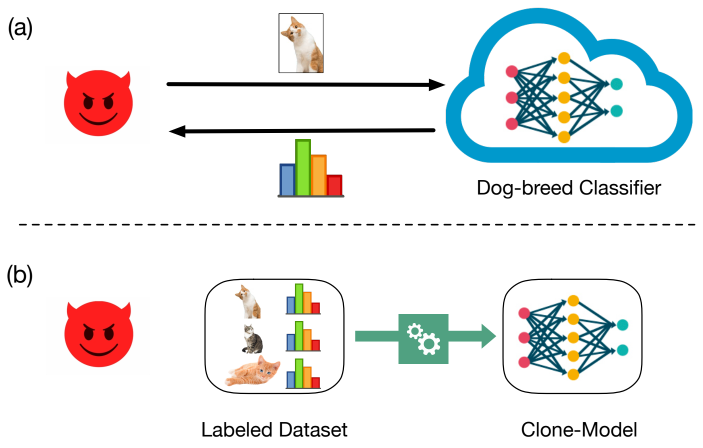


Figure 6.1: Model Stealing Attack: (a) An adversary queries the target model (dog-breed classifier) using synthetic/surrogate data (cat images) and constructs a labeled dataset using the predictions of the model (b) The labeled dataset can then be used to train a clone-model that replicates the functionality of the target model.

query the model with any input and observe the output probabilities.

2. The adversary is *data-limited* and does not have access to a large number of inputs representative of the training data of the target model.

Attacks that work under these constraints rely on one of two methods for generating the data necessary for querying the target model: (a) *Synthetic Data*: [67, 6] produce synthetic data from a small set of in-distribution *seed* examples by iteratively adding heuristic-based perturbations to the seed examples. (b) *Surrogate Data*: Several attacks [5, 79] simply use a surrogate dataset to query the target model. For example, a cat dataset can be used as the surrogate dataset to query the dog-breed classifier as shown in Figure 6.1a. A labeled dataset can be constructed from these queries, which can be used by the adversary to train a clone model that mimics the functionality of the target model (Figure 6.1b). Such attacks make it viable for an adversary to create a clone of the target model even with limited/no access to the target model's data distribution. The goal of this paper is to propose an effective defense for model stealing attacks carried out by a data-limited adversary with black-box access to the target model.

We observe that all existing attacks invariably generate Out-Of-Distribution (OOD) queries. One way to check if the data is OOD is by plotting the Maximum Softmax Prob-

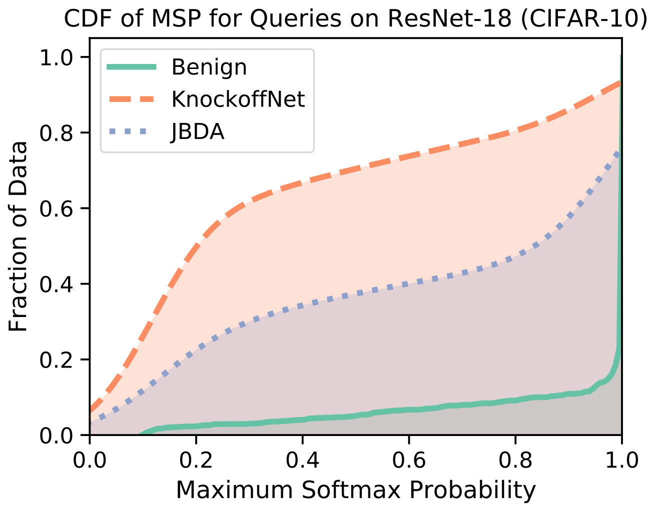


Figure 6.2: CDF of Maximum Softmax Probability (MSP) for queries from: (a) Benign User (b) KnockoffNet Attacker (c) Jacobian-Based Dataset Augmentation (JBDA) Attacker. Queries from benign user produce high values of MSP indicating in-distribution data while queries generated from attacks produce low values of MSP indicating out-of-distribution data.

ability (MSP) of the data produced by the target model. High values of MSP indicate In-Distribution (ID) data and low values indicate OOD data [80]. As an example, we characterize the MSP values using a ResNet-18 network trained on the CIFAR-10 dataset. We plot the CDF of MSP for benign queries sampled from a held-out test set as well as adversarial queries from two representative attacks: 1. Knockoffnets [5], using surrogate data from CIFAR-100 dataset and 2. Jacobian-Based Data Augmentation (JBDA) [6], which uses synthetic data, in Figure 6.2. Notice that the CDFs of the queries from both attacks are concentrated towards lower values of MSP, indicating OOD data, compared to the inputs from the benign user which produce high MSP values, implying that the inputs are ID.

Motivated by this observation, we propose *Adaptive Misinformation* (AM) to defend against model stealing attacks. AM selectively sends incorrect predictions for queries that are deemed OOD, while ID queries are serviced with correct predictions. Since a large fraction of the adversary's queries is OOD, this leads to the mislabeling of a significant portion of the adversary's dataset. Training a model on this mislabeled dataset results in a low-quality clone with poor accuracy, reducing the effectiveness of model stealing attacks. Recent works [7, 8] have used a similar insight of misleading the adversary, by injecting perturbations to the predictions of the model. Compared to these perturbation based defenses, our proposal is more scalable and offers a significantly better trade-off between model accuracy and security due to the following key attributes:

1. *Adaptive Nature:* The adaptive nature of our defense allows using incorrect predictions to selectively service suspicious OOD queries, instead of indiscriminately adding perturbations to the probabilities for all inputs. This results in a better trade-off between model accuracy and security against model stealing attacks.

2. *Reduced Correlation through Misinformation:* Prior works add perturbations to the original prediction in order to mislead the adversary. However, we find that these perturbed predictions remain correlated with the original predictions, leaking information about the original predictions of the model. In contrast, our defense uses an uncorrelated misinformation function to generate incorrect predictions, which reveals no information about the original predictions, resulting in better security.

3. *Low Computational Overhead:* Our proposal only requires a single inference pass with a modest increase in the amount of computation over an undefended model ($< 2\times$). In contrast, existing defenses like Prediction Poisoning [8] (PP) requires multiple gradient computations and thus incurs several orders of magnitude increase in computational cost and inference latency.

6.2 Threat Model

Our problem setting involves a *data-limited adversary* who is trying to perform model stealing attack on a *defender*'s model, just using black-box query access. In this section, we outline the attack and defense objectives.

6.2.1 Attack Objective

The adversary's goal is to replicate the functionality of the defender's model $f(x;\theta)$ by training a clone model $f'(x;\theta')$ that achieves high classification accuracy on the defender's classification task, as shown in Equation 6.1. Here, $P_{def}(X)$ denotes the distribution of data from the defender's problem domain, θ represents the parameters of the defender's model, and θ' represents the parameters of the clone model.

$$\max_{\theta'} \mathop{\mathbb{E}}_{x\sim\mathcal{P}_{def}(X)} Acc(f'(x;\theta')) \tag{6.1}$$

If the adversary had access to a labeled dataset of inputs sampled from $P_{def}(X)$, the adversary could simply use this to train the clone-model f'. However, in a lot of real-world classification problems, the adversary is data-limited and lacks access to a sufficiently large dataset that is representative of $P_{def}(X)$.

6.2.2 Defense Objective

The defender's aim is to prevent an adversary from being able to replicate the functionality of the model. Thus the defender's objective involves minimizing the accuracy of the cloned model f' trained by the adversary (Equation 6.2).

$$\min \mathop{\mathbb{E}}_{x\sim\mathcal{P}_{def}(X)} [Acc(f'(x;\theta'))] \tag{6.2}$$

The defender is also constrained to provide high classification accuracy to benign users of the service in order to retain the utility of the model for the classification task at hand. We formalize this by stating that the classification accuracy of the model for in-distribution examples has to be above a threshold T.

$$\mathop{\mathbb{E}}_{x \sim \mathcal{P}_{def}(X)} [Acc(f(x;\theta))] \geq T \tag{6.3}$$

Equation 6.2, Equation 6.3 describe a constrained optimization problem for the defender. This formulation of the problem allows the defense to trade off the accuracy of the model for improved security, as long as the accuracy constraint (Equation 6.3) is satisfied. We term defenses that work within these constraints as *accuracy-constrained* defenses. Our proposed defense falls under this framework and allows improvement in security at the cost of a reduction in classification accuracy.

6.3 Related Work

We discuss the various defenses against model stealing attacks that have been proposed in literature and describe their limitations that motivate our work. Existing defenses can broadly be categorized into *Stateful Detection Defenses* and *Perturbation Based Defenses*.

6.3.1 Stateful Detection Defenses

Several works [67, 81] have proposed analyzing the distribution of queries from individual users to classify the user as adversarial or benign. For instance, [67] uses the L_2 distance between successive queries to detect adversarial attacks based on the assumption that adversarial users send highly correlated queries. Unfortunately, such methods requires the defender to maintain a history of past queries limiting scalability. Moreover, these defenses are ineffective against adaptive attacks, attacks involving multiple colluding adversarial users and attacks that use surrogate datasets, which do not have correlated queries.

6.3.2 Perturbation-Based Defenses

In an undefended setting, the attacker has reliable access to the predictions of the target model f for any arbitrary input x. Perturbation-based defenses modify the original prediction of the model $y = f(x; \theta)$ to produce a perturbed prediction y', preventing the adversary from having reliable access to the target model's predictions.

Consequently, training a clone model with the surrogate dataset: $\{x, y'\}$ formed by the adversary results in a low-quality clone model with reduced accuracy. There are several defenses that work under different constraints for generating perturbed predictions. They can be broadly categorized into defenses that preserve the accuracy of the model and defenses that trade-off accuracy for security. We briefly describe each of these works before detailing our solution in the following section.

Accuracy Preserving Defenses

These defenses ensure that the accuracy of the model on ID examples is unchanged after adding the perturbations. For instance, [7] constrains the perturbation to leave the top-1 class of the perturbed output unchanged i.e $argmax(y'_i) = argmax(y_i)$.

This preserves the accuracy of the model, while removing the information present in the probabilities outputted by the defender's model. Similarly, [79] avoids exposing the output probabilities by only sending the hard labels of the top-1 or top-K classes while servicing requests. Both these defenses prevent the adversary from accessing the true prediction probabilities either implicitly or explicitly, while retaining the accuracy of the defender's model. Unfortunately, subsequent works have shown that the effectiveness of these defenses are limited as the adversary can still use the top-1 prediction of the model to perform model stealing attacks [8].

Accuracy-Constrained Defenses

Unlike accuracy preserving attacks, accuracy-constrained defenses do not require the classification accuracy to be retained after perturbing the predictions of the model. This allows the defender to trade off model accuracy for better security by injecting a larger amount of perturbation to the predictions of the model. However, the amount of perturbation that can be injected is bound by the accuracy constraint (Equation 6.3), which ensures that the accuracy of the model is above a specified threshold for ID examples. Prediction Poisoning [8] (PP) is a recent work that proposes such an accuracy-constrained defense, whereby the defender perturbs the prediction of the model as shown in Equation 6.4

$$y' = (1 - \alpha) f(x; \theta) + \alpha\eta \tag{6.4}$$

The perturbed output y' is computed by taking a weighted average between the predictions of the true model f and a poisoning probability distribution η. The poisoning distribution η is computed with the objective of mis-training the adversary's clone model. This is done by maximizing the angular deviation between the weight gradients of the perturbed prediction with that of the original predictions of the model. α is a tunable parameter that controls the weightage given to the poisoned distribution and the true output of the model. Thus, increasing α allows the defender to trade off the accuracy of the model for increased security against model stealing attacks by increasing the amount of perturbation injected into the model's original predictions. Note that an inherent limitation of such defenses (including our proposed defense) is that they cannot be used in applications where the end-user is reliant on the confidence of the predictions for downstream processing since the predictions are perturbed to improve security.

6.4 Adaptive Misinformation

We propose *Adaptive Misinformation* (AM), an accuracy-constrained defense to protect against model stealing attacks. Our defense is based on the observation that existing model stealing attacks generate a large number of OOD examples to query the defender's model. This is because the adversary is data-limited and does not have access to a large dataset representative of the defender's training dataset. AM takes advantage of this observation by adaptively servicing queries that are OOD with *misinformation*, resulting in most of the attacker's queries being serviced with incorrect predictions. Consequently, a large fraction of the attacker's dataset is mislabeled, degrading the accuracy of the clone model trained on this poisoned dataset. Compared to existing perturbation based defenses like PP, our proposal has the following distinguishing qualities:

1. AM selectively modifies the predictions only for OOD queries, leaving the predictions unchanged for ID inputs. This is in contrast to prior works, which perturb the predictions indiscriminately for all inputs

2. In existing perturbation based defenses, there is a significant amount of correlation between the perturbed prediction y' and the original prediction y, which leaks information that can be exploited by an adversary (discussed further in Section subsubsection 6.5.2). In contrast, AM ensures that y' is uncorrelated with y for OOD queries and therefore avoids leaking information about the original predictions.

3. PP requires expensive gradient computations to determine the perturbations. In contrast, AM has low computational overheads and just requires evaluation of an auxiliary *misinformation* model.

These advantages allow our defense to achieve a better trade-off between classification accuracy and security compared to existing defenses, with a low computational overhead. Figure 6.3 shows the block diagram of our proposed Adaptive Misinformation defense. In addition to the defender's model f, there are three components that make up our defense: (1) An OOD detector (2) A misinformation function (f') (3) A mechanism to gradually switch between the predictions of f and f' depending on the input.

For an input query x, AM first determines if the input is ID or OOD. If the input is ID, the user is assumed to be benign and AM uses the predictions of f to service the request. On the other hand, if x an OOD input, the user is considered to be malicious and the query is serviced using the incorrect predictions generated from $\hat{f}$. In the remainder of this section, we explain the different components of our defense in more detail.

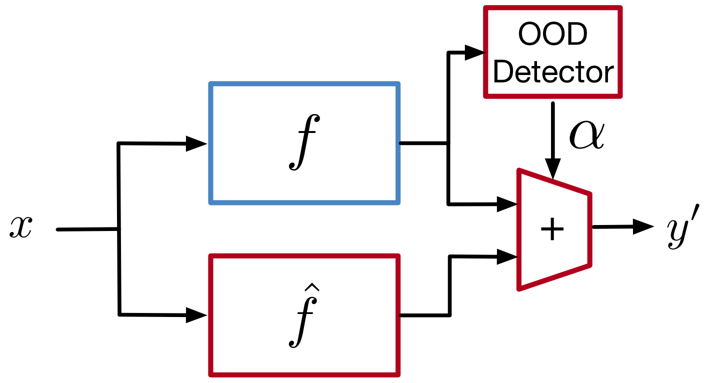


Figure 6.3: Adaptive Misinformation: We use an OOD detection mechanism to selectively service OOD inputs with the predictions of the misinformation function f', while the ID inputs are serviced with the original predictions of the model f.

6.4.1 Out of Distribution Detector

Out of Distribution detection is a well-studied problem in deep learning [80, 82, 83, 84, 85, 86], where the objective is to determine if an input received by the model during testing is dissimilar to the inputs seen during training. This can be used to detect and flag anomalous or hard to classify inputs which might require further examination or human intervention. A simple proposal to detect OOD examples [80] involves using the Maximum Softmax Probability (MSP) of the model. For a model that outputs a set of K output probabilities $\{y_i\}_{i=1}^{K}$ for an input x, OOD detection can be done by thresholding the MSP as shown in Equation 6.5.

$$Det(x) = \begin{cases} ID & \text{if } \max_i(y_i) > \tau \\ OOD & otherwise \end{cases} \tag{6.5}$$

The idea here is that the model produces confident predictions on ID inputs, similar to the ones seen during training and less confident predictions on OOD examples that are dissimilar to the training dataset. Outlier Exposure [87] is a recent work that improves the performance of the threshold-based detector by exposing the classifier to an auxilary dataset of outliers $\mathcal{D}_{out}$. The model is trained to produce uniform probability distribution ($\mathcal{U}$) on inputs from $\mathcal{D}_{out}$ by adding an extra term to the loss function during training.

$$\mathbb{E}_{(x,y)\in\mathcal{D}_{\text{in}}}\left[\mathcal{L}\left(f\left(x\right),y\right)\right] + \lambda\mathbb{E}_{x'\in\mathcal{D}_{\text{out}}}\left[\mathcal{L}\left(f\left(x'\right),\mathcal{U}\right)\right] \tag{6.6}$$

This ensures that the model produces accurate and confident predictions for inputs sampled from $\mathcal{D}_{in}$, while OOD examples produce less confident predictions, improving the ability of the detector to distinguish them. We train the defender's model with outlier exposure and use a threshold-based detector in our defense to perform OOD detection.

6.4.2 Misinformation Function

For queries which are deemed OOD by the detector, we want to provide incorrect predictions that are dissimilar to the predictions of the true model in order to deceive the adversary. We obtain the incorrect predictions by using a *misinformation function* $\hat{f}$, which is trained to minimize the probability of the correct class $\hat{f}(x, y)$. We can use categorical cross entropy to define a loss function to train $\hat{f}$ as described by Equation 6.7 to achieve this objective.

$$loss = \mathbb{E}_{(x,y)\in\mathcal{D}_{\text{in}}}\left[-log(1-\hat{f}(x,y))\right] \tag{6.7}$$

This loss term is minimized when the misinformation model produces low probability for the correct class y. We use this model to provide misleading information to OOD queries, making it harder for an adversary to train a clone model that obtains high accuracy on the classification task.

6.4.3 Adaptively Injecting Misinformation

Finally, we need a mechanism to gradually switch between the outputs of the defender's model (f) and the misinformation model ($\hat{f}$), depending on whether the input x is ID or OOD. In order to achieve this, we first pass x through an OOD detector, which simply requires computing the maximum softmax probability y_{max} of all the output classes.

$$y_{max} = \max_i(y_i) \tag{6.8}$$

A larger value of y_{max} indicates that the input is ID, while a smaller value indicates an OOD input. We use a threshold τ to classify between ID and OOD inputs as shown in Equation 6.5. The predictions of f and $\hat{f}$ are combined by using a reverse sigmoid function $S(x)$ to produce the final output probabilities y' as shown in Equation 6.9 and Equation 6.10.

$$y' = (1-\alpha)f(x;\theta) + (\alpha)\,\hat{f}(x;\hat{\theta}) \tag{6.9}$$

$$where \quad \alpha = S(y_{max}-\tau) \tag{6.10}$$

$$S(z) = \frac{1}{1+e^{\nu z}} \tag{6.11}$$

Thus for an ID input, with $y_{max} > \tau$, we obtain $\alpha < 0.5$ with $y' \to f(x;\theta)$ as $\alpha \to 0$. Similarly, for an OOD input with $y_{max} < \tau$, we obtain $\alpha > 0.5$ with $y' \to \hat{f}(x;\hat{\theta})$ as $\alpha \to 1$.

ν in Equation 6.11 indicates the growth rate of the sigmoid. We set $\nu = 1000$ for all of our experiments. Thus, an adversary accessing the model with OOD inputs obtains the predictions of f' instead of the true predictions of model f, while inputs from benign users of the service sending ID queries would be serviced by f. This results in the adversary's dataset containing examples which have been mislabeled, leading to a degradation in the accuracy of the clone model trained on this data.

Security vs Accuracy Trade-off: The OOD detector has a trade-off between true and false positive rates. In general, by lowering the value of the detector threshold τ, we can increase the number of OOD inputs classified correctly (true positive rate), which improves security as more OOD queries are serviced with misinformation. However, this also results in a higher number of ID inputs misclassified as OOD (false positive rate), leading to a greater number of ID inputs being serviced with misinformation, degrading the accuracy of the defender's model for benign ID examples. By appropriately setting the value of τ, the defender can pick a trade-off point between security and accuracy that satisfies the accuracy-constraint (Equation 6.3).

6.5 Experiments

We perform experiments to evaluate our defense against various model stealing attacks. Additionally, we compare AP against existing defenses and show that our defense offers better protection against model stealing compared to prior art. We describe our experimental setup followed by the results in this section.

6.5.1 Setup

Our experimental setup involves a defender who hosts a model f, trained for a specific classification task. The attacker aims to produce the clone model f', which achieves high classification accuracy on the same classification task. We briefly describe the classification tasks as well as the attacks and defenses that we use in our evaluations.

Datasets and model architecture: We focus on vision based classification tasks using DNNs in our experiments. Table Table 6.1 lists the various datasets and model architectures used to train the defender's model f as well as the test accuracy achieved by the models. As mentioned in section subsection 6.4.1, we train our model with *outlier exposure* [87] to improve the performance of OOD detection. For this purpose, we use KMNIST [88] for MNIST and FashionMNIST, ImageNet1k for CIFAR-10, and Indoor67 [89] for Flowers-17 [90] as the outlier datasets.

Dataset	DNN Architecture	Accuracy(%)
MNIST	LeNet	99.4
FashionMNIST	LeNet	91.47
CIFAR-10	ResNet-18	93.6
Flowers-17	ResNet-18	98.2

Table 6.1: Datasets and model architectures used to train the defender's model

Attacks: We evaluate our defense against two representative model stealing attacks:

1. KnockoffNets [5]: This attack uses surrogate data to perform model stealing. We use EMNISTLetters/EMNIST/CIFAR-100/ImageNet1k as the surrogate datasets to attack the MNIST/FashionMNIST/CIFAR-10/Flowers-17 models respectively. We assume a query budget of 50000 examples and train all clone models for 50 epochs.

2. Jacobian-Based Dataset Augmentation (JBDA) [67, 6]: This attack constructs a synthetic dataset by iteratively augmenting an initial set of *seed* examples with perturbed examples constructed using the jacobian of the clone model's loss function. We use a seed dataset of 150 examples with 6 rounds of augmentation to construct the adversary's dataset. Between each augmentation round, the clone model is trained for 10 epochs with $\lambda = 0.1$.

To improve the efficacy of the attacks, we use the same model architecture as the defender's model to train the clone model. Additionally, for the ResNet-18 models, we initialize the weights using a network pre-trained on the ImageNet dataset. We use a learning rate of 0.1 and 0.001 for LeNet and ResNet-18 models respectively.

Comparison with existing Defenses: Only a small number of defenses currently exist

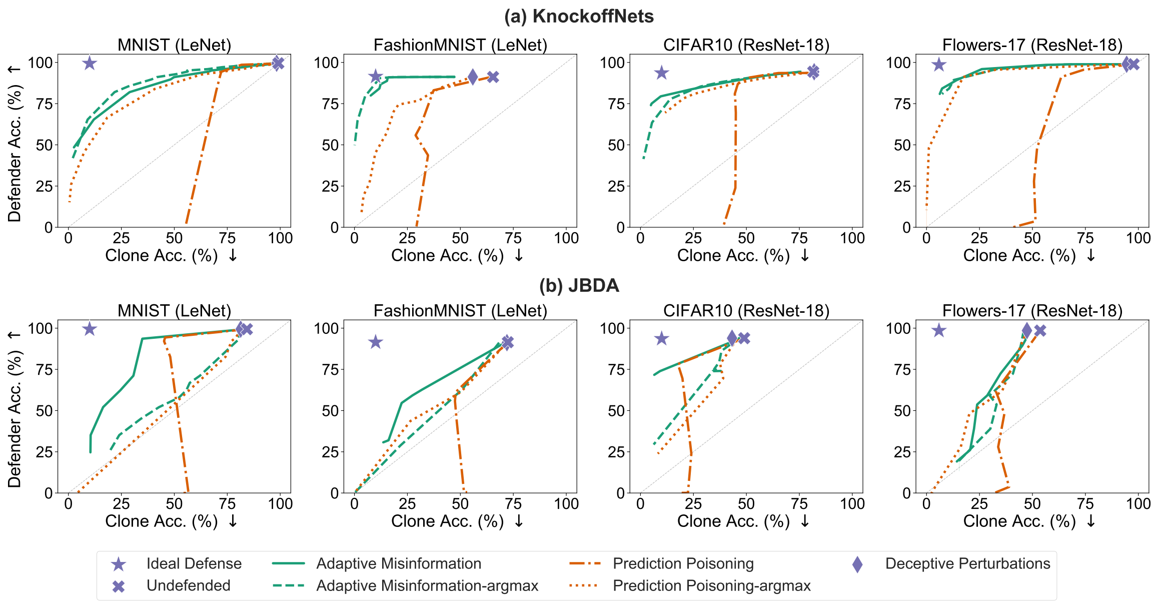


Figure 6.4: Defender Accuracy vs Clone Accuracy trade-off for defenses evaluated against two attacks: (a) *KnockoffNets* (b) *Jacobian Based Dataset Augmentation*. Perturbation based defenses can improve security (lower clone model accuracy) at the expense of reduced defender accuracy. Our proposal *Adaptive Misinformation* offers a better trade-off compared to existing defenses. E.g. in case of Flowers-17 dataset with KnockoffNets attack, PP achieves a clone accuracy of 63.6% with a defender accuracy of 91.1%. In comparison, AM yields a much lower clone accuracy of 14.3% (-49.3%) for the same defender accuracy, significantly improving the trade-off compared to PP.

for model stealing. We compare our defense against two recent perturbation-based defenses:

1. Deceptive Perturbation (DCP): This is a accuracy-preserving defense that adds deceptive perturbations to the predictions of the model but leaves the top-1 class unchanged[7]. Injecting perturbations removes information about prediction probabilities but preserves information about the *argmax* prediction of the model.

2. Prediction Poisoning (PP): This is an accuracy-constrained defense that perturbs the predictions of the model with an objective of mistraining the adversary's clone model[8]. Increasing the amount of perturbation allows the defender to trade off the model accuracy for increased security, similar to our defense.

6.5.2 Results

Our defense allows the defender to trade off the defender's model accuracy for increased security against model stealing attacks by varying the threshold τ of the OOD detector. We measure security by the accuracy of the clone model trained using model stealing attacks, with a lower clone-model accuracy indicating better security. We plot this trade-off curve of defender's model accuracy vs clone model accuracy evaluated against different attacks and show that our defense offers a better trade-off compared to existing defenses for various classification tasks.

KnockoffNets Attack

Figure 6.4a shows the trade-off curve of Adaptive Misinformation (AM) evaluated against the *KnockoffNets* attack. Our results show that AM is able to reduce clone model accuracy significantly, with only a small degradation in defender model accuracy. Additionally, we compare our results with the trade-offs offered by two existing defenses: Prediction Poisoning (PP) and Deceptive Perturbations (DCP). Note that PP allows a variable amount of perturbation to be added, leading to a trade-off curve whereas DCP has a fixed perturbation leading to a single trade-off point. We also plot the trade-off points for an ideal defense and an undefended model for reference.

Comparison with PP: Our results show that for a given defender accuracy, AM has a lower clone accuracy compared to PP for all datasets, offering a better trade-off between security and accuracy. For instance, AM lowers clone accuracy by 49.3% compared to PP with comparable defender accuracy (91.1%) in the case of the Flowers-17 dataset. We highlight and explain two key differences between the trade-off curves of AM and PP:

1. For PP, as we increase security (reduce clone model accuracy), the accuracy of the defender's model declines sharply to 0%, while AM retains high defender model accuracy. This is because PP indiscriminately perturbs predictions for all queries. As the amount of perturbation is increased, the top-1 class of y' changes to an incorrect class, leading to a

steep drop in the accuracy of benign inputs. AM avoids this problem by using an adaptive mechanism that only modifies the probabilities for OOD queries, allowing ID queries to retain a high classification accuracy.

2. For PP, even as the defender accuracy falls to 0%, the clone accuracy continues to be high (close to 50% for MNIST and Flowers-17). This is because there is a high correlation between the original predictions y and the perturbed predictions y'. We can quantify the correlations between y and y' by using Hellinger distance. We plot the CDF of Hellinger distance for a LeNet model (trained with MNIST) under KnockoffNets attack in Figure 6.5, comparing AM and PP for the same defender accuracy (92%). We find that

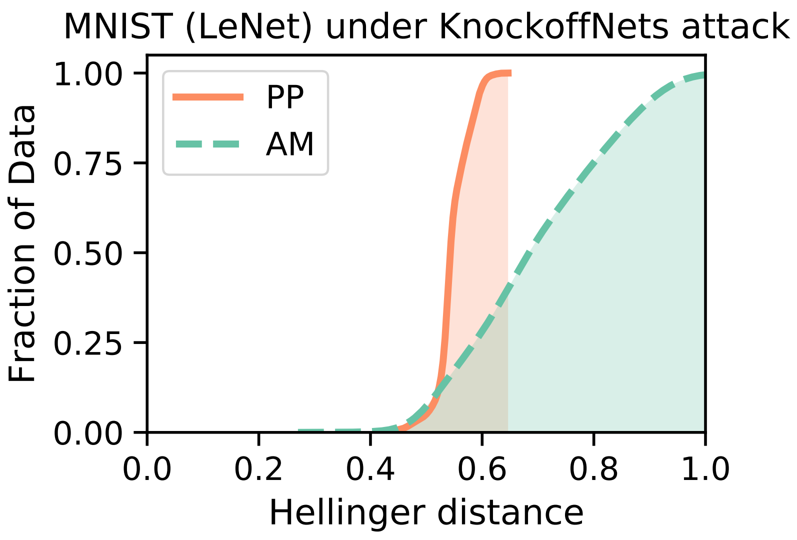


Figure 6.5: CDF of Hellinger distance comparing true prediction y and poisoned predictions y', computed for AM and PP for a LeNet model trained on MNIST under KnockoffNets attack, with comparable defender accuracy. The Hellinger distance is larger for AM compared to PP indicating less correlation between y' and y

the predictions of PP have lower Hellinger distance indicating a higher correlation with the true predictions of the defender's model. Sending correlated predictions allows the adversary to learn a clone model with higher accuracy. In contrast, AM avoids leaking information about the predictions of the original model f by switching to the predictions of the misinformation model $\hat{f}$ when an OOD input is encountered. Therefore, by using uncorrelated probabilities to service OOD queries, AM can offer better security without severely degrading the defender's model accuracy.

Comparison with DCP: For the DCP defense, since the top-1 class of the model remains unchanged after adding perturbations, the optimal strategy for the adversary is to use argmax labels from the perturbed predictions to train the clone model. Our results show that DCP only marginally improves security compared to an undefended model. In contrast, our defense is able to lower the clone model accuracy significantly. We also evaluate AM and PP with the attacker using an argmax-label strategy. We find this strategy to be less effective for the attacker, as it results in a lower accuracy compared to using the model's predictions to train the clone-model.

Jacobian Based Dataset Augmentation Attack

Figure 6.4b shows the trade-off curve for the JBDA attack. We find that this attack produces clones with lower accuracies compared to the KnockoffNets attack. The results for the PP defense shows that the defender accuracy quickly drops to 0%, even as the clone accuracy remains high, similar to the KnockoffNets attack. Our defense does not suffer from this problem and offers a better trade-off compared to PP. Additionally, we find that using the argmax labels offers a better clone model accuracy for this attack depending on the trade-off point. In this case, AM has a comparable or slightly better trade-off curve compared to PP. As before, the security offered by the DCP defense is marginally better than the undefended case, provided the attacker uses argmax labels to train the clone model.

6.6 Discussions on Adaptive Attacks

In this section, we discuss adaptive attacks against AM and provide simple solutions that can prevent such attacks.

Can the defense be treated as part of the black box to perform model stealing?

In order to train a high accuracy clone model with a limited query budget, the adversary needs to maximize the number of inputs that get serviced by f. Since our defense returns the predictions of f only for in-distribution inputs, just a small fraction of the adversary's queries (which are misclassified as in-distribution by the OOD detector) get serviced by f. In the absence of a way to reliably generate in-distribution inputs, the adversary would require a much larger query budget compared to other defenses to reach the desired level of clone accuracy. Furthermore, this would expose the adversary to other detection mechanisms. E.g. a user sending a large fraction of OOD examples can be blacklisted.

Can the adversary distinguish when the inputs are being serviced by f vs $\hat{f}$?

For an input to be serviced by f, it has to be classified as an ID example producing high MSP on f. Thus, the adversary can potentially use the confidence of the top-1 class as an indication of when the input is serviced by f and only use these inputs to train the clone model. While this can improve the accuracy of the clone model, the adversary would still need a much larger query budget since only a small fraction of the adversary's queries are serviced by f. Additionally, we can easily prevent such detection by smoothing the posterior probabilities of f or sharpening the probabilities of $\hat{f}$ to make the distribution of MSP identical between the outputs of f and $\hat{f}$.

6.7 Summary

We propose *Adaptive Misinformation* – a practical defense to protect against black-box model stealing attacks in the data-limited setting. We identify that existing model stealing attacks invariably use out of distribution data to query the target model. Our defense exploits this observation by identifying out of distribution inputs and selectively servicing such inputs with incorrect predictions (misinformation). Our evaluations against existing attacks show that AM degrades clone model accuracy by up to 40% with a minimal impact on the defender accuracy ($< 0.5\%$). In addition, our defense has lower computational overhead ($< 2\times$) and significantly better security vs accuracy trade-off (up to 49.3% reduction in clone accuracy) compared to existing defenses.

CHAPTER 7
CONCLUSION

The need for privacy poses a challenge to the training and deployment of deep learning models for several privacy-sensitive applications. To address this need, frameworks such as federated learning and split learning have been developed to train a model on distributed private data, and remote-inference is used to offer inference services while protecting the privacy of the model. These frameworks try to provide privacy by preventing the direct access of private model/data by an untrusted party. Unfortunately, these frameworks lack formal guarantees, making it hard to reason about their privacy properties. Furthermore, the limitations of prior attacks continue to provide an appearance of privacy for these frameworks, without any theoretically backed reason. The goal of this thesis is to investigate the privacy vulnerabilities that exist in these frameworks, and to show that the information exchanged during training/inference can indeed leak private data, even under challenging circumstances where prior attacks are ineffective.

First, in chapter 3, we develop *Cocktail Party Attack*– a gradient inversion attack on FL that can leak private training data from aggregate gradients. CPA uses a novel formulation of the attack that overcomes the limitations of prior works by scaling to high-dimensional inputs (like ImageNet) and large batch sizes (1024). The efficacy of the attack demonstrates that aggregation alone does not provide any privacy in FL.

Second, we propose a label-leakage attack *ExPLoit* in chapter 4 to demonstrate that split learning does not protect label privacy. *ExPLoit* uses the gradient information exchanged during split learning to frame the attack as a optimization problem, which enables the recovery of private labels with high accuracy of up to 99.53%.

Third, we propose a data-free model stealing attack *MAZE* to show that an adversary can create a functionally equivalent clone model, without using any data. The efficacy of our attack highlights the risk to model privacy in remote inference by showing that lack of access to data is not sufficient to prevent an adversary from carrying out a high-accuracy model stealing attack.

Finally, motivated by the need to protect model privacy, we propose the *adaptive misinformation* defense in chapter 6. AM protects the model from a data-limited adversary by servicing out-of-distribution queries with misinformation. AM retains high utility for benign queries while significantly deterring the effectiveness of model stealing attacks.

In summary, our attacks on FL, SL, and remote inference expose new vulnerabilities that can be exploited to leak private data, breaking the illusion of privacy created by these frameworks. The efficacy of our attacks demonstrates that it is possible to carry out practical attacks on these frameworks to leak private models and data, even under challenging settings that were previously considered to be hard. Our work highlights the need for defenses that can improve the privacy offered by these frameworks. Differential privacy (DP) is an effort in this direction that provides principled privacy guarantees by bounding information leakage during FL. However, its adoption remains challenging as it can cause significant degradation in utility. Reducing the utility loss of DP is an interesting direction for future exploration. Additionally, defenses like AM can be used to reduce the attack surface without causing significant degradation in utility in remote inference. Developing defenses that can provide principled bounds on privacy leakage while causing minimal degradation in utility remains an open challenge that is vital for the practical adoption of these frameworks.

REFERENCES

[1] B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[2] P. Kairouz *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[3] P. Vepakomma *et al.*, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[4] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[5] T. Orekondy *et al.*, "Knockoff nets: Stealing functionality of black-box models," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4954–4963.

[6] N. Papernot *et al.*, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[7] T. Lee *et al.*, "Defending against model stealing attacks using deceptive perturbations," *CoRR*, vol. abs/1806.00054, 2018. arXiv: 1806.00054.

[8] T. Orekondy *et al.*, "Prediction poisoning: Towards defenses against dnn model stealing attacks," in *ICLR*, 2020.

[9] F. Mireshghallah *et al.*, "Not all features are equal: Discovering essential features for preserving prediction privacy," *arXiv preprint arXiv:2003.12154*, 2020.

[10] F. Mireshghallah *et al.*, "Shredder: Learning noise distributions to protect inference privacy," in *ASPLOS*, 2020, pp. 3–18.

[11] F. McKeen *et al.*, "Innovative instructions and software model for isolated execution.," *Hasp@ isca*, vol. 10, no. 1, 2013.

[12] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," in *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, 2008, pp. 21–30.

[13] S. Kariyappa *et al.*, "Enabling inference privacy with adaptive noise injection," *arXiv preprint arXiv:2104.02261*, 2021.

[14] B. McMahan and D. Ramage, *Federated learning: Collaborative machine learning without centralized training data*, https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, Apr. 2017.

[15] R. G. Christophe Dupuy Jwala Dhamala, *Advances in trustworthy machine learning at alexa ai*, https://www.amazon.science/blog/advances-in-trustworthy-machine-learning-at-alexa-ai, Apr. 2022.

[16] L. Zhu *et al.*, “Deep leakage from gradients,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[17] C. Fu *et al.*, “Label inference attacks against vertical federated learning,”

[18] F. Boenisch *et al.*, “When the curious abandon honesty: Federated learning is not private,” *arXiv preprint arXiv:2112.02918*, 2021.

[19] L. Fowl *et al.*, “Robbing the fed: Directly obtaining private data in federated learning with modified models,” *arXiv preprint arXiv:2110.13057*, 2021.

[20] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” 2009.

[21] B. Zhao *et al.*, “Idlg: Improved deep leakage from gradients,” *arXiv preprint arXiv:2001.02610*, 2020.

[22] J. Geiping *et al.*, “Inverting gradients-how easy is it to break privacy in federated learning?” *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.

[23] L. I. Rudin *et al.*, “Nonlinear total variation based noise removal algorithms,” *Physica D: nonlinear phenomena*, vol. 60, no. 1-4, pp. 259–268, 1992.

[24] H. Yin *et al.*, “See through gradients: Image batch recovery via gradinversion,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 337–16 346.

[25] A. Hatamizadeh *et al.*, “Gradvit: Gradient inversion of vision transformers,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 021–10 030.

[26] K. Hsieh *et al.*, “The non-iid data quagmire of decentralized machine learning,” in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 4387–4398.

[27] T. Li *et al.*, “Federated learning: Challenges, methods, and future directions,” *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, 2020.

[28] H. Yin *et al.*, “Dreaming to distill: Data-free knowledge transfer via deepinversion,” *arXiv preprint arXiv:1912.08795*, 2019.

[29] J. Jeon *et al.*, “Gradient inversion with generative image prior,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 898–29 908, 2021.

[30] Y. Wen *et al.*, “Fishing for user data in large-batch federated learning via gradient magnification,” *arXiv preprint arXiv:2202.00580*, 2022.

[31] A. Hyvärinen and E. Oja, “Independent component analysis: Algorithms and applications,” *Neural networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

[32] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[33] R. Zhang *et al.*, “The unreasonable effectiveness of deep features as a perceptual metric,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.

[34] C. Dwork, A. Roth, *et al.*, “The algorithmic foundations of differential privacy,” *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[35] M. Abadi *et al.*, “Deep learning with differential privacy,” in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[36] S. Yeom *et al.*, “Privacy risk in machine learning: Analyzing the connection to overfitting,” in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, IEEE, 2018, pp. 268–282.

[37] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[38] O. Li *et al.*, “Label leakage and protection in two-party split learning,” *arXiv preprint arXiv:2102.08504*, 2021.

[39] E. Erdogan *et al.*, “Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning,” *arXiv preprint arXiv:2108.09033*, 2021.

[40] C. Fu *et al.*, “Label inference attacks against vertical federated learning,”

[41] X. He *et al.*, "Practical lessons from predicting clicks on ads at facebook," in *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, 2014, pp. 1–9.

[42] Y. Yang *et al.*, "Image clustering using local discriminant models and global integration," *IEEE Transactions on Image Processing*, vol. 19, no. 10, pp. 2761–2773, 2010.

[43] A. Halevy *et al.*, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

[44] I. Goodfellow *et al.*, "Explaining and harnessing adversarial examples," in *ICLR*, 2015.

[45] C. Szegedy *et al.*, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013. arXiv: 1312.6199.

[46] F. Tramèr *et al.*, "The space of transferable adversarial examples," *arXiv*, 2017.

[47] R. Shokri *et al.*, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2017, pp. 3–18.

[48] M. Nasr *et al.*, "Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks," *arXiv preprint arXiv:1812.00910*, 2018.

[49] M. Fredrikson *et al.*, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.

[50] Y. Zhang *et al.*, "The secret revealer: Generative model-inversion attacks against deep neural networks," *arXiv preprint arXiv:1911.07135*, 2019.

[51] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," *FoCM*, vol. 17, no. 2, pp. 527–566, 2017.

[52] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAMOPT*, vol. 23, no. 4, pp. 2341–2368, 2013.

[53] F. Tramèr *et al.*, "Stealing machine learning models via prediction apis," in *USENIX-SS*, 2016, pp. 601–618.

[54] D. Lowd and C. Meek, "Adversarial learning," in *KDD*, 2005, pp. 641–647.

[55] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 36–52.

[56] S. J. Oh *et al.*, "Towards reverse-engineering black-box neural networks," in *ICLR*, 2018.

[57] N. Roberts *et al.*, "Model weight theft with just noise inputs: The curious case of the petulant attacker," *arXiv preprint arXiv:1912.08987*, 2019.

[58] G. Hinton *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[59] P. Micaelli and A. J. Storkey, "Zero-shot knowledge transfer via adversarial belief matching," in *Advances in Neural Information Processing Systems*, 2019, pp. 9547–9557.

[60] G. Fang *et al.*, "Data-free adversarial distillation," *arXiv preprint arXiv:1912.11006*, 2019.

[61] I. Goodfellow *et al.*, "Generative adversarial nets," in *NeurIPS*, 2014.

[62] B. T. Polyak, "Introduction to optimization. optimization software," *Inc., Publications Division, New York*, 1987.

[63] J. C. Duchi *et al.*, "Optimal rates for zero-order convex optimization: The power of two function evaluations," *IEEE Trans. Inf. Theory.*, 2015.

[64] S. Liu *et al.*, "Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications," *arXiv preprint arXiv:1710.07804*, 2017.

[65] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, Elsevier, 1989, pp. 109–165.

[66] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[67] M. Juuti *et al.*, "Prada: Protecting against dnn model stealing attacks," in *IEEE-EuroSP*, IEEE, 2019, pp. 512–527.

[68] M. Arjovsky *et al.*, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[69] S. Sanyal *et al.*, "Towards data-free model stealing in a hard label setting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 15 284–15 293.

[70] X. Yuan *et al.*, “Es attack: Model stealing against deep neural networks without data hurdles,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2022.

[71] X. Li *et al.*, “Qair: Practical query-efficient black-box attacks for image retrieval,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3330–3339.

[72] Z. Yue *et al.*, “Black-box attacks on sequential recommenders via data-free model extraction,” in *Fifteenth ACM Conference on Recommender Systems*, 2021, pp. 44–54.

[73] J. Chen and M. I. Jordan, “Boundary attack++: Query-efficient decision-based adversarial attack,” *arXiv preprint arXiv:1904.02144*, vol. 2, no. 7, 2019.

[74] M. Cheng *et al.*, “Query-efficient hard-label black-box attack: An optimization-based approach,” *arXiv preprint arXiv:1807.04457*, 2018.

[75] A. Ilyas *et al.*, “Black-box adversarial attacks with limited queries and information,” *arXiv preprint arXiv:1804.08598*, 2018.

[76] C. Dwork, A. Roth, *et al.*, “The algorithmic foundations of differential privacy.,” *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3-4, pp. 211–407, 2014.

[77] S. Kariyappa and M. K. Qureshi, *Defending against model stealing attacks with adaptive misinformation*, 2019. arXiv: 1911.07100 `[stat.ML]`.

[78] S. Kariyappa *et al.*, “Maze: Data-free model stealing attack using zeroth-order gradient estimation,” in *CVPR*, 2021, pp. 13 814–13 823.

[79] F. Tramèr *et al.*, “Stealing machine learning models via prediction apis,” *CoRR*, vol. abs/1609.02943, 2016. arXiv: 1609.02943.

[80] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” *arXiv*, 2016.

[81] S. Chen *et al.*, “Stateful detection of black-box adversarial attacks,” *arXiv*, 2019.

[82] B. Lakshminarayanan *et al.*, “Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles,” *arXiv*,

[83] S. Liang *et al.*, “Principled detection of out-of-distribution examples in neural networks,” *CoRR*, vol. abs/1706.02690, 2017. arXiv: 1706.02690.

[84] K. Lee *et al.*, “Training Confidence-calibrated Classifiers for Detecting Out-of-Distribution Samples,” *arXiv*, 2017.

[85] A. Vyas *et al.*, “Out-of-distribution detection using an ensemble of self supervised leave-out classifiers,” *CoRR*, vol. abs/1809.03576, 2018. arXiv: 1809.03576.

[86] A. R. Dhamija *et al.*, “Reducing network agnostophobia,” *CoRR*, vol. abs/1811.04110, 2018. arXiv: 1811.04110.

[87] D. Hendrycks *et al.*, “Deep anomaly detection with outlier exposure,” *ICLR*, 2019.

[88] T. Clanuwat *et al.*, “Deep Learning for Classical Japanese Literature,” *arXiv*, arXiv:1812.01718, arXiv:1812.01718, 2018. arXiv: 1812.01718 `[cs.CV]`.

[89] A. Quattoni and A. Torralba, “Recognizing indoor scenes,” in *CVPR*, 2009, pp. 413–420.

[90] M.-E. Nilsback and A. Zisserman, “A visual vocabulary for flower classification,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, IEEE, vol. 2, 2006, pp. 1447–1454.