DATA PRIVACY IN THE MODERN MACHINE LEARNING ECOSYSTEM

A Dissertation
Presented to
The Academic Faculty

By

Stacey C. Truex

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology

May 2021

Copyright © Stacey C. Truex 2021
DATA PRIVACY IN THE MODERN MACHINE LEARNING ECOSYSTEM

Approved by:

Dr. Ling Liu, Advisor  
School of Computer Science  
*Georgia Institute of Technology*

Prof. Peter Swire  
Scheller College of Business  
*Georgia Institute of Technology*

Dr. Margaret Loper  
Georgia Tech Research Institute  
*Georgia Institute of Technology*

Dr. Calton Pu  
School of Computer Science  
*Georgia Institute of Technology*

Dr. Mustaque Ahamad  
School of Computer Science  
*Georgia Institute of Technology*

Date Approved: March 17, 2021
To Lauren Louise Truex. May you pursue all you can dream.
ACKNOWLEDGEMENTS

I owe my deepest appreciation to multiple people, each of whom was instrumental in the realization of this dissertation.

First, I would like to acknowledge the unwavering encouragement, support, and guidance of my advisor, Professor Ling Liu, throughout my time at Georgia Tech. Her advice and leadership has been invaluable in navigating all aspects of graduate school and I remain fortunate to count her as an advocate and mentor.

I would also like to thank Professors Ling Liu, Peter Swire, and Mustaque Ahamad for their support as letter writers and providing invaluable career guidance.

I would additionally like to thank all the members of my doctoral dissertation committee: Professor Ling Liu, Professor Peter Swire, Dr. Margaret Loper, Professor Calton Pu, and Professor Mustaque Ahamad. The critical eyes and thoughtful consideration of these individuals as well as the commitment of their valuable time has both contributed to the improvement of this work as well as helped to broaden my own research perspective and interests. And, while their thoughtful input has influenced this work, I would like to remark that I retain full responsibility for errors or inconsistencies, if any, that may remain in this document.

I have additionally been privileged to spend three summers as a research intern with IBM Research where I was fortunate enough to experience industrial research and engineering in practice. I would like to particularly thank Arun Iyengar for being a great mentor and good friend, for the great leadership and mentorship of Sandeep Gopisetty, and the collaboration of Heiko Ludwig, Nathalie Baracaldo, and the entire AI Security and Privacy Solutions team at IBM Research Almaden.

The research within this dissertation was also very generously sponsored by a number of organizations which I would like to thank including: the National Science Foundation (NSF CISE SaTC 1564097), the IBM PhD Fellowship, Georgia Institute of Technology Presiden-
tial, Haley, and IISP Cybersecurity Fellowships, and the Microsoft Research Women’s Fel-
lowship. However, any opinions, findings, and conclusions or recommendations expressed
in this material are those of the author(s) and do not necessarily reflect the views of the
National Science Foundation or other funding agencies and companies mentioned above.

I would also like to thank the other coauthors of the research presented in this disserta-
tion and every member, past and present, of the DiSL research group at Georgia Tech. It
was a distinct pleasure to work with such a broad range of dynamic researchers throughout
my time at Georgia Tech. I would like to express particular thanks to M. Emre Gursoy,
Vale Tolpegin, Lei Yu, Wenqi Wei, Ka-Ho Chow, and Yanzhao Wu for their research col-
laboration and friendship.

Finally, and most importantly, I must express my deepest gratitude to my husband
William Truex for his constant motivation, love, and support throughout our years in At-
lanta. I could not have endeavored to imagine, let alone complete, this journey without him
in my corner.
## TABLE OF CONTENTS

Acknowledgments ................................................................................................. iv

List of Tables ......................................................................................................... x

List of Figures ......................................................................................................... xii

Summary ................................................................................................................... xvi

Chapter 1: Introduction .......................................................................................... 1

Chapter 2: Privacy Risks in Machine Learning ...................................................... 8

2.1 Characterization of Privacy Risks in Machine Learning ................................. 9

2.1.1 Vulnerabilities in Inductive Learning ......................................................... 10

2.2 Inference-Based Privacy Attacks on Machine Learning Models ................... 14

2.2.1 Attack Definition ......................................................................................... 15

2.2.2 Threat Model and Assumptions ................................................................. 18

2.2.3 Membership Inference Risk: Data Skewness ............................................. 20

2.3 A Framework for Evaluating Membership Inference Risks in ML ............... 25

2.3.1 Defining the Membership Inference Attack .............................................. 25

2.3.2 Characterization of Membership Inference .............................................. 37

2.3.3 MPLens: System Overview .................................................................... 55
2.4 Privacy Vulnerabilities Specific to Distributed Machine Learning . . . . . . 57
  2.4.1 Membership Inference Insider Attacks in Federated Learning . . . . 60
  2.4.2 Poisoning Attacks in Federated Learning . . . . . . . . . . . . . . . 65

Chapter 3: Privacy-Preserving Machine Learning . . . . . . . . . . . . . . . . . 72
  3.1 Techniques for Privacy-Preserving Machine Learning . . . . . . . . . . . 72
    3.1.1 Attack-Based Mitigation Techniques . . . . . . . . . . . . . . . . . . 73
    3.1.2 Randomization-Based Privacy . . . . . . . . . . . . . . . . . . . . . 75
    3.1.3 Differential Privacy . . . . . . . . . . . . . . . . . . . . . . . . . . 77
    3.1.4 Secure Multiparty Computation . . . . . . . . . . . . . . . . . . . . 79
    3.1.5 Comparison of Approaches with Decision Tree Models . . . . . . . 87
    3.1.6 Federated Learning . . . . . . . . . . . . . . . . . . . . . . . . . . 93
  3.2 Application of Privacy-Preserving Techniques . . . . . . . . . . . . . . . . 93
    3.2.1 Differentially Private Deep Learning . . . . . . . . . . . . . . . . . . 94
    3.2.2 Privacy-Preserving Regression Training with SMC . . . . . . . . . . 100
    3.2.3 Privacy-Preserving Model Evaluation with SMC . . . . . . . . . . . 122
    3.2.4 Real World and System Considerations in Privacy-Preserving ML . . . 145
  3.3 Inference Attacks Against Privacy-Preserving Machine Learning . . . . . . 147
    3.3.1 Model and Problem Complexity . . . . . . . . . . . . . . . . . . . . 147
    3.3.2 Implications for Skewed Datasets . . . . . . . . . . . . . . . . . . . . 150

Chapter 4: Privacy-Preserving Federated Learning . . . . . . . . . . . . . . . . . 154
  4.1 Federated Learning Systems . . . . . . . . . . . . . . . . . . . . . . . . . . 154
    4.1.1 Federated Machine Learning . . . . . . . . . . . . . . . . . . . . . . 154
4.1.2 Privacy in Federated Machine Learning . . . . . . . . . . . . . . . 156
4.1.3 Deployment Considerations in Federated Learning Systems . . . . 159

4.2 Applying Privacy-Preserving Techniques to Federated Learning Systems . . 167
4.2.1 Developing Privacy-Preserving Federated Learning Algorithms . . . . 167
4.2.2 Practical Considerations in Privacy-Preserving FL Systems . . . . . 172

Chapter 5: The TSC-PFed Architecture for Privacy-Preserving FL . . . . . . . . 176
5.1 Components in the Private Federated Learning Architecture . . . . . . . . 176
5.2 Trust and Security Enhancements in TSC-PFed . . . . . . . . . . . . . . . 180
5.2.1 Introducing Trust in Private Federated Learning . . . . . . . . . . . . 180
5.2.2 Defending Against Label Flipping Attacks in TSC-PFed . . . . . . . . 196
5.3 Customizable Privacy in TSC-PFed . . . . . . . . . . . . . . . . . . . . . . 199
5.3.1 Customizing the Type of Privacy Protection . . . . . . . . . . . . . . 199
5.3.2 Tiered Participant Selection for Individualized Privacy Budgets . . . 204

Chapter 6: Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 207
6.1 Machine Learning Vulnerabilities . . . . . . . . . . . . . . . . . . . . . . . 207
6.1.1 Membership Inference Attacks . . . . . . . . . . . . . . . . . . . . . 207
6.1.2 Label Flipping Data Poisoning Attacks . . . . . . . . . . . . . . . . . 209
6.2 Privacy-Preserving Machine Learning . . . . . . . . . . . . . . . . . . . . 210
6.2.1 Differential Privacy to Mitigate Membership Inference . . . . . . . . 211
6.3 Privacy-Preserving Federated Learning . . . . . . . . . . . . . . . . . . . . 212

Chapter 7: Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 215
7.1 Summary ................................................................................. 215
7.2 Open Issues and Future Work ..................................................... 216
  7.2.1 Optimizing Private Systems ............................................... 217
  7.2.2 ML/AI Trust and Ethics ..................................................... 217
  7.2.3 Multi-Disciplinary Applications for ML Systems .................. 218
  7.2.4 Privacy Policy ................................................................. 218

References ....................................................................................... 241

Vita .................................................................................................. 242
LIST OF TABLES

2.1 Membership Inference attack accuracies targeting decision tree models. . . 17
2.2 Vulnerability to membership inference attacks for subsets of the LFW dataset. 21
2.3 Examples of target and attack model performance and confidence on vari-
ous images from the LFW dataset. . . . . . . . . . . . . . . . . . . . . . . 22
2.4 Comparison of datasets versus membership inference attack accuracy using
a decision tree model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
2.5 Precision of membership inference attack across 5 model types. . . . . . . 42
2.6 Accuracy of membership inference attack across 4 model types. . . . . . . 42
2.7 Model configuration reporting maximum membership inference attack ac-
curacy compared to attack accuracies when configuration is varied across
the target, generation, and attack models. . . . . . . . . . . . . . . . . . . . 43
2.8 Impact of noisy target data and noisy shadow data on membership inference
attack accuracy. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
2.9 Membership inference attack accuracy against a decision tree model trained
on the Purchases-20 dataset. . . . . . . . . . . . . . . . . . . . . . . . . . 50
2.10 Membership inference attack accuracy with various attack, data generation,
and target models using the CIFAR-100 dataset. . . . . . . . . . . . . . . . 52
2.11 Standard deviation of membership inference attack accuracy given (1) fixed
target models, (2) fixed generation models, and (3) fixed attack models. . . 53
2.12 Precision of insider membership inference attack within a 3-party federated
learning system with a decision tree target model. . . . . . . . . . . . . . 62
2.13 Loss in source class recall due to label flipping data poisoning attacks for three source → target class settings with differing baseline misclassification counts. ................................. 69

2.14 Changes in source class recall, target class recall, and total recall for all remaining classes (non-source, non-target) due to label flipping data poisoning attacks. ................................. 70

3.1 Effectiveness of mitigation techniques against the membership inference attack with a neural network target model trained on the Texas hospital admissions dataset. ................................. 74

3.2 High-level comparison of representative privacy-preserving decision tree learning approaches. ................................. 92

3.3 Actual time required (in seconds) for the online phase of our secure protocol training a linear regression predictive model. ................................. 121

3.4 Estimated time required (in seconds) for the offline phase of our secure protocol training a linear regression predictive model. ................................. 122

3.5 Results of the experiments for both securely and non-privately trained decision tree classifiers. The classification time is given as the computing time plus the number of half roundtrip times (RTT/2). ................................. 140

3.6 Results of the experiments for both securely and non-privately trained SVM and logistic regression classifiers with 2 classes. The classification time is given as the computing time plus the number of half roundtrip times (RTT/2) ................................. 141

3.7 Problem complexity vs membership inference vulnerability and differential privacy utility loss with a DP guarantee of $(\epsilon, \delta) = (10, 10^{-5})$. ................................. 149

4.1 Final source class recall when at least one malicious party conducting a label flipping data poisoning attack participates in the final round of training vs. when all participants in the final round of training are non-malicious. . . . 163
LIST OF FIGURES

2.1

Comparison of 2-D PCA for dog and truck classes in the CIFAR-10 dataset.
Plots show (1) training data, (2) data predicted as training data, and (3) data
predicted as non-training data by the membership inference attack. . . . . . 13

2.2

Minority class (automobile class) percentage vs membership inference vulnerability evaluated with the CIFAR-10 dataset. . . . . . . . . . . . . . . . 20

2.3

Development process for the membership inference attack model. . . . . . 26

2.4

Workflow of the membership inference attack generation process. . . . . . . 28

2.5

Development process for the creation of a shadow dataset. . . . . . . . . . 29

2.6

Shadow dataset development example. Figures adapted from [19] and [20].

2.7

Generation process for the development of an attack model training set. . . 33

2.8

Overview of the training and deployment processes for the membership
inference attack model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

2.9

Impact of size of the shadow dataset on membership inference attack accuracy with logistic regression models. . . . . . . . . . . . . . . . . . . . . . 49

31

2.10 Vulnerability of the Adult dataset to membership inference attack with decision tree models and highlighting the minority (>50K) class. . . . . . . . 54
2.11 Overview of the MPLens system for personalized evaluation of membership inference attack vulnerability. . . . . . . . . . . . . . . . . . . . . . . 55
2.12 Decision boundaries for 3 different participants in a federated system using
the Adult dataset. An area where the decision boundaries are significantly
different is highlighted as well as the training instances provided to each
participant relevant to the highlighted area. . . . . . . . . . . . . . . . . . . 62

xii


2.13 In-class distance between different parties compared with insider membership inference attack accuracy for the Adult dataset and a 3-party federated learning system. ............................................................. 64

2.14 Evaluation of label flipping data poisoning attack feasibility and impact of malicious participant percentage on attack effectiveness. The black bars are mean over the 10 runs and the green error bars denote standard deviation. . . 68

2.15 Relationship between global model accuracy and source class recall across changing percentages of malicious participants launching a label flipping data poisoning attack. ................................................................. 70

3.1 The Trusted Initializer functionality. ....................................................... 83

3.2 Protocol for secure distributed matrix multiplication in regression training. . 105

3.3 Protocol for secure truncation in regression training. .............................. 108

3.4 Protocol for securely computing the inverse of a covariance matrix. ....... 112

3.5 Protocol for securely substituting the trusted initializer in linear regression training. .............................................................. 115

3.6 The distributed matrix multiplication functionality. ............................... 123

3.7 The protocol for secure distributed matrix multiplication. .................... 123

3.8 The distributed comparison functionality. ............................ 125

3.9 The \( \text{arg max} \) functionality. ..................................................... 126

3.10 The secure \( \text{arg max} \) protocol. .................................................... 127

3.11 The \( \mathcal{F}_{\text{decomp}} \) functionality. .............................................. 128

3.12 The secure two-party bit-decomposition protocol. ............................ 129

3.13 The oblivious input selection functionality. ...................................... 131

3.14 The oblivious input selection protocol. ............................................. 132

3.15 Example of decision tree with 7 nodes and 2 classes. ......................... 133

3.16 The decision tree functionality. ...................................................... 133
3.17 The protocol for secure evaluation of a decision tree. .......................... 134

3.18 Impact of differentially private model training on minority classes. ......... 151

4.1 Source class recall by round for experiments with “early round poisoning”,
i.e., malicious participation only in the first 75 rounds ($r < 75$). The blue
line indicates the round at which malicious participation is no longer allowed. 162

4.2 Source class recall by round for experiments with “late round poisoning”,
i.e., malicious participation only after round 75 ($r \geq 75$). The blue line
indicates the round at which malicious participation starts. .................. 163

4.3 Evaluation of impact from malicious participants’ availability $\alpha$ on source
class recall. Results are averaged from 3 runs for each setting. ............... 165

4.4 Source class recall by round when malicious participants’ availability is
close to that of honest participants ($\alpha = 0.6$) vs significantly increased
($\alpha = 0.9$). The blue line indicates the round in which attack starts. ...... 166

4.5 Relationship between change in source class recall in consecutive rounds
versus change in number of malicious participants in consecutive rounds. .. 166

4.6 Decision tree training time in the presence of encryption. .................... 167

4.7 Overview of the TiFL system for participant selection in federated learning. 173

5.1 Overview of the workflow for differentially private federated DNN training
with TSC-PFed. .......................................................... 178

5.2 Overview of a step $s$ in the TSC-PFed system with the trust enhancement. . 181

5.3 Effect of privacy budgets on the overall F1-score for decision trees trained
with differential privacy. .............................................. 186

5.4 Effect of increasing number of parties in TSC-PFed on the overall F1-score
for decision trees. ....................................................... 187

5.5 Query epsilons values in decision tree training with varying rates of trust
(50 parties). Epsilon 1 is defined as the privacy budget for count queries
while Epsilon 2 is used for class counts. ............................... 188

5.6 Differentially private DNN training with MNIST data (10 parties and $\sigma =
8, (\epsilon, \delta) = (0.5, 10^{-5}))$. ............................................ 190
5.7 Impact of rate of trust on the degree of noise required for differentially private DNN training. ............................................................. 191

5.8 Linear SVM training with differential privacy (10 parties and \((\epsilon, \delta) = (5, 0.0059))\). ............................................................ 192

5.9 PCA plots with 2 components demonstrating the ability of Algorithm 9 to identify updates originating from a malicious versus honest participant. Blue Xs represent gradients from malicious participants while yellow Os represent gradients from honest participants. ............................................. 198

5.10 Performance of the LDP customization in TSC-PFed with \(\alpha\)-CLDP \((\alpha=1)\) for the Fashion MNIST dataset. ......................................................... 205
SUMMARY

The explosion of data collection and advances in artificial intelligence and machine learning have motivated a robust economy around cloud-based machine learning services. While such services provide opportunities for a broad array of individuals and companies to leverage the power of modern machine learning, they also introduce new vulnerabilities and privacy risks such as membership inference attacks, attribute inference attacks, and data poisoning attacks. Such attacks can allow for the malicious manipulation of model outcomes and cause serious violations to the data privacy of those individuals who have contributed to the model learning process.

Federated learning (FL) is a decentralized collaborative machine learning paradigm developed in response to such privacy risks. In FL, machine learning models are trained via multiple rounds of communication using a distributed computing platform. This allows FL participants to share only their local training model updates, therefore allowing individual participants’ private training data to remain local. While FL systems protect raw data from explicit disclosure, such systems remain vulnerable to inference based privacy risks, such as membership or attribute inference attacks, as well as private training data poisoning attacks. This dissertation research is dedicated to making original contributions towards addressing the growing public concern and legislative action surrounding data privacy in the modern machine learning ecosystem.

This dissertation research first takes a holistic approach to create a structured and comprehensive analysis of privacy risks in machine learning including a characterization of privacy vulnerabilities in both centralized and decentralized settings, an in-depth study on inference-based privacy attacks, specifically membership inference, against machine learning models, and a framework for evaluating membership inference risks in machine learning. The second contribution is the development of privacy-preserving machine learning solutions. This includes an analysis on privacy-preserving techniques in machine learn-
ing as well as protocols for the private training and evaluation of machine learning models under formal privacy frameworks including differential privacy and secure multiparty computation.

The next contribution consists of an analysis of the challenges and system considerations for extending privacy protection to the growing domain of federated learning. The final contribution is a proposed architecture, TSC-PFed, for trust and security enhanced customizable private federated learning. To this end, we propose the development of a privacy-enhanced federated learning system which incorporates both differential privacy and secure multiparty computation (SMC) to privately train accurate predictive models. Within our TSC-PFed system we include support for considering trust dynamics within a federated learning system which allow FL participants to decrease the degree of noise injected locally by a customizable trust factor $t$ while still adhering to a global differential privacy guarantee. We additionally provide support for security enhancements as well as customizable settings which allow participants to tune the type and level of privacy provided by TSC-PFed.
CHAPTER 1
INTRODUCTION

Machine learning and platforms for supporting machine learning workflows, known as machine learning-as-a-service (MLaaS), have seen an explosion of interest with the development of cloud platform services and the modern boom in data collection. Many cloud service providers, such as Amazon [1], Google [2], IBM [3], and Microsoft [4], have launched such MLaaS platforms to support this growing demand. These services allow consumers and application companies to leverage powerful machine learning and artificial intelligence technologies without requiring in-house domain expertise. Most MLaaS platforms offer two categories of services. (1) Machine learning model training. This type of service allows users and application companies to upload their datasets (often sensitive) and perform task-specific analysis including private machine learning and data analytics. The ultimate goal of this service is to construct one or more trained predictive models. (2) Hosting service for pre-trained models. This service provides pre-trained models with a prediction API. Consumers are able to select and query such APIs to obtain task specific data analytic results on their own query data.

With the exponential growth of digital data in governments, enterprises, and social media, there has also been a growing demand for data privacy protections, leading to legislation such as HIPAA [5], GDPR [6], and the 2018 California privacy law [7]. Such legislation puts limits on the sharing and transmission of the data analyzed by these platforms and used to train predictive models. All MLaaS providers and platforms are therefore subject to the compliance of such privacy regulations.

With the new opportunities of MLaaS and the growing attention on privacy compliance, we have seen a rapid increase in the study of potential vulnerabilities involved in deploying MLaaS platforms and services. Two specific categories of such attacks include (1) infer-
ence attacks and (2) adversarial machine learning attacks. With more mission critical cyber applications and systems using machine learning algorithms as a critical functional component, such vulnerabilities are a major and growing threat to the safety of cyber-systems in general and the trust and accountability of algorithmic decision making in particular.

As a result of such burgeoning privacy and security issues, it is imperative to design machine learning systems that enforce formal privacy, security, and robustness guarantees. In therefore crucial to develop state-of-the-art techniques for protecting privacy and evaluate their effectiveness in (1) maintaining model efficacy and (2) mitigating vulnerability to attack.

Modern systems are additionally becoming more frequently distributed with users who are more hesitant to (a) share their personal data or (b) allow those who collect their data to share it with others. This has led to the growing prominence of federated learning (FL) systems. These systems have unique privacy properties but also require the development of distinct privacy-preserving techniques when compared with traditional or centralized learning settings.

To that end this dissertation research and development have been conducted by making the following research propositions:

1. We need to create a systematic approach to understanding and analyzing privacy risks and new attack surfaces in computing systems with AI and ML as critical decision making components.

2. We need to create a theoretical foundation for developing privacy-enhancing methodologies and techniques for privacy preserving machine learning.

3. We need to leverage trust and security mechanisms in designing and developing privacy preserving federated learning architecture, algorithms and systems.

In the remaining of this chapter, I will briefly describe the problems and the solution approach with original contributions on each of the above three objectives.
In addressing the first research objective, a systematic approach to understanding and analyzing privacy risks, we address the following problems with corresponding solution approaches and contributions. (i) Can we identify areas of vulnerability in machine learning and why such vulnerabilities are important to the stakeholders of modern ML systems? In answering this question we consider the entire ecosystem of stakeholders to understand risk in machine learning including users, predictive model owners, as well as individuals represented by the data leveraged in the training of models by the model owners. We then identify both membership inference attacks as well as adversarial ML vulnerabilities in addition to how these risks impact the aforementioned stakeholders.

(ii) Can we develop a systematic approach to constructing and consequently evaluating one such vulnerability: the membership inference attack? We approach this issue by providing a formalized framework for the membership inference attack including attack definition, threat model and assumptions, as well as highlighting real world considerations such as the implications of data skewness on vulnerability. We consequently provide a foundation for scientifically analyzing and understanding the membership inference attack as well as demonstrate the importance of considering vulnerabilities at a more granular level when data is skewed.

(iii) How can we demystify the membership inference attack to specifically understand when and how membership inference attacks work and why certain models and datasets are more vulnerable? We approach our comprehensive analysis of the membership inference attack with extensive experimentation which allows us to characterize membership inference attack vulnerability. We are able to demonstrate that membership inference attacks are (a) data-driven, (b) transferable, and (c) impacted by model choice. We then reflect these factors in our privacy analysis and compliance evaluation system MPLens which allows machine learning practitioners to evaluate membership privacy through a multidimensional, customizable lens.

And, finally, (iv) extend our vulnerability analysis to include risks specific to the area
of distributed machine learning, an area which continues to see more and more prominence in modern systems. We consider both aspects of privacy attacks and adversarial ML vulnerability which may be impacted by the unique characteristics of distributed systems such as data distribution, change in attack landscape with the introduction of more knowledgeable system participants, and the loss of a centralized data authority. We demonstrate that both membership inference and adversarial ML attacks can persist in distributed systems through malicious system participants.

We then move to the second research hypothesis, methodologies and techniques for privacy-preserving machine learning. (i) Given the established vulnerability of ML systems, we seek to review state-of-the-art approaches to protecting privacy in machine learning and their efficacy in addressing vulnerabilities. We discuss formal and informal frameworks for privacy protection including attack-based techniques, randomization-based obfuscation, the differential privacy framework, and secure multiparty computation protocols and then provide a comparison of methods through representative approaches used to protect a specific, popular ML model: the decision tree. Our review of concepts and techniques in the space of privacy-preserving ML and discussion on state-of-the-art results consequently lay the foundation for our study in developing and delivering privacy-preserving machine learning as a service.

(ii) We aim to develop privacy-preserving protocols under the formal frameworks of differential privacy and secure multiparty computation which allow for model training and deployment for privacy sensitive big data science and engineering applications. We consider both model training including of both deep learning models as well as more traditional linear models in addition to the evaluation of a broad set of classifiers and demonstrate the complexity of applying privacy techniques in the development of privacy-preserving machine learning protocols and algorithms as well as how real world system considerations are impacted by the integration of such techniques. We provide privacy-preserving approaches to model training and evaluation which extend the state-of-the-art in privacy-preserving
ML as well as discussion and experimental evaluation to understand their usability in the context of real world machine learning as a service systems.

(iii) We want to evaluate the effectiveness of using differentially private learning algorithms in mitigating membership inference vulnerabilities. While differential privacy and SMC provide formal frameworks to analyze privacy protection we augment this with an experimental approach in evaluating effectiveness in mitigating membership inference attacks. We demonstrate that there are important trade-offs in implementing differentially private deep learning strategies particularly as vulnerability relates to model and problem complexity as well as the implications for skewed data.

We finally address the third research hypothesis, designing and developing a privacy preserving federated learning architecture. We first (i) identify attack vulnerabilities in the federated learning landscape. We accomplish this by first identifying the stakeholders in a federated learning system and then formalize the relevant privacy threats as well as deployment considerations unique to federated learning. Through this approach we provide a foundation for our study on privacy-preserving federated learning including developing a thread model, differentiating inference threads under this model, and establishment of deployment considerations such.

(ii) We then want to understand challenges in applying previously established privacy-preserving machine learning techniques to federated learning systems. We consider both theoretical implications of extending protocols to more distributed environments as well as practical elements such as variation in privacy policies or computing resources at various participants. This provides the framework for the discussion of challenges in extending privacy protection to the growing area of federated learning.

(iii) We finally formalize the federated learning environment considered in developing our private federated learning system TSC-PFed which allows for trust and security enhanced customizable private federated learning. We identify model and data agnostic components and their relationship to one another within a federated learning environment.
We then provide a general algorithm and architecture for federated learning in our system.

(iv) We then introduce components into the private federated learning architecture which leverage trust and account for security threats such as data poisoning adversarial attacks. We combine secure multiparty computation and differential privacy to allow participants to leverage known trust dynamics which allow for increased ML model accuracy while preserving privacy guarantees and introduce an update auditor to protect against malicious participants launching dangerous label flipping data poisoning. We consequently propose and develop a hybrid version of our FL algorithm as well as an approach for identifying malicious participants in TSC-PFed and provide experimental validation for both system enhancements.

(v) We also aim to consider the participant heterogeneity that is likely to exist in a real world federation in a deployed setting of TSC-PFed. We introduce modules into the TSC-PFed ecosystem which (a) allow users to customize the type of privacy protection provided and (b) provide a tiered participant selection approach which considers variation in privacy budgets. We outline and experimentally validate our approach to providing parameter-level (rather than member-level) privacy protection with local differential privacy as well as propose our tiered-based participant selection algorithm, both of which provide support for further customization by users of the TSC-PFed system.

In addressing our three research objectives, this dissertation is consequently organized as follows. First, in Chapter 2, we will provide a review of vulnerabilities in machine learning systems followed by a detailed introduction to membership inference attacks and a comprehensive characterization and framework for evaluating such attacks (membership inference) and finally a discussion on vulnerabilities specific to the growing area of distributed machine learning. Next, in Chapter 3 we will consider state-of-the-art techniques for protecting privacy and their effectiveness in machine learning as it relates to two objectives: (1) maintaining model efficacy and (2) mitigating vulnerability to attack. In Chapter 4 we then focus on the challenges and system considerations for extending privacy pro-
tection to the previously discussed growing domain of distributed machine learning and specifically the area of federated learning. Finally, in Chapter 5 we introduce our system for trust and security enhanced customizable private federated learning: TSC-PFed. This dissertation then concludes with a review of related work in Chapter 6 and concluding remarks in Chapter 7.
CHAPTER 2

PRIVACY RISKS IN MACHINE LEARNING

Machine learning and platforms for supporting machine learning workflows, known as machine learning-as-a-service (MLaaS), have seen an explosion of interest with the development of cloud platform services and the modern boom in data collection. Many cloud service providers, such as Amazon [1], Google [2], IBM [3], and Microsoft [4], have launched such MLaaS platforms to support this growing demand. These services allow consumers and application companies to leverage powerful machine learning and artificial intelligence technologies without requiring in-house domain expertise. Most MLaaS platforms offer two categories of services. (1) **Machine learning model training.** This type of service allows users and application companies to upload their datasets (often sensitive) and perform task-specific analysis including private machine learning and data analytics. The ultimate goal of this service is to construct one or more trained predictive models. (2) **Hosting service for pre-trained models.** This service provides pre-trained models with a prediction API. Consumers are able to select and query such APIs to obtain task specific data analytic results on their own query data.

With the exponential growth of digital data in governments, enterprises, and social media, there has also been a growing demand for data privacy protections, leading to legislation such as HIPAA [5], GDPR [6], and the 2018 California privacy law [7]. Such legislation puts limits on the sharing and transmission of the data analyzed by these platforms and used to train predictive models. All MLaaS providers and platforms are therefore subject to the compliance of such privacy regulations.

With the new opportunities of MLaaS and the growing attention on privacy compliance, we have seen a rapid increase in the study of potential vulnerabilities involved in deploying MLaaS platforms and services. Two specific categories of such attacks include (1) infer-
ence attacks and (2) adversarial machine learning attacks. With more mission critical cyber applications and systems using machine learning algorithms as a critical functional component, such vulnerabilities are a major and growing threat to the safety of cyber-systems in general and the trust and accountability of algorithmic decision making in particular.

In this chapter we review and evaluate vulnerabilities in machine learning with particular attention to privacy risks. This includes a general review of machine learning vulnerabilities, a detailed introduction to inference-based privacy attacks in machine learning, a comprehensive characterization and framework for evaluating one such attack (membership inference), and finally a discussion on vulnerabilities specific to the growing area of distributed machine learning.

2.1 Characterization of Privacy Risks in Machine Learning

Today, most industries are rapidly increasing the amount of data they collect while researchers continue to develop tools and techniques to mine valuable insights from this data. As more data is collected and mined by various machine learning (ML) tools and algorithms, data privacy awareness is increasing, and the need to protect personal privacy as well as organizational privacy is pushing to the forefront of many business and industry operations.

Consider one popular mining approach: inductive learning. Inductive learning is a predictive analytic method that learns a target model through iterative inductions over the training sample set. This is done by finding an approximation to an optimal hypothesis through optimizing the objective learning function. A popular objective function is defined by minimizing a given loss function. Formally, a training dataset of \( n \) samples is denoted by \( \{(x_1, c_1), \ldots, (x_n, c_n)\} \), in which \( c_i \) is the true label of \( x_i \). We also assume a target function \( c = \Gamma(x) \) for the training dataset, where \( c \) is the true label for the sample object \( x \) in the training set. An inductive learner produces a model \( y = g(x) \) which approximates the true function \( \Gamma(x) \) such that a given loss function \( L(c, y) \) is minimized, where \( y \) is the
label predicted by the model for the sample object $x$.

An optimal model is the one that minimizes the average loss defined by $L(c, y)$ for all samples in the training set, weighted by their posterior probability. The posterior probability, $P_c(y|x)$, is defined as the probability of class $y$ being the label of sample $x$. For many application specific problems, $c = \Gamma(x)$ is a non-deterministic function. That is, if $x$ is sampled repeatedly, different values of $c$ may be given. In this case, the optimal choice of the label for sample object $x$ among all candidate labels is the label $y_{ml}$ that minimizes the expected loss for a given sample $x$, i.e., $\forall y = f(x), \exists y_{ml} = f(x), y_{ml} \neq y$, s.t. $E_c(L(c, y_{ml})) \leq E_c(L(c, y))$.

Inductive learning has been shown to be a powerful data analytic approach. However, recent studies have also demonstrated that inductive learning has multiple vulnerabilities to attack.

2.1.1 Vulnerabilities in Inductive Learning

Consider a machine learning model deployment scenario in the context of automated medical diagnosis service. A company that provides automated medical assessments will likely maintain risk profiles for various diseases, including the DNA profiles, medical history, and job-specific information of previously diagnosed individuals. The first privacy concern in such a scenario is the risk profiles, which are sensitive by themselves. The second is the predictive machine learning model built from this sensitive risk profile data as the model often contains sensitive meta-data; therefore, revealing the model can compromise sensitive information and violate certain laws and regulations. A third privacy concern comes from the use of the model for evaluation. The consumers of such a predictive model service may submit either their own risk profile or, in the case of a medical provider, the risk profile of a patient for classification. Without proper protection, this service request leaks the sensitive data of consumers or their patients to the service provider. The classification result can also be highly sensitive. For example, revealing the classification result of a predictive
model for cancer diagnosis would compromise the consumer’s privacy. In this scenario it is critical to protect (1) the training dataset, (2) the predictive model, and (3) the evaluation and testing data and results.

Another important category of vulnerabilities to consider in modern machine learning is adversarial input attacks, also referred to as adversarial machine learning. Adversarial machine learning attacks are broadly classified into two categories: (1) evasion attacks wherein attackers aim to mislead pre-trained models and cause inaccurate output; and (2) poisoning attacks which aim to generate a poisonous trained model by manipulating its construction during the training process. Such poisoned models will misbehave at prediction time and can be deceived by the attacker. Adversarial deep learning research to date has been primarily centered on the generation of adversarial examples by injecting the minimal amount of perturbation to benign examples required to either (1) cause a pre-trained classification model to misclassify the query examples with high confidence or (2) cause a training algorithm to produce a learning model with inaccurate or toxic behavior.

We next discuss both privacy and adversarial vulnerabilities in inductive learning in more detail as well as the potential for linkage between the two risks.

Membership Inference Attacks

One specific threat to privacy consideration (1) the training dataset is the membership inference attack. To understand the risk of membership inference, let us consider the case of cancer diagnosis where a cancer treatment center has leveraged their large database of valuable patient data to train a predictive model which, when given a patient’s data as input, can predict cancer-related health outcomes. The treatment center then utilizes a cloud deployment option to create a service of their own wherein users can log in, provide their own health information, and receive predictions in return. A black-box membership inference attack considers a scenario wherein a user of such a prediction service is an adversary. This adversary can provide the health information of another individual $X$ and, based on
the model’s output, try to infer if $X$ is a cancer patient at the treatment center.

There are two primary parties who are interested in protecting against such membership inference attacks: patient $X$ and the cancer treatment center. Previous patients of the cancer treatment center, such as patient $X$, consider their membership private and do not want their patronage to be public knowledge. For example, consider the case of a treatment center patient, Alice. Let Alice be under consideration for a job at Bob’s company. Bob can leverage the cancer treatment center’s service to infer whether or not Alice is a patient. Upon learning of Alice’s inclusion in the cancer treatment center’s database, Bob decides not to hire Alice in favor of a candidate who, he believes, will have lower healthcare costs for the company.

In addition to concerns of patients such as Alice, we also must consider the interests of the cancer treatment center, the owner of the training dataset and the trained model under attack. In today’s market, across many domains, data is considered an organizational asset [8]. While internal data has always been a driver of decision making for most companies, the role of data has been moving steadily closer to the core of many industries. A company’s data therefore holds intrinsic value to the organization. Additionally, this training data is the source of the machine learning model under attack. Training this model not only requires front-end capital, time, and resources but also holds competitive business value for the treatment center. The treatment center may even charge a fee per evaluation. It is therefore essential from the cancer treatment center’s perspective to protect their private database.

*Adversarial Attacks*

Our second category of risk, adversarial ML, has triggered a flurry of attention and research efforts on understanding such deception attacks as well as developing defense methods as predictive models take critical roles in many sensitive or mission critical systems and application domains. Given that most of the prediction models targeted are hosted by MLaaS
providers and kept private with only a black-box access API, one common approach in developing adversarial example attacks is to use a substitute model of the target prediction model. This substitute model can be constructed in two steps. First, use membership inference methods to infer the training data of the target model and its distribution. Then, utilize this data and distribution information to train a substitute model.

Therefore, privacy vulnerabilities related to modern machine learning models should not only be considered within the strict definition of individual privacy concerns but should also be considered within the context of the ability of a malicious party to launch an effective adversarial attack against the model.

Specifically, in [9] we demonstrate how an adversary can leverage an effective membership inference attack model to develop a representative dataset for the training of an effective substitute model used in launching an adversarial attack. Figures 2.1a-2.1c provide visualization plots for the comparison of 2-D PCA given the images of the dog and truck classes in the CIFAR-10 dataset. The publicly available CIFAR-10 dataset contains 60,000 color images [10] categorized into 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Each image is formatted to be 32 x 32 and each class has 6,000 available images. The problem is therefore a 10-class classification problem where the task is to identify which of the 10 classes is depicted in a given image. The plots are divided relative to the membership inference attack prediction output including
the true target model training data (Figure 2.1a), the data predicted by the membership inference attack as training data (Figure 2.1b), and the data predicted as non-training data by the membership inference attack (Figure 2.1c).

These plots illustrate the accuracy of the distribution of instances predicted as in the target model’s training dataset through membership inference. They clearly demonstrate how even with the inclusion of false positives, an attacker can create a good representation of the training data distribution, particularly compared to those instances not predicted to be in the target training data. An attacker can therefore easily train a substitute model on this representative dataset which then enables the generation of adversarial examples by attacking this substitute model. Examples developed to successfully attack a substitute model trained on the instances in Figure 2.1b are likely to also be successful against a model trained on the instances in Figure 2.1a.

Given the significance of the vulnerabilities in modern machine learning systems, the rest of this chapter is organized as follows. Section 2.2 formalizes the membership inference attack including an attack definition as well as a demonstration of predictive model vulnerabilities, including under skewed data distributed settings. Section 2.3 proposes a framework for evaluating membership inference risk in machine learning which includes details on the development and deployment of the membership inference attack model as well as highlighting key vulnerability properties. Finally, Section 2.4 looks at privacy vulnerabilities specific to the growing area of distributed machine learning, including specific membership inference risks as well as the increased risk for a specific adversarial attack: label flipping.

2.2 Inference-Based Privacy Attacks on Machine Learning Models

Membership inference refers to the ability of an attacker to infer the membership of training examples used during model training. We call a membership inference a black-box attack if the attacker only has the access to the prediction API of a privately trained model hosted by
a MLaaS provider. A black-box attacker therefore does not have any knowledge of either the private training process or the privately trained model.

Specifically, a black-box membership inference attack considers a scenario wherein a user of the prediction service is an attacker. This attacker provides data of a target individual $X$ and, based on the model’s output, the attacker tries to infer if $X$ had been used in the training of the predictive model.

2.2.1 Attack Definition

In studying membership inference attacks there are two primary sets of processes at play: (1) the training, deployment, and use of the machine learning model which the attacker is targeting for inference and (2) the development and use of the membership inference attack. Each of these two elements has guiding pre-defined objectives impacting respective outputs.

Machine Learning Model Training and Prediction

The training of and prediction using the machine learning model which the attacker is targeting may be formalized as follows. Consider a dataset $D$ comprised of $n$ training instances $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with each instance $x_i$ containing $m$ features, denoted by $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,m})$, and a class value $y_i \in \mathbb{Z}_k$, where $k$ is a finite integer $\geq 2$. Let $F_t : \mathbb{R}^m \rightarrow \mathbb{R}^k$ be the target model trained using this dataset $D$. $F_t$ is then deployed as a service such that users can provide a feature vector $x \in \mathbb{R}^m$, and the service will then output a probability vector $p \in \mathbb{R}^k$ of the form $p = (p_1, p_2, \ldots, p_k)$, where $p_i \in [0, 1] \forall i$, and $\sum_{i=1}^{k} p_i = 1$. The prediction class label $y$ according to $F_t$ for a feature vector $x$ is the class with highest probability value in $p$. Therefore $y = \arg \max_{i \in \mathbb{Z}_k} F_t(x)$. 
Membership Inference Definition

Given some level of access to the trained model $F_t$ the attacker conducts his or her own training to develop a binary classifier which serves as the membership inference attack model. The most limited access environment in which an attacker may conduct the membership inference attack is the black-box access environment.

In machine learning-as-a-service systems, the machine learning service provider publishes the trained target model $F_t$ through a black-box access API, which accepts service requests from users in the form of a prediction query for input $x$ and returns the predicted class and the prediction probability vector. The input and output formats of the API are given by the service provider as part of a service agreement. However, $F_t$ and the dataset $D$ on which $F_t$ is trained remain private. Only the prediction API is exposed to users, thus ensuring only black-box access to $F_t$.

A question one may ask is: “How can an adversary with only black-box access to the prediction API perform membership inference attack without knowing anything about $F_t$ and the dataset $D$?” Recall from the details on inductive learning in Section 2.1 that $F_t$ is trained as an approximation of an ideal function $y = \Gamma(x)$ for a training dataset $D$, where $y$ is the true class for the sample instance $x \in D$. Let $y_c$ denote the output of a candidate model $F_c$. An optimal model is then the one that minimizes the average loss defined by a chosen loss function $L(y, y_c)$ for all samples in the training set $D$, weighted by their posterior probability. The posterior probability, $P_y(y_c|x)$, is defined as the probability of class $y_c$ being the label of sample $x$. For many application specific problems, $y = \Gamma(x)$ is a non-deterministic function. That is, if $x$ is sampled repeatedly, different values of $y$ may be given. In this case, the optimal choice of the class for sample object $x$ among all candidate class labels is the class that minimizes the expected loss for a given sample $x$. The target model $F_t$ is then assigned to be the optimal model for the training dataset $D$ upon the completion of the training and testing phases.

Our target function therefore creates a decision boundary which separates the feature
space \( \mathbb{R}^m \) into \( k \) sets in which each set is associated with a candidate class value in \( \mathbb{Z}_k \). As these sets and corresponding class assignments are chosen to minimize the loss function \( L(y, y_c) \) over the training dataset \( D \), the decision boundaries are strongly informed by the training dataset \( D \) and will in turn be the core of the trained machine learning model \( F_t \).

Let us consider an attacker with such black-box access to \( F_t \). Given only a query input \( x \) and output \( F_t(x) \) from some target model \( F_t \) trained using a dataset \( D \), the membership inference attacker attempts to identify whether or not \( x \in D \).

Table 2.1: Membership Inference attack accuracies targeting decision tree models.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Accuracy of Membership Inference (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult</td>
<td>59.89</td>
</tr>
<tr>
<td>MNIST</td>
<td>61.75</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>90.44</td>
</tr>
<tr>
<td>Purchases-10</td>
<td>82.29</td>
</tr>
<tr>
<td>Purchases-20</td>
<td>88.98</td>
</tr>
<tr>
<td>Purchases-50</td>
<td>93.71</td>
</tr>
<tr>
<td>Purchases-100</td>
<td>95.74</td>
</tr>
</tbody>
</table>

Many different datasets and model types have demonstrated vulnerability to membership inference attacks in black-box settings. Table 2.1 reports 5 accuracy results for black-box attackers targeting decision tree models for problems ranging from binary classification (Adult) to 100-class classification (Purchases-100). The Adult dataset is available on the UCI Machine Learning Repository [11] and contains 48,842 instances described by 14 different features. The feature set contains both continuous (ex: age, hours per week) and discrete (ex: education, marital status) values. This dataset presents a binary classification problem wherein one wishes to identify if an individual makes \( \geq \$50K \) or \( < \$50K \) in yearly salary. MNIST is a publicly available dataset containing 70,000 images of handwritten digits [12]. Each image is formatted to be 32 x 32 and processed such that the digit is at the center of the image. The MNIST dataset constitutes a 10-class classification problem where the task is to identify which digit between 0 and 9, inclusive, is contained within a given image. We additionally developed a number of purchases datasets similar to what
was done in [13]. The purchases datasets were developed from the Kaggle Acquire Val-
ued Shoppers Challenge dataset which contains the shopping history of several thousand
individuals. From this dataset we create new datasets wherein each instance represents an
individual and each feature represents a particular product. If an individual has purchased
this product, there will be a 1 for the feature and otherwise a 0. The instances are then
clustered into different shopping profile types. These cluster assignments are treated as
the classes. We created different datasets with 10, 20, 50, and 100 shopping profile types.
The classification problem then becomes: given a shopper’s purchase history, identify their
shopping profile type. Finally, recall from Section 2.1.1 that the CIFAR-10 dataset is a
10-class image classification problem.

Finally,

These results demonstrate both the viability of membership inference attacks as well as
the variation in vulnerability between datasets. This accentuates the need for practitioners
to evaluate their system’s specific vulnerability.

2.2.2 Threat Model and Assumptions

Adversarial Knowledge

The membership inference threat model can be characterized based on prior adversarial
knowledge. We broadly categorize this adversarial knowledge into three categories: black-
box, grey-box, and white-box data knowledge.

Black-Box Knowledge. An adversary is said to have black-box knowledge when the ad-
versary does not have any specialized knowledge of the training data. However, black-box
knowledge may include the input and output of the service API as well as publicly avail-
able information about the target prediction model $F_t$. For example, if the service provider
is our cancer treatment center, then the adversary may have access to relevant statistics
curated by the government and published for the public good including demographic infor-
mation, such as the likeliness of different age groups or genders to contract certain cancers,
or clinical information, such as the prevalence of co-occurrence of different diseases with various cancer types.

**Grey-Box Knowledge.** We characterize grey-box knowledge as *specialized* population-level knowledge. This may include population-level statistics that describe the distribution of features in the target model’s training data. For example, in addition to publicly available distributions on the average age of cancer patients (black-box knowledge), the adversary may know the average age of a cancer patient seen at the target treatment center (*specific* knowledge).

**White-Box Knowledge.** White-box knowledge characterizes scenarios where the training data for $F_t$ is sampled from a constrained population or in a skewed fashion such that an adversary has access to some versions of real data in the training data $D$ of the target model $F_t$ or some leaked portion of $D$ but not the complete training set $D$. For example, a noisy version of the real data may be accessible which resembles $D$ with the addition of some noise or missing values [13]. Adversaries with white-box knowledge can therefore develop or access **true** “in” samples and employ active learning techniques on these known samples to develop a very accurate dataset to mirror $D$.

Recently, researchers showed similar membership inference vulnerability in settings where attackers have white-box access to the target model, including the output from the intermediate layers of a pre-trained neural network model or the gradients for the target instance [14]. Interestingly, this study showed that the intermediate layer outputs, in most cases, do not lead to significant improvements in attack accuracy. For example, with the CIFAR-100 dataset and AlexNet model structure, a black-box attack achieves 74.6% accuracy while the white-box attack achieves 75.18% accuracy. This result further supports the understanding that the attackers can gain sufficient knowledge from only the black-box access to the pre-trained models which is common in MLaaS platforms. Attackers do not require either full or even partial knowledge of the pre-trained target model as black-box attacks include the *primary* source of membership inference vulnerability.
The setting we use to formulate membership inference attacks and to characterize adverse effects and divergence of membership inferences throughout the rest of this thesis is the black-box setting, wherein adversaries are limited to (i) publicly available information, (ii) black-box queries to the prediction API, and (iii) the output of classification prediction from the target model $F_t$.

### 2.2.3 Membership Inference Risk: Data Skewness

Related work prior to our study in [9] considered the membership inference attack either under uniform class distributions or without specific consideration of the impact of any data skewness. However, our work demonstrates that the risk of membership inference vulnerability can vary when class representation is skewed. Specifically, minority classes can display increased risk to membership inference attack as models struggle to more effectively generalize past the training data when fewer instances are given.

For example, in Figure 2.2 we investigate the impact of data skewness on membership inference vulnerability by controlling the representation of a single class. We reduce the automobile images from the CIFAR-10 dataset to only 1% of the data and then increase the representation until the dataset is again balanced with automobiles representing 10% of the training images. We then plot the aggregate membership inference vulnerability which is the overall membership inference attack accuracy evaluated across all classes as well as the
vulnerability of just the automobile class.

Figure 2.2 demonstrates that in cases where the automobile class constitutes 5% or less of the total training dataset, i.e. the automobile class has fewer than half as many instances as each of the other classes, this skewness will result in the automobile class displaying more severe vulnerability to membership inference attack.

Interestingly, the automobile class displays lower membership inference vulnerability than the average vulnerability reported in the CIFAR-10 dataset when the dataset is balanced. However, when the automobile class becomes a minority class with fewer than half the instances of each of the other classes, the vulnerability shifts to be greater than that reported by the overall model. This gap becomes greater as continued decreased representation results in continued increased vulnerability for the automobile class.

Table 2.2: Vulnerability to membership inference attacks for subsets of the LFW dataset.

<table>
<thead>
<tr>
<th>Target Population</th>
<th>Attack Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>70.14</td>
</tr>
<tr>
<td>Male Images</td>
<td>68.18</td>
</tr>
<tr>
<td>Female Images</td>
<td>76.85</td>
</tr>
<tr>
<td>White Race Images</td>
<td>62.77</td>
</tr>
<tr>
<td>Racial Minority Images</td>
<td>89.90</td>
</tr>
</tbody>
</table>

Table 2.2 additionally shows the vulnerability of a deep neural network target model trained on the Labeled Faces in the Wild (LFW) dataset to membership inference attacks. The LFW database contains face photographs for unconstrained face recognition with more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1,680 of the people pictured have two or more distinct photos in the data set. Each person is then labeled with a gender and race (including mixed races). Data is then selected for the top 22 classes which were represented with a sufficient number of data points.

We analyze this vulnerability by breaking down the aggregated vulnerability across the top 22 classes into four different (non-disjoint) subsets of the LFW dataset: Male, Female,
White Race, and Racial Minority. We observe that the training examples of racial minorities experience the highest attack success rate (89.90%) and are thus highly vulnerable to membership inference attacks compared to images of white individuals. Similarly, female images, which represent less than 25% of the training data, demonstrate higher average vulnerability (76.85%) compared with images of males (68.18%).

To provide deeper insight and more intuitive illustration for the increased vulnerability of minority groups under membership inference attacks, Table 2.3 provides 7 individual examples of images in the LFW dataset targeted by the membership inference attack. That is, given a query with each example image, the target model predicts the individual’s race and gender (22 separate classes) which the attack model, a binary classifier, uses to predict if that image was “in” or “out” of the target model’s training dataset. The last row reports the ground truth as to whether or not the image was in the target model training set.

Table 2.3: Examples of target and attack model performance and confidence on various images from the LFW dataset.

<table>
<thead>
<tr>
<th>Target Confidence (%)</th>
<th>Attacker Confidence (%)</th>
<th>In Training Data?</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ 99.99</td>
<td>✓ 86.10</td>
<td>in</td>
</tr>
<tr>
<td>✓ 65.81</td>
<td>✓ 50.49</td>
<td>out</td>
</tr>
<tr>
<td>✓ 72.56</td>
<td>✓ 61.85</td>
<td>out</td>
</tr>
<tr>
<td>✓ 62.30</td>
<td>✓ 72.06</td>
<td>out</td>
</tr>
<tr>
<td>✓ 99.99</td>
<td>✓ 56.40</td>
<td>in</td>
</tr>
<tr>
<td>✓ 99.63</td>
<td>✓ 99.88</td>
<td>out</td>
</tr>
<tr>
<td>✓ 98.38</td>
<td>✓ 53.29</td>
<td>out</td>
</tr>
</tbody>
</table>

Through Table 2.3, we highlight how minority populations are more likely to be identified by an attacker with a higher degree of confidence. We next discuss each example from left to right to articulate the impact of data skewness on model vulnerability to the membership inference attack.

For the 1st image, the target model is highly confident with its prediction and its prediction is indeed correct. Using the membership inference attack model, the attacker predicts that the model must have seen this example with high confidence and it succeeds in the
For the 2nd image, the target model is less confident with its prediction, although the prediction outcome is correct. The attacker succeeds in the membership inference attack because it correctly predicts that this example is not in the training set, though the attacker’s confidence on this membership inference is much less certain (close to 50%) compared to that of the attack to the 1st image. We conjecture that the relatively low prediction confidence by the target model may likely contribute to the fact that the attacker is unable to obtain a high confidence for his membership inference attack.

The 3rd image is predicted by the target model correctly with a confidence of 72.56%, which is about 11.5% more confidence than that for the 2nd image. However, the attacker wrongly predicts that the example is in the training set when the ground truth shows that this example is not in the training set. Assuming the same logic as with the 1st image, i.e., the confidence and accuracy of target model prediction may indicate that the image was in the training dataset, could have caused the attacker to be misled.

For the 4th image, the target model has an incorrect prediction with the confidence of 62.30%. The attacker correctly predicts that this example is not in the training set. It is clear that a somewhat confident and yet incorrect prediction by the target model is likely to result in high attacker confidence that this minority individual has not been seen during the training.

These four images highlight the compounding downfall for minority populations. Models are more likely to overfit these populations. This leads to poor test accuracy for these populations and makes them more vulnerable to attack. As the 3rd example shows, the way to fool attackers is to have an accurate target model that can show reasonable confidence when classifying minority test images.

We next compare the results from the previous four images which represented minority classes with results from three images representing the majority class (white male images). For the 5th image in Table 2.3, the target model predicts correctly with high confidence and
the attacker is able to correctly predict that the image was in the training data. This result can be interpreted through comparison with the attacker performance for the 1st image. For the 5th query image which is from the majority group, the attacker predicts correctly that the image has been seen in training with barely over 50% in confidence, showing relatively high uncertainty compared with the 1st image from a minority class. This indicates that model accuracy and confidence are weaker indicators with respect to membership inference vulnerability for the majority class.

For the 6th image, the target model produces an incorrect prediction with high confidence. The attacker is very confident that the query image is not in the training dataset, which is indeed the truth. This demonstrates rare a potential vulnerability for the majority class: When the target model has high confidence in an inaccurate prediction, an attack model is able to confidently succeed in the membership inference attack. Through this example and the above analysis, we see that the majority classes have two advantages compared to the minority classes: (1) it is less common for the target model to demonstrate this vulnerability of misclassification with high confidence; and (2) for majority classes, the accuracy and privacy are aligned rather than as competing objectives.

For the 7th image, the target model makes a correct prediction with high confidence. The attacker makes the incorrect prediction that the example was in the training dataset of the target model. But the truth is that the example is not in the training set. This membership attack failed and is the flip side of the 5th image, with both reporting low attack confidence. Again by comparing with the 1st image of minority, it shows how model confidence and accuracy may lead to membership inference vulnerability for the minority classes in a way that is not true for the majority classes.

As our experimental results have demonstrated, membership inference presents a practical risk to multiple machine learning models (Table 2.1, can inform adversarial attackers (Figures 2.1a-2.1c) and provides a vector of increased vulnerability to minority populations (Tables 2.2 and 2.3 and Figure 2.2), we next provide a comprehensive analysis in
demystifying the membership inference attack and propose our framework for evaluating membership inference risks in machine learning.

2.3 A Framework for Evaluating Membership Inference Risks in ML

We now present our work in developing a framework for evaluating membership inference risks. We first describe a systematic approach to constructing a membership inference attack model with a general formulation of each component of the attack model generation framework. We show that generating a membership inference attack model is a complex and multi-step strategic process. Second, to understand when and how membership inference attacks work and why certain models and datasets are more vulnerable, we take a holistic approach with extensive empirical evidence to study and characterize membership inference attacks across different target model types, different types of training datasets, and different combinations of model types for generating attack training datasets and attack models. Finally, we present our privacy analysis and compliance evaluation system, MPLens, which investigates Membership Privacy through a multi-dimensional Lens. MPLens aims to expose membership inference vulnerabilities, including those unique to varying distributions of the training data. Our privacy analysis system can serve for both MLaaS providers and data scientists to conduct privacy analysis and privacy compliance evaluation.

2.3.1 Defining the Membership Inference Attack

Recall that the problem of membership inference can be formally defined as follows: given a query input $x$ and black-box access to the target model $F$, the membership inference attack answers the question of whether $x \in D$ is true or false. The attack is successful if the attacker can determine with high confidence that $x \in D$ is true or with high confidence that $x \notin D$.

At the most abstract level, membership inference attack models are binary classifiers.
Given an instance $x$ and a target model $F_t$, the goal of a membership inference attack model is to identify whether or not $x$ was contained within the dataset $D$ used to train $F_t$.

Let $D$ consist of $n$ training instances $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ where $x_i$ consists of $m$ features, denoted by $x_i = (x_{i,1}, x_{i,2}, ..., x_{i,m})$, and $y_i \in \mathbb{Z}_k$, where $k$ is a finite integer value $\geq 2$. Let $F_t : \mathbb{R}^m \rightarrow \mathbb{R}^k$ be the target model trained using this dataset $D$. Given a particular feature vector $x \in \mathbb{R}^m$, $F_t$ will then output a probability vector $p \in \mathbb{R}^k$ of the form $p = (p_1, p_2, ..., p_k)$, where $p_i \in [0, 1] \forall i$, and $\sum_{i=1}^{k} p_i = 1$. The prediction class label $y$ for a feature vector $x$ is the class with highest probability value in $p$. Therefore $y = \text{arg max}_{i \in \mathbb{Z}_k} F_t(x)$.

Given the adversary’s black-box access to $F_t$ via the prediction service API, an adversary is able to query $F_t$ with any number of instances to receive corresponding probability vectors. The adversary uses this probing access, along with any prior knowledge, to generate $I$, a representation of adversarial knowledge of $D$. The first building block for implementing a black-box membership inference attack is to leverage $I$ to generate a synthetic labeled dataset $D'$ to mirror the data in $D$. This synthetic, labeled dataset $D'$ is artificially simulated and called a shadow dataset of $D$. Although the word “shadow” was borrowed...
from shadow copying for systems creating back up data copies [15], the shadow dataset in our context should be thought of as a synthetic version of the real training dataset \( D \). \( D' \) is then used to generate an attack training dataset \( D^* \), which is required to train the final membership attack model, a binary classifier \( F_a \).

Figure 2.3 highlights these three primary phases in the development of the membership inference attack: (1) development of a shadow dataset, (2) generation of an attack model training dataset, and (3) training and deployment of the membership inference attack model. In phase (1) the attacker’s goal is to develop a dataset denoted as \( D' \) which closely emulates, or shadows, the dataset \( D \) which was used to train the target model. In phase (2) the attacker uses this shadow dataset to develop a shadow model which is considered to emulate the behavior of the target model. The attacker may then observe the behavior of this shadow model in response to instances which the attacker knows were given during training versus those that were not. This behavior is used to develop an attack dataset which captures the difference between output for instances in the training data and those previously unseen by the model. Finally, in phase (3), this attack data is used to generate a binary classifier which provides predictions on whether an instance was previously known to a model based on that model’s output from that instance. We now discuss each of these phases in further detail.

Generating the Membership Attack Model

Generating Shadow Data and Substitute Models  In the shadow model technique, an attacker must first generate or access a shadow dataset, a synthetic labeled dataset \( D' \) to mirror the data in \( D \). While [13] and [16] both outline potential approaches to generating such a synthetic dataset from scratch, we would like to note that in many cases, attackers may also have examples of their own which can be used as seeds for the shadow data generation process or to bootstrap their shadow dataset. Consider our example of the financial institution. A competitor to the target institution may in fact have their own customer data,
which could be leveraged to bootstrap a shadow dataset.

Once the attacker has developed the shadow dataset $D'$, the next phase of the membership inference attack is to leverage $D'$ to train and observe a series of shadow models. Specifically, the shadow dataset is used to train multiple shadow models each of which is designed to emulate the behavior of the target model. Each shadow model is trained on a subset of the shadow dataset $D'$. As the attacker knows which portion of $D'$ was provided to each shadow model, the attacker may then observe the shadow models’ behavior in response to instances which were in their training set versus behavior in response to those that were held out.

Given a target model $F_t$, its training dataset $D$, and black-box adversarial knowledge, the development of a shadow dataset $D'$ is the first step in generating a membership inference attack model. $D'$ consists of $n'$ training instances $(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_{n'}, y'_{n'})$ where each $x'_i$ consists of $m$ features equivalent to those in $D$ and each $y'_i$ is a predicted class label in $\mathbb{Z}_k$. Note that $k$ and $m$ are known via the service API and thus consistent across $D$ and $D'$. The cardinality of $D$, however, remains unknown and therefore $n$ and $n'$ are likely to differ. The shadow dataset generation process leverages the prediction service API to manage the creation and control the quality of $D'$, as shown in Figure 2.5.

**API Probing.** While the training set $D$ and the cardinality (size) $n$ of $D$ are unknown to
an adversary, the adversary can probe the service API to reveal structural information, such as the number of features $m$, the data types of those features, and the number of classes $k$. This knowledge can be obtained by the adversary through sending in trial query instances and observing responses.

We refer to the complete set of adversarial knowledge as $I$. This includes prior knowledge as well as that inferred from this API probing. We do not state any limitations on $I$ except that $D \not\subset I$, as the membership inference attack becomes trivial when $D \subset I$. The cost of launching an effective membership inference attack includes the work of this probing phase by the adversary.

As a result of this API probing, the adversary can construct a skeleton dataset $D'$, which is similar to $D$ in structure and, ideally, any $x' \in D'$ should be a viable instance that could be included in $D$.

There are several ways to generate a quality shadow dataset with a small number of query probing attempts. Below we highlight four categories of techniques: statistics-based, active learning-based, query-based, and region-based generation.

**Statistics-Based Generation.** In statistics-based generation, the adversary leverages population-level statistics of the features in $D$ to create samples for $D'$. Given known dis-
tributions for features, an adversary may conduct random sampling to construct these new samples. Features may be treated independently where an instance is generated through $m + 1$ random samplings of $m + 1$ distributions, each distribution corresponding to either a different feature or the class label. Alternatively, sampling may account for feature relationships. This may be done, for example, when our adversary has knowledge of statistics on disease co-occurrence.

*Active Learning-Based Generation.* Active learning is a technique developed in the semi-supervised machine learning domain [17]. Active learning has been developed to address the problem of a largely unlabeled training dataset when assigning accurate labels is an expensive task. For example, in the development of a spam filter one may have access to a large number of unlabeled emails [18]. It is a very expensive proposal to suggest that a human read millions of emails to provide labels, yet labels are necessary to develop an accurate filter. To address this problem, representative samples are selected and labeled. Given this subset of training instances which now have accurate labels, an automated process takes over and propagates the label logic to other instances. Active learning techniques may be combined with statistics-based generation in black-box or grey-box data knowledge scenarios where a large number of samples may be generated through random sampling of the features but class labels assigned through intervention by the adversary followed by active learning.

*Query-Based Generation.* When using query-based generation, an adversary will generate a random sample and then query the target model. The target model will then provide a probability vector output. In this case the adversary will want to identify instances for which the machine learning service provides a class label with relatively high confidence. That is, the adversary will search for instances in which the output $p$ has a $\max(p)$ value above some predefined threshold. This, again, may be combined with other techniques. For example, query-based generation may be used to provide a seed for active learning or statistics-based generation may inform the development of the instances sent to the service.
Region-Based Generation. Region-based generation follows a clustering-based logic. Given an instance $x$ with label $y$, region-based generation will seek to generate instances $x'$ where $\text{dist}(x, x')$ is below some pre-determined threshold for a pre-chosen distance function $\text{dist}$. The new instance $x'$ is then assigned the same label $y$. One way an adversary may use region-based generation is in conjunction with white-box data knowledge. Given knowledge of some instances either in $D$ or very similar to those in $D$ an adversary can use region-based generation to expand this knowledge into a larger number of highly accurate instances to construct $D'$.

Several factors may determine which concrete technique will be chosen by an adversary, such as the knowledge contained in $I$ or the query probing results. For example, an adversary who has grey-box data knowledge may be more likely to rely on statistics-based generation due to the specificity of the available statistics to $D$ whereas an adversary with black-box knowledge may want to augment statistics-based generation with query-based generation due to lower confidence in their non-specific, population-level knowledge of $D$. On the other hand, active learning-based or region-based generation would likely be a popular choice for adversaries with white-box data knowledge where the adversary may leverage their insider knowledge of $D$. These are just some of the considerations in the development of an effective membership inference attack model.

Figure 2.6: Shadow dataset development example. Figures adapted from [19] and [20].
In Figure 2.6 we show an example of the shadow dataset development process using a combination of query-based and region-based techniques for an adversary with black-box data knowledge. The adversary first randomly generates a starting point \( d_s \) using distributions in \( I \). This point \( d_s \) is then queried to the service provider which provides \( p_{d_s} \) in return. The point \( d_s \) is then updated randomly and queried again. This is continued until a confidence threshold or stopping condition is reached. That is, the process stops when either (1) a point \( d_f \) is found such that \( \max(p_{d_f}) \) meets a predefined confidence threshold or (2) the point has been updated and queried without meeting the confidence threshold the maximum number of times as determined by the attacker. If condition (2) is met a new \( d_s \) is chosen until the process results in condition (1) and an accepted point \( d_f \).

A hyper-cube is then constructed surrounding \( d_f \). A set of new samples \( \{d^1_f, d^2_f, d^3_f, \ldots\} \) are then generated by randomly sampling from the hyper-cube region. Each point \( d^i_f \) is assigned the class \( \arg\max_{i \in \mathbb{Z}} p_{d_f} \) and added, along with \( d_f \) to the shadow dataset \( D' \). The entire process is then repeated beginning with sampling a new starting point from \( I \) and ending with a new set of samples added to \( D' \). The adversary will continue this repetition until a satisfactory number of samples \( n' \) have been added to \( D' \).

**Generating Attack Datasets and Models** Attackers use the observations of the shadow models to develop an attack dataset \( D^* \) which captures the difference between the output generated by the shadow models for instances included in the training data and those previously unseen by models.

Once the attack dataset \( D^* \) has been developed, \( D^* \) is used to generate a binary classifier \( F_a \) which provides predictions on whether an instance was previously known to a model based on the model’s output from that instance. At attack time this binary classifier may the be deployed against the target model service in a black-box setting. The attack model takes as input prediction vectors of the same structure as those provided by the shadow models and contained within \( D^* \) and produces as output a prediction of 0 or 1 representing “out”
and “in” respectively with the former indicating an instance that was not in the training dataset of the target model and the latter indicating an instance that was included.

Figure 2.7: Generation process for the development of an attack model training set.

Upon the completion of the development of the shadow dataset $D'$, the adversary will proceed to utilize the shadow dataset $D'$ to develop the membership attack dataset for training a binary classifier as the final attack model, as shown in Figure 2.7. Given that each instance in $D'$ consists of a feature-vector and its known-class, denoted by $(x', y')$, the adversary can define an attack generation function denoted by $F_g : (\mathbb{R}^m, \mathbb{Z}_k) \rightarrow (\mathbb{R}^k, \mathbb{Z}_2)$. $F_g$ takes a feature vector-known class pair $(x', y')$ as input and outputs an attack training instance, consisting of two pieces of information: a probability vector $p = (p_1, p_2, ..., p_k)$ and a binary class label, indicating “in” or “out”. There are several approaches to generate $F_g$ using $D'$. For example, the adversary can train a new model over $D'$ which simulates the private target model $F_t$. In this case, we call $F_g$ a shadow model of $F_t$. Given that the adversary does not know the original training set $D$ nor the size of $D$, the adversary may leverage ensemble learning techniques [21], such as data partition-based ensembles, model-based ensembles, or hybrid ensemble models, to improve the quality of the shadow model with the goal of accurately simulating the target model $F_t$. The attack model training
set generation function $F_g : (\mathbb{R}^m, \mathbb{Z}_k) \rightarrow (\mathbb{R}^k, \mathbb{Z}_2)$ can therefore be viewed as an ensemble of a set of shadow models. These shadow models seek to characterize the decision boundary of the target model. More specifically, the shadow models aim at mirroring the sensitivity of the target decision boundary to individual instances.

**Effect of Ensemble Methods.** Combining multiple different models reduces the risk of choosing the wrong hypothesis within the hypothesis space of a particular problem. Also, multiple models allows for more effective local search, which many machine learning algorithms perform in various ways, and limits the impact of the local optima problem. Finally, a combining of chosen hypotheses allows for an expansion of the hypothesis space [21]. Two common ways to accomplish this diversity are bagging and boosting.

Bagging is accomplished by either drawing a sample of training examples from the original dataset randomly and with replacement, or by creating disjoint subsets of the original training data called cross-validated committees [22]. A commonly used implementation of bagging is the Random Forest ensemble model [23].

Using the boosting technique, a set of weights is maintained for each instance within the training dataset. Each model is then trained iteratively to minimize the weighted error of the training dataset. The weights of the training instances are updated to put more emphasis on the misclassified examples. The adaboost ensemble model [24] is a popular implementation of boosting.

The use of multiple models via ensemble learning for attack data generation decreases the risk of choosing the wrong hypothesis. Given an adversary who has only black-box knowledge of the target model (regardless of the adversary’s knowledge type of the dataset $D$), there is no guaranteed method to reproducing the target model’s behavior on $D$. A diversity of generation models will minimize the risk that the adversary is only capturing one behavior type or candidate decision boundary shape. This again accentuates the need for boosting or bagging to ensure that the model set is diverse.

Consider a data partition based ensemble approach [13]. The adversary partitions the
shadow dataset $D'$ into $D'_\text{train}$ and $D'_\text{test}$. $D'_\text{train}$ is then divided into $q$ partitions ($q > 1$), one partition for each shadow model. Each partition of $D'_\text{train}$ will then be used to train a single shadow model $F_{g_i}$. Here we intentionally do not specify the machine learning model type of $F_{g_i}$ as this is yet another design choice made by an adversary. The decision may be informed if the adversary knows the model type of $F_t$ or chosen using some other criteria, we leave this decision unspecified to remove the constraint that an adversary must know the model type of $F_t$. Next, $D'_\text{test}$ will be evaluated against $F_{g_i}$. The corresponding outputs will then be labeled as “\text{out}”. Additionally, a sample of size $|D'_\text{test}|$ is taken from the $D'_\text{train}$ partition used to train $F_{g_i}$ and evaluated against $F_{g_i}$ with the corresponding outputs labeled as “\text{in}”. By combining these output-label pairs, we obtain the attack train data $D^*$.

Figure 2.7 highlights the workflow of generating an attack model training set. An adversary may choose a single model for efficiency or an ensemble of models to increase the size or generality of $D^*$. An adversary may diversify model types in an ensemble when $F_t$’s type is unknown. If an ensemble is used, there are choices on size, sampling, and aggregation that must be made and can be informed by the adversary’s knowledge of the target in various ways. We stress that while we formulate the membership inference attack using a general model, there exist many implementation variants.

![Figure 2.8](image-url)  
Figure 2.8: Overview of the training and deployment processes for the membership inference attack model.
The attack model training set $D^*$ contains the outputs from the generation function $F_g : (\mathbb{R}^m, Z_k) \rightarrow (\mathbb{R}^k, Z_2)$. $D^*$ consists of $n^*$ instances, $(x_1^*, y_1^*), (x_2^*, y_2^*), \ldots, (x_{n^*}^*, y_{n^*}^*)$, and each instance is the output of $F_g$ for some input $(x', y') \in (\mathbb{R}^m, Z_k)$. This attack model training dataset $D^*$ will then be used to generate the final attack model $F_a : \mathbb{R}^k \rightarrow \{\text{in}, \text{out}\}$, which takes as input a probability vector output for an instance $x$ and outputs a binary classification of “in” or “out”. Similarly, we make no assumptions on how $D^*$ is used to inform $F_a$ but rather say that $D^*$ is available to an adversary during the generation of the membership attack model $F_a$. A number of machine learning models and techniques can be leveraged to train a binary classification-based attack model $F_a$ using $D^*$. $F_a$ can then be deployed against the output of the target model $F_t$ such that, ideally, given an instance $x$, $F_a(F_t(x)) = \text{“in”}$ if the instance $x \in D$ and $F_a(F_t(x)) = \text{“out”}$ if $x \notin D$. Figure 2.8 visualizes this final phase of developing a membership inference attack.

Regardless of how complex the chosen training process is, whether it is a distance evaluation or a complex machine learning model, this phase produces the final attack model $F_a$ which will be deployed for membership inference attack against $D$ in real time.

The attack generation process can vary significantly based on the power of the attacker. For example, the attack proposed in [25] requires knowledge of the training error of $F_t$. The attack technique proposed in [13], however, requires computational power and involves the training of multiple machine learning models. The techniques proposed in [26] are different still in that they require the attacker to develop effective threshold values. Figure 2.4 gives a workflow sketch of membership inference attack generation algorithm. We use the shadow model technique documented in [13] and [16] to describe the attack generation process of membership inference attacks, while noting that many of the processes may be applicable to other attack generation techniques.

The totality of these two phases: (1) generating shadow data and substitute models and (2) generating attack datasets and models, constitute the primary processes for constructing the membership inference attack.
**Attack Value vs. Attack Cost**

It is generally accepted that systems security should never operate as an all-or-nothing mechanism. Systems must always seek to optimize two sets of factors: cost of defense vs value of assets to the system owner and cost of attack vs value of assets to the adversary. These principles hold true to deployed learning systems with respect to membership inference attacks.

In the context of membership inference attacks, value can be characterized by attack accuracy as an evaluation of what level of leakage is present in $F_t$ or what amount of knowledge an attacker can expect to gain. The cost of attack refers to the knowledge and work necessary for an adversary to launch a successful attack. When characterizing cost, we consider knowledge cost as well as development cost. For example, to launch an effective attack, how much knowledge of $D$ or $F_t$ does an adversary need? Gaining this knowledge should be considered a type of cost for the adversary. Alternatively, cost can also be characterized by how computationally expensive it is to develop an effective attack model $F_a$.

The accuracy of an attack can be characterized by a number of metrics. For example, an accuracy measure may be defined as the likelihood that $F_a$ correctly identifies $x \in D$ or $x \notin D$. Alternatively, accuracy can be defined as a precision measure which indicates the fraction of the instances inferred as members that are indeed members of the training dataset $D$, focusing on the probability $Pr[x \in D | F_a$ says $x \in D]$.

**2.3.2 Characterization of Membership Inference**

We have thus far provided a general formulation of membership inference attacks. In this section, we characterize such attacks through a systematic evaluation of a variety of machine learning models and model combinations using multiple datasets. We show that membership inference vulnerability is data-driven and its attack models are largely transferable. Although the target model is a dominating factor in determining vulnerability,
attack data generation techniques need not explicitly mirror the target model. Finally, we show that membership inference attacks can persist as insider attacks in federated systems.

**Impact of Data-Driven Factors on Membership Inference**

Table 2.4: Comparison of datasets versus membership inference attack accuracy using a decision tree model.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>In-Class Standard Deviation</th>
<th>Number of Classes</th>
<th>Accuracy of Membership Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult</td>
<td>0.1433</td>
<td>2</td>
<td>59.89</td>
</tr>
<tr>
<td>MNIST</td>
<td>0.1586</td>
<td>10</td>
<td>61.75</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>0.2301</td>
<td>10</td>
<td>90.44</td>
</tr>
<tr>
<td>Purchases-10</td>
<td>0.3820</td>
<td>10</td>
<td>82.29</td>
</tr>
<tr>
<td>Purchases-20</td>
<td>0.3873</td>
<td>20</td>
<td>88.98</td>
</tr>
<tr>
<td>Purchases-50</td>
<td>0.3873</td>
<td>50</td>
<td>93.71</td>
</tr>
<tr>
<td>Purchases-100</td>
<td>0.3832</td>
<td>100</td>
<td>95.74</td>
</tr>
</tbody>
</table>

Recall from Section 2.3.1 that there are many choices an adversary can make during the three phases of developing the membership inference attack model, each of which may impact the model accuracy and consequently the attack success rate. Two elements that the adversary cannot control, however, are the training dataset $D$ and the target model $F_t$. Our experimental results show that membership inference attacks are data-driven. That is, the make-up of $D$ strongly correlates with the corresponding target model $F_t$'s vulnerability to membership inference attacks. Potential targets of such attacks can therefore use knowledge of their data to evaluate their risk. Table 2.4 compares the seven different datasets measuring three characteristics: (1) feature distribution, (2) number of classes (size of $k$), and (3) accuracy of (vulnerability to) membership inference.

The number of classes is important as it characterizes the number of regions into which the input space $\mathbb{R}^m$ is divided. The more classes, the smaller each region. With smaller regions, there is less uninformed space and in fact the regions will more tightly surround the provided training instances in $D$. This will make any single instance more likely to alter the decision boundary as space is “tighter” between the regions. If an instance is more
likely to impact the decision boundary of the target model, then an adversary will be more likely to infer its inclusion in the training dataset.

Another side of this same argument can be seen in the in-class standard deviation metric. This value captures feature distributions by addressing the following: within a dataset, given all instances of the same class, how similar are the feature vectors? The Adult and MNIST datasets, for example, have significantly lower standard deviations than the Purchases datasets despite having more instances of each class. This demonstrates more uniformity within classes of the Adult and MNIST datasets as compared to the Purchases datasets. If an instance is exceedingly similar to other instances of the same class then it will be less likely to noticeably impact the decision boundary during training. If, however, instances within a class are notably different, then the inclusion of each instance may significantly impact the decision boundary. Therefore, the uniformity of the target training data within each class will, in addition to the number of classes, impact an adversary’s ability to identify the inclusion of a particular instance.

The variation in attack accuracy results reported in Table 2.4 from experiments using the same attack development process targeting decision tree models trained on each of the seven datasets demonstrates that factors in the dataset such as the in-class standard deviation and the number of classes contribute to a model’s susceptibility to membership inference. We make two interesting observations. First, the impact of in-class standard deviation varies for different types of datasets and different scales of \( k \)-class classification problems. Second, membership inference vulnerability is related to a number of factors, many of which are tied to the dataset itself. Concretely, for image datasets the in-class standard deviation increases as the complexity of the dataset increases. For example, MNIST represents images of grey scale for 10 single digits and CIFAR-10 represents color images of 10 different complex entities ranging from flowers to animals. It is observed that the attacks have higher success rate for more complex images which also show larger in-class SD. On the other hand, consider the Purchase datasets where the value of \( k \) varies from 2 to
10, 20, 50 and, finally, 100. The attack accuracy increases as $k$ increases without the standard deviation increasing. This shows, from a different perspective, that attack accuracy is higher for more complex classification problems. For example, consider the Purchases-50 and Purchases-100 datasets. The increase in attack accuracy demonstrates that in-class standard deviation may not always be a dominant factor in membership inference vulnerability, other data-driven factors, such as the complexity of the classification problem and the complexity of the dataset itself, can also be important factors in membership inference vulnerability.

Table 2.4 shows more explicitly the impact on the attack accuracy of two parameters: the variation of in-class standard deviation and the parameter $k$ for the $k$-class classification problem. While we want to highlight the impact of these important parameters, neither the in-class standard deviation nor the value of $k$ is the sole dominating factor. Rather, the results in Table 2.4 demonstrate that the membership inference vulnerability is not isolated to the learning process itself but also highly related to several attributes of the training dataset. This leads us to characterize membership inference attacks as data-driven.

*Impact of Model-Based Factors on Membership Inference*

The most widely acknowledged factor impacting vulnerability to membership inference attacks is the degree of overfitting in the trained target model. Shokri et al. [13] demonstrate that the more over-fitted a deep neural network model is, the more it leaks under membership inference attacks. Yeom et al. [25] investigated the role of overfitting from both the theoretical and the experimental perspectives. While their results confirm that models become more vulnerable as they overfit more severely, the authors also state that overfitting is not the only factor leading to model vulnerability under the membership inference attack. We further demonstrate the impact that the type of machine learning model targeted has on its vulnerability to membership inference. We additionally study the impact of model choices on the attacker side on attack effectiveness, specifically the impact of the type of
model used as the generation model \( F_g \).

**Attacks Across Model Types**  Many works in adversarial machine learning focus on the vulnerability of deep learning models whose uses range from image classification [27], [28] to speech recognition [29], [30] to natural language processing [31]. By generating adversarial examples which are tweaked in ways that are unnoticeable to humans, adversaries can exploit the models’ complexities to force misclassification [32], [33]. This trend has also influenced the study of membership inference problems.

Most existing efforts on membership inference [34], [35], [36], [13] have been focused on deep learning models. We argue that the complexity exploited in traditional adversarial learning attacks is not explicitly leveraged in membership inference attacks. Additionally, areas where membership inference would be most alarming, such as healthcare [37], e-commerce, banking, and government often deploy simpler model types, such as decision trees as model understanding is prioritized for safe use of predictive services. This leads to a natural question: are model types outside of deep learning methods susceptible to membership inference? Our empirical study on all seven datasets and four types of models demonstrates that not only are other model types vulnerable to membership inference attacks but that the model type is in fact very influential in determining the extent of that vulnerability.

The basic hypothesis of membership inference attack is it that models respond differently to instances which they have “seen” versus those they have not. In this hypothesis it is clear that model behavior and sensitivity are likely to impact vulnerability. We therefore consider a variety of model types outside of neural networks. We consider models from 4 different major categories: linear models (logistic regression), Bayesian models (Naïve Bayes), cluster models (k-nearest neighbor with \( k=5 \)), and tree models (CART decision trees).

For equal comparison to the previous work in [13], we use the shadow model imple-
mentation of the membership inference attack. All results from our experiments use 10-fold cross-validation and are averaged across 10 runs for a randomly selected sample of 10,000 instances.

Table 2.5: Precision of membership inference attack across 5 model types.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LR</th>
<th>k-NN</th>
<th>DT</th>
<th>NB</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult</td>
<td>50.13</td>
<td>51.39</td>
<td><strong>55.49</strong></td>
<td>50.22</td>
<td>50.30</td>
</tr>
<tr>
<td>MNIST</td>
<td>53.25</td>
<td>50.44</td>
<td><strong>56.66</strong></td>
<td>50.48</td>
<td>51.70</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>70.25</td>
<td>65.99</td>
<td><strong>83.94</strong></td>
<td>50.03</td>
<td>78.00</td>
</tr>
<tr>
<td>Purchases-10</td>
<td>64.56</td>
<td>53.53</td>
<td><strong>73.85</strong></td>
<td>50.61</td>
<td>55.00</td>
</tr>
<tr>
<td>Purchases-20</td>
<td>75.85</td>
<td>55.36</td>
<td><strong>81.94</strong></td>
<td>50.79</td>
<td>59.00</td>
</tr>
<tr>
<td>Purchases-50</td>
<td>81.61</td>
<td>58.19</td>
<td><strong>88.88</strong></td>
<td>52.08</td>
<td>86.00</td>
</tr>
<tr>
<td>Purchases-100</td>
<td>83.78</td>
<td>60.11</td>
<td>92.19</td>
<td>54.93</td>
<td><strong>93.50</strong></td>
</tr>
</tbody>
</table>

Table 2.6: Accuracy of membership inference attack across 4 model types.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LR</th>
<th>k-NN</th>
<th>DT</th>
<th>NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult</td>
<td>50.17</td>
<td>51.22</td>
<td><strong>59.89</strong></td>
<td>50.18</td>
</tr>
<tr>
<td>MNIST</td>
<td>54.38</td>
<td>50.59</td>
<td><strong>61.75</strong></td>
<td>50.81</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>67.40</td>
<td>68.32</td>
<td><strong>90.37</strong></td>
<td>50.01</td>
</tr>
<tr>
<td>Purchases-10</td>
<td>66.82</td>
<td>53.78</td>
<td><strong>82.29</strong></td>
<td>51.00</td>
</tr>
<tr>
<td>Purchases-20</td>
<td>80.50</td>
<td>55.92</td>
<td><strong>88.98</strong></td>
<td>51.29</td>
</tr>
<tr>
<td>Purchases-50</td>
<td>88.60</td>
<td>59.57</td>
<td><strong>93.71</strong></td>
<td>53.49</td>
</tr>
<tr>
<td>Purchases-100</td>
<td>90.23</td>
<td>62.19</td>
<td><strong>95.74</strong></td>
<td>57.61</td>
</tr>
</tbody>
</table>

We can clearly see in Tables 2.5 and 2.6 that other models are in fact vulnerable to membership inference and that both the training data (as discussed in Section 2.3.2) and the model type play an important role in understanding a particular model's risk. As shown in Tables 2.5 and 2.6, despite the variety in our datasets, the highest precision and accuracy are seen with the decision tree model for all datasets except Purchases-100 while the Naïve Bayes models consistently show exceedingly low precision across all datasets.

In general, a target model whose decision boundary is unlikely to be drastically impacted by a particular instance will be more resilient to membership inference attacks. For example, the Naïve Bayes algorithm independently considers the probability of a given class for each feature. Therefore, given significant training samples, a single instance only
Table 2.7: Model configuration reporting maximum membership inference attack accuracy compared to attack accuracies when configuration is varied across the target, generation, and attack models.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model Types for $(F_{\text{max}}^{\text{max}}, F_{\text{max}}^{\text{gen}}, F_{\text{max}}^{\text{att}})$</th>
<th>Accuracy $(F_{\text{max}}^{\text{max}}, F_{\text{max}}^{\text{gen}}, F_{\text{max}}^{\text{att}})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult</td>
<td>(DT, DT, NB)</td>
<td>59.91%</td>
</tr>
<tr>
<td>MNIST</td>
<td>(DT, DT, LR)</td>
<td>61.80%</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>(DT, LR, NB)</td>
<td>90.52%</td>
</tr>
<tr>
<td>Purchases-10</td>
<td>(DT, k-NN, DT)</td>
<td>82.45%</td>
</tr>
<tr>
<td>Purchases-20</td>
<td>(DT, LR, NB)</td>
<td>89.05%</td>
</tr>
<tr>
<td>Purchases-50</td>
<td>(DT, LR, LR)</td>
<td>93.77%</td>
</tr>
<tr>
<td>Purchases-100</td>
<td>(k-NN, LR, DT)</td>
<td>95.86%</td>
</tr>
</tbody>
</table>

marginally affects these probabilities. This explains the low numbers continuously seen when attacking a Naïve Bayes model. By contrast, a decision tree leaf node will consider a unique feature combination to determine class rather than each feature in isolation. The introduction of a single instance, if that instance displays a unique feature set-class combination, may cause a decision tree to grow and entire new branch. This sensitivity to single instances makes membership inference attacks more successful when targeting decision tree models.

Consequently, it is important to understand that while different datasets display different vulnerabilities to membership inference, so do different model types. This also indicates that machine learning-as-a-service providers who are wary of membership inference attacks against their deployed models may also be able to use model choice to help mitigate vulnerability.

We next study to impact of model choice by the membership inference attacker on the accuracy of their attack.
Variation in Generation Model  Instinctually, one may believe that the model used to generate attack data must be of the same type as the target model. We briefly discuss why previous research has made this same assumption and then investigate its veracity and seek to explain why it does not strictly hold.

In [13], the authors claim that the shadow model implementation of a membership inference attack requires the shadow models be trained in a similar way to the target model, an assumption followed by later work such as [38]. Consistent with our formalization of membership inference attacks, this claim is equivalent to saying that the attack data generation technique $F_g$ must mirror the behavior of the target model $F_t$ under attack.

The reasoning behind this assumption is intuitive. Let us say the target model $F_t$ is an approximation of an ideal function $\Gamma$ for a dataset $D$. Then, given a data point $x$ for which the adversary aims to identify membership in the training data $D$, the output provided to the adversary will be $F_t(x)$. This output will then be provided to the attack model to determine classification of $x$ as “in” or “out”. Let the binary classifier which serves as the attack model be trained from the output of a generation model $F_g$. That is, the output $F_g(x')$ for all $x' \in D'$ make up the attack model training data $D^*$.

We can clearly see that our attack model is therefore trained on the output of a function $F_g$ and deployed against the output of a function $F_t$. It is understandable then that previous work would seek to have the behavior of $F_g$ mirror the behavior of $F_t$. This assumption naturally extends to say that $F_g$, in an attempt to mirror $F_t$, must be, or intuitively should be, of the same model type as $F_t$ in a successful membership inference attack model.

However, this assumption is not necessary to launch an effective membership inference attack. We conducted a set of experiments varying model type combinations of $F_t$, $F_g$, and $F_a$ considering the four candidate model types. The combination with the highest accuracy for each dataset is reported in Table 2.7. Let $\text{type}(F_t^{\text{max}}), \text{type}(F_g^{\text{max}}), \text{type}(F_a^{\text{max}})$ be the target model, attack data generation model, and attack model types respectively for the membership inference attack which reported the highest accuracy. These types will
naturally vary for different datasets. We also report accuracy for scenarios when all three model types are set to $\text{type}(F_t^{max})$. We similarly report when all three types are equivalent to $\text{type}(F_g^{max})$ and $\text{type}(F_a^{max})$.

The highest accuracy seen in the membership inference attack against the CIFAR-10 model across all model combinations was reported when the target model was a decision tree, the data generation model was a logistic regression model, and the attack model was a Naïve Bayes model. Therefore, $\text{type}(F_t^{max}) = \text{DT}$, $\text{type}(F_g^{max}) = \text{LR}$, and $\text{type}(F_a^{max}) = \text{NB}$, as shown in column 2 of Table 2.7 along with the attack accuracy under these settings. For the CIFAR-10 dataset, we then also report the attack accuracy when all the models are a decision tree (i.e. equivalent to $\text{type}(F_t^{max})$), when all models are logistic regression models ($\text{type}(F_g^{max})$), and finally when all models are Bayesian ($\text{type}(F_a^{max})$). We repeat this process for all seven datasets.

We observe that, across datasets, it is not necessary for all models to be of the same type, as no maximally accurate combination contained the same model type across all three phases of the membership attack development. Additionally, the attack data generation model, as previously assumed, does not need to strictly mirror the target model for a successful membership inference attack. In fact, for 5 out of the 7 datasets the highest accuracy was reported when the attack data generation model was of a different type than the target model, i.e. $\text{type}(F_t^{max}) \neq \text{type}(F_g^{max})$. The reason behind this non-intuitive phenomenon lies in a more precise understanding of the role of the generation model. Although it was previously assumed that the role was to mirror the behavior of the target model, we assert that the generation model’s role is to characterize how the target model may be impacted by the inclusion of a particular instance. That is, how the decision boundary of the target model may reveal the inclusion of an instance.

The generation algorithm is therefore trying to characterize probability distributions related to a decision boundary which either has or has not been informed by the instance. From this perspective, it is now more clear that the vulnerability is more closely related to
two elements: distribution of the data and sensitivity of the decision boundary. If a decision boundary for a given dataset is notably impacted by the inclusion of a given instance then membership inference attacks are likely to be more successful.

In summary, we have demonstrated that, despite natural intuition, it is not strictly necessary for the attack data generation technique to be of the same type as the target model for a membership inference attack to be successful. Additionally, we again see the dominating factors are the target model type and the training dataset. We note that when all model types are equivalent to $F_t^{max}$ the resulting accuracy is within 0.16% of the maximum reported accuracy. By comparison, setting all model types to $F_g^{max}$ decreases attack accuracy by up to 28.67%. Setting all model types to $F_a^{max}$ demonstrates an even larger decrease in many cases as $F_a^{max}$ is reported to be a Naïve Bayes model for multiple datasets. As we previously identified the Naïve Bayes model as robust against membership inference attack, it is not surprising that attack accuracy will decrease significantly in these cases.

While success is close to maximal in all scenarios when the target model type is known, we re-accentuate the attack success seen in Table 2.7 with a mixture of model types. This again supports the conclusion that an adversary need not have this level of insider knowledge to launch a successful attack. Rather, when a model and its training dataset are particularly vulnerable, a variety of attack scenarios are likely to demonstrate success.

**Impact of Attacker Knowledge on Membership Inference**

Another category of factors that may cause model vulnerability to the membership inference attacks is the type and scope of knowledge which attackers may have about the target model and its training parameters. To further understand and evaluate the impact that attacker knowledge with respect to both the training data of the target model and the target data have on the accuracy of the membership inference attack, we launch membership inference attacks while varying the degree of noise in the shadow dataset and target data used by the attacker. Table 2.8 shows the experimental results from this study on four datasets.
Table 2.8: Impact of noisy target data and noisy shadow data on membership inference attack accuracy.

<table>
<thead>
<tr>
<th>Degree of Noise</th>
<th>Attack Accuracy (%) with Noisy Target Data</th>
<th>Attack Accuracy (%) with Noisy Shadow Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>Purchases-10</td>
</tr>
<tr>
<td>0</td>
<td>67.49</td>
<td>66.69</td>
</tr>
<tr>
<td>0.1</td>
<td>65.37</td>
<td>68.72</td>
</tr>
<tr>
<td>0.2</td>
<td>63.88</td>
<td>66.01</td>
</tr>
<tr>
<td>0.3</td>
<td>60.43</td>
<td>62.90</td>
</tr>
<tr>
<td>0.4</td>
<td>60.48</td>
<td>60.07</td>
</tr>
<tr>
<td>0.5</td>
<td>58.33</td>
<td>58.29</td>
</tr>
<tr>
<td>0.6</td>
<td>57.53</td>
<td>57.58</td>
</tr>
<tr>
<td>0.7</td>
<td>55.97</td>
<td>54.94</td>
</tr>
<tr>
<td>0.8</td>
<td>55.35</td>
<td>54.44</td>
</tr>
<tr>
<td>0.9</td>
<td>54.07</td>
<td>54.03</td>
</tr>
<tr>
<td>1.0</td>
<td>53.95</td>
<td>52.72</td>
</tr>
</tbody>
</table>

with four types of learning tasks: the CIFAR-10 10-class image dataset and the 10, 20, and 50-class Purchases datasets categorizing shoppers based on purchase history.

The experiments in Table 2.8 demonstrate the impact of the attacker knowledge of the target data points by evaluating how adding varying degrees of to data features may impact on the success rate of membership inference attacks. Noise uniformly sampled from $[0, \sigma], \sigma \leq 1$ and added to features normalized within $[0, 1]$. Given a level of uncertainty of $x$ or inaccuracy in $D'$ on the part of the attacker, represented by a corresponding degree of noise $\sigma$, Table 2.8 evaluates how effective the attacker remains in launching a membership inference attack to identify if $x \in D$ or $x \notin D$. The results reported in Table 2.8 are
reported for four logistic regression models, each one trained on a different dataset with gradually increasing $\sigma$ values (degree of noise).

We make two interesting observations from Table 2.8. First, for all four datasets, the more accurate (the less noise) the attacker knowledge about $x$ is, the higher the model vulnerability (attack success rate) to membership inference. This shows the accuracy of attacker knowledge about the targeted examples is an important factor in determining model vulnerability and attack success rate (in terms of attack accuracy). Second, in comparison to the noisy target data, adding noise to the shadow dataset results in a less severe drop in accuracy. Similar trends are however still observed with slightly higher attack success rates under smaller $\sigma$ values for all four datasets.

While it is unsurprising that the noisier the target instance the less accurate the attack, it is also interesting to note that accurate attacks may still be launched when small amounts of noise are added. That is, if the attacker knows 90% of the data for the target instance but perhaps has to make an educated guess for the final 10%, then that attacker is able to identify whether or not the target instance was within the training data with similar accuracy as was seen when 100% of the target data was known.

Finally, in Figure 2.9 we report the effect that the size of the shadow dataset has on attack accuracy. For these experiments we select shadow datasets of various sizes which are again disjoint from the target training data but from the same distribution. No noise is added for these experiments to isolate the impact of the shadow data size. The experiments are evaluated on the same four datasets again with logistic regression models.

Each dataset in these experiments appears to result in different behavior. The CIFAR-10 dataset continues to show increased accuracy as the shadow dataset size increases. The Purchases-10 dataset however shows an initial accuracy increase before decreasing again and leveling out by the time the shadow dataset reaches 1,000 instances. The Purchases-20 dataset shows a fast initial trend of accuracy improvement before leveling off at a shadow dataset size around 700 instances. Finally, the Purchases-50 dataset results show a similar
Figure 2.9: Impact of size of the shadow dataset on membership inference attack accuracy with logistic regression models.

trend to that seen with the CIFAR-10 dataset where the accuracy continues to increase with the shadow dataset size.

Overall through Table 2.8 and Figure 2.9 it is clear that accuracy in the target instances is the most important attacker knowledge factor in determining attack success. With limited resources an attacker should therefore focus on acquiring sufficient information with respect to target instances. The next most deterministic element is the size of the shadow dataset. Rather than allocating resources to ensuring completely accurate instances, attackers should instead put the focus on quantity in the development of their shadow datasets. This set of experiments demonstrates that attackers with different knowledge and different levels of resources may have different success rates in launching a membership inference attack. Thus, model vulnerability should be evaluated by taking into account potential or available attacker knowledge.
Transferability of Membership Inference

Inspired by the transferability of adversarial examples [39], [40], [41], [42], we also demonstrate that membership inference attacks can also be transferable. That is, attack model $F_a$ trained on an attack dataset $D*$ containing the outputs from a set of shadow models is effective not only when shadow models and the target model are of the same type but also when the shadow model type varies. This property further opens the door to the black-box attackers who do not have any knowledge of the target model.

Table 2.9: Membership inference attack accuracy against a decision tree model trained on the Purchases-20 dataset.

<table>
<thead>
<tr>
<th>Purchases-20</th>
<th>Shadow Model Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack Model</td>
<td>DT</td>
</tr>
<tr>
<td>DT</td>
<td>88.98</td>
</tr>
<tr>
<td>k-NN</td>
<td>88.23</td>
</tr>
<tr>
<td>LR</td>
<td><strong>89.02</strong></td>
</tr>
<tr>
<td>NB</td>
<td>88.96</td>
</tr>
</tbody>
</table>

Table 2.9 demonstrates this property of membership inference attacks. It reports the membership inference attack accuracy for various attack configurations against a decision tree model trained on the Purchases-20 dataset. It shows the transferability of membership inference attacks for different combinations from four different model types used as the attack model type $F_a$ (rows) and the shadow model type (columns). In this experiment, the target prediction model is a decision tree. We observe that while using decision tree as the shadow model results in the most consistent membership inference attack success compared to other combinations, multiple combinations with both $k$-NN and logistic regression (LR) shadow models achieve attack success within a 5% gap compared to the most successful attack configuration using decision tree shadow models. Table 2.9 also shows that multiple types of models can be successful as the binary attack classifier $F_a$. This set of experiments also shows that the worst attack performances against the DT target prediction model are seen when the shadow models are trained using Naive Bayes (NB), with the
worst performance reported when NB is the model type of the attack model as well. These results indicate that (1) the same strategy used for selecting the shadow model type may not be optimal for the attack model and (2) shadow models of different types other than the target model type may still lead to successful membership inference attacks.

The transferability study in Table 2.9 indicates an attacker does not always need to know the exact target model configuration to launch an effective membership inference attack as attack models can be transferable from one target model type to another. And although finding the most effective attack strategy can be a challenging task for attackers, vulnerability to membership inference attack remains serious even with suboptimal attack configurations with almost all configurations reporting attack accuracy about 70% and many above 85%.

This suggests that membership inference attack models are transferable from one target model to another, provided that targets are trained on the same dataset $D$.

We further investigate transferability in the membership inference attack by evaluating the standard deviation of attack results for a membership inference attack model $F_a$ learned over the output of an attack data generation model $F_g$ and deployed against a target model $F_t$ across the seven datasets for the following three scenarios: (1) vary the model types for $F_g$ and $F_a$ while keeping $F_t$ consistent, (2) varying $F_t$ and $F_a$ while keeping $F_g$ consistent, and (3) varying $F_t$ and $F_g$ while keeping $F_a$ consistent.

For example, Table 2.10 shows the results of the experiments on various combinations of model types for the CIFAR-10 dataset. We calculate the average standard deviation for scenario (1) as follows: $DT_{sd}$ denotes the standard deviation of all accuracy values in rows 1, 5, 9, and 13 where the target model is a decision tree (DT). Similarly, $k$-$NN_{sd}$ corresponds to rows 2, 6, 10, and 14, $LR_{sd}$ with 3, 7, 11, and 15, and $NB_{sd}$ with 4, 8, 12, and 16. Then we consider the average standard deviation in accuracy for scenario (1) to be the average of $DT_{sd}$, $k$-$NN_{sd}$, $LR_{sd}$, $NB_{sd}$. For scenario (3) we follow a similar process but with row sets 1-4, 5-8, 9-12, and 13-16. Scenario (2) is calculated by averaging the standard deviations for each of the 4 $F_g$ columns.
Table 2.10: Membership inference attack accuracy with various attack, data generation, and target models using the CIFAR-100 dataset.

<table>
<thead>
<tr>
<th>Attack Model</th>
<th>Target Model</th>
<th>Attack Data Generation Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DT</td>
<td>k-NN</td>
</tr>
<tr>
<td>DT</td>
<td>DT</td>
<td>90.44%</td>
</tr>
<tr>
<td></td>
<td>k-NN</td>
<td>54.92%</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>53.84%</td>
</tr>
<tr>
<td></td>
<td>NB</td>
<td>50.46%</td>
</tr>
<tr>
<td>k-NN</td>
<td>DT</td>
<td>89.96%</td>
</tr>
<tr>
<td></td>
<td>k-NN</td>
<td>55.33%</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>51.34%</td>
</tr>
<tr>
<td></td>
<td>NB</td>
<td>50.12%</td>
</tr>
<tr>
<td>LR</td>
<td>DT</td>
<td>90.37%</td>
</tr>
<tr>
<td></td>
<td>k-NN</td>
<td>51.72%</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>50.01%</td>
</tr>
<tr>
<td></td>
<td>NB</td>
<td>50.54%</td>
</tr>
<tr>
<td>NB</td>
<td>DT</td>
<td>90.42%</td>
</tr>
<tr>
<td></td>
<td>k-NN</td>
<td>50.33%</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>50.00%</td>
</tr>
<tr>
<td></td>
<td>NB</td>
<td>50.58%</td>
</tr>
</tbody>
</table>

We follow this process for each of the seven datasets Adult, MNIST, CIFAR-10, along with the 4 Purchases datasets and summarize the results in Table 2.11. This allows us to compare the impact of model variation for each $F_t$, $F_g$, and $F_a$. We observe that the standard deviation of membership inference attack results is relatively small against a fixed target model when compared to a fixed attack data generation model or fixed attack model. A smaller standard deviation is indicative of a larger impact. That is, the accuracy is stable when the standard deviation is small. A small standard deviation would indicate that the fixed model has more influence over attack accuracy than the varied models.

Table 2.11 clearly shows that for all datasets the deviation is minimized when $F_t$ is fixed. This indicates that variation in $F_g$ and $F_a$ have a comparatively low impact on attack success rates. This supports the evidence in Table 2.9 that an adversary need not have particularly informed choices in $F_g$ or $F_a$ to develop an attack model. We therefore characterize membership inference attacks, like others in the adversarial learning domain, as transferable.
Table 2.11: Standard deviation of membership inference attack accuracy given (1) fixed target models, (2) fixed generation models, and (3) fixed attack models.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Standard Deviation in Accuracy Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed $F_t$</td>
</tr>
<tr>
<td>Adult</td>
<td>0.0093</td>
</tr>
<tr>
<td>MNIST</td>
<td>0.0126</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>0.0643</td>
</tr>
<tr>
<td>Purchases-10</td>
<td>0.0396</td>
</tr>
<tr>
<td>Purchases-20</td>
<td>0.0545</td>
</tr>
<tr>
<td>Purchases-50</td>
<td>0.0705</td>
</tr>
<tr>
<td>Purchases-100</td>
<td>0.0849</td>
</tr>
</tbody>
</table>

In summary, Table 2.9 highlights the role of the attack data generation model using the Purchases-20 dataset. It articulates that a variety of attack model structures may be used against a single target model. Tables 2.10 and 2.11 cover combinations of target models, attack models, and attack data generation models with the CIFAR-10 dataset to illustrate a broader, more comprehensive picture that both the target model and the target dataset are the most important factors for membership inference vulnerability. For each of the four attack models (DT, k-NN, LR, NB), the results show the impact in varying target models on attack accuracy under one of the four given attack data generation models. Table 2.11 shows the derivation of accuracy results for each of the seven datasets under a fixed target, a fixed generation model, or a fixed attack model. These results indicate that an adversary may be able to develop an attack model without knowing “best” attack model or the “best” attack data generation model.

**Impact of Training Data Skewness on Membership Inference Vulnerability**

The fourth important dimension of membership inference vulnerability is the risk imbalance across different prediction classes when the training data is skewed. Even when the overall membership inference vulnerability appears limited with attack success close to the 50% baseline (in or out random guess), there may be subgroups within the training data, which display significantly more vulnerability.
For example, Figure 2.10 illustrates the impact of data skewness on membership inference vulnerability. In this set of experiments, we measure the membership inference attack accuracy for a decision tree target model trained on the Adult dataset where the class distribution is skewed with less than 25% of instances being labeled >$50K. As overfitting is widely considered a key factor in membership inference vulnerability, we simulate overfitting by increasing the depth of the target decision tree model. Figure 2.10 shows that the impact of overfitting (increasing in X-axis) on both the aggregate membership inference vulnerability in terms of membership inference attack accuracy (accuracy over all classes) and the minority membership inference vulnerability (attack accuracy over the minority class). In this case aggregate vulnerability is the accuracy of the membership inference attack evaluated on an equal number of randomly selected examples seen by the target model (“in”) as as unseen (“out”) while the minority vulnerability reports the membership inference attack accuracy evaluated on only the subset of the previously selected examples whose class is >$50K (the minority class).

We observe from Figure 2.10 that, the minority class has an increased risk under the membership inference attack as the model over-fits more severely. This follows the intuition that minority class members have fewer other instances amongst whom they can hide in the training set and thus are more easily exposed under membership inference attacks. This aligns well to some extent with the observation that smaller training dataset sizes can
lead to a greater overall risk for membership inference [13]. We argue that it is important to consider vulnerability not just for the entire training dataset, but also the level of risk for minority populations specifically when evaluating privacy compliance.

We next introduce our membership privacy framework, MPLens, which can be leveraged in real world MLaaS deployment scenarios for such privacy compliance evaluation.

2.3.3 MPLens: System Overview

MPLens is designed as a privacy analysis and privacy compliance evaluation system for both data scientists and MLaaS providers to understand the membership inference vulnerability risk involved in their model training and model prediction process. Figure 2.11 provides an overview of the system architecture. The system allows providers to specify a set of factors that are critical to privacy analysis. Example factors include the data used to train their model, what data might be held by the attacker, what attack technique might be used, the degree of data skewness in the training set, whether the prediction model is constructed using the differentially private model training, what configurations are used for the set of differential privacy parameters, and so forth. Given the model input, the MPLens evaluation system reports the overall statistics on the vulnerability of the model, the per-class vulnerability, as well as the vulnerability of any sensitive populations such as specific
minority groups. Example statistics include attack accuracy, precision, recall, and f-1 score. We also include attacker confidence for true positives, false positives, true negatives, and false negatives, the average distance from both the false positives and the false negatives to the training data, and the time required to execute the attack.

Target Model Training

Over-fitting is the first factor that MPLens measures for conducting membership vulnerability analysis. MPLens specifically highlights the overfitting analysis by reporting the Accuracy Difference between the target model training accuracy and testing accuracy. This enables MLaaS providers and domain-specific data scientists to understand whether their vulnerability might be linked to overfitting. As previously indicated, while overfitting is strongly correlated with membership inference vulnerability, it is not the only source of vulnerability. Thus, when MPLens indicates undesirable vulnerability an absence of significant overfitting, analysis may be triggered to investigate if vulnerability is linked to other model or data characteristics as those discussed in Section 2.3.2.

Attacker Knowledge

Our MPLens system is by design customizable to understand multiple attack scenarios. For instance, users may specify the shadow data, which is used by the attacker. This allows the user to consider a scenario in which the attacker has access to some subset of the target model’s training data, one where the attacker has access to some data that are drawn from the same distribution as the training data of the target model, or one where the attacker has noisy and inaccurate data, such as that evaluated in Table 2.8 and possibly generated through black-box probing [13, 16].

The MPLens system is additionally customizable with respect to the attack method, including the shadow model based attack techniques [13], and the threshold-based attack techniques [25]. Furthermore, when using a threshold-based attack, our MPLens sys-
tem can either accept pre-determined values representing attacker knowledge of the target model error or it can also determine good threshold values through the shadow model training.

These customizations allow MLaaS providers and users of MPLens to specify the types of attackers they wish to analyze, analyze their model vulnerability against such attackers, and evaluate the privacy compliance of their model training and model prediction services.

Transferability

Another aspect of privacy analysis is related to the specific model training methods used to generate membership inference attacks and whether different methods result in significant variations in membership inference vulnerability. Consider the attack method from [13], the user can specify not only the attacker’s data but also the shadow model training algorithm and the membership inference binary classifier training algorithm. Each element is customizable as a system parameter when configuring MPLens for specific privacy risk evaluation. The MPLens system makes no assumption on how the attacker develops the shadow dataset, what knowledge is included in the data, whether the attacker has knowledge of the target model algorithm, or what attack technique is used. This flexibility allows MPLens to support evaluation across various transferable attack configurations.

Thus far we have studied the risks in machine learning and machine learning services under settings where all data is held by single party, e.g. one healthcare provider. We now extend our considerations to include settings with multiple data holders and discuss the privacy vulnerabilities specific to such distributed machine learning scenarios.

2.4 Privacy Vulnerabilities Specific to Distributed Machine Learning

In traditional machine learning environments, training data is centrally held by one organization executing the learning algorithm. Early distributed learning systems extend this approach by using a set of learning nodes accessing shared data or having the data sent
to the participating nodes from a central node, all of which are fully trusted. For example, MLlib from Apache Spark assumes a trusted central node to coordinate distributed learning processes [43]. Another approach is the parameter server [44], which again requires a fully trusted central node to collect and aggregate parameters from the many nodes learning on their different datasets.

However, some learning scenarios must address less open trust boundaries, particularly when multiple organizations are involved. While a larger dataset improves the performance of a trained model, organizations often cannot share data due to legal restrictions or competition between participants. For example, consider three hospitals with different owners serving the same city. Rather than each hospital creating their own predictive model forecasting cancer risks for their patients, the hospitals want to create a model learned over the whole patient population. However, privacy laws prohibit them from sharing their patients’ data. Similarly, a service provider may collect usage data both in Europe and the United States. Due to legislative restrictions, the service provider’s data cannot be stored in one central location. When creating a predictive model forecasting service usage, however, all datasets should be used.

The area of federated learning addresses these more restrictive environments. Federated ML has seen increased adoption in recent years [45, 46, 47] in response to the growing legislative demand to address user privacy [5, 48, 6]. Federated learning (FL) allows data to remain at the edge with only model parameters being shared with a central server. Specifically, there is no centralized data curator who collects and verifies an aggregate dataset. Instead, each data holder (participant) is responsible for conducting training on their local data.

Federated learning can either be loosely federated or tightly federated. In the former, each participant fully trains their own predictive model which is then leveraged in coordination with other participants’ models at evaluation time such that the set of models functions as an ensemble. In this setting, model predictions are aggregated via some pre-determined
approach such as a majority vote or weighted average through some third party aggregation service before being returned to users of the predictive service. However, in a tightly federated setting, participants work together to train on aggregate model. In regular intervals participants are send model parameter values learned from local training to a central parameter server or aggregator. A global model is then created through aggregation of the individual updates. A global model can thus be trained over all participants’ data without any individual participant needing to share their private raw data.

While FL systems allow participants to keep their raw data local, new threats are also introduced in the form of a third party aggregation service as well as the other data holder participants which have the potential to be malicious. We first look at the privacy risk of insider membership inference given this new ecosystem for attack. Additionally, with no central authority able to validate data, malicious participants can poison a trained global model under a tightly federated setting. For example, consider Microsoft’s AI chat bot Tay. Tay was released on Twitter with the underlying natural language processing model set to learn from the Twitter users it interacted with. Thanks to malicious users, Tay was quickly manipulated to learn offensive and racist language [49].

We specifically study the vulnerability of FL systems to malicious participants with minimal capabilities – each can only manipulate the raw training data on their device. This allows for non-expert malicious participants to achieve poisoning with no knowledge of model type, parameters, and FL process. Under this set of assumptions, label flipping attacks become a feasible strategy to implement data poisoning, attacks which have been shown to be effective against traditional, centralized ML models [50, 51, 52, 53]. We investigate their application to FL systems using complex deep neural network models.
2.4.1 Membership Inference Insider Attacks in Federated Learning

**Insider Attack Model**

To this point we have exclusively discussed “outsider” membership inference attacks. That is, membership inference attacks which are launched by an adversary who is only a user of the target model through black-box access to the target service prediction API. We now introduce the threat of insider membership inference attacks. We define these attacks to be those launched by a participant in a federated learning system.

In recent years there has been an increased interest in the role of federated learning systems in addressing privacy concerns in data mining. The intuition behind federated systems to protect against membership inference is as follows: if an adversary is able to identify that a certain instance is contained within the training data of a model and that model is the result of a federated learning system, then any individual participant will have plausible deniability with respect to their individual dataset. However, such federations open the door to a new risk through insider attacks.

The difference between an attack on federated systems and outsider membership inference attacks is that, in federated systems, the training dataset $D$ is divided amongst multiple parties who engage in collaborative learning to provide predictions to the machine learning service. We consider the following loosely federated system: given parties $P_1, P_2, ..., P_\ell$ there exist $\ell$ independent dataset $D_1, D_2, ..., D_\ell$, one belonging to each party. Each party $P_i$ will then train a model $F_{t_i}$ using $D_i$ as the corresponding training data.

Within this environment, new instances will be evaluated as follows. On input of $x$, each model $F_{t_i}$ will output a probability vector $p_i = F_{t_i}(x)$. The individual parties will then share their output either openly with one another or with some aggregation service to compute the final output $p = \text{ave}(p_1, p_2, ..., p_\ell)$ where $\text{ave}$ refers to point-wise averaging of the probability vectors. Any outside adversary using this service will only have access to the final probability vector $p$. Any individual party $P_i$ will therefore have plausible de-
niability because, if an instance is identified as a member of the training data, the adversary is unlikely to identify which training set specifically.

However we must also consider adversaries who are members of the federated learning systems. That is, the aggregation service or a participating party. Under this scenario the “insider” will have access to the individual probability vectors $p_1, p_2, ..., p_\ell$. The insider membership inference attack then becomes: given these probability vectors, is the adversary able to identify which dataset a training instance belongs to.

We now consider when parties may participate in a federated system. Let $P_1, P_2,$ and $P_3$ represent three candidate parties with training datasets $D_1$, $D_2$, and $D_3$ respectively. Let us assume the extreme case that $D_1$, $D_2$, and $D_3$ are statistically equivalent. Then, the trained models $F_{t_1}$, $F_{t_2}$, and $F_{t_3}$ will approximate the same ideal function $\Gamma$. Given this set up, it is then likely that, on any input $x$, the outputs $F_{t_1}(x)$, $F_{t_2}(x)$, and $F_{t_3}(x)$ will also be statistically equivalent. Here, there is no accuracy gain for $P_1$, $P_2$, or $P_3$ through collaboration. They are therefore unlikely to be motivated to create a federated learning system.

Alternatively, consider such significantly different $D_1$, $D_2$, and $D_3$ such that $F_{t_1}$, $F_{t_2}$, and $F_{t_3}$ may be considered independent. Let us now assume accuracies of 75%, 80%, and 70% and a majority voting aggregation scheme. Such a federation, on input $x$, has an 84.5% chance of accurately classifying $x$, an accuracy higher than any individual model. Under these conditions, $P_1$, $P_2$, and $P_3$ are much more likely to form a federation.

It is therefore reasonable to assume parties are likely to form federated learning systems when their individual datasets are sufficiently different. Unfortunately, this leads to sufficiently different decision boundaries for different parties. These diverging decision boundaries open the door to effective insider membership inference attacks as an adversary will notice differences in $p_1$, $p_2$, ..., $p_\ell$. 
Table 2.12: Precision of insider membership inference attack within a 3-party federated learning system with a decision tree target model.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Outsider Inference Precision (Accuracy)</th>
<th>Insider Inference Precision (Accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult</td>
<td>55.49 (59.89)</td>
<td>73.26 (69.33)</td>
</tr>
<tr>
<td>MNIST</td>
<td>56.66 (61.75)</td>
<td>68.47 (68.17)</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>83.94 (90.44)</td>
<td>82.02 (82.05)</td>
</tr>
<tr>
<td>Purchases-10</td>
<td>73.85 (82.29)</td>
<td>74.42 (74.30)</td>
</tr>
</tbody>
</table>

Vulnerability to Insider Membership Inference Attacks in Federated Learning Systems

In Table 2.12 we see that even datasets showing resilience to outsider membership inference are vulnerable to an insider membership inference attack. We created a federated system where $n = 3$ for datasets with $k \leq 10$ so that each party has sufficient instances of each class to learn a meaningful decision boundary. Given a federation where $n = 3$, any party behaving as an adversary will have an attack precision baseline of 50%. This allows for comparison with the outsider inference attacks. Both the Adult and MNIST dataset showed minimal vulnerability in the outsider membership inference attacks and experienced significant jumps in vulnerability in the insider attack scenario while the CIFAR-10 and Purchases-10 datasets show similar precision results, notably outperforming the baseline.

Figure 2.12: Decision boundaries for 3 different participants in a federated system using the Adult dataset. An area where the decision boundaries are significantly different is highlighted as well as the training instances provided to each participant relevant to the highlighted area.
In Figure 2.12, we plot the decision boundaries created by three different decision tree models trained on disjoint subsets of the Adult dataset to articulate how the variation in decision boundaries informs insider attacks. We plot the decision boundaries relative to the *capital loss* and *education number* features. The section enclosed within the blue box highlights a portion of the decision boundary which notably differs between each plot. The second level of Figure 2.12 is a zoomed-in view of this section for all three plots. On the third level we then plot the positive training instances that informed each decision boundary in this region. It is clear that the long region identified as the positive class in the third plot was informed by significantly more positive instances than the other two decision plots. It is decision boundary differences such as those demonstrated here, and what they reveal of the underlying training data, that reveals ownership in the insider membership inference attack.

More specifically, we know that a participant who returns prediction vectors consistent with the third plot (i.e. higher probabilities for the positive class within the long horizontal region), is more likely to have seen a participant with a capital loss value just under 2,000 and a salary above $50,000. The probabilities returned from the participant with a model whose decision boundary is more consistent with the first plot however will return higher probabilities for the negative class. It is therefore much less likely that this participant has an individual with a capital loss value within this range and a salary greater than $50,000. An attacker, therefore, will know that such an individual (i.e. one who falls within this range with a higher salary) is much more likely to be in the dataset of the third participant.

This is supported by the accuracy seen for federations with different data distributions characteristics. When constructing our federated learning systems for these experiments we first sampled target class distributions at random to create different scenarios which may be seen in deployed federated learning environments.

In Figure 2.13 we show the relationship between the accuracy of the insider membership inference attack and the distance between the two targeted parties’ data using the Adult
dataset. That is, let a 3-party federation be formed by parties $P_1$, $P_2$, and $P_3$ wherein $P_3$ behaves maliciously and launches an insider membership inference attack against $P_1$ and $P_2$. Then, we look at how similar all the instances of class $k_i$ in $D_1$ are to those of class $k_i$ in $D_2$. This similarity is averaged across all $k_i \in [1, k]$. If the datasets are more similar then their in-class distance will be lower. In Figure 2.13 we can see that $P_3$ will be less successful than when $D_1$ and $D_2$ have a closer in-class measure than if the datasets are very different.

Unfortunately this leads to a catch-22 scenario. Parties with very similar looking training datasets will not be motivated to participate in a federation as they are less likely to see significant increases in classification accuracy. Parties with very different looking training datasets, however, will be more vulnerable to insider membership inference.

We argue that the risk of insider membership inference attack is of particular concern as participants are likely to assume they are less vulnerable than in outsider scenarios due to the plausible deniability protections inherent in federated learning systems. The potential for insider attacks, however, calls for a robust trust policy for any federated learning system to consider such risk.

Figure 2.13: In-class distance between different parties compared with insider membership inference attack accuracy for the Adult dataset and a 3-party federated learning system.
2.4.2 Poisoning Attacks in Federated Learning

We also consider adversarial machine learning attacks within the context of federated learning. Specifically, let’s consider the area of adversarial ML known as poisoning attacks. Poisoning attacks are highly relevant in domains such as spam filtering [54, 55], malware and network anomaly detection [56, 57, 58], disease diagnosis [59], computer vision [60], and recommender systems [61, 62]. Several poisoning attacks were developed for popular ML models including SVM [63, 64, 51, 65, 52, 53], regression [66], dimensionality reduction [67], linear classifiers [64, 68, 69], unsupervised learning [70], and more recently, neural networks [64, 71, 72, 65, 73, 74].

There are two types of poisoning attacks in FL: data poisoning and model poisoning. We will specifically look at the data poisoning category. In data poisoning, a malicious FL participant manipulates their training data, e.g., by adding poison instances or adversarially changing existing instances [75, 76]. The local learning process is otherwise not modified. In model poisoning, the malicious FL participant modifies its learning process in order to create adversarial gradients and parameter updates. [77] and [78] demonstrated the possibility of causing high model error rates through targeted and untargeted model poisoning attacks. While model poisoning is also effective, data poisoning may be preferable or more convenient in certain scenarios, since it does not require adversarial tampering of model learning software on participant devices, it is efficient, and it allows for non-expert poisoning participants.

Label Flipping Attacks in Federated Learning

Threat and Adversary Model When discussing the threat of label flipping in federated learning, we consider the scenario in which a subset of FL participants are malicious or are controlled by a malicious adversary. We denote the percentage of malicious participants among all participants $P$ as $m%$. Malicious participants may be injected to the system by adding adversary-controlled devices, compromising $m%$ of the benign participants’ de-
VICES, or incentivizing (bribing) m% of benign participants to poison the global model for a certain number of FL rounds. As the aggregator holds no data to manipulate, the aggregator is considered honest and not compromised.

The goal of the adversary is to manipulate the learned parameters such that the final global model $M$ has high errors for particular classes (a subset of $C$). The adversary is thereby conducting a targeted poisoning attack. This differs from untargeted attacks which instead seek indiscriminate high global model errors across all classes [63, 78, 67]. Targeted attacks have the desirable property that they decrease the possibility of the poisoning attack being detected by minimizing influence on non-targeted classes.

Label flipping attacks further allow for a realistic adversary model with the following constraints. Each malicious participant can manipulate the training data $D_i$ on their own device, but cannot access or manipulate other participants’ data or the model learning process, e.g., SGD implementation, loss function, or server aggregation process. The attack is not specific to the model architecture, loss function or optimization function being used. It requires training data to be corrupted, but the learning algorithm remains unaltered.

**Vulnerability to Label Flipping Attacks in Federated Learning Systems** We consider label flipping attacks to study targeted data poisoning in FL. Given a source class $c_{src}$ and a target class $c_{target}$ from $C$, each malicious participant $P_i$ modifies their dataset $D_i$ as follows: For all instances in $D_i$ whose class is $c_{src}$, change their class to $c_{target}$. We denote this attack by $c_{src} \rightarrow c_{target}$. For example, in CIFAR-10 image classification, airplane $\rightarrow$ bird denotes that images whose original class labels are airplane will be poisoned by malicious participants by changing their class to bird. The goal of the attack is to make the final global model $M$ more likely to misclassify airplane images as bird images at test time.

Label flipping is a well-known attack in centralized ML [76, 51, 52, 53]. It is also suitable for the FL scenario given the adversarial goal and capabilities above. Unlike other
types of poisoning attacks, label flipping does not require the adversary to know the global
distribution of $\mathcal{D}$, the DNN architecture, loss function $\mathcal{L}$, etc. It is time and energy-efficient,
an attractive feature considering FL is often executed on edge devices. It is also easy to
carry out for non-experts and does not require modification or tampering with participant-
side FL software.

Figure 2.14 demonstrates the feasibility of poisoning federated learning systems using
label flipping attacks by outlining the global model accuracy and source class recall in
scenarios with malicious participant percentage $m$ ranging from 2% to 50%. We conduct
our attacks using two popular image classification datasets: CIFAR-10 [10] and Fashion-
MNIST [79]. Recall from Section 2.1.1 that the CIFAR-10 dataset is a 10-class color
image classification problem. Similar to the MNIST dataset, Fashion-MNIST consists of
a training set of 60,000 images and a test set of 10,000 images. Each image in Fashion-
MNIST is gray-scale and associated with one of 10 classes of clothing such as pullover,
ankle boot, or bag. CIFAR-10 experiments in Figure 2.14 are for the $5 \rightarrow 3$ setting while
Fashion-MNIST experiments are for the $4 \rightarrow 6$ setting.

In experiments with CIFAR-10, we use a convolutional neural network with six con-
volutional layers, batch normalization, and two fully connected dense layers. This DNN
architecture achieves a test accuracy of 79.90% in the centralized learning scenario, i.e.
$N = 1$, without poisoning. In experiments with Fashion-MNIST, we use a two layer con-
volutional neural network with batch normalization, an architecture which achieves 91.75%
test accuracy in the centralized scenario without poisoning. Further details of the datasets
and DNN model architectures can be found in [80].

By default in our poisoning experiments, we have 50 participants, one central aggre-
gator, and select 5 participants for learning at each round. We use an independent and
identically distributed (iid) data distribution, i.e., we assume the total training dataset is
uniformly randomly distributed among all participants with each participant receiving a
unique subset of the training data. The testing data is used for model evaluation only and is
therefore not included in any participant’s train dataset. Observing that both DNN models converge after fewer than 200 training rounds, we set our FL experiments to run for 200 total rounds.

Results demonstrate that as the malicious participant percentage, increases the global model utility (test accuracy) decreases. Even with small $m$, we observe a decrease in model accuracy compared to a non-poisoned model (denoted by $M_{NP}$ in the graphs), and there is an even larger decrease in source class recall. In experiments with CIFAR-10, once $m$ reaches 40%, the recall of the source class decreases to 0% and the global model accuracy decreases from 78.3% in the non-poisoned setting to 74.4% in the poisoned setting. Experiments conducted on Fashion-MNIST show a similar pattern of utility loss. With $m = 4\%$ source class recall drops by $\sim 10\%$ and with $m = 10\%$ it drops by $\sim 20\%$. It is therefore clear that an adversary who controls even a minor proportion of the total participant population is capable of significantly impacting global model utility.

While both datasets are vulnerable to label flipping attacks, the degree of vulnerability varies between datasets with CIFAR-10 demonstrating more vulnerability than Fashion-MNIST. For example, consider the 30% malicious scenario, Figure 2.14b shows the source class recall for the CIFAR-10 dataset drops to 19.7% while Figure 2.14d shows a much lower decrease for the Fashion-MNIST dataset with 58.2% source class recall under the same experimental settings.
Table 2.13: Loss in source class recall due to label flipping data poisoning attacks for three source $\rightarrow$ target class settings with differing baseline misclassification counts.

<table>
<thead>
<tr>
<th>$c_{src}$ $\rightarrow$ $c_{target}$</th>
<th>$m_{crit_src}$</th>
<th>Percentage of Malicious Participants ($m%$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 $\rightarrow$ 2</td>
<td>16</td>
<td>1.42%</td>
</tr>
<tr>
<td>1 $\rightarrow$ 9</td>
<td>56</td>
<td>0.69%</td>
</tr>
<tr>
<td>5 $\rightarrow$ 3</td>
<td>200</td>
<td>0%</td>
</tr>
<tr>
<td>Fashion-MNIST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 $\rightarrow$ 3</td>
<td>18</td>
<td>0.12%</td>
</tr>
<tr>
<td>4 $\rightarrow$ 6</td>
<td>51</td>
<td>0.61%</td>
</tr>
<tr>
<td>6 $\rightarrow$ 0</td>
<td>118</td>
<td>-1%</td>
</tr>
</tbody>
</table>

On the other hand, vulnerability variation based on source and target class settings is less clear. In Table 2.13, we report the results of three different combinations of source $\rightarrow$ target attacks for each dataset. Consider the two extreme settings for the CIFAR-10 dataset: on the low end the 0 $\rightarrow$ 2 setting has a baseline misclassification count of 16 while the high end count is 200 for the 5 $\rightarrow$ 3 setting. Because of the DNN’s relative challenge in differentiating class 5 from class 3 in the non-poisoned setting, it could be anticipated that conducting a label flipping attack within the 5 $\rightarrow$ 3 setting would result in the greatest impact on source class recall. However, this was not the case. Table 2.13 shows that in only two out of the six experimental scenarios did 5 $\rightarrow$ 3 record the largest drop in source class recall. In fact, four scenarios’ results show the 0 $\rightarrow$ 2 setting, the setting with the lowest baseline misclassification count, as the most effective option for the adversary. Experiments with Fashion-MNIST show a similar trend, with label flipping attacks conducted in the 4 $\rightarrow$ 6 setting being the most successful rather than the 6 $\rightarrow$ 0 setting which has more than $2 \times$ the number of baseline misclassifications. These results indicate that identifying the most vulnerable source and target class combination may be a non-trivial task for the adversary, and that there is not necessarily a correlation between non-poisoned misclassification performance and attack effectiveness.

We additionally study a desirable feature of the label flipping attack: they appear to be targeted. Specifically, Table 2.14 reports the following quantities for each source $\rightarrow$ target
Table 2.14: Changes in source class recall, target class recall, and total recall for all remaining classes (non-source, non-target) due to label flipping data poisoning attacks.

<table>
<thead>
<tr>
<th>(c\text{\textsubscript{src}}) → (c\text{\textsubscript{target}})</th>
<th>(\Delta c\text{\textsubscript{src}}^{\text{recall}})</th>
<th>(\Delta c\text{\textsubscript{target}}^{\text{recall}})</th>
<th>(\sum\text{all other }\Delta c\text{\textsubscript{recall}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 → 2</td>
<td>-6.28%</td>
<td>1.58%</td>
<td>0.34%</td>
</tr>
<tr>
<td>1 → 9</td>
<td>-6.22%</td>
<td>2.28%</td>
<td>0.16%</td>
</tr>
<tr>
<td>5 → 3</td>
<td>-6.12%</td>
<td>3.00%</td>
<td>0.17%</td>
</tr>
<tr>
<td>Fashion-MNIST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 → 3</td>
<td>-2.23%</td>
<td>0.25%</td>
<td>0.01%</td>
</tr>
<tr>
<td>4 → 6</td>
<td>-9.96%</td>
<td>2.40%</td>
<td>0.09%</td>
</tr>
<tr>
<td>6 → 0</td>
<td>-8.87%</td>
<td>2.59%</td>
<td>0.20%</td>
</tr>
</tbody>
</table>

Figure 2.15: Relationship between global model accuracy and source class recall across changing percentages of malicious participants launching a label flipping data poisoning attack.

flipping scenario: loss in source class recall, loss in target class recall, and loss in recall of all remaining classes. We observe that the attack causes substantial change in source class recall (> 6% drop in most cases) and target class recall. However, the attack impact on the recall of remaining classes is an order of magnitude smaller. CIFAR-10 experiments show a maximum of 0.34% change in class recalls attributable to non-source and non-target classes and Fashion-MNIST experiments similarly show a maximum change of 0.2% attributable to non-source and non-target classes, both of which are relatively minor compared to source and target classes. Thus, the attack is causing the global model to misclassify instances belonging to \(c\text{\textsubscript{src}}\) as \(c\text{\textsubscript{target}}\) at test time while other classes remain relatively unimpacted, demonstrating its targeted nature towards \(c\text{\textsubscript{src}}\) and \(c\text{\textsubscript{target}}\). Considering the large impact of
the attack on source class recall, changes in source class recall therefore make up the vast majority of the decreases in global model accuracy caused by label flipping attacks in FL systems. This observation can also be seen in Figure 2.15 where the change in global model accuracy closely follows the change in source class recall.

The targeted nature of the label flipping attack allows for adversaries to remain under the radar in many FL systems. Consider systems where the data contain 100 classes or more, as is the case in CIFAR-100 [10] and ImageNet [81]. In such cases, targeted attacks become much more stealthy due to their limited impact to classes other than source and target. Our results demonstrate that data poisoning attacks, and label flipping attack specifically, are important considerations in evaluating vulnerabilities in the growing area of distributed machine learning.
CHAPTER 3
PRIVACY-PRESERVING MACHINE LEARNING

As a result of the burgeoning privacy and security issues discussed in Chapter 2, it is imperative to design machine learning systems that enforce formal privacy, security, and robustness guarantees. In this chapter we therefore look to the state-of-the-art techniques for protecting privacy and their effectiveness in (1) maintaining model efficacy and (2) mitigating vulnerability to attack.

3.1 Techniques for Privacy-Preserving Machine Learning

Let us consider privacy-preserving machine learning within the contest of the decision tree predictive model. The decision tree is one of the most fundamental inductive learning models, with widespread deployment in big data services and applications in multiple industries, including healthcare cost prediction [82], disease diagnosis [83, 84], computer network analysis [85], and credit-risk assessment [86, 87]. Each of these domains has major privacy concerns from legal constraints around data privacy, e.g., the HIPAA rule for healthcare data [5], to business concerns over revealing their private organizational data to competitors.

Using the decision tree as a representative model there are a number of privacy-preserving techniques to consider. The notion of privacy-preserving decision tree learning was introduced in the seminal paper [88]. A number of privacy-preserving data mining methods have since been proposed [88, 89, 90, 91, 92, 93, 94, 95, 96], of which some leverage randomization techniques [88, 90], some use other statistical approaches such as differential privacy [89, 91, 92, 93], and some leverage cryptographic mechanisms in the secure multiparty computation (SMC) framework [94, 95, 96]. We discuss the landscape of privacy-preserving machine learning techniques which mitigate machine learning vul-
nerability to privacy attacks through four broad categories: (1) attack-based techniques, (2) randomization-based, (3) differentially private, and (4) SMC.

3.1.1 Attack-Based Mitigation Techniques

We further categorize attack based mitigation techniques to protect against privacy attacks such as membership inference into two categories: model hardening and API hardening.

Model Hardening

Model hardening mitigation techniques are implemented during the training phase of the target model $F_t$. We consider four such techniques: (1) model choice, (2) fit control, (3) regularization, and (4) anonymization. In model choice, a service provider may introduce concerns of membership inference into their model selection process. For example, as was demonstrated in our experimentation in Chapter 2, a Naïve Bayes target model will be much more resilient to membership inference attacks than a decision tree and therefore may be the preferred model type for a particular machine learning service. For fit control, service providers may leverage parameters such as the decision tree’s complexity parameter for tree pruning to prevent overfitting and therefore decrease inference risk. Another technique is regularization where noise is added to a model’s loss function. This technique is particularly relevant to deep learning models. Finally, the service provider may introduce anonymization techniques into $D$ prior to training. That is, if $D$ is made to be $k$-anonymous prior to the training of $F_t$ then the impact of a single instance may be hid amongst $k$ others prior to training.

API Hardening

API hardening techniques are implemented during the prediction phase through the machine learning service. For example, the service API may introduce noise into the prediction vector before returning it to the user. This will reduce the adversary’s understanding
of exactly where an instance lies with respect to the target model’s decision boundary. Another option is to reduce the dimensionality of p. This can be done either by limiting the return value to the top \( k' < k \) values in p or even returning only the prediction label. Machine learning-as-a-service APIs may additionally be hardened through limiting queries and/or allocating a differential privacy budget to query responses [97]. This however requires service providers to track and control user queries which may not always be a suitable option.

Table 3.1: Effectiveness of mitigation techniques against the membership inference attack with a neural network target model trained on the Texas hospital admissions dataset.

<table>
<thead>
<tr>
<th>Mitigation</th>
<th>Parameter</th>
<th>Model Accuracy</th>
<th>Attack Accuracy</th>
<th>Utility Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td>55%</td>
<td>83%</td>
<td></td>
</tr>
<tr>
<td>Dimension Reduction [13]</td>
<td>( k' = 3 )</td>
<td>55%</td>
<td>83%</td>
<td>▼</td>
</tr>
<tr>
<td></td>
<td>( k' = 1 )</td>
<td>55%</td>
<td>82%</td>
<td>▼</td>
</tr>
<tr>
<td></td>
<td>label</td>
<td>55%</td>
<td>73%</td>
<td>▼</td>
</tr>
<tr>
<td>Regularization [13]</td>
<td>( L2 \lambda = 1e - 4 )</td>
<td>56%</td>
<td>80%</td>
<td>▲ 1%</td>
</tr>
<tr>
<td></td>
<td>( L2 \lambda = 5e - 4 )</td>
<td>57%</td>
<td>73%</td>
<td>▲ 2%</td>
</tr>
<tr>
<td></td>
<td>( L2 \lambda = 1e - 3 )</td>
<td>56%</td>
<td>66%</td>
<td>▲ 2%</td>
</tr>
<tr>
<td></td>
<td>( L2 \lambda = 5e - 3 )</td>
<td>35%</td>
<td>52%</td>
<td>▼ 20%</td>
</tr>
<tr>
<td>Adversarial Regularization [98]</td>
<td>0</td>
<td>51.9%</td>
<td>63%</td>
<td>▼ 3.1%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>47.5%</td>
<td>51%</td>
<td>▼ 7.5%</td>
</tr>
</tbody>
</table>

An issue that is pervasive in each mitigation technique is a loss of utility. This is demonstrated in Table 3.1, where we compare mitigation techniques with the Texas Hospital Admissions dataset which is developed from discharge data released by the Texas Department of State Health Services from 2006 to 2009 with a classification task of predicting the patient’s main procedure, focusing on the 100 most frequent procedures, making this dataset a 100-class classification problem.

We note here that there is not significant reduction in attack accuracy until only the label is returned when using a dimension reduction technique. Consider a hospital processing images of mass scans for cancer classification. A service which says “This mass is cancerous.” is significantly less useful than a service which says “There is a 56% chance that this mass is cancerous.” Additionally, even with the strongest dimension reduction, the
attack accuracy is still notably outperforming the baseline at 73%. Regularization on the other hand is able to successfully decrease attack accuracy to 52%. Unfortunately, to gain this level of protection the noise introduced to the model decreases model accuracy to 35%. This is a significant challenge in the mitigation of membership inference attacks.

Recent work in [98] has proposed a regularization technique specifically targeting membership inference attacks. While this approach is able to decrease attack accuracy to 51% with a comparatively moderate decrease in test accuracy to 47.5%, the training accuracy drops from 81.6% to 55%. It is therefore unclear to what degree regularization mitigation is simply a decrease in learning. If the model learns less from the training data in the first place it will naturally have less to reveal when under attack.

3.1.2 Randomization-Based Privacy

Randomization-based techniques have been popular in privacy-preserving data mining research for more than a decade [88, 89, 90, 91, 92, 93], with randomized distortion being one of the dominant techniques. The privacy-preserving algorithms belonging to the randomization group work by first perturbing the training data using randomization techniques and then using the distorted data as input to extract patterns and models. The randomization methodology attempts to hide the sensitive information from data miners by randomly modifying the data values under certain model-specific constraints, e.g., preserving the underlying probabilistic properties.

The most straightforward randomization technique is to preserve data privacy by adding random noise while ideally maintaining data utility. On one hand, the random noise should be added in such a way that the individual data values are distorted. On the other hand, with added random noise, the perturbed dataset should still preserve the model-specific feature signals from the data, such as underlying distribution properties, so that the model specific data utility patterns can still be “observed" and estimated with certain accuracy. An example of value distortion is additive noise based perturbation, which takes a training
dataset of \( n \) samples \( \{x_1, \ldots, x_n\} \) and returns a perturbed dataset in which value \( x_i \) is replaced by \( x_i + r_i \) \((i = 1, \ldots, n)\), where \( r_i \) is a random value drawn from some distribution. Alternatively, one can also use random data swapping (random shuffle) techniques to design a value distortion function such that a value returned from a field of a data sample record is still a true value in the training dataset, but from the same field in some other randomly chosen record [99].

However, randomization techniques may suffer from both low accuracy and inference attack vulnerability problems. With respect to accuracy, when the model-specific data utility is defined by latent and complex hidden statistical features, the accuracy of randomization methods will suffer significantly due to the lack of support for latent feature understanding and extraction. At the same time, several open challenges have been put forward with respect to the vulnerabilities of randomization techniques. As pointed out in [100], when the noise added can be statistically separated from the perturbed data, the input data privacy can be seriously compromised by inference over the model and its classification output. It is also true that when the dataset has significant skewedness, random data shuffling may no longer protect the individuals in the training dataset.

To address the additive random noise problem, multiplicative noise based random perturbation techniques are proposed, including the geometric rotation based technique [101, 102] and the random project-based approach [103].

In addition, we are also concerned with the unwanted exposure of statistical features that are repeatedly used for prediction.

In summary, randomness and uncertainty may not be equivalent for all cases. If random events and their properties can be captured and analyzed by probabilistic theorems and if we can easily and intuitively interpret probabilistic characterization of random processes, then the randomness may be exploited to compromise privacy unless one pays extra careful attention to the exposure of such structures.
By contrast with attack-based and randomization-based techniques, differential privacy (DP) is a formal privacy framework with a theoretical foundation and rigorous mathematical guarantees when effectively employed [104]. A machine learning algorithm is defined to be differentially private if and only if the inclusion of a single instance in the training dataset will cause only statistically insignificant changes to the output of the algorithm. Theoretical limits are set on such output changes in the definition of differential privacy, which is given formally as follows:

**Definition 1** (Differential Privacy [104]). A randomized mechanism $K$ provides $(\epsilon, \delta)$-differential privacy if for any two neighboring database $D_1$ and $D_2$ that differ in only a single entry, $\forall S \subseteq \text{Range}(K)$,

$$\Pr(K(D_1) \in S) \leq e^{\epsilon} \Pr(K(D_2) \in S) + \delta \quad (3.1)$$

If $\delta = 0$, $K$ is said to satisfy $\epsilon$-differential privacy.

To achieve $(\epsilon, \delta)$-differential privacy (DP), noise is added to the algorithm’s output. This noise is proportional to the sensitivity of the output. Sensitivity measures the maximum change of the output due to the inclusion of a single data instance.

**Definition 2** (Sensitivity [104]). For $f : D \rightarrow \mathbb{R}^k$, the sensitivity of $f$ is

$$\Delta = \max_{D_1, D_2} \| f(D_1) - f(D_2) \|_2 \quad (3.2)$$

for all $D_1$, $D_2$ differing in at most one element.

The noise mechanism which is used is therefore bounded by both the sensitivity of the function $f$, $S_f$, and the privacy parameters $(\epsilon, \delta)$. For example, consider the Gaussian mechanism defined as follows:

77
Definition 3 (Gaussian Noise Mechanism). \( M(d) \triangleq f(D) + N(0, S_f^2\sigma^2) \)

where \(N(0, S_f^2\sigma^2)\) is the normal distribution with mean 0 and standard deviation \(S_f\sigma\). A single application of the Gaussian mechanism in Definition 3 to a function \(f\) with sensitivity \(S_f\) satisfies \((\epsilon, \delta)\)-differential privacy if \(\delta \geq \frac{5}{4}\exp\left(-\frac{(\sigma\epsilon)^2}{2}\right)\) and \(\epsilon < 1\) [105].

Additionally, there exist several nice properties of differential privacy for either multiple iterations of a differentially private function \(f\) or the combination of multiple different functions \(f_1, f_2, \ldots\) wherein each \(f_i\) satisfies a corresponding \((\epsilon_i, \delta_i)\)-differential privacy. These composition properties are important for machine learning processes which often involve multiple passes over the training dataset \(D\). The formal composition properties of differential privacy include the following:

Definition 4 (Composition properties [105, 106]). Let \(f_1, f_2, \ldots, f_n\) be \(n\) algorithms, such that for each \(i \in [1, n]\), \(f_i\) satisfies \((\epsilon_i, \delta_i)\)-DP. Then, the following properties hold:

- **Sequential Composition**: Releasing the outputs \(f_1(D), f_2(D), \ldots, f_n(D)\) satisfies \((\sum_{i=1}^n \epsilon_i, \sum_{i=1}^n \delta_i)\)-DP.

- **Parallel Composition**: Executing each algorithm on a disjoint subset of \(D\) satisfies \((\max_i(\epsilon_i), \max_i(\delta_i))\)-DP.

- **Immunity to Post-processing**: Computing a function of the output of a differentially private algorithm does not deteriorate its privacy, e.g., publicly releasing the output of \(f_i(D)\) or using it as an input to another algorithm does not violate \((\epsilon_i, \delta_i)\)-DP.

The definition of membership inference leads to a natural assumption that a solution lies with differential privacy. There are, however, remaining questions on the exact relationship between differential privacy and membership inference attacks as the former is a theoretical framework whereas the latter is an attack with empirical results. Further investigation into this relationship is crucial in understanding what attackers are truly learning in a membership inference attack and to what degree differential privacy curtails attackers’ ability to learn this information.
For example, Table 2.8 shows that even if an instance which is relatively close to the target instance is provided, the membership inference attack can still be successful. Therefore, is the attacker learning that \( x \) is in \( D \), is the attacker learning that something similar to \( x \) is in \( D \), or is the attacker simply learning that \( x \) is consistent with a decision boundary informed by \( D \)?

Additionally, if differential privacy is the answer to mitigating membership inference vulnerability, what level of privacy (value of \( \epsilon \) or \((\epsilon, \delta)\)) is sufficient to provide protection? For example, the authors in [107] launch membership inference attacks against deep learning models trained with differentially private mechanisms. While a privacy parameter of \( \epsilon = 1 \) is able to protect against membership inference, dropping attack accuracy to 50.8% with the CIFAR-10 dataset, the training accuracy also drops from 94.4% in the non-private case to 24.7%. Even with a modest privacy parameter of \( \epsilon = 8 \) the authors report an attack accuracy 58.3% with a training accuracy of just 68.6%. The authors in [108] also remark that defense mechanisms based on differential privacy are not always effective, particularly when an attacker is able to mimic the behavior of the perturbation.

Further research is necessary to understand which differential privacy mechanisms under what settings are able to provide viable protection to membership inference without forfeiting model utility.

3.1.4 Secure Multiparty Computation

We next consider cryptographic techniques for privacy-preserving machine learning under the umbrella of secure multiparty computation. This can include deployment of non-machine learning specific cryptographic methods such as homomorphic encryption or may also include machine learning task-specific protocols developed within the SMC framework. We discuss both next.
Encryption Methods for Machine Learning

The mathematical functions required in the training and evaluation of machine learning models make many cryptographic systems unsuitable for privacy preserving machine learning. However, one category of cryptographic methods has seen rising popularity in the machine learning space: homomorphic encryption.

Let’s consider a sub-category of homomorphic encryption known as additively homomorphic encryption. An additively homomorphic encryption scheme is one wherein the following property is guaranteed:

\[ \text{Enc}(m_1) \circ \text{Enc}(m_2) = \text{Enc}(m_1 + m_2), \]

for some predefined function \( \circ \). Such schemes are popular in privacy-preserving data analytics as untrusted parties can perform operations on encrypted values.

One such additive homomorphic scheme is the Paillier cryptosystem [109], a probabilistic encryption scheme based on computations in the group \( \mathbb{Z}_{n^2}^* \), where \( n \) is an RSA modulus. An additional feature relevant in some machine learning settings is proposed in [110] where the authors extend this encryption scheme and propose a threshold variant. In the threshold variant, a set of participants is able to share the secret key such that no subset of the parties smaller than a pre-defined threshold is able to decrypt values.

While such cryptosystems are not designed for specific machine learning tasks, they can be directly deployed in both training and evaluation settings.

Secure Machine Learning Protocols

Another category of cryptographic approaches is the development of secure protocols specific to a machine learning task which provide provable security under the SMC framework. For example, secure model evaluation with a predictive model owner and a user of the trained model. We define (exact) secure two-party computation following Goldreich
and consider honest-but-curious adversaries (i.e., adversaries that follow the protocol instructions but try to learn additional information about the other parties’ inputs).

A two-party computation is specified by an ideal functionality that is a (possibly randomized) mapping $f: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$ from inputs $(a, b)$ to outputs $(c, d)$. Let $f_1(a, b)$ denote the first output of $f$ and $f_2(a, b)$ the second. A two-party protocol is a pair of polynomial-time interactive algorithms $\pi = (\pi_{Alice}, \pi_{Bob})$. Alice executes $\pi_{Alice}$ with input $a$ and randomness $r_{Alice}$ and Bob executes $\pi_{Bob}$ with input $b$ and randomness $r_{Bob}$. The execution proceeds in rounds, each party is able to send one message in each round to the other party. The messages are specified by $\pi$, given the party’s view, which consists of his input, randomness, and messages exchanged so far. Each party can also terminate, at any point, outputting some value based on his view. Let $\text{view}_{Alice}(a, b)$ denote Alice’s view of the protocol execution, i.e., her input, her randomness, and all the exchanged messages. Let $\text{view}_{Bob}(a, b)$ similarly denote Bob’s view of the protocol execution. Let $\text{output}_{Alice}(a, b)$ and $\text{output}_{Bob}(a, b)$ denote Alice’s and Bob’s outputs respectively.

**Definition 5.** A protocol $\pi$ privately computes $f$ with statistical security if for all possible inputs $(a, b)$ the following properties hold:

- **Correctness:**
  $$\{\text{output}_{Alice}(a, b), \text{output}_{Bob}(a, b)\} \equiv \{f(a, b)\}$$

- **Privacy:** There are simulators $S_{Alice}$ and $S_{Bob}$ such that:
  $$\{S_{Alice}(a, c), f_2(a, b)\} \approx \{\text{view}_{Alice}(a, b), \text{output}_{Bob}(a, b)\}$$
  $$\{f_1(a, b), S_{Bob}(b, d)\} \approx \{\text{output}_{Alice}(a, b), \text{view}_{Bob}(a, b)\}$$

where $\approx$ denotes statistical indistinguishability.

Definition 5 outlines the requirements for proving a protocol $\pi$ is a private computation of some ideal functionality $f$. The privacy requirement ensures that whatever an honest-
but-curious adversary learns from interactions within the protocol can also be learned by an ideal adversary that only learns the input and output of that party. A very useful paradigm for building private protocols is designing them in a modular way, using the Theorem 1, the Composition Theorem [111]:

**Theorem 1.** Let \( f, g \) be two-party functionalities. Let \( \pi^{f|g} \) be a private protocol for computing \( f \) using oracle calls to \( g \) and suppose that there is a private protocol \( \pi^g \) computing \( g \). Let \( \pi^f \) be the protocol obtained from \( \pi^{f|g} \) by independently using one instance of \( \pi^g \) for implementing each oracle call to \( g \). Then \( \pi^f \) privately computes \( f \).

**Secure Approximations** Additionally relevant to the development of secure protocols for machine learning is the concept of approximations. Recall from Section 2.1 that machine learning models are likely to be approximations of some target function \( c = \Gamma(x) \) according to the given training dataset. We note that the private computation of an approximation \( \overline{f} \) of a target functionality \( f \) can reveal more information than the target functionality itself. Imagine, for instance, the case where the output of \( \overline{f} \) is equal to the output of \( f \) in all bits except the least significant one, in which \( \overline{f} \) encodes one bit of the input of the other party. To ensure that the approximation \( \overline{f} \) does not leak additional information, we use the framework of Feigenbaum et al. [112, 113] for private approximations, using the notation of Kiltz et al. [114]. Only deterministic target functionalities \( f \) are considered, but the approximations \( \overline{f} \) can be randomized.

**Definition 6.** The functionality \( \overline{f} \) is an \( \varepsilon \)-approximation of \( f \) if for all possible inputs \((a, b)\), 
\[
|f(a, b) - \overline{f}(a, b)| < \varepsilon.
\]

**Definition 7.** The functionality \( \overline{f} \) is functionally private with respect to \( f \) if there is a simulator \( S \) such that for all possible inputs \((a, b)\), 
\[
\{S(f(a, b))\} \approx \{\overline{f}(a, b)\}.
\]

Note that functional privacy is a property of the functionality \( \overline{f} \) itself, and not of any protocol \( \pi \) implementing it. It captures the fact that the approximation error is independent from the inputs when conditioned on the output of the exact functionality.
Definition 8. A protocol $\pi$ is a private computation of an $\varepsilon$-approximation with respect to $f$ if $\pi$ privately computes a (possibly randomized) function $\overline{f}$ such that $\overline{f}$ is functionally private with respect to $f$ and is an $\varepsilon$-approximation of $f$.

Commodity Based Cryptography  Just as threshold variants provide a potentially valuable extension to some cryptosystems in ML settings, the commodity-based cryptography model can be a powerful tool for secure ML protocols. In the commodity based cryptography model [115, 116], a trusted initializer (TI) distributes values (i.e., the commodities) to the parties before the start of the protocol execution. The TI has no access to the parties’ secret inputs and does not communicate with the parties except for delivering the pre-distributed values during the setup. One main advantage of this model is the high computational efficiency that arises from the fact that the parties often only need to derandomize the pre-computed instances to match their own inputs. Another advantage is the computations are pre-distributed by a trusted initializer, and therefore most protocols yield perfect security. The trusted initializer functionality $F^D_{TI}$ is parametrized by an algorithm $D$, which is executed upon initialization to generate the correlated randomness $(P_{Alice}, P_{Bob})$ that is distributed to Alice and Bob respectively.

The commodity-based model [115, 116] is a setup assumption in which there is a trusted initializer who pre-distributes correlated data to the protocol participants during a setup phase, which is performed before the protocol execution (possibly far before the inputs are even fixed) and is independent of the protocol inputs. The trusted initializer does not
take part in the protocol execution after the setup phase; in particular, he does not learn
the parties’ inputs. The trusted initializer is modeled in this work by an ideal functionality
$F^D_{TI}$, which is parametrized by an algorithm $D$ that samples the correlated data to be pre-
distributed to the parties. See Figure 3.1 for details.

The main advantage of using this model is that, for many problems, it allows very ef-

cient solutions with unconditional security (in some cases even perfect security). This
follows from the fact that, in these problems, the trusted initializer can pre-distribute in-
stances computed on random inputs, which the parties later on only derandomize to match
their actual inputs. One such example is the case of the multiplication of secret shared val-
ues, which is an expensive operation in the plain model (i.e., the model in which there is no
setup assumption), but quite simple in the commodity-based model (see Sections 3.2.2 and
3.2.3). This model was already used to obtain very efficient solutions for primitives such
as commitments [117, 118, 119], oblivious transfer [115, 116], inner-product [120, 121],
linear algebra [122], string equality [121], verifiable secret sharing [123, 124], set intersec-
tion [121] and oblivious polynomial evaluation [125]. In the context of privacy preserving
machine learning, this model was used in [126, 127].

In practice, this correlated data can be obtained in different ways: (1) it can be dis-
tributed by a single trusted center that runs the setup phase and delivers the data to the
participants; (2) it can be pre-distributed by many not entirely trusted centers that do not
interact with (or even know) each other. In this case only a majority of honest centers is
needed [115, 128]; (3) it can be pre-computed by the parties themselves, using a multi-party
computation protocol in order to emulate the trusted initializer (in this case the main ad-

vantage is offloading the heavy computational steps to an offline phase that can be executed
at any idle time).

**The Universal Composability Framework** Finally, we must also discuss how to analyze
the security of proposed protocols. We specifically consider the Universal Composability
(UC) framework [129] security model. Only a short overview of the UC model is provided here, for more details please refer to the book of Cramer et al. [130]. The UC framework analyzes the security of cryptographic protocols under arbitrary composition, i.e., it considers scenarios where multiple copies of a protocol are executed concurrently with themselves and other protocols in a complex environment, such as the Internet. The UC composition theorem guarantees that any protocol proven UC-secure can also be securely composed with other copies of itself and other protocols. Apart from guaranteeing security in a realistic scenario, this framework also enables the design of complex protocols in a modular way.

In the UC model there are a set of parties $P_1, \ldots, P_u$, an adversary $A$ and an environment $Z$ that interact with each other. The main insight of the UC framework is that $Z$ captures all activity external to the current execution of the protocol. $Z$ is responsible for providing the inputs for the parties and $A$, and for receiving their outputs. The adversary $A$ is responsible for delivering the messages between the parties in the protocol execution (thus modeling that the adversary controls the network scheduling) and for corrupting parties, in which case he gains control over them. All entities are modeled as Interactive Turing Machines. For defining security, one first defines an idealized version $F$ of the functionality that the protocol is supposed to perform. The ideal functionality $F$ does what the protocol should do in a black box manner, i.e., given the inputs, the ideal functionality follows the primitive specification and returns the output as specified; however, the functionality must also deal with the actions of corrupted parties, such as invalid inputs and deviations from the protocol. After that, one shows that for every adversary $A$ there exists a simulator $S$ such that no environment $Z$ can distinguish between an execution of the protocol $\pi$ with the parties $P_1, \ldots, P_u$ and $A$, and an ideal execution with dummy parties that only forward inputs/outputs, $F$ and $S$. Some interesting points are: $S$ has no access to the contents of the messages sent between a party and $F$ if the party is not corrupted; $Z$ cannot see the messages sent between the parties and $F$ and also cannot see the messages.
sent between the parties in the real protocol execution. A protocol \( \pi \) securely UC-realizes an ideal functionality \( F \) if for every adversary \( A \) in the real world there exists a simulator \( S \) in the ideal world such that no \( Z \) can distinguish an execution of the protocol \( \pi \) with the parties and \( A \) from an execution of the ideal functionality \( F \) with the dummy parties and \( S \). This is stated formally as follows:

**Definition 9.** ([129]) A protocol \( \pi \) is said to UC-realize an ideal functionality \( F \) if, for every adversary \( A \), there exists a simulator \( S \) such that, for every environment \( Z \), the following holds:

\[
\text{EXEC}_{\pi,A,Z}(n) \approx \text{IDEAL}_{F,S,Z}(n)
\]

where \( \approx \) denotes computational indistinguishability, \( \text{EXEC}_{\pi,A,Z}(n) \) represents the view of \( Z \) in the real protocol execution with \( A \) and the parties (with security parameter \( n \)) and \( \text{IDEAL}_{F,S,Z}(n) \) represents the view of \( Z \) in the ideal execution with the functionality \( F \), the simulator \( S \) and the dummy parties. The probability distribution is taken over the randomness of the parties.

Computational indistinguishability between real and ideal executions guarantees that the protocol is secure against probabilistic polynomial time adversaries. Even though this is enough for the security requirements of many protocols and applications, it is also very interesting to achieve perfect security against computationally unbounded adversaries, which is the case considered in this work.

**Setup Assumption** It is a well-known fact that two-party computation and multi-party computation with dishonest majority is only possible with additional assumptions, either computational or setup assumptions. In the case of UC-secure protocols, the restriction is even bigger: non-trivial two-party and multi-party functionalities cannot be realized without setup assumptions [131, 132]. Some setup assumptions allowing the realization of non-trivial functionalities are: the existence of a common reference string [131, 132,
133], a public-key infrastructure [134] or noisy-channels [135, 136], the random oracle model [137], signature cards [138] and tamperproof hardware [139, 140, 141]. Pre-distributed correlated randomness, i.e., the commodity-based model, constitutes an additional attractive setup assumption.

**Simulation Strategy** The simulation strategy for proving the security of proposed protocols is simple and will be described briefly: all the messages look uniformly random from the recipient’s point of view, except for the messages that open some secret share to a party, but these ones can be easily simulated using the output of the respective functionalities. Therefore a simulator $S$, having the leverage of being able to simulate the trusted initializer functionality $F_{TI}^D$ in the ideal world, can easily perform a perfect simulation of a real protocol execution; therefore making the real and ideal worlds indistinguishable for any environment $\mathcal{Z}$.

In the ideal functionalities the messages are public delayed outputs, meaning that the simulator is first asked whether they should be delivered or not (this is due to the modeling that the adversary controls the network scheduling). This fact as well as the session identifications are omitted from our functionalities’ descriptions in later sections for the sake of readability.

3.1.5 Comparison of Approaches with Decision Tree Models

We now look to compare randomization-based, differentially private, and secure multiparty computation approaches to machine learning. We will do so via the decision tree model as a representative learner. We will first review work in each category as it relates to decision tree learning before comparing method pros and cons.
Random forest and random decision trees are candidate randomization techniques for providing baseline privacy protection of decision tree models. Random forests are an ensemble learning method that use multiple decision trees and outputs the class that is the mean prediction of the individual trees, aiming at correcting for decision trees which over-fit their training set [142]. The first algorithm for random decision forests was developed by Tin Kam Ho [143] who uses the random subspace method to implement the "stochastic discrimination" approach to classification [144]. Random forests was later trademarked using an extension with bagging developed by Leo Breiman [23] and others.

Random decision trees represent another orthogonal dimension of efforts. A random decision tree is a variant of the traditional decision tree proposed by Fan et al. [145] where nodes split on randomly chosen features rather than features chosen using statistical measures. The main idea of random decision trees is to build the structure of $N$ random decision trees solely based on the given feature set without using the training data samples. Each discrete feature can be used only once in a decision path from the root to a leaf node. Continuous features can be either discretized and treated as discrete features or alternatively be divided by randomly picking a splitting point (i.e., the dividing value with less than and greater or equal to as the two branches) with a different splitting point chosen each time. After the trees are constructed, we only need to scan the dataset once to update the statistics of the leaf nodes in each tree. Random decision trees are shown to be efficient implementations of Bayes’ Optimal Classifier. With large datasets, random decision trees are shown to be as accurate as the single best decision tree carefully built using statistical criteria simply because a single decision tree will often either significantly over-estimate or under-estimate [145]. Interestingly, random decision trees can be much more robust against certain privacy risks of public release of decision tree models.
**Differentially Privacy Decision Tree Learning**

When using the differential privacy framework to support privacy-preserving decision tree learning, the choice of $\varepsilon$ represents a significant trade-off. More noise is likely to decrease the accuracy and usability of the resulting decision tree, while less noise will decrease privacy. Additionally, one must consider the number of queries which are made against the data in creating any high-level structure [89]. While differential privacy allows the “release of coarse-grained information while keeping private the details” [146], there is a constant trade-off between the level of privacy and the usability of that information.

There exist multiple differential privacy solutions for decision tree learning including [91, 92, 89, 147, 148, 93], with each taking a slightly different approach to achieving a differentially private learning algorithm.

The first theoretical contribution made by Blum et al. [91] built a differentially private ID3 decision tree through differentially private queries to the original dataset using the proposed SuLQ framework (SubLinear Queries). This approach, however, suffers from poor accuracy [92, 89]. This illustrates two important points: (1) the addition of noise in privacy-preserving decision tree training can dramatically impact the accuracy of the generated model and therefore solutions within the differential privacy framework should always be tested experimentally, and (2) for the resulting model to be usable, privacy must be guaranteed in the presence of reasonable query counts on the model structure.

Friedman et al. [92] extends the above approach [91] from two aspects: (1) it ensures that each step in the inductive learning process for decision tree training is differentially private using mechanisms from [149, 150] and (2) it experimentally highlights the tension between privacy, accuracy, and dataset size.

To better address the trade-offs between privacy, accuracy, and dataset size, Jagannathan et al. leverage random decision trees in an ensemble instead of relying on the traditional decision tree. As the structure of each tree is randomly chosen, no privacy protection is required for the tree structure. Thus, the noise traditionally incorporated in differential
privacy solutions is limited to the leaf nodes. Jagannathan et al. show that an ensemble of these random decision trees with noisy leaf nodes can perform well when compared to single decision trees trained without privacy.

Several extensions have been made to the approach by Jagannathan et al.: Bojarski et al. similarly utilize the random decision tree structure with noise restricted to the class distributions within the leaf nodes in [147]. Patil and Singh [148] extend [89] by using multiple noisy trees to combat decreased accuracy. Rana et al. [93] use the step-wise differential privacy technique in [92] to train a random forest model by distributing the privacy parameter amongst the decision trees within the ensemble, thus making each individual tree less noisy while preserving the privacy of the entire ensemble.

**Decision Tree Learning with Secure Multiparty Computation**

When considering security based in the simulation paradigm with respect to privacy preserving decision tree learning, we must keep in mind that security under the ideal/real simulation paradigm works in isolation. While one execution may be considered secure within the simulation paradigm and information leaked from one single output may be acceptable, repetitions can lead to greater information leakage. Consider the following example: Alice and Bob would like to know which one of them earns a higher salary, but neither wishes to reveal their exact salary. This is commonly referred to as the Millionaires’ problem. Alice and Bob can participate in a protocol \( \pi \) which allows each to provide their salaries as input and receive as output the name of the individual with the higher salary. We will assume that \( \pi \) is secure under the ideal/real simulation paradigm. Given this scenario, Alice can run \( \pi \) repeatedly, altering her input each time, to discover Bob’s exact salary. Since Alice could lodge this same attack on the ideal execution, \( \pi \) remains a secure computation under ideal/real simulation paradigm. This weakness however leads to the leakage of Bob’s private information. Researchers in privacy-preserving decision tree learning must consider the scenario where the tree structure is used to evaluate different instances many times over.
**Private Multiparty Training**  The first to propose a privacy-preserving training solution for decision trees were Lindell and Pinkas in [151]. They proposed a secure two-party protocol for training an ID3 tree over horizontally partitioned data.

The most efficient secure multiparty protocol for the privacy-preserving training of decision trees is proposed by de Hoogh et al. in [96]. This solution is implemented within the Virtual Ideal Functionality Framework (VIFF) using operations over large finite fields based on Shamir’s secret sharing scheme [152] and provides provable security against semi-honest adversaries. As noted by [95] however, this result is limited to the consideration of categorical attributes and does not scale well for fine grained numerical attributes. Additionally, the complexity of the protocol increases exponentially on the bit-length representation of a category.

**Private Evaluation**  There exist multiple secure multiparty computation solutions for decision tree evaluation including [94, 153, 95, 96]. Decision tree classification is traditionally a two-party scenario where there is a server which holds the machine learning structure, a decision tree, and a client which holds the classification instance. At the end of a secure evaluation protocol the ideal outcome would give the client the classification result of their instance based on the server’s decision tree without revealing any information on the server’s model while the server would learn nothing.

Three different approaches have been studied. The first approach provides solutions for decision tree evaluation which use additive homomorphic encryption schemes and the oblivious transfer scheme of Naor and Pinkas [154] to allow for computation over the private data. [153] provide privacy against the weaker semi-honest adversarial model, and [94] offers solutions under both the semi-honest and malicious adversarial models. The second approach [155] uses trusted SGX processors for data-oblivious decision tree evaluation. Our approach, which is detailed in Section 3.2.3 and [95] uses Shamir’s secret sharing scheme in the commodity-based model and provides unconditionally secure proto-
Table 3.2: High-level comparison of representative privacy-preserving decision tree learning approaches.

<table>
<thead>
<tr>
<th>comparison metric</th>
<th>random perturbation</th>
<th>differential privacy</th>
<th>secure multiparty computation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mining w/ Random</td>
<td>SuLQ Framework [91]</td>
<td>Decision Tree: Evaluate [94]</td>
</tr>
<tr>
<td></td>
<td>Noise [88]</td>
<td>Decision Tree</td>
<td>Decision Tree w/ TI [95]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Framework [90]</td>
<td>Secure Tree Learn [96]</td>
</tr>
<tr>
<td>input data during training</td>
<td>✓ perturbed ✓</td>
<td>✓ perturbed ✓</td>
<td>✓ original ✓</td>
</tr>
<tr>
<td>input data during evaluation</td>
<td>✓ original</td>
<td>✓ perturbed ✓</td>
<td>✓ original ✓</td>
</tr>
<tr>
<td>learning model construction</td>
<td>public private</td>
<td>public public</td>
<td>public public public public</td>
</tr>
<tr>
<td>process</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decision tree produced by</td>
<td>✓</td>
<td>✓ ✓</td>
<td></td>
</tr>
<tr>
<td>inductive learner output of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>learner</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>computational complexity</td>
<td>statistical</td>
<td>statistical</td>
<td>homomorphic</td>
</tr>
<tr>
<td>requirements</td>
<td>operations</td>
<td>operations</td>
<td>encryptions</td>
</tr>
<tr>
<td>level of accuracy loss</td>
<td>0 − 10%</td>
<td>&gt; 10%</td>
<td>≥ 10%</td>
</tr>
<tr>
<td>cols for decision tree</td>
<td></td>
<td></td>
<td>large finite field operations</td>
</tr>
<tr>
<td>evaluation in the presence</td>
<td></td>
<td></td>
<td>large finite field operations</td>
</tr>
<tr>
<td>of honest-but-curious</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adversaries, with a set-up</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phase which uses a trusted</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>intializer to make evaluation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>more efficient.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comparison of Representative Works

When considering privacy-preserving decision tree processes we must consider what is being protected, what level of protection is provided, and against whom this protection is guaranteed. Across the three frameworks for privacy-preserving machine learning, randomization based obfuscation, differential privacy, and secure multiparty computation, each contribute different approaches to these considerations. Table 3.2 highlights some representative works for each of these approaches and the treatment of privacy, efficiency, and accuracy considerations within each work.

We can see at a high level that randomization and differential privacy techniques generally require access to the data during the training phase. Afterward, the evaluation phase is essentially trusted to be in the clear as the generated model is perturbed to protect the underlying training data. This leads to relatively fast approaches (statistical operations) but limits the resulting accuracy of the model. Comparatively, secure multiparty computation results usually protect all data elements throughout the life cycle of decision tree learning.
and allow for accuracy equivalent to that seen in privacy-free environments. These gains are made at significant efficiency costs as secure multiparty computation requires either large finite field operations or homomorphic encryptions.

3.1.6 Federated Learning

One final approach to privacy in modern machine learning which has gained attention in recent years is federated learning. In traditional machine learning (ML) environments, training data is centrally held by one organization executing the learning algorithm.

Recall however from Section 2.4 that the area of federated learning (FL) aims to address more restrictive, privacy-conscious environments wherein data holders collaborate throughout the learning process rather than relying on a trusted third party to hold data [156, 157]. Specifically, data holders in tightly federated ML systems run a learning algorithm locally and only exchange model parameters, which are aggregated and redistributed by one or more central entities. We note, however, that this approach is not sufficient to provide reasonable data privacy guarantees as shown in Section 2.4.1. Beyond the explicit data disclosure protected by the FL approach, we must also consider that information can be inferred from the learning process [14] and that information that can be traced back to its source in the resulting trained model [13].

The most recent research efforts in FL have been focusing on the functionality [158], scalability [47], privacy [157, 159], tackling heterogeneity [160, 161], and system optimization [162, 158, 163, 164].

For further background information, [165, 166] provide in-depth discussion for the current challenges and state-of-the-art systems in federated learning.

3.2 Application of Privacy-Preserving Techniques

Up to this point we have broadly discussed the techniques available for mitigating privacy vulnerabilities in machine learning. We next look to our work in applying such techniques
to specific machine learning tasks including the training of deep neural networks with differential privacy, training linear regression models with secure multiparty computation, and evaluating a range of classifiers with SMC as well as real world and system considerations for the practical use of privacy-preserving machine learning.

3.2.1 Differentially Private Deep Learning

Recall that differential privacy provides a formal mathematical framework, which bounds the impact of individual instances on the output of a function when this function is constructed in a differentially private manner. In the context of deep learning (DL), a deep neural network model is said to be differentially private if its training function is differentially private therefore guaranteeing the privacy of its training data. Thus, conceptually, differential privacy provides a natural mitigation strategy against membership inference threats. If training processes could limit the impact that any single individual instance may have on the model output, then the differential privacy theory [104] would guarantee that an attacker would be incapable of identifying with high confidence that an individual example is included in the training dataset. Additionally, recent research has indicated that differential privacy also has a connection to the model robustness against adversarial examples [167].

Applying differential privacy to deep neural network (DNN) training requires changes to the vanilla DNN training algorithm. DNNs are complex, sequentially stacked neural networks containing multiple layers of interconnected nodes. Each node represents the dataset in a unique way and each layer of networked nodes processes the input from previous layers using learned weights and a pre-defined activation function. The objective in training a DNN is to find the optimal weight values for each node in the multi-tier networks. This is accomplished by making multiple passes over the entire dataset with each pass constituting one epoch. Within each epoch, the entire dataset is partitioned into many mini-batches of equal size and the algorithm processes these batches sequentially, each including only a
subset of the data. When processing one batch, the data is fed forward through the network using the existing weight values. A pre-defined loss function is computed for the errors made by the neural network learner with respect to the current batch of data. An optimizer, such as stochastic gradient descent (SGD), is then used to propagate these errors backward through the network. The weights are then updated according to the errors and the learning rate set by the training algorithm. The higher the learning rate value, the larger the update made in response to the backward propagation of errors.

Differentially private deep learning can be implemented by adding a small amount of noise to the updates made to the network such that there is only a marginal difference between the following two scenarios: (1) when a particular individual is included within the training dataset and (2) when the individual is absent from the training dataset. The noise added to the updates is sampled from a Gaussian distribution with scale determined by an appropriate noise parameter $\sigma$ corresponding to a desired level of privacy and controlled sensitivity. That is, the privacy budget $\epsilon$, according to Definition 1, should constrain the value of $\sigma$ at each epoch. Let $n$ denote the number of epochs for the DNN training, a pre-defined hyper-parameter set at the training configuration as the termination condition. Let one epoch satisfy $\epsilon_i$-differential privacy given $\sigma_i$ from the Gaussian mechanism in Definition 3. Then, a traditional accounting using the composition properties of differential privacy (Definition 4) would dictate that $n$ epochs would result in an overall privacy guarantee of $\epsilon$ if $\epsilon_i = \epsilon/n$, and each epoch employed the Gaussian mechanism with the same value $\sigma_i$. We refer to this approach the fixed noise perturbation method [168]. We provide an alternative approach in [169] advocating a variable noise perturbation approach, which uses a decaying function to manage the total privacy budget $\epsilon$ and define variable noise scale $\sigma$ based on different settings of $\epsilon_i$ for each different epoch $i$ ($1 \leq i \leq n$), aiming to add variable amount of noise to the $i^{th}$ epoch in a decreasing manner as the training progresses in epochs. Thus, we have $\epsilon_i \neq \epsilon_j$ for $i \neq j$, $1 \leq i, j \leq n$ and $\sigma_i$ for a given epoch $i$ is bounded by its allocated privacy budget $\epsilon_i$. The same overall privacy guarantee is met
when $\sum_{i=1}^{n} \epsilon_i = \epsilon$ for the differentially private DNN training of $n$ epochs.

**Differentially Private Deep Learning with Fixed $\sigma$**

The first differentially private approach to training deep neural networks is proposed by Abadi et al. [168] and implemented on the tensorflow deep learning framework [170]. A summary of their approach is given in Algorithm 1. To apply differential privacy, the sensitivity of each epoch is bounded by a clipping value $C$, specifying that an instance may impact weight updates by at most the value $C$. To achieve differential privacy, weight updates at the end of each batch include noise injection according to the sensitivity defined by $C$ and the scale of noise $\sigma$. The choice of $\sigma$ is directly related to the overall privacy guarantee.

**Algorithm 1** Differentially Private Deep Learning: Fixed $\sigma$

**Input:** Dataset $D$ containing training instances $x_1, ..., x_N$, loss function $L(\theta) = 1/N \sum_i L(\theta, x_i)$, learning rate $\eta_t$, noise scale $\sigma$, batch size $L$, norm bound $C$, number of epochs $E$, $N$: Gaussian noise mechanism

**Initialize** $\theta_0$ randomly

**Set** $T = E \cdot N/L$

**for** $t \in [T]$ **do**

- Set sampling probability $q = L/N$
- Take a random sample $L_t$ from $D$

**Compute gradient**
- For each $x_i \in L_t$, compute $g_t(x_i) \leftarrow \nabla_{\theta_t} L(\theta_t, x_i)$

**Clip gradient**
- $\bar{g}_t(x_i) \leftarrow g_t(x_i) / \max(1, \|g_t(x_i)\|_2)$

**Add Noise**
- $\tilde{g}_t \leftarrow 1/L (\sum_i \bar{g}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 I))$

**Descent**
- $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{g}_t$

**Output** $\theta_T$ and compute the overall privacy cost ($\epsilon, \delta$) using a privacy accounting method.

Let $\sigma = \frac{\sqrt{2 \log(1/\delta)}}{\epsilon}$, then according to differential privacy theory [105], each step (processing of a batch) is $(\epsilon, \delta)$-differentially private. If $L_t$ is randomly sampled from $D$ then additional properties of random sampling [171, 172] may be applied. Each step then becomes $(O(q\epsilon), q\delta)$-differentially private. The moments accountant privacy accounting
method is also introduced in [168] to prove that Algorithm 1 is \(O(q\epsilon \sqrt{T}), \delta\)-differentially private given appropriate parameter settings.

We refer to Algorithm 1 as the fixed noise perturbation approach as each epoch is treated equally by introducing the same noise scale to every parameter update.

**Differentially Private Deep Learning with Variable \(\sigma\)**

We propose the variable noise perturbation approach to differentially private deep learning in [169]. We extends the fixed noise scale of \(\sigma\) over the total \(n\) epochs in [168] by introducing two new capabilities. First, we point out a limitation of the approach outlined in Algorithm 1. Namely, Algorithm 1 specifically calls for random sampling, wherein each batch is selected randomly with replacement. However, the most popular implementation for partitioning a dataset into mini-batches in many deep learning frameworks is random shuffling, wherein the dataset is shuffled and then partitioned into evenly sized batches. In order to develop a differentially private DNN model under random shuffling, we extend Algorithm 1 of [168] by introducing a new privacy accounting method.

Additionally, we analyze the problem of using fixed noise scale (fixed \(\sigma\) values), and propose employing different noise scales to the weight updates at different stages of the training process. Specifically, we propose a set of methods for privacy budget allocation, which improve model accuracy by progressively reducing the noise scale as the training progresses. The variable \(\sigma\) noise scale approach is inspired by two observations. First, as the training progresses, the model begins to converge causing the noise being introduced to the updates to potentially become more impactful. This slows down the rate of model convergence and causes later epochs to no longer increase model accuracy compared to non-private scenarios. Second, the research on improving training accuracy and convergence rate of DNN training has led to a new generation of learning rate functions that replace the constant learning rate baseline by decaying learning rate functions and cyclic learning rates [173, 174]. We employ a similar set of decay functions to add noise at a
decreased scale. That is, the noise defined by $\sigma_i$ at the \(i^{th}\) epoch as the training progresses is less than $\sigma_j$ at the \(j^{th}\) epoch given \(1 \leq j < i \leq n\). The performance of four different types of decay functions to introduce variable noise scale by partitioning $\sigma$ over \(n\) epochs were evaluated.

**Important Implementation Factors**

Unfortunately, differential privacy can be challenging to implement efficiently in deep neural network training for a number of reasons including determining the privacy budget, the availability of accuracy boosting methods such as transfer learning, and the complexity of optimizing the multitude of parameters in privacy-preserving deep learning.

**Choosing Epsilon** In differentially private algorithms, the $\epsilon$ value dictates the amount of noise which must be introduced into the DNN model training and therefore the theoretical privacy bound. Choosing the correct $\epsilon$ value requires a careful balance between tolerable privacy leakage given the practical setting as well as the tolerable utility loss.

For example, Naldi and D’Acquisto propose a method in [175] for finding the optimal $\epsilon$ value to meet a certain accuracy for Laplacian mechanisms. Lee and Clifton alternatively employ the approach in [176] to analyze a particular adversarial model. Hsu et. al [177], on the other hand, takes an individual’s perspective on data privacy by opt-in incentivization. Kohli and Laskowski [178] also promote choosing an $\epsilon$ based on individual privacy preferences.

Despite these existing approaches, determining the “right” $\epsilon$ value remains a complex problem and is likely to be highly dependent on the privacy policy of the organization that owns the model and the dataset, the vulnerability of the model, the sensitivity of the data, and the tolerance to utility loss in the given setting. Additionally, there might be scenarios, such as the healthcare setting, in which even small degrees of utility loss are intolerable and are combined with stringent privacy constraints given highly sensitive data. In these cases,
it may be hard or even impossible to find a good $\epsilon$ value for existing differentially private DNN training techniques.

**The Role of Transfer learning** Another key consideration is the use of transfer learning for dealing with more complex datasets [168, 169]. For example, model parameters may be initialized by training on a non-private, similar dataset. The private dataset is then only used to further hone a subset of the model parameters. This helps to reduce the number of parameters affected by the noise addition required by exercising differential privacy. The use of transfer learning however relies on a strong assumption that such a non-private, similar dataset exists and is available. In many cases, this assumption may be unrealistic.

**Parameter Optimization** In addition to the privacy budget parameters $(\epsilon, \delta)$, differentially private deep learning introduces influential privacy parameters, such as the clipping value $C$ that bounds the sensitivity for each epoch and the noise scale approach (including potential decay parameters) for $\sigma$. The settings of these privacy parameters may impact both the training convergence rate, and thus the training time, and the training and testing accuracy. However, tuning these privacy parameters becomes much more complex as we need to take into account the many learning hyper-parameters already present in DNN training, such as the number of epochs, the batch size, the learning rate policy, and the optimization algorithm. These hyper-parameters need to be carefully configured for high performance training of deep neural networks in a non-private setting. For deep learning with differential privacy, one needs to configure the privacy parameters by considering the side effect on other hyper-parameters and re-configure the previously tuned learning hyper-parameters to account for the privacy approach.

For example, for a fixed total privacy budget values $(\epsilon, \delta)$, too small of a number of epochs may result in reduced accuracy due to insufficient time to learn. A higher number of epochs however will result in a higher $\sigma$ value required each epoch (recall Algorithm 1). Therefore, a carefully tuned hyper-parameter for a non-private setting, such as the optimal
number of epochs, may no longer be effective when differentially private deep learning is enabled.

A number of challenging questions remain open problems in differentially private DNN training, such as at what point do more epochs lead to accuracy loss under a particular privacy setting? What is the right noise decaying function for effective deep learning with differential privacy? Can we learn privately with high accuracy given complex datasets? The balance of the many parameters in a differentially private deep learning system presents new challenges to practitioners.

### 3.2.2 Privacy-Preserving Regression Training with SMC

#### Secure Training of Linear Regression Models

There are many attempts in the literature at obtaining secure linear regression protocols over distributed databases. Most of them clearly do not even aim at obtaining the level of privacy usually required by modern cryptographic protocols (such as Karr et al. [179] and Du et al.[180], see also [181, 182]).

The pioneering work of Hall et al. [183] actually presents a protocol that achieves cryptographic security within the framework of secure two-party protocols and simulation based definitions of security, as proposed by Goldreich in [111]. However, we remark that as some of the protocols they propose rely on approximations, rather than exact computations, the correct framework to be used is that of Feigenbaum et al. [112, 113], instead of Goldreich’s. Additionally, Hall et al. [183] uses operations in a finite field and homomorphic encryption as a building block. However, the (interesting version of the) linear regression problem deals with numbers in \( \mathbb{R} \), or at least in \( \mathbb{Q} \). To cope with this problem, a fixed-point data type and its representation in the finite field are defined in [183]. In such an approach, it is necessary to perform a truncation after each multiplication, but the description of the truncation protocol of [183] has a small (correctable) problem as explained in [127] Appendix A. Finally, the overall computing time for solving the linear regression problem
for 51K input vectors, each with 22 features, is two days [183]. The online phase of our protocol solves this problem in a few seconds. Even when considering the running time of the offline phase of our computationally secure protocol, by exploiting its embarrassingly parallelization property, the overall running time is still in the order of minutes for such a number of features and vectors.

In [184], a solution is proposed based on homomorphic encryption and garbled circuits for a scenario where many parties upload their data to a third party responsible for obtaining the regression model (with the help of a Crypto Service Provider, responsible for performing heavy cryptographic operations). The Crypto Service Provider is a semi-honest trusted party that is assumed to not collude with other players and actively engages in the protocol during its execution.

However, we now present our information theoretical solution for linear regression model training leveraging the trusted initializer functionality. Our protocol consists of two phases: the offline setup phase and the online training phase. The trusted initializer does not engage in the protocol after the setup phase and our online phase is much faster than the protocol presented in [184]. Even when we add up the offline phase and the online phase running times, in the case of our computationally secure protocol, when multiple cores are available for the offline phase computations, the overall running time is less for our protocol.

We additionally assess our secure linear regression training approach by implementing and analyzing the protocol results using ten real datasets. We chose a variety of different datasets based on their number of features and instances. Some of our datasets have millions of vectors. We are unaware of any other work on secure linear regression where real datasets of this size have been analysed before. For example, in [183] and in [184], the real datasets used had thousands of vectors.
SMC Building Blocks for the Training of a Linear Regression Model

Assume that we have a set of training examples (real vectors)

\[(a_1(x_i), a_2(x_i), \ldots, a_m(x_i), y_i)\]

where \(a_j(x_i)\) is the value of the input attribute \(a_j\) for the training example \(x_i\) \((i = 1, \ldots, n)\) and \(y_i\) is the associated output. The goal is to leverage these training examples to predict the unknown outcome for a previously unseen input as accurately as possible. To this end, we want to learn a linear function

\[y = \beta_1 a_1(x) + \beta_2 a_2(x) + \ldots + \beta_m a_m(x) + b\]

that best approximates the relation between the input variables \(a_1(x), a_2(x), \ldots, a_m(x)\) and the response variable \(y\). Throughout this section we assume that all variables are real numbers and that we aim to find real values for the parameters \(\beta_1, \beta_2, \ldots, \beta_m\) and \(b\) that minimize the empirical risk function

\[\frac{1}{n} \sum_{i=1}^{n} ((\beta_1 a_1(x_i) + \beta_2 a_2(x_i) + \ldots + \beta_m a_m(x_i) + b) - y_i)^2\]  \(3.3\)

which is the mean squared error over the training instances. For notational convenience, we switch to the homogeneous version of the linear function and we use vector notation, i.e. let

- \(\mathbf{x}_i = (a_0(x_i), a_1(x_i), a_2(x_i), \ldots, a_m(x_i))\), with \(a_0(x_i) = 1\) for all \(i \in \{1, \ldots, n\}\)
- \(\mathbf{\beta} = (\beta_0, \beta_1, \ldots, \beta_m)\), with \(\beta_0 = b\)
Using \( \langle \beta, x_i \rangle \) to denote the dot product of \( \beta \) and \( x_i \), minimizing (3.3) amounts to calculating the gradient and comparing it to zero, i.e. solving

\[
\frac{2}{n} \sum_{i=1}^{n} (\langle \beta, x_i \rangle - y_i) x_i = 0
\]  

(3.4)

The solution to (3.4) is\(^1\)

\[
\beta = (X^T X)^{-1} X^T y
\]  

(3.5)

with

\[
X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}
\]

The scenarios that we are interested in are those in which the training data is not owned by a single party but is instead distributed across multiple parties who are not willing to disclose it. Our experiments presented later in this section correspond to scenarios in which \( X \) is partitioned column-wise across two parties, i.e. Alice and Bob have information about different features of the same instances. However, as will become clear below, our protocols work in all scenarios in which Alice has a share \( X_{\text{Alice}} \) and Bob has a share \( X_{\text{Bob}} \) such that \( X_{\text{Alice}} + X_{\text{Bob}} = X \), regardless of whether the dataset \( X \) is sliced column-wise, row-wise, or a mixture of the two. In our experiments we also assume that Bob has the vector \( Y \). However, our protocol can also handle the case when \( Y \) is distributed over two or more players.

We next give an overview of our solution. The basic idea is to reduce the problem of securely computing linear regression to the problem of securely computing products of matrices. The protocol for computing products of matrices works only for elements of the matrices belonging to a finite field. Thus, Alice and Bob should be able to map their real

\(^1\text{Assuming that } X^T X \text{ is invertible}\)
valued fixed precision inputs to elements of a finite field (as described later in this section). Our protocol works in a shared input model in which each party holds some elements of the design matrix. Each party creates its share of the design matrix by mapping their respective real valued inputs to elements of a finite field and putting them on the respective position of the matrix and then filling the remaining positions of the matrix’s share with zeros. I.e., the shares $X_{\text{Alice}}$ and $X_{\text{Bob}}$ are such that $X_{\text{Alice}} + X_{\text{Bob}} = X$ where $X$ is the design matrix mapped into the finite field. The two protocol phases can then be summarized as follows:

1. Offline phase: in the information-theoretically secure protocol, Alice and Bob receive correlated data from the Trusted Initializer. In the case of the computationally secure protocol, we provide a protocol later in this section for replacing the trusted initializer assumption.

2. Online Phase:

(a) The players map their fixed precision real valued inputs to elements of a finite field and create the shares of $X$ as described above.

(b) The players compute over their shares using the protocols for matrix multiplication and for computing the inverse of a Covariance Matrix (protocols for each are described later in this section) in order to obtain shares of the estimated regression coefficient vector.

(c) The players exchange their shares of the estimated regression coefficient vector and reconstruct it.

After next presenting the building blocks for our protocols, we will reiterate the information theoretically secure and the computationally secure protocol for linear regression at a more concrete level of detail.

In presenting our protocols, we first introduce an ideal functionality that captures the behaviour of a secure instance of the protocol in question. We then present the protocol and prove that the protocol is as secure as the ideal functionality.
Secure Distributed Matrix Multiplication Protocol $\pi_{\text{DMM}}$ for Linear Regression Training

1. At the setup, the trusted initializer chooses uniformly random $A_{\text{Alice}}, A_{\text{Bob}} \in \mathbb{Z}^{n_1 \times n_2}_q$, $B_{\text{Alice}}, B_{\text{Bob}} \in \mathbb{Z}^{n_2 \times n_3}_q$, and $T \in \mathbb{Z}^{n_1 \times n_3}_q$, and distributes the values $A_{\text{Alice}}, B_{\text{Alice}}, T$ to Alice and the values $A_{\text{Bob}}, B_{\text{Bob}}, C = (A_{\text{Alice}}B_{\text{Bob}} + A_{\text{Bob}}B_{\text{Alice}} - T)$ to Bob.

2. Bob sends $(X_{\text{Bob}} - A_{\text{Bob}})$ and $(Y_{\text{Bob}} - B_{\text{Bob}})$ to Alice.

3. Alice chooses a random $T' \in \mathbb{Z}^{n_1 \times n_3}_q$, computes $W = A_{\text{Alice}}(Y_{\text{Bob}} - B_{\text{Bob}}) + (X_{\text{Bob}} - A_{\text{Bob}})B_{\text{Alice}} + X_{\text{Alice}}Y_{\text{Alice}} - T'$ and sends $W, (X_{\text{Alice}} - A_{\text{Alice}})$ and $(Y_{\text{Alice}} - B_{\text{Alice}})$ to Bob. Alice outputs $T + T'$.

4. Bob outputs $U = (X_{\text{Alice}} - A_{\text{Alice}})Y_{\text{Bob}} + X_{\text{Bob}}(Y_{\text{Alice}} - B_{\text{Alice}}) + X_{\text{Bob}}Y_{\text{Bob}} + W + C$.

Figure 3.2: Protocol for secure distributed matrix multiplication in regression training.

Secure Distributed Matrix Multiplication Protocol. Let’s first take a look at a direct extension of Beaver’s protocol for secure multiplication in the commodity based model [185] for matrices. Alice and Bob hold shares of matrices $X \in \mathbb{Z}^{n_1 \times n_2}_q$ and $Y \in \mathbb{Z}^{n_2 \times n_3}_q$ to be multiplied and the goal is to obtain shares of the multiplication result $X \cdot Y \in \mathbb{Z}^{n_1 \times n_3}_q$ in such a way that the result remains hidden from both of them individually. Let $X_{\text{Alice}} \in \mathbb{Z}^{n_1 \times n_2}_q$ and $Y_{\text{Alice}} \in \mathbb{Z}^{n_2 \times n_3}_q$ be Alice’s shares of the inputs and $X_{\text{Bob}} \in \mathbb{Z}^{n_1 \times n_2}_q$ and $Y_{\text{Bob}} \in \mathbb{Z}^{n_2 \times n_3}_q$ be Bob’s shares. Note that $XY = (X_{\text{Alice}} + X_{\text{Bob}})(Y_{\text{Alice}} + Y_{\text{Bob}}) = X_{\text{Alice}}Y_{\text{Alice}} + X_{\text{Alice}}Y_{\text{Bob}} + X_{\text{Bob}}Y_{\text{Alice}} + X_{\text{Bob}}Y_{\text{Bob}}$. The terms $X_{\text{Alice}}Y_{\text{Alice}}$ and $X_{\text{Bob}}Y_{\text{Bob}}$ can be computed locally, but the computation of the terms $X_{\text{Alice}}Y_{\text{Bob}}$ and $X_{\text{Bob}}Y_{\text{Alice}}$ is more complex. Beaver’s protocol solves the problem of computing the last two terms by having the trusted initializer distribute random values $A_{\text{Alice}}, A_{\text{Bob}}, B_{\text{Alice}}, B_{\text{Bob}}$ to the parties and also random shares of the value $A_{\text{Alice}}B_{\text{Bob}} + A_{\text{Bob}}B_{\text{Alice}}$. Then the parties only need to derandomize this pre-distributed instance to the actual input values. The protocol $\pi_{\text{DMM}}$ is parametrized by the size $q$ of the field and by the dimensions $n_1, n_2, n_3$ of the matrices to be multiplied and works as outlined in Figure 3.2.
This protocol securely implements the ideal distributed matrix multiplication functionality \( F_{DMM} \) that works as follows: \( F_{DMM} \) is parametrized by the size \( q \) of the field and the dimensions \( n_1, n_2, n_3 \) of the matrices to be multiplied. Given Alice’s input shares \( X_{Alice} \in \mathbb{Z}_{q}^{n_1 \times n_2} \) and \( Y_{Alice} \in \mathbb{Z}_{q}^{n_2 \times n_3} \), and Bob’s input shares \( X_{Bob} \in \mathbb{Z}_{q}^{n_1 \times n_2} \) and \( Y_{Bob} \in \mathbb{Z}_{q}^{n_2 \times n_3} \), it computes \( V = (X_{Alice} + X_{Bob})(Y_{Alice} + Y_{Bob}) \), chooses a random matrix \( R \in \mathbb{Z}_{q}^{n_1 \times n_3} \) and sends \( R \) to Alice and \( V - R \) to Bob.

**Theorem 2.** The distributed matrix multiplication protocol \( \pi_{DMM} \) is correct and securely implements the distributed matrix multiplication functionality \( F_{DMM} \) against honest but curious adversaries in the commodity based model.

The correctness of the protocol can be trivially verified by inspecting the value of \( T + T' + U \). The security of this protocol lies in the fact that all values exchanged between the parties are blinded by a value which is completely random in the underlying field from the point of view of the message receiver. This protocol can in fact be proved secure even against malicious parties and in the stronger Universal Composability (UC) framework [129]. The idea is that the simulator simulates an instance of the adversary and an instance of the protocol execution with the adversary, allowing the adversary to communicate with the environment. The leverage used by the simulator is the fact that, in the ideal execution, he is the one simulating the trusted initializer. By simulating the TI, he is able, at the same time, to generate a protocol transcript for the adversary that is indistinguishable from the real protocol execution and also to extract the input of the corrupted parties in order to forward them to the ideal functionality.

**Dealing with Real Numbers.** The security proof of the (matrix) multiplication protocol essentially relies on the fact that the blinding factors are uniformly random in \( \mathbb{Z}_{q} \). If one tries to design similar protocols working directly with integers or real numbers there would be a problem, since it is not possible to sample uniformly in \( \mathbb{Z} \) or \( \mathbb{R} \). Similarly, in protocols that use homomorphic encryption as building blocks, the encryption is normally done for messages which are members of a finite group. But in secure protocols for functionalities
such as linear regression, one needs to deal with inputs which are real numbers. Thus it is necessary to develop a way to approximate the computations on real numbers by using building blocks which work on fields $\mathbb{Z}_q$.

We adapt the method of Catrina and Saxena [186] with a fixed-point representation. Let $k$, $e$ and $f$ be integers such that $k > 0$, $f \geq 0$ and $e = k - f \geq 0$. Let $\mathbb{Z}_{(k)}$ denote the set 
\[ \{ x \in \mathbb{Z} : -2^{k-1} + 1 \leq x \leq 2^{k-1} - 1 \} \].

The fixed-point data type with $k$ bits, resolution $2^{-f}$, and range $2^{e}$ is the set $\mathbb{Q}_{(k,f)} = \{ \tilde{x} \in \mathbb{Q} : \tilde{x} = \hat{x}2^{-f}, \hat{x} \in \mathbb{Z}_{(k)} \}$. The signed integers in $\mathbb{Z}_{(k)}$ are then encoded in the field $\mathbb{Z}_q$ (with $q > 2^k$) using the function

\[ g: \mathbb{Z}_{(k)} \rightarrow \mathbb{Z}_q, g(\hat{x}) = \hat{x} \mod q. \]

In secure computation protocols using secret sharing techniques, the values in $\mathbb{Z}_q$ are actually shared between the two parties. Using this encoding, we have that $\hat{x} + \hat{y} = g^{-1}(g(\hat{x}) + g(\hat{y}))$, where the second addition is in $\mathbb{Z}_q$, i.e., we can compute the addition for signed integers in $\mathbb{Z}_{(k)}$ by using the arithmetic in $\mathbb{Z}_q$. The same holds for subtraction and multiplication.

For the fixed-point data type we can do additions using the fact that $\tilde{w} = \tilde{x} + \tilde{y} = (\hat{x} + \hat{y})2^{-f}$, so we can trivially obtain the representation of $\tilde{w}$ with resolution $2^{-f}$ by computing $\tilde{w} = \hat{x} + \hat{y}$, i.e., we can do the addition of the fixed-point type by using the addition in $\mathbb{Z}_{(k)}$, which itself can be done by performing the addition in $\mathbb{Z}_q$. The same holds for subtraction. But for multiplication we have that $\tilde{w} = \tilde{x}\tilde{y} = \hat{x}\hat{y}2^{-2f}$, and therefore if we perform the multiplication in $\mathbb{Z}_q$, we will obtain (if no overflow occurred) the representation of $\tilde{w}$ with resolution $2^{-2f}$. Such representation uses $\mathbb{Z}_{(k+f)}$ instead of the original $\mathbb{Z}_{(k)}$. In order to have the size of the signed integers representation be independent from the amount of multiplication operations performed with the fixed-point data, we need to scale the resolution of $\tilde{w}$ down to $2^{-f}$. For that purpose we use a slightly modified version of the truncation protocol of Catrina and Saxena [186].
Secure Truncation Protocol $\pi_{\text{Trunc}}$ for Linear Regression Training

1. At the setup, the trusted initializer chooses uniformly random $r' \in \{0, 1\}^f$ and $r'' \in \{0, 1\}^{\lambda+k}$ and computes $r = r''2^f + r'$. He also chooses uniformly random $r'_{\text{Bob}}, r_{\text{Bob}} \in \mathbb{Z}_q$ and then sets $r'_{\text{Alice}} = r' - r'_{\text{Bob}}$ and $r_{\text{Alice}} = r - r_{\text{Bob}}$. He sends $r'_{\text{Alice}}, r_{\text{Alice}}$ to Alice and $r'_{\text{Bob}}, r_{\text{Bob}}$ to Bob.

2. Bob sends $z_{\text{Bob}} = (w_{\text{Bob}} + r_{\text{Bob}})$ to Alice and outputs $(w_{\text{Bob}} + r'_{\text{Bob}})^{-1}$.

3. Alice computes $c = z_{\text{Bob}} + w_{\text{Alice}} + r_{\text{Alice}} + 2^{k+f-1}$ and $c' = c \mod 2^f$. Then she outputs $(w_{\text{Alice}} + r'_{\text{Alice}} - c')^{-1}$.

Figure 3.3: Protocol for secure truncation in regression training.

The central idea of this truncation protocol is to reveal the number $w$ to be truncated to one of the parties, but blinded by a factor $r$ which is from a domain exponentially bigger than the domain of the value $w$ and thus statistically hides $w$. The value $r$ is generated so that the parties have shares of both $r$ and $r'$ (which represents the $f$ least significant bits of $r$). Then Bob can reveal $w + r$ to Alice and they can compute shares of the truncated value by using local computations. In more detail, let $\lambda$ be a security parameter and let the field $\mathbb{Z}_q$ in which the signed integers are encoded be such that $q > 2^{k+f+\lambda+1}$. Note that the multiplicative inverse of $2^f$ in $\mathbb{Z}_q$ is $((q+1)/2)^f$ and let $F^{-1}$ denote it. The parties have, as inputs, shares $w_{\text{Alice}}, w_{\text{Bob}} \in \mathbb{Z}_q$ such that $w = (w_{\text{Alice}} + w_{\text{Bob}}) \in \{0, 1, \ldots, 2^{k+f-1} - 1\} \cup \{q - 2^{k+f-1} + 1, \ldots, q - 1\}$. The protocol $\pi_{\text{Trunc}}$ for truncating the output works outlined in Figure 3.3.

This protocol securely implements the functionality $F_{\text{Trunc}}$ that captures the approximate truncation without leakage. $F_{\text{Trunc}}$ is parametrized by the size $q$ of the field and reduces the resolution by $2^{-f}$. Given Alice and Bob’s shares of the input, $w_{\text{Alice}}, w_{\text{Bob}} \in \mathbb{Z}_q$, and a blinding factor $r'_{\text{Bob}} \in \mathbb{Z}_q$ from Bob, it computes $\hat{w} = g^{-1}(w_{\text{Alice}} + w_{\text{Bob}} \mod q)$ and samples $u$ such that $\Pr[u = 1] = (\hat{w} \mod 2^f)/2^f$. Then it computes $v = (w_{\text{Alice}} - (w_{\text{Alice}} + w_{\text{Bob}} \mod 2^f) - r'_{\text{Bob}})^{-1} + u$ and sends it to Alice (Bob’s output is $(w_{\text{Bob}} + r'_{\text{Bob}})^{-1}$ and can be locally computed).
Theorem 3. The truncation protocol $\pi_{\text{Trunc}}$ privately computes the approximate truncation functionality $F_{\text{Trunc}}$.

Proof. Correctness: Let $\hat{w} = g^{-1}(w_{\text{Alice}} + w_{\text{Bob}} \mod q)$. We have that the value $\hat{w} \in \{-2^{k+f-1} + 1, -2^{k+f-1} + 2, \ldots, 2^{k+f-1} - 1\}$. Let $b = \hat{w} + 2^{k+f-1}$ and let $b' = b \mod 2^f$.

We have that $b \in \{1, \ldots, 2^{k+f} - 1\}$ and since $k > 0$ also that

$$b' = b \mod 2^f = \hat{w} + 2^{k+f-1} \mod 2^f = \hat{w} \mod 2^f.$$ 

Since $r < 2^{k+f+\lambda}$ and $q > 2^{k+f+\lambda+1}$ we have that $c = b + r$ and thus

$$c' = (b' + r') \mod 2^f = b' + r' - u2^f$$

where $u \in \{0, 1\}$ and $\Pr[u = 1] = \Pr[r' \geq 2^f - b'] = (\hat{w} \mod 2^f)/2^f$ with the probability over the choices of random $r'$. Hence

$$c' - r'_{\text{Alice}} - r'_{\text{Bob}} = g(\hat{w} \mod 2^f - u2^f),$$

$$w_{\text{Alice}} + w_{\text{Bob}} + r'_{\text{Alice}} + r'_{\text{Bob}} - c' = g(\hat{w} - (\hat{w} \mod 2^f) + u2^f)$$

$$= g\left(\left\lfloor \frac{\hat{w}}{2^f} \right\rfloor 2^f + u2^f\right),$$

$$(w_{\text{Alice}} + w_{\text{Bob}} + r'_{\text{Alice}} + r'_{\text{Bob}} - c')^{F^{-1}} = g\left(\left\lfloor \frac{\hat{w}}{2^f} \right\rfloor + u\right),$$

and so the shares output by the parties $(w_{\text{Alice}} + r'_{\text{Alice}} - c')^{F^{-1}}$ and $(w_{\text{Bob}} + r'_{\text{Bob}})^{F^{-1}}$ are correct.

Security: The only message exchanged is $z_{\text{Bob}} = (w_{\text{Bob}} + r_{\text{Bob}})$ that reveals $c = b + r$ to Alice, but since $r$ is a uniformly random $(k + f + \lambda)$-bits integer and $b$ is a $(k + f)$-bits integer, we have that the statistical distance between $c$ and $r$ is at most $2^{-\lambda}$, i.e., $c$ is statistically indistinguishable from a uniformly random value. \qed
Theorem 4. $\mathcal{F}_{\text{Trunc}}$ is an 1-approximation\footnote{in the representation, $2^{-f}$ in the fixed-point data type} and is functionally private with respect to an exact truncation functionality that computes the truncation using the floor function.

Proof. The only difference between the two functionalities is that in the approximate truncation an error factor $u$ is present in the shared output. But note that $u \in \{0, 1\}$ and $\Pr[u = 1] = (\hat{w} \mod 2^f)/2^f$, but $u$ is independent from the specific shares $w_{\text{Alice}}, w_{\text{Bob}}$ used to encode $g(\hat{w})$. Thus the protocol rounds $\hat{w}/2^f$ to the nearest integer with probability $1 - \alpha$, where $\alpha$ is the distance between $\hat{w}/2^f$ and the nearest integer. \qed

We should mention that in the case of matrix multiplication the truncations only have to be performed in the elements of the resulting matrix instead of once per element multiplication, which would be less efficient and also increase the error due to truncation.

Computing the Inverse of a Covariance Matrix. In order to be able to compute the linear regression model from a design matrix and the response vector we need to compute the inverse of the covariance matrix. Let $A$ be the covariance matrix. In order to compute $A^{-1}$ we use a generalization for matrices of the Newton-Raphson division method.

The algorithms for division of fixed-point numbers are divided in two main classes: digit recurrence (subtractive division) and functional iteration (multiplicative division). The Newton-Raphson division method is from the functional iteration class, which is more amenable to secure implementation and converges faster. Additionally this method is self correcting, i.e., truncation errors in one iteration decrease quadratically in the next iterations. The inverse of a number $a$ is computed by defining the function $h(x) = x^{-1} - a$ and then applying the Newton-Raphson method for finding successively better approximations to the roots of $h(x)$. The iterations follow the form:

$$x_{s+1} = x_s(2 - ax_s).$$

This algorithm can be generalized for computing the inverse of the matrix $A$. A numerical
stable iteration for computing $A^{-1}$ is [183, 187]:

$$c = \text{trace}(A)$$

$$X_0 = c^{-1}I$$

$$X_{s+1} = X_s(2 - AX_s)$$

where $I$ is the identity matrix with the same dimensions as $A$. Note that $A$ is a covariance matrix and thus it is positive semi-definite and the trace of $A$ dominates the largest eigenvalue of $A$. It is convenient to use $c = \text{trace}(A)$ because the trace of $A$ can be computed locally by parties that have shares of $A$. To compute $c^{-1}$ the Newton-Raphson is also used with $x_0$ set to an arbitrarily small value, as the convergence happens if the magnitude of the initial value is smaller than that of $c^{-1}$.

Note that, in our case, we use this method to securely compute the inverse of the covariance matrix, i.e, each party has a share of the covariance matrix as input and should receive, as output, random shares of its inverse, but no additional information should be learned by the parties. Hence we cannot perform a test after each iteration in order to check if the values already converged with resolution $2^{-f}$ (and thus stop the iterations at the optimal point) because this would leak information about the input based on how many iterations were performed. We have to use an upper bound $\ell$ on the number of iterations such that all possible covariance matrices converge with resolution $2^{-f}$ in $\ell$ iterations. A very conservative upper bound is $\ell = 2k$ [183]; further studies will be done to try to determine a tighter upper bound.

The protocol to securely compute the inverse of a covariance matrix is parametrized by the size $q$ of the field. Let $A \in \mathbb{Z}_q^{n \times n}$ be the encoding in $\mathbb{Z}_q$ of a covariance matrix where the elements are fixed-point numbers. Alice has input $A_{\text{Alice}} \in \mathbb{Z}_q^{n \times n}$ and Bob has input $A_{\text{Bob}} \in \mathbb{Z}_q^{n \times n}$ such that $A_{\text{Alice}} + A_{\text{Bob}} = A$. Then the protocol proceeds as outlined in Figure 3.4.
Secure Inverse of a Covariance Matrix for Linear Regression Training

1. Alice and Bob locally compute shares of \( c = \text{trace}(A) \), i.e., \( c_{\text{Alice}} = \sum_{i=1}^{n} A_{\text{Alice}}[i,i] \) and \( c_{\text{Bob}} = \sum_{i=1}^{n} A_{\text{Bob}}[i,i] \)

2. Alice and Bob use the Newton-Raphson division method to compute shares \( c^{-1}_{\text{Alice}} \) and \( c^{-1}_{\text{Bob}} \) of \( c^{-1} \) with resolution \( 2^{-f} \). The subtractions can be performed locally and the multiplications using the distributed (matrix) multiplication functionality \( F_{\text{DMM}} \) followed by the approximate truncation functionality \( F_{\text{Trunc}} \).

3. Alice and Bob use the generalized Newton-Raphson method to obtain shares \( A^{-1}_{\text{Alice}} \) and \( A^{-1}_{\text{Bob}} \) of \( A^{-1} \) with resolution \( 2^{-f} \) for the elements. The subtractions can be performed locally and the multiplications using the distributed matrix functionality \( F_{\text{DMM}} \) followed by the approximate truncation functionality \( F_{\text{Trunc}} \).

Figure 3.4: Protocol for securely computing the inverse of a covariance matrix.

We emphasize that the truncation used has some intrinsic error, however the self-correcting properties of the Newton-Raphson method compensate for that.

**Computing the Linear Regression Coefficients**  
We consider the setting in which there is a design matrix \( \tilde{X} \) and a response vector \( \tilde{y} \). We are interested in analyzing the statistical regression model

\[
\tilde{y} = \tilde{X} \tilde{\beta} + \epsilon,
\]

and therefore want to compute the estimated regression coefficient vector

\[
\tilde{\beta} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \tilde{y}
\]

The design matrix is a \( n \times m \) matrix where the elements are of the fixed-point data type with precision \( 2^{-f} \) and range \( 2^{k-f} \) (i.e., \( \tilde{X} \in Q_{(k,f)}^{n \times m} \)) and the response vector \( \tilde{y} \in Q_{(k,f)}^{n} \). Let \( \hat{X} \) be the element-wise representation of \( \tilde{X} \) as signed integers and let \( X = \hat{X} \mod q \) be the element-wise encoding of \( \hat{X} \) as elements of the field \( \mathbb{Z}_q \). Define \( \hat{y} \) and \( y \) in the same
way from \( \tilde{y} \).

It is assumed that the parties Alice and Bob hold shares of \( X \) and \( y \). Alice and Bob can then use the protocols for matrix multiplication, truncation and covariance matrix inversion that were described in the previous sections in order to compute shares of

\[
\beta = (X^T X)^{-1} X^T y
\]

Then they only need to reveal their final shares and convert the result back to the fixed-point data type in order to get \( \bar{\beta} \). Our approach is outlined in detail in Algorithm 2.

**Algorithm 2** Online Phase for Secure Linear Regression Training

1. The players map their fixed precision real valued inputs to elements of a finite field and create the shares of \( X \).
2. The players compute \( X^T X \) by using the matrix multiplication protocol \( \pi_{DMM} \). Once the multiplication is finished they ran the truncation protocol \( \pi_{Trunc} \) for each element of the matrix \( X^T X \).
3. Alice and Bob compute the inverse of \( (X^T X) \) by running the protocol for computing the inverse of a covariance matrix. Within the covariance matrix inversion protocol there are several calls to the matrix multiplication and truncation protocols.
4. Alice and Bob run the matrix multiplication and the truncation protocols twice to obtain \( (X^T X)^{-1} X^T \) and finally \( (X^T X)^{-1} X^T y \).
5. The players exchange their shares of the estimated regression coefficient vector and reconstruct it.
6. The coefficients \( \beta \) obtained by the players are mapped back from finite field elements to real values with finite precision.

The security of the composed protocol follows from the composition theorem (Theorem 1) using the facts that \( \pi_{DMM} \) securely implements the distributed matrix multiplication functionality \( F_{DMM} \) and \( \pi_{Trunc} \) privately computes the approximate truncation functionality \( F_{Trunc} \). It is assumed that a big enough \( k \) is used so that no overflow occurs and hence the correctness of the protocol follows. The final protocol implements the linear regression functionality \( F_{Reg} \) that upon getting the shares \( X_{Alice} \) and \( X_{Bob} \) of the design matrix \( X \) and \( y_{Alice} \) and \( y_{Bob} \) of the response vector \( y \), compute \( \beta = (X^T X)^{-1} X^T y \) and output \( \beta \) to Alice and Bob.
**Substituting the Trusted Initializer** If a trusted initializer is not desired, it is possible to obtain a solution where the parties, during the setup phase, compute the correlated data themselves. The idea is to use the homomorphic properties of the Paillier’s encryption scheme [109]. For two large prime numbers \( p \) and \( q \), the secret key of Paillier’s cryptosystem is \( \text{sk} = (p, q) \). The corresponding public key is \( \text{pk} = N = pq \) and the encryption of a message \( x \in \mathbb{Z}_N \) is done by picking a random \( r \in \mathbb{Z}_N^* \) and computing \( \text{Enc}(\text{pk}, x) = (N + 1)x r^N \mod N^2 \). The following homomorphic properties of Paillier’s encryption scheme are used:

\[
\text{Enc}(\text{pk}, x) \cdot \text{Enc}(\text{pk}, y) = \text{Enc}(\text{pk}, x + y \mod N)
\]

\[
\text{Enc}(\text{pk}, x^y) = \text{Enc}(\text{pk}, xy \mod N).
\]

Given two vectors \( x = (x_1, \ldots, x_n) \) and \( y = (y_1, \ldots, y_n) \) where the second is given in clear and the first is encrypted element-wise (i.e., \( \text{Enc}(\text{pk}, x_i) \) are revealed), one can compute a ciphertext corresponding to the inner product:

\[
\text{Enc}(\text{pk}, \langle x, y \rangle \mod N) = \prod_{i=1}^{n} \text{Enc}(\text{pk}, x_i)^{y_i}.
\]

The idea for computing the necessary correlated data for the distributed matrix multiplication protocol is to use the above fact in order to compute the non-local multiplication terms. Bob has a pair of public/secret keys for Paillier’s encryption scheme and sends to Alice the element-wise encryption under his own public key of the elements of the column/row that needs to get multiplied. Alice, having the plaintext corresponding to her own values on the appropriate column/row, can compute an encryption of the inner product under Bob’s public key. She then adds a random blinding factor and sends the ciphertext to Bob, who can decrypt it, thus yielding distributed shares of the inner product between Alice and Bob.

The protocol is parametrized by the dimensions \( n_1, n_2, n_3 \) of the matrices to be multi-
Securely Substituting the Trusted Initializer for Linear Regression Training

1. Alice chooses uniformly random \( A_{\text{Alice}} \in \mathbb{Z}_N^{n_1 \times n_2} \), \( B_{\text{Alice}} \in \mathbb{Z}_N^{n_2 \times n_3} \) and \( T \in \mathbb{Z}_N^{n_1 \times n_3} \).
2. Bob chooses uniformly random \( A_{\text{Bob}} \in \mathbb{Z}_N^{n_1 \times n_2} \) and \( B_{\text{Bob}} \in \mathbb{Z}_N^{n_2 \times n_3} \), element-wise encrypts them under his own public key and send the ciphertexts to Alice.
3. For \( i = 1, \ldots, n_1 \), \( j = 1, \ldots, n_3 \), Alice computes the ciphertext
   \[
   \tilde{c}[i, j] = \text{Enc}(pk, t[i, j]) \cdot \prod_{k=1}^{n_2} \left( \text{Enc}(pk, b_{\text{Bob}}[k, j])^a_{\text{Alice}}[i, k] \right) \cdot \text{Enc}(pk, a_{\text{Bob}}[i, k])^{b_{\text{Alice}}}[k, j]
   \]
   and sends them to Bob. Alice outputs \( A_{\text{Alice}}, B_{\text{Alice}} \) and \( T \).
4. Bob decrypts the ciphertexts in order to get the matrix \( C = (A_{\text{Alice}}B_{\text{Bob}} + A_{\text{Bob}}B_{\text{Alice}} + T) \). Bob outputs \( A_{\text{Bob}}, B_{\text{Bob}} \) and \( C \).

The security of this protocol follows trivially from the IND-CPA security of Paillier’s encryption scheme [109].

Note that the values \( r \) and \( r' \) that are distributed by the trusted initializer for performing the truncation protocol can be trivially computed by the parties themselves using distributed multiplications.

Securing Against Malicious Adversaries

Thus far we have exclusively considered the honest-but-curious, also known as semi-honest, adversary. Recall from Section 3.1.4 that honest-but-curious adversaries follow the protocol instructions but try to learn additional information about other parties’ inputs. There may be settings however when a stronger
adversary, a malicious adversary, that may deviate from the protocol should be considered.

One possibility for obtaining security against malicious adversaries is to use the compute with MACs approach [188, 189, 190] to protect the shared values from being manipulated by a malicious adversary. In this technique there is an unconditionally secure message authentication code (MAC) associated to each shared secret value. At the beginning of the protocol the parties commit to their inputs by opening the difference between their inputs and some random shared value (that has an associated MAC). Whenever an operation is performed over some shared values, the MAC for the output value is also computed (using the MACs of the input values). This approach prevents the malicious adversary from successfully deviating from the specified protocol instructions (if he deviates, the honest parties will notice it during the checking of the MACs). The only problem for applying this technique is that the truncation protocol used before is probabilistic and so it would not allow propagation of the MACs, but this can be solved by using a deterministic truncation procedure [191] (at the cost of having to perform one execution of a comparison protocol per truncation). See [188, 189, 190] for further details about the compute with MACs technique.

Evaluating the Secure Linear Regression Training Algorithm

We assessed our secure linear regression algorithm by implementing it and analyzing the results using ten real datasets. We chose a variety of different datasets based on the number of features and the number of instances. We used C++ as our programming language which we augmented with the BOOST libraries for functionality such as lexical_cast for type casting and asio for work with sockets. We also made use of the GMP and NTL libraries within C++ to implement our protocols. We built our system on top of a Microsoft Azure G4 series machine with Intel Xeon processor E5 v3 family, 224GB RAM size, 3072GB of disk size and 16 cores. Finally, we chose Ubuntu 12.04 as our operating system. We have merged the matrix multiplication and truncation protocols within one protocol for
implementation purposes.

All our datasets are contained within the UCI repository\(^3\), with the exception of the State Inpatient Database (WA) which is provided by HCUP\(^4\). The UCI repository includes 48 regression task datasets from which we chose 9. Our datasets range in size from 395 instances to over 4 million and from 7 attributes to 367.

The Gas Sensor Array Under Dynamic Gas Mixtures dataset represents data from 16 chemical sensors exposed to ethylene and CO mixtures at various concentration levels in the air. We added together the concentration of ethylene and the concentration of CO to create one continuous response variable of gas concentration and removed the time feature from the dataset. We then designated the first 8 sensor readings to Alice and the second 8 to Bob. This left us with a total of 4,208,261 sets of 16 sensor readings to different total concentrations of ethylene and CO.

In the Communities and Crime dataset we used 122 attributes describing 1,993 communities and their law enforcement departments in 1990 to create this dataset. The goal with this dataset is to predict the number of violent crimes per capita for each community. All missing values present in the dataset (of which there were 36,850 distributed throughout 1,675 different communities and 22 different attributes) were replaced with 0s. These missing values were largely relevant to the communities’ police departments. We also removed 5 variables that were present in the original data but described by the UCI documentation as non-predictive, namely state, county, community, community name, and fold. The final 122 attributes were then divided in half between Alice and Bob.

The Auto MPG dataset contains attributes describing 398 automobiles in attempt to predict MPG (miles per gallon) for each. We removed the car name attribute which was present in the original data and were left with 7 predictive features. We then replaced the 6 missing horsepower values with 0s. In the end we designated the cylinders, displacement,

\(^3\)UC Irvine Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets.html]

\(^4\)http://www.ahrq.gov/research/data/hcup/
and horsepower features to Alice and the weight, acceleration, model year, and origin features to Bob.

In an attempt to predict the number of comments a blog post will receive in the upcoming 24 hours, the BlogFeedback dataset originally had 280 attributes. Since our complete dataset must be linearly independent, to enable the inversion of $X^TX$ required in our protocol, we removed 57 of these original attributes leaving us with 223 predictors describing 52,397 blog posts. An example of such a feature would be the binary indicator of whether or not a blog post was published on a Sunday. There are binary indicators of whether publication occurred on any of the other days of the week and therefore this feature, publication on a Sunday, is linearly dependent on the other six. Finally, the dataset was divided column wise, designating 111 attributes to Alice and the other 112 to Bob.

The Wine Quality dataset takes 11 attributes related to the white variant of Portuguese “Vinho Verde” wine which are used to predict the quality score of 4,897 wines. We designated the fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, and free sulfur dioxide features to Alice and the total sulfur dioxide, density, pH, sulphates, and alcohol features to Bob.

In the Bike Sharing dataset we took attributes describing a certain hour and day and attempted to predict the number of users for a bike sharing system. We removed the record index which was present in the original data as well as the count of casual users and the count of registered users and targeted the total rental bikes used ($\text{casual} + \text{registered}$) for our prediction. We were left with 13 predictors of 17,379 hour/day combinations which were used to model bike use. Alice received information on the dates, seasons, years, months, hours, and holidays while Bob was given information on weekdays, working days, weather situations, temperatures, feel temperatures, humidities, and wind speeds.

We used 30 attributes describing 395 students across two schools to create the Student Performance dataset. The goal with this dataset is to predict the final grade of each student in their math class. We removed two columns from the original dataset – one detailing
students’ performances in the first period and one detailing their performances in the second period. We identified the student’s final grade as our sole response variable. The final 30 attributes were then divided evenly between Alice and Bob.

In the YearPredictionMSD dataset, attributes describe audio features of 515,344 songs and are used to predict the release year of each song. We kept all 90 features that were present in the original data provided by the UCI repository. In allocating the data we gave Alice the first 45 features and the second 45 to Bob.

From the HCUP State Inpatient Database (WA) we extract attributes describing 25,180 beneficiaries who had at least one hospital admission within the state of Washington during the first nine months of the year between the years 2009 and 2012. The goal with this data is to predict the cost each beneficiary will incur in the final three months of the same year. We extracted demographic, medical, and previous cost information from the original data and replaced any missing values with a 0 value. We then designated the age, gender, race, number of chronic conditions, length of stay, and number of admits attributes to Alice. Bob was given a Boolean matrix of comorbidities as well as previous cost information.

Finally, in an attempt to predict the relative location of a CT slice on the axial axis of the human body, the original Relative Location of CT Slices on Axial Axis dataset had 384 attributes describing CT images. Since our complete dataset must be linearly independent, we removed 17 of these original attributes leaving us with 367 predictors describing 53,500 CT images in the Relative Location of CT Slices on Axial Axis dataset. We then divided this dataset column wise, designating 183 attributes to Alice and the other 184 to Bob.

The online phase is very fast and capable of handling millions of records within less than an hour, which is a huge improvement to the previous results. We only use addition and multiplication of matrices on our online phase which makes it simple and easy to manage.

In the case when a trusted initializer is not desired one can use our computationally secure protocol, at the cost of having a costier offline phase. However, because Alice and Bob only work over random inputs during the offline phase, the encryption, decryption and
mathematical operations are all embarrassingly parallelizable.

**Online Phase** We present in Table 3.3 the running times for the online phase of our protocol building a predictive linear regression model. Our online phase is very fast, computing a linear regression model for a matrix of over 4 million rows and 16 columns in under one hour. The regression coefficients computed with our secure protocol agree to the 5th decimal digit with regression coefficients computed without any security.

We briefly work out the theoretical computational complexity of computing

\[
\beta = (X^T X)^{-1} X^T y
\]

with our online protocol. If our dataset (which is denoted by \( X \) in this formula), has \( n \) features and \( m \) records, then the total runtime for computing the \( \beta \) values is \( O(mn^2) \) which means that the number of records in the dataset has only a linear effect on the run time of our implementation.

We used NTL for matrix multiplication with modular arithmetic. We also used GMP (the GNU Multi-Precision library) in conjunction with NTL to increase our performance. In the NTL library, the basic algorithm is used (with time complexity \( O(n^3) \) when all matrix dimensions are \( n \)).

Note that \( (X^T X) \) is a square matrix with both dimensions equal to \( n \), and for datasets in which the number of features is small relative to the number of records, computing \( (X^T X)^{-1} \) is very fast and negligible in respect to, for example, computing \( X^T X \). Our online phase is faster and independent from the semi-honest trusted party unlike similar implementations, such as Nikolaenko et al.’s implementation [184].

**Computationally Secure Offline Phase** In the pre-processing of the computationally secure offline phase of the matrix multiplication protocol \( \pi_{DMM} \), we use Paillier for encryption and decryption, but any additive homomorphic encryption scheme can be used. The down
Table 3.3: Actual time required (in seconds) for the online phase of our secure protocol training a linear regression predictive model.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Train Time: Data Shared in the Clear</th>
<th>Train Time: Using Proposed Secure Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Performance</td>
<td>395</td>
<td>30</td>
<td>0.3 sec</td>
<td>11.7 sec</td>
</tr>
<tr>
<td>Auto MPG</td>
<td>398</td>
<td>7</td>
<td>0.09 sec</td>
<td>1.2 sec</td>
</tr>
<tr>
<td>Communities and Crime</td>
<td>1,993</td>
<td>122</td>
<td>9 sec</td>
<td>147 sec</td>
</tr>
<tr>
<td>Wine Quality</td>
<td>4,897</td>
<td>11</td>
<td>0.9 sec</td>
<td>5.2 sec</td>
</tr>
<tr>
<td>Bike Sharing</td>
<td>17,379</td>
<td>13</td>
<td>3.7 sec</td>
<td>16.5 sec</td>
</tr>
<tr>
<td>State Inpatient Database (WA)</td>
<td>25,180</td>
<td>36</td>
<td>21 sec</td>
<td>93 sec</td>
</tr>
<tr>
<td>BlogFeedback</td>
<td>52,397</td>
<td>223</td>
<td>1,800 sec</td>
<td>9,000 sec</td>
</tr>
<tr>
<td>Relative Location of CT Slices on Axial Axis</td>
<td>53,500</td>
<td>367</td>
<td>6,000 sec</td>
<td>30,000 sec</td>
</tr>
<tr>
<td>YearPredictionMSD</td>
<td>515,344</td>
<td>90</td>
<td>3,800 sec</td>
<td>18,000 sec</td>
</tr>
<tr>
<td>Gas Sensor Array Under Dynamic Gas Mixtures</td>
<td>4,208,261</td>
<td>16</td>
<td>1,100 sec</td>
<td>4,500 sec</td>
</tr>
</tbody>
</table>

Side of these schemes is that their encryption and decryption times are computationally intensive and, if the given dataset is large, the pre-processing phase can take a long time. This issue can be tackled by noticing that Alice and Bob, during this phase, only work over random inputs and thus one can use heavy parallelization to speed-up the running time.

For a dataset with $m$ records and $n$ features, in order to get coefficients securely and correctly, we use $i = 50$ iterations in the computation of inversion. Overall, we need $5mn + (3i + 3)n^2 + n + 3i$ encryptions and $mn + (i + 1)n^2 + n + i$ decryptions. We also have two matrix multiplications and 3 matrix additions between encryption and decryption. Each encryption in an Azure VM takes about 0.005 seconds for each core. It is then easy to see that the encryption phase is the bottleneck of the pre-processing phase and easily parallelizable. Since we have $5mn$ number of encryptions, by multiplying this number to the runtime of a single encryption time divided by number of cores, a good estimate of the pre-processing phase is achievable.

The estimated running time for the offline phase is given in Table 3.4. These estimated results are a huge improvement when compared to the previous result [183] which took two days given a dataset with only about 50,000 records and are comparable to the total running time presented in [184] in the case of 256 cores.
Table 3.4: Estimated time required (in seconds) for the offline phase of our secure protocol training a linear regression predictive model.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Offline Time With 16 Cores</th>
<th>Offline Time With 64 Cores</th>
<th>Offline Time With 256 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Performance</td>
<td>395</td>
<td>30</td>
<td>20 sec</td>
<td>6 sec</td>
<td>2 sec</td>
</tr>
<tr>
<td>Auto MPG</td>
<td>398</td>
<td>7</td>
<td>4 sec</td>
<td>1 sec</td>
<td>0.3 sec</td>
</tr>
<tr>
<td>Communities and Crime</td>
<td>1,993</td>
<td>122</td>
<td>400 sec</td>
<td>100 sec</td>
<td>30 sec</td>
</tr>
<tr>
<td>Wine Quality</td>
<td>4,897</td>
<td>11</td>
<td>100 sec</td>
<td>30 sec</td>
<td>10 sec</td>
</tr>
<tr>
<td>Bike Sharing</td>
<td>17,379</td>
<td>13</td>
<td>350 sec</td>
<td>100 sec</td>
<td>30 sec</td>
</tr>
<tr>
<td>State Inpatient Database (WA)</td>
<td>25,180</td>
<td>36</td>
<td>1,500 sec</td>
<td>400 sec</td>
<td>100 sec</td>
</tr>
<tr>
<td>BlogFeedback</td>
<td>52,397</td>
<td>223</td>
<td>15,000 sec</td>
<td>4,000 sec</td>
<td>1,000 sec</td>
</tr>
<tr>
<td>Relative Location of CT Slices on Axial Axis</td>
<td>53,500</td>
<td>367</td>
<td>30,000 sec</td>
<td>10,000 sec</td>
<td>3,000 sec</td>
</tr>
<tr>
<td>YearPredictionMSD</td>
<td>515,344</td>
<td>90</td>
<td>70,000 sec</td>
<td>20,000 sec</td>
<td>6,000 sec</td>
</tr>
<tr>
<td>Gas Sensor Array Under Dynamic Gas Mixtures</td>
<td>4,208,261</td>
<td>16</td>
<td>100,000 sec</td>
<td>30,000 sec</td>
<td>10,000 sec</td>
</tr>
</tbody>
</table>

3.2.3 Privacy-Preserving Model Evaluation with SMC

Thus far we have considered applying formal privacy-preserving techniques, both differential privacy and secure multiparty computation, to model training. We now switch our focus to the other half of the ML model equation: evaluating model users’ data on altread trained models deployed via some MLaaS provider.

SMC Building Blocks for the Evaluation of Classification Models

Secure Distributed Matrix Multiplication Recall from Section 3.2.2 the protocol $\pi_{\text{DMM}}$ for performing distributed matrix multiplication. In Figure 3.7 we present an alternative $\pi_{\text{DMM}}$ using terminology consistent with our evaluation protocols for clarity.

The parties have as input $[X]_q$ and $[Y]_q$, for matrices $X \in \mathbb{Z}_q^{i \times j}$ and $Y \in \mathbb{Z}_q^{j \times k}$, and want to obtain shares of the product. The trusted initializer pre-distributes a random matrix multiplication triple to the parties, i.e., secret sharings $[U]_q$, $[V]_q$ and $[W]_q$ for $U$ and $V$ uniformly random in $\mathbb{Z}_q^{i \times j}$ and $\mathbb{Z}_q^{j \times k}$, respectively, and $W = UV$. The parties then derandomize the random matrix multiplication triple during the protocol execution in order to compute a secret sharing $[Z]_q$ corresponding to $Z = XY$ without leaking any information about the input values $X$ and $Y$ or the output value $Z$. Figure 3.6 describes the distributed matrix multiplication functionality $F_{\text{DMM}}$ that is considered and Figure 3.7
**Functionality** $\mathcal{F}_{\text{DMM}}$

$\mathcal{F}_{\text{DMM}}$ runs with parties $P_1, \ldots, P_n$ and is parametrized by the size $q$ of the ring and the dimensions $(i, j)$ and $(j, k)$ of the matrices.

**Input:** Upon receiving a message from a party with its shares of $[X]_q$ and $[Y]_q$, verify if the share of $X$ is in $\mathbb{Z}^{i \times j}_q$ and the share of $Y$ is in $\mathbb{Z}^{j \times k}_q$. If it is not, abort. Otherwise, record the shares, ignore any subsequent message from that party and inform the other parties about the receipt.

**Output:** Upon receipt of the shares from all parties, reconstruct $X$ and $Y$ from the shares, compute $Z = XY$ and create a secret sharing $[Z]_q$ to distribute to the parties: the corrupt parties fix their shares of the output to any constant values and the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

Figure 3.6: The distributed matrix multiplication functionality.

**Secure Distributed Matrix Multiplication Protocol** $\pi_{\text{DMM}}$

The protocol is parametrized by the size $q$ of the ring and the dimensions $(i, j)$ and $(j, k)$ of the matrices, and runs with the parties $P_1, \ldots, P_n$. The trusted initializer chooses uniformly random $U$ and $V$ in $\mathbb{Z}^{i \times j}_q$ and $\mathbb{Z}^{j \times k}_q$, respectively, computes $W = UV$ and pre-distributes secret sharings $[U]_q, [V]_q, [W]_q$ to the parties. The parties have inputs $[X]_q, [Y]_q$ and interact as follows:

1. Locally compute $[D]_q \leftarrow [X]_q - [U]_q$ and $[E]_q \leftarrow [Y]_q = [V]_q$, then open $D$ and $E$.
2. Locally compute $[Z]_q \leftarrow [W]_q + E[U]_q + D[V]_q + DE$.

Figure 3.7: The protocol for secure distributed matrix multiplication.
presents the protocol $\pi_{\text{DMM}}$ that implements such functionality\(^5\).

**Theorem 5.** The protocol $\pi_{\text{DMM}}$ is correct and securely implements the distributed matrix multiplication functionality $\mathcal{F}_{\text{DMM}}$ against honest-but-curious adversaries in the commodity based model.

**Proof.** **Correctness:** For verifying correctness, first notice that $Z = XY = (U + D)(V + E) = UV + UE + DV + DE = W + UE + DV + DE$ and therefore $[Z]_q \leftarrow [W]_q + E[U]_q + D[V]_q + DE$ obtains a secret sharing corresponding to $Z = XY$. The fact that the resulting shares are uniformly random with the constraint that $Z = XY$ follows trivially from the fact that the pre-distributed multiplication triple has this property.

**Security:** The simulation is very simple and proceeds as follows. The simulator $S$ runs internally a copy of the adversary $A$ and reproduces the real world protocol execution perfectly for $A$. For that, it simulates the protocol execution with dummy inputs for the uncorrupted parties. The leverage of the simulator is the fact that it can simulate the trusted initializer functionality $\mathcal{F}_{\text{DTI}}$ for $A$. Using this leverage, whenever a corrupted party announces its shares of $D$ and $E$ in the simulated protocol execution, $S$ can extract the respective shares of $X$ and $Y$ to give to the distributed matrix multiplication functionality $\mathcal{F}_{\text{DMM}}$. And whenever an honest party sends its shares to the functionality, $S$ simulates the announced messages for $A$ by sending random messages, which from $A$’s point of view are indistinguishable from the messages in the real protocol execution as the shares of $U$ and $V$ are uniformly random and unknown to $A$. Given its knowledge about $[U]_q, [V]_q, [W]_q, D$ and $E$ by the end of the simulated execution, $S$ knows, for each corrupted party, which value its share of the output is supposed to take, and therefore $S$ can fix these values in $\mathcal{F}_{\text{DMM}}$ so that the sum of the uncorrupted parties’ shares is compatible with the simulated execution. Therefore no environment $Z$ can distinguish the real and ideal worlds. \hfill $\square$

---

\(^5\)Notation: We denote by $\pi_{\text{DM}}$ the protocol for the special case of multiplication of single elements. The special case of inner-product computation will be denoted as $\pi_{\text{IP}}$. Henceforth $[Z]_q \leftarrow [X]_q[Y]_q$ will denote the secure distributed multiplication of secret shared values using the above protocol.
\textbf{Functionality} $\mathcal{F}_{\text{DC}}$

$\mathcal{F}_{\text{DC}}$ runs with parties $P_1, \ldots, P_n$ and is parametrized by the bit-length $\ell$ of the values being compared.

\textbf{Input:} Upon receiving a message from a party with its shares of $[x]_2$ and $[y]_2$ for all $i \in \{1, \ldots, \ell\}$, record the shares, ignore any subsequent messages from that party and inform the other parties about the receipt.

\textbf{Output:} Upon receipt of the inputs from all parties, reconstruct $x$ and $y$ from the bitwise shares. If $x \geq y$, then create and distribute to the parties the secret sharing $[1]_2$; otherwise the secret sharing $[0]_2$. Before the deliver of the output shares, the corrupt parties fix their shares of the output to any constant values. In both cases the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig3_8.png}
\caption{The distributed comparison functionality.}
\end{figure}

\textbf{Secure Distributed Comparison} For performing secure distributed bitwise comparison we use the protocol of Garay et al. \cite{Garay192} with secret sharings in the field $\mathbb{Z}_2$. That protocol has $\lceil \log \ell \rceil + 1$ rounds and uses $3\ell - \lfloor \log \ell \rfloor - 2$ multiplications. The protocol will be denoted by $\pi_{\text{DC}}$ and it securely implements the distributed comparison functionality $\mathcal{F}_{\text{DC}}$ that is described in Figure 3.8. For a more detailed description of the protocol please see the original paper of Garay et al. \cite{Garay192} or Section 4.3.3 of De Hoogh’s PhD thesis \cite{DeHoogh193}. While the original proof of security of this protocol does not consider UC-security, the UC-security of the protocol against honest-but-curious adversaries is trivial.

\textbf{Secure ArgMax} Suppose that the parties $P_1, \ldots, P_n$ have bitwise shares of a tuple of values $(v_1, \ldots, v_k)$ and want one of them, let’s say $P_1$, to learn all the arguments $m \in \{1, \ldots, k\}$ such that $v_m \geq v_j$ for all $j \in \{1, \ldots, k\}$, but no party should learn any $v_j$ or the relative order between the elements, i.e., the parties just want $P_1$ to learn

\[
m = \arg \max_{j \in \{1, \ldots, k\}} v_j.
\]
Functionality $\mathcal{F}_{\arg \max}$

$\mathcal{F}_{\arg \max}$ runs with parties $P_1, \ldots, P_n$ and is parametrized by the bit-length $\ell$ of the values being compared and the number $k$ of values being compared.

**Input:** Upon receiving a message from a party with its bitwise shares of $[v_{j,i}]_2$ for all $j \in \{1, \ldots, k\}$ and $i \in \{1, \ldots, \ell\}$, record the shares, ignore any subsequent messages from that party and inform the other parties about the receipt.

**Output:** Upon receipt of the inputs from all parties, reconstruct the values $v_j$ from the bitwise shares $v_{j,i}$, compute $m = \arg \max_{j \in \{1, \ldots, k\}} v_j$, and send $m$ to $P_1$.

Figure 3.9: The argmax functionality.

The argmax functionality $\mathcal{F}_{\arg \max}$ is described in Figure 3.9. Using the protocol for secure distributed comparison it is possible to give simple and practical solutions for securely computing this function. An idea, which optimizes the number of communication rounds, is having the parties comparing in parallel each ordered pair of vectors and then using the result of the comparisons to determine the argmax. Note that when considering all executions of the comparison protocol involving a specific value $v_j$ as the first argument, they will all return one if and only if the value is a maximum. The protocol $\pi_{\arg \max}$ is described in Figure 3.10.

**Theorem 6.** The argmax protocol $\pi_{\arg \max}$ is correct and securely implements the argmax functionality against honest-but-curious adversaries in the commodity-based model.

**Proof.** **Correctness:** The correctness follows trivially as for a maximum value, all comparison involving it as the first argument will return one, and so the product of the comparison results will also be one and the index will be added to the output. For all values which are not a maximum, at least one comparison will return zero, and so the product will be zero and the index will not be added.

**Security:** The first two steps only involve invocations of the distributed comparison $\pi_{\text{DC}}$
Secure Argmax Protocol $\pi_{\text{arg max}}$

Let $\ell$ be the bit length of the $k$ values to be compared. The trusted initializer pre-distributes all the correlated randomness necessary for the execution of the instances of the distributed multiplication and comparison protocols. The parties have as input bitwise shares $[v_{j,i}]_q$ for all $j \in \{1, \ldots, k\}$, $i \in \{1, \ldots, \ell\}$ and proceed as follows:

1. For all $j = 1, \ldots, k$ and $n \in \{1, \ldots, k\} \setminus j$, the parties compare in parallel $[v_{j,i}]_2$ and $[v_{n,i}]_2$ ($i = 1, \ldots, \ell$). Let $[w_{j,n}]_2$ denote the output obtained.

2. For all $j = 1, \ldots, k$, the parties computed in parallel

$$[w_j]_2 = \prod_{n \in \{1, \ldots, k\} \setminus j} [w_{j,n}]_2$$

3. The parties open $w_j$ for $P_1$. If $w_j = 1$, $P_1$ append $j$ to the value to be output in the end.

Figure 3.10: The secure $\text{arg max}$ protocol.

and multiplication $\pi_{\text{DM}}$ protocols, while the last step only opens one bit of information per index, indicating whether it corresponds to a maximum value or not; but this information is exactly the information contained in the output of the functionality $F_{\text{arg max}}$; hence the security of the protocol follows easily. Using the fact that $\pi_{\text{DC}}$ securely realizes $F_{\text{DC}}$ and $\pi_{\text{DM}}$ securely realizes $F_{\text{DMM}}$, the simulator $S$ runs internally a protocol execution for the adversary $A$ in which he simulates the ideal functionalities and uses dummy inputs for the uncorrupted parties. Using this leverage, it is trivial for $S$ to extract the inputs of the corrupted parties in order to give to $F_{\text{arg max}}$. If $P_1$ is corrupted, $S$ can then use the output it gets from $F_{\text{arg max}}$ to adjust the output of the simulated protocol by picking an uncorrupted party and changing its share of each $w_j$ appropriately before the opening. The real and ideal worlds are then indistinguishable to all environments $\mathcal{Z}$.
Functionality $F_{\text{decomp}}$

$F_{\text{decomp}}$ runs with parties $P_1, \ldots, P_n$ and is parametrized by the bit-length $\ell$ of the value $x$ being converted from additive sharings $[x]_q$ in $\mathbb{Z}_q$ to additive bitwise sharings $[x_i]_2$ in $\mathbb{Z}_2$ such that $x = x_\ell \cdots x_1$.

**Input:** Upon receiving a message from a party with its share of $[x]_q$, record the share, ignore any subsequent messages from that party and inform the other parties about the receipt.

**Output:** Upon receipt of the inputs from all parties, reconstruct the value $x = x_\ell \cdots x_1$ from the shares, and for $i \in \{1, \ldots, \ell\}$ distribute new sharings $[x_i]_2$ of the bit $x_i$. Before the output deliver, the corrupt parties fix their shares of the outputs to any constant values. The shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraints.

---

**Secure Bit-Decomposition** We next deal with the problem of converting from shares $[x]_q$ of a value $x$ in a large field $\mathbb{Z}_q$ to shares of $[x_i]_2$ in the field $\mathbb{Z}_2$, where $x_\ell \cdots x_1$ is the binary representation of $x$. The bit-decomposition functionality $F_{\text{decomp}}$ is described in Figure 3.11. The usefulness of such functionality comes from the fact that it allows to convert from a representation that allows the efficient execution of algebraic operations to a representation that allows the efficient execution of Boolean operations, such as a comparison. We present in Figure 3.12 a bit-decomposition protocol $\pi_{\text{decomp}}$ that is specialized for the two-party case with $q = 2^\ell$. Alice and Bob know shares $a$ and $b$, respectively, such that $x = a + b \mod 2^\ell$. The main observation is that the difference between the sum of $a = a_\ell \cdots a_1$ and $b = b_\ell \cdots b_1$ modulo $2^\ell$ and two bit strings that xor to the bit string $x_\ell \cdots x_1$ is exactly equal to the carry bits.\(^6\) Therefore we use a carry computation to obtain the bitwise secret sharings $[x_i]_2$ starting from $a_\ell \cdots a_1$ and $b_\ell \cdots b_1$.

\(^6\)The protocol is similar to the one of Laud and Randmets [194]
Secure Two-Party Bit-Decomposition Protocol $\pi_{\text{decomp}}$

Let $\ell$ be the bit length of the value $x$ to be reshared. All distributed multiplications are over $\mathbb{Z}_2$ and the required correlated randomness is pre-distributed by the trusted initializer. The parties, Alice and Bob, have as input $[x]_q$ for $q = 2^\ell$ and proceed as follows:

1. Let $a$ denote Alice’s share of $x$, which corresponds to the bit string $a_\ell \ldots a_1$. Similarly, let $b$ denote Bob’s share of $x$, which corresponds to the bit string $b_\ell \ldots b_1$. Define the secret sharings $[y_i]_2$ as the pair of shares $(a_i, b_i)$ for $y_i = a_i + b_i \mod 2$, $[a_i]_2$ as $(a_i, 0)$ and $[b_i]_2$ as $(0, b_i)$.

2. Compute $[c_1]_2 \leftarrow [a_1]_2 [b_1]_2$ and set $[x_1]_2 \leftarrow [y_1]_2$.

3. For $i = 2, \ldots, \ell$:
   (a) Compute $[d_i]_2 \leftarrow [a_i]_2 [b_i]_2 + 1$
   (b) $[e_i]_2 \leftarrow [y_i]_2 [c_{i-1}]_2 + 1$
   (c) $[c_i]_2 \leftarrow [e_i]_2 [d_i]_2 + 1$
   (d) $[x_i]_2 \leftarrow [y_i]_2 + [c_{i-1}]_2$

4. Output $[x_i]_2$ for $i \in \{1, \ldots, \ell\}$.

Figure 3.12: The secure two-party bit-decomposition protocol.
curely implements the bit-decomposition functionality $F_{\text{decomp}}$ for the special case of two players against honest-but-curious adversaries in the commodity-based model.

**Proof. Correctness:** The protocol implements the carry of a full adder logic $c_i = (a_i \land b_i) \lor ((a_i \oplus b_i) \land c_{i-1})$, which can be similarly expressed as $c_i = \neg((a_i \land b_i) \land \neg((a_i \oplus b_i) \land c_{i-1}))$ to obtain the carry bit string. By adding $c_{i-1}$ into $y_i$, we convert from bit strings that sum to $x$ modulo $2^\ell$ to bit strings that xor to $x$, thus obtaining the shares of $x_i$ modulo 2.

**Security:** The only non-local operations are the invocations of the distributed multiplication protocol $\pi_{\text{DM}}$, which securely realizes $F_{\text{DMM}}$. Therefore the security follows essentially from the security of that protocol. $S$ runs a copy of $A$ and simulates an execution of the protocol using dummy inputs for the uncorrupted party. Since $S$ is the one simulating the distributed multiplication functionality $F_{\text{DMM}}$, it can easily extract the corrupted party’s share of the input in order to give it to $F_{\text{decomp}}$ and also derive the corrupted party’s shares of the outputs in order to fix then in $F_{\text{decomp}}$. Consequently the real and ideal worlds are indistinguishable to any possible environment $Z$.

**Optimization:** The idea to optimize the number of rounds to logarithmic is to compute speculatively. In the first round the bit strings are divided in blocks of size 1 and the values of $i$ and $c_i$ are computed speculatively using both $c_{i-1} = 1$ and $c_{i-1} = 0$ for all but $i = 1$, for which we know that there is no carry in and so only one computation is needed. The second round divides the bit strings in blocks of size 2 and uses the information from the previous round to compute $x_{i+1}x_i$ and $c_{i+1}c_i$ speculatively using both $c_{i-1} = 1$ and $c_{i-1} = 0$ (except for the least significant block that only needs one computation). The third round proceeds analogously with blocks of size 4 by joining the blocks of size 2, and so on. After $\lceil \log \ell \rceil$ rounds one gets the desired bit strings $x_{\ell} \ldots x_1$ and $c_{\ell} \ldots c_1$. The first iteration uses $3\ell$ instances of the multiplication protocol and needs two rounds of communication as there are pairs of sequential multiplications, all other iterations only need one round of communication and use $2\ell$ multiplications each. Therefore in total the optimized protocol
Oblivious Input Selection

In our applications there are also circumstances in which Alice holds a vector of inputs $x = (x_1, \ldots, x_n)$ and Bob holds an index $k$, and they want to obtain bitwise secret sharings of $x_k$ for further uses in the protocol, but without revealing any information about the inputs or $k$. The oblivious input selection functionality $F_{OIS}$, which captures this task, is described in Figure 3.13. In Figure 3.14 a protocol $\pi_{OIS}$ realizing this functionality is presented. This idea was previously used by Toft [195, 196], where it was called “secret indexing”\(^7\).

**Theorem 8.** The oblivious input selection protocol $\pi_{OIS}$ is correct and securely implements the oblivious input selection functionality $F_{OIS}$ against honest-but-curious adversaries in the commodity-based model.

**Proof.** **Correctness:** Straightforward to verify.

---

\(^7\)Remark: As pointed out by an anonymous reviewer of [95], an alternative would be to run the equivalent of the first two steps in $\mathbb{Z}_{2\ell}$ instead of $\mathbb{Z}_2$ and then execute the bit decomposition protocol. Overall this would result in $n$ multiplications in $\mathbb{Z}_{2\ell}$ (plus the ones for the bit decomposition) instead of $n\ell$ multiplications in $\mathbb{Z}_2$, and the amount of data communicated in the two steps would be the same.
Oblivious Input Selection Protocol $\pi_{\text{OIS}}$

Let $\ell$ be the bit length of the inputs to be shared and $n$ the dimension of the input vector. The trusted initializer pre-distributes all the correlated randomness necessary for the execution of $\pi_{\text{DM}}$ over $\mathbb{Z}_2$. Alice has as input a vector of values, $x = (x_1, \ldots, x_n)$, and Bob has as input $k$, the index of the desired input value. They proceed as follows:

1. Define $y_k = 1$ and, for $j \in \{1, \ldots, n\} \setminus \{k\}$, $y_j = 0$. For $j \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, \ell\}$, let $x_{j,i}$ denote the $i$-th bit of $x_j$. Define $[y_j]_2$ as the pair of shares $(0, y_j)$ and $[x_{j,i}]_2$ as $(x_{j,i}, 0)$

2. Compute in parallel $[z_i]_2 \leftarrow \sum_{j=1}^{n} [y_j]_2 [x_{j,i}]_2$ for $i = 1, \ldots, \ell$.

3. Output $[z_i]_2$ for $i \in \{1, \ldots, \ell\}$.

Figure 3.14: The oblivious input selection protocol.

**Security:** Similarly to the previous proofs, $S$ uses the fact that the only messages exchanged are for performing the distributed multiplications and the leverage of being able to simulate $\mathcal{F}_{\text{DMM}}$ in order to simulate an execution of the protocol to $\mathcal{A}$ and at the same time being able to extract the inputs and the output shares of a corrupted party in order to forward to $\mathcal{F}_{\text{OIS}}$. By doing so, the real and ideal worlds are indistinguishable to $\mathcal{Z}$.

*Assembling the Building Blocks for Privacy-Preserving Classification*

We now present our privacy-preserving classifiers using the building blocks previously introduced. We use the same fixed-point representation as in Catrina and Saxena [186] to deal with real numbers. This representation maps fixed-point precision real numbers into integers. We assume a fixed precision for all of our inputs (and truncate any digit beyond that precision) and multiply them by a constant big enough so that the result is an integer for the whole range of inputs we work with. While in [186] a downscale (truncation) protocol is used after each multiplication in order to reduce multiplied numbers to the original precision, this step is not necessary in our implementation, since: (i) in our decision
Figure 3.15: Example of decision tree with 7 nodes and 2 classes.

**Functionality $\mathcal{F}_{DT}$**

$\mathcal{F}_{DT}$ is parametrized by the tree depth $d$, which is revealed to Alice.

**Input:** Upon receiving the feature vector $x$ from Alice or the decision tree model $D = (d, G, H, w)$ from Bob, store it, ignore any subsequent message from that party, and inform the other party about the receipt.

**Output:** Upon receipt of the inputs from both parties, evaluate the decision tree $D$ with the input $x$. Let $j$ be the reached leaf. Output $G(j)$ to Alice.

Figure 3.16: The decision tree functionality.

tree protocol there is no multiplication of fixed-point numbers; and (ii) in our hyperplane-based classifier the multiplication depth of the inner product is 1, so we can avoid rounding by just running the argmax protocol on the outputs of the innerproduct protocol.

**Secure Decision Trees** Here, Alice inputs $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and the classification algorithm will result in one of the $k$ possible classes $c_1, \ldots, c_k$. Bob holds the model $D = (d, G, H, w)$, where $d$ is the depth of the tree, $G$ maps the leaves to classes, $H$ maps internal nodes (always considered in level-order) to input features and $w$ is a vector of
Secure Decision Tree Protocol $\pi_{DT}$

Alice has as input a feature vector $x$ and Bob has a decision tree model $D = (d, G, H, w)$. Alice and Bob proceed as follows:

1. For $i = 1, \ldots, 2^d - 1$, Alice and Bob obtain bitwise secret sharings of $x_{H(i)}$ by using $\pi_{\text{OIS}}$ with inputs $x_1, \ldots, x_n$ from Alice and input $H(i)$ from Bob.

2. For $i = 1, \ldots, 2^d - 1$, Alice and Bob securely compare $x_{H(i)}$ and $w_i$. For the input $w_i$, Bob inputs its bit representation and Alice inputs zeros. Let $\langle z_i \rangle_2$ denote the result.

3. For $j = 0, \ldots, 2^d - 1$, let $j_d \ldots j_1$ be the binary representation of $j$ with $d$ bits and let $b_\alpha \ldots b_1$ for $\alpha = \lceil \log k \rceil$ be the binary representation of $G(j + 1) - 1$. For $r = 1, \ldots, \alpha$, initialize $\langle y_{j,r} \rangle_2$ with the shares $(0, b_r)$. Initialize $u = 1$ and $s = d$. While $s > 0$ do:
   
   (a) For $r = 1, \ldots, \alpha$, $\langle y_{j,r} \rangle_2 \leftarrow \langle y_{j,r} \rangle_2 (\langle z_u \rangle_2 + j_s)$.
   
   (b) Update $u \leftarrow 2u + j_s$ and $s \leftarrow s - 1$.

4. For all $r = 1, \ldots, \alpha$ compute $\langle \sigma_r \rangle_2 \leftarrow \sum_{j=0}^{2^d-1} \langle y_{j,r} \rangle_2$ and open $\sigma_r$ to Alice. Alice reconstructs $\sigma$ from the bit string $\sigma_\alpha \ldots \sigma_1$ and outputs $k^* = \sigma + 1$.

Figure 3.17: The protocol for secure evaluation of a decision tree.
thresholds. Each internal node of the tree structure tests the value of a particular feature against a corresponding threshold and branches according to the results. Each leaf node specifies a class. In all our secure protocols, we assume without loss of generality that we have a full binary tree. In case a decision tree is not full, one can always fill it with dummy nodes and obtain a full one. Let \( z_i \) be the Boolean variable denoting the result of comparing \( x_{H(i)} \) with \( w_i \). We recall the classification algorithm:

- Starting from the root node, for the current internal node \( v_i \), evaluate \( z_i \). If \( z_i = 1 \), take the left branch; otherwise, the right branch.
- The algorithm terminates when a leaf is reached. If the \( j \)-th leaf is reached, then the output is \( c_{G(j)} \).

Similar to Bost et al. \[153\], the classification can be expressed as a polynomial \( P_G : \{0, 1\}^{2^d-1} \rightarrow \{1, \ldots, k\} \) that depends on the mapping \( G \) from the leaves to the classes. On input \( z = (z_1, \ldots, z_{2^d-1}) \), \( P_G \) gives the classification result. This polynomial is a sum of terms such that each term corresponds to one possible path in the tree: the term corresponding to path taken by \( x \) in the tree evaluates to the classification result (i.e., the class associated to that leaf), while the remaining terms evaluate to zero. For example, for the tree portrayed in Figure 3.15, the polynomial \( P_G \) that represents the tree is:

\[
P_G(z_1, z_2, z_3) = z_1 z_2 c_1 + z_1 \bar{z}_2 c_2 + \bar{z}_1 z_3 c_1 + \bar{z}_1 \bar{z}_3 c_2\]

where \( \bar{x} \) denotes \( 1 - x \).

The idea of our secure protocol is that, for each internal node, Alice and Bob use the oblivious input selection protocol \( \pi_{OIS} \) to obtain bitwise secret sharings of the value \( x_{H(i)} \) that will be compared against the threshold \( w_i \) of this node. Note that, as Alice does not learn any information from the execution of \( \pi_{OIS} \), she does not know which feature will be used in the comparison at each internal node. Then the comparisons are performed using the secure distributed comparison protocol \( \pi_{DC} \) in order to obtain \( z \), which is then used to evaluate the polynomial \( P_G \) using the secure multiplication protocol \( \pi_{DM} \) and local addition of secret sharings. The only information leaked about the tree structure to Alice is its depth.
The decision tree functionality $F_{DT}$ is described in Figure 3.16 and a more detailed description of the protocol $\pi_{DT}$ realizing $F_{DT}$ is in Figure 3.17.

**Theorem 9.** The decision tree protocol $\pi_{DT}$ is correct and securely implements the decision tree functionality $F_{DT}$ against honest-but-curious adversaries in the commodity-based model.

**Proof.** **Correctness:** For each leaf $j \in \{1, \ldots, 2^d\}$, the secret sharings $[y_{j-1,r}]_2$ with $r = 1, \ldots, \lceil \log k \rceil$ obtained in step 3 correspond to a binary representation of the index of its associated class (offset by 1) if $j$ is the leaf that would be reached by using the model $D$ on input $x$; otherwise they correspond to zeros as at least one of the terms $[z_u]_2 + j_s$ in the multiplication would be zero. Thus in step 4, by summing all $[y_{j-1,r}]_2$ for $j \in \{1, \ldots, 2^d\}$, opening the results and adding 1, Alice obtains the result of the classification $k^*$.

**Security:** Alice learns the depth $d$ of the tree in order to allow the execution, but this is leaked by $F_{DT}$ as well. In the first three steps messages are only exchanged in order to execute the sub-protocols $\pi_{OIS}, \pi_{DC}$ and $\pi_{DM}$ respectively, which securely realize the functionalities $F_{OIS}, F_{DC}$ and $F_{DMM}$ respectively. Then the last step simply reveals the bit string encoding the class that was the result of the classification to Alice. The simulation strategy is similar to the one in the previous sections. The simulator $S$ internally runs a protocol execution for the adversary $A$ in which $S$ simulates $F_{OIS}, F_{DC}$ and $F_{DMM}$ and uses dummy inputs for the uncorrupted parties. Using this leverage $S$ can easily extract the inputs of the corrupted party, $x$ in case Alice is corrupted or $D = (d, G, H, w)$ in case Bob is corrupted, in order to forward to $F_{DT}$. In case Alice is corrupted, upon learning the correct output from $F_{DT}, S$ can adjust appropriately Bob’s shares of $\sigma_r$ in the simulated protocol in order to match the right result. The real and ideal worlds are thus indistinguishable to $Z$.

**Optimization:** All independent operations are run in parallel and the round complexity of step 3a can be reduced using techniques similar to the previous sections.
Secure Hyperplane-Based Classifiers  A privacy-preserving hyperplane-based classifier is easily achievable using our building blocks. One just needs to represent the model and features in $\mathbb{Z}_q$, compute each inner product between $w_i$ and $x$ by using $\pi_{ip}$, input the results into the bit-decomposition protocol $\pi_{decomp}$ and then into the argmax protocol $\pi_{arg\ max}$ to obtain the classification result

$$k^* = \arg \max_{i \in [k]} \langle w_i, x \rangle.$$  

In the specific case of SVM, the overall idea for obtaining a privacy-preserving classifier is as follows: Alice inputs her personal vector $x$ and Bob inputs his model vector $a$ to the secure distributed inner product protocol $\pi_{ip}$. After that, the result is run through the bit-decomposition protocol $\pi_{decomp}$. The resultant bitwise shares, together with $b$, are used in the comparison protocol $\pi_{DC}$ to determine the final result, which is then opened to Alice as her prediction. To privately score a logistic regression classifier with threshold 0.5 we can use exactly the same protocol as for support vector machines. The security of these compositions follows from the security of the sub-protocols and the fact that no values are ever opened before the final result; each party only sees shares, which appear completely random.

Removing the Trusted Initializer

Our evaluation protocols assume that pre-distributed data is made available to the players by a trusted initializer: random binary multiplication triples (binary Beaver triples) in the case of decision trees, random binary multiplication triples and random inner product evaluations for the support vector machines and logistic regression classifiers.

In case a trusted initializer is not available or desirable, Alice and Bob can run pre-computations during a setup phase (see, for instance, [189, 190, 197]). In the case of the
protocol evaluating decision trees, to obtain the binary random multiplication triples, Alice and Bob can run oblivious transfer protocols on random inputs. The outcome of these evaluations can be easily transformed in the random binary multiplication triples (see, for instance, [198]). The nice point of this solution is that oblivious transfer can be extended efficiently by using symmetric cryptographic primitives [199, 200, 201]. The online phase of our protocols would remain the same - using solely modular additions and multiplications. Therefore, even considering the offline phase, our protocol would still be substantially more efficient than the protocols proposed in [153] and in [54]. We also remark that the protocol for evaluating decision trees in [94] does not allow its computationally heavy steps (Paillier encryptions and uses of a somewhat homomorphic encryption scheme) to be pre-computed. We also note that while the oblivious transfer executions in [94] could also be pre-computed, the Paillier encryption scheme would still be needed in the online phase.

Evaluating the Secure Algorithms for Evaluation of Classification Models

For decision trees, SVM and logistic regression models we report accuracy (calculated using 10-fold cross validation) for 7 different datasets within the UCI Repository. We also report average classification time for an instance in each dataset when following our privacy-preserving protocol as well as average time required when the classification is done in the clear. Note that the bit-length used to express the values should be large enough as not to compromise the accuracy of the algorithms. It is no real gain for applications if the performance is improved at the cost of drastically decreasing the accuracy, therefore the accuracy is also reported.

Support Vector Machine: For this study, we tested SVM with a linear kernel, and we report the results for accuracy for 7 different datasets from the UCI repository. We lever-aged the e1071 package within R [202], setting type to ‘C-classification’, indicating our problems were classification tasks.

Decision Trees: We used an implementation of the classification and regression tree
algorithm (CART) [203] in R [204]. The minimum deviance (mean squared error) is used as the test parameter for proceeding with a new split. That is, adding a node should reduce the error by at least a certain amount. For our models, we set the complexity parameter to 0.01 and report the corresponding accuracy.

Logistic Regression: For our experimentation, we used R’s base glm function [205], setting the family parameter to binomial(link=“logit”) to obtain a logistic regression model.

The following datasets were chosen for our experimentation: (1) Breast Cancer Wisconsin (Diagnostic): The goal with this dataset is to classify 568 different tumors as malignant or benign. Each tumor is characterized by 30 different continuous features derived from an image of the tumor (i.e. perimeter, area, symmetry, etc.). (2) Pima Indians Diabetes: This dataset includes 767 females of at least 21 years of age, all with Pima Indian decent, and we wish to identify those with diabetes. We leverage 8 different continuous features which describe each woman’s health (examples: body mass index, diastolic blood pressure). (3) Parkinsons: Here, the task is to differentiate between patients with and without Parkinsons. To this end, the dataset includes 22 features, all of which are measures derived from voice recordings of 195 different patients (example: average vocal fundamental frequency). (4) Connectionist Bench (Sonar, Mines vs. Rocks): The goal with this dataset is to differentiate whether 207 sonar signals were bounced off of a metal cylinder vs. a roughly cylindrical rock. Each of the 60 features is within the range of 0.0 to 1.0 and represents energy within a particular frequency band over a certain period of time. (5) Hill-Valley: The task for this dataset is to identify hills vs. valleys in terrain. Each of the 100 continuous features is a point on a 2-D graph. We chose the dataset which did not contain any noise. (6) LSVT Voice Rehabilitation: This dataset includes 126 patients who have undergone voice rehabilitation treatment and we wish to determine the success of their treatment, i.e. whether their phonations are considered acceptable or unacceptable. To do this, we leverage 312 features, each of which is the results of a different speech signal algorithm. (7) Spambase: Here, the goal is to identify 4,600 emails as either spam or not spam. This dataset includes 57
features which describe the contents of each email (examples: word frequencies, number of capital letters).

**Implementation Specifics** To generate preliminary results, the privacy-preserving algorithms were implemented in Java, and compared against a simple implementation without any privacy preservation. For our experiments with the privacy-preserving classifiers, a general bit length, $\ell$, of 64 bits was used for representing all the inputs and throughout all calculations, as this allowed for a good trade off between complexity and space for precision. For some trials, a smaller bit length might have served with sufficient precision.

All values had to be converted to integers to properly work in the proposed algorithms. This was accomplished by choosing a multiplier value and applying it to the features and the weights for SVM and logistic regression or the thresholds for decision trees and rounding any remaining decimals. Furthermore, since calculations were done over a ring, any negative values had to be expressed as their additive inverses. This means in addition to precision considerations, the bit length must be selected in such a way that the positive values and negative values will remain distinctly separate in the lower half and upper half of the values, respectively. This allows us to differentiate between positive and negative values by comparing against $2^{\ell-1}$ instead of 0.

Table 3.5: Results of the experiments for both securely and non-privately trained decision tree classifiers. The classification time is given as the computing time plus the number of half roundtrip times (RTT/2).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Depth of Tree</th>
<th>Number of Features</th>
<th>Accuracy</th>
<th>Classification Time in the Clear (ms)</th>
<th>Classification Time Secure Protocol (ms)</th>
<th>Communication Complexity Uplink+Downlink(kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>4</td>
<td>30</td>
<td>95.95%</td>
<td>0.07 + 1 RTT/2</td>
<td>3.20 + 10 RTT/2</td>
<td>7.96</td>
</tr>
<tr>
<td>Diabetes</td>
<td>9</td>
<td>8</td>
<td>77.18%</td>
<td>0.02 + 1 RTT/2</td>
<td>9.11 + 11 RTT/2</td>
<td>95.94</td>
</tr>
<tr>
<td>Parkinson’s</td>
<td>4</td>
<td>22</td>
<td>88.72%</td>
<td>0.40 + 1 RTT/2</td>
<td>3.62 + 10 RTT/2</td>
<td>6.09</td>
</tr>
<tr>
<td>Connectionist Bench</td>
<td>4</td>
<td>60</td>
<td>73.91%</td>
<td>0.10 + 1 RTT/2</td>
<td>9.64 + 10 RTT</td>
<td>14.99</td>
</tr>
<tr>
<td>Hill-Valley</td>
<td>3</td>
<td>100</td>
<td>49.83%</td>
<td>0.14 + 1 RTT/2</td>
<td>4.85 + 9 RTT/2</td>
<td>11.37</td>
</tr>
<tr>
<td>LSVT rehabilitation</td>
<td>3</td>
<td>310</td>
<td>79.37%</td>
<td>0.75 + 1 RTT/2</td>
<td>12.79 + 9 RTT/2</td>
<td>34.34</td>
</tr>
<tr>
<td>Spambase</td>
<td>6</td>
<td>57</td>
<td>88.89%</td>
<td>0.10 + 1 RTT/2</td>
<td>9.33 + 11 RTT/2</td>
<td>60.04</td>
</tr>
</tbody>
</table>

Table 3.5 presents the results for the case of decision tree classifiers and Table 3.6 for SVM and logistic regression classifiers. These results were generated using a laptop computer with 16 GB DDR4 RAM at 2133 MHz and an Intel Core i7 6700HQ at 2.6 GHz.
Table 3.6: Results of the experiments for both securely and non-privately trained SVM and logistic regression classifiers with 2 classes. The classification time is given as the computing time plus the number of half roundtrip times (RTT/2).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Features</th>
<th>Accuracy</th>
<th>Classification Time in the Clear (ms)</th>
<th>Classification Time Secure Protocol (ms)</th>
<th>Communication Complexity Uplink+Downlink (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td></td>
<td></td>
<td>SVM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>30</td>
<td>97.71%</td>
<td>0.06 + 1 RTT/2</td>
<td>3.47 + 16 RTT/2</td>
<td>0.92</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>77.05%</td>
<td>0.02 + 1 RTT/2</td>
<td>3.04 + 16 RTT/2</td>
<td>0.57</td>
</tr>
<tr>
<td>Parkinson’s</td>
<td>22</td>
<td>87.18%</td>
<td>0.04 + 1 RTT/2</td>
<td>3.36 + 16 RTT/2</td>
<td>0.79</td>
</tr>
<tr>
<td>Connectionist Bench</td>
<td>60</td>
<td>74.70%</td>
<td>0.10 + 1 RTT/2</td>
<td>4.12 + 16 RTT/2</td>
<td>1.39</td>
</tr>
<tr>
<td>Hill-Valley</td>
<td>100</td>
<td>57.59%</td>
<td>0.17 + 1 RTT/2</td>
<td>4.89 + 16 RTT/2</td>
<td>2.01</td>
</tr>
<tr>
<td>LSVT rehabilitation</td>
<td>310</td>
<td>80.16%</td>
<td>0.51 + 1 RTT/2</td>
<td>9.16 + 16 RTT/2</td>
<td>5.29</td>
</tr>
<tr>
<td>Spambase</td>
<td>57</td>
<td>92.72%</td>
<td>0.10 + 1 RTT/2</td>
<td>4.06 + 16 RTT/2</td>
<td>1.34</td>
</tr>
</tbody>
</table>

Logistic Regression

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Features</th>
<th>Accuracy</th>
<th>Classification Time in the Clear (ms)</th>
<th>Classification Time Secure Protocol (ms)</th>
<th>Communication Complexity Uplink+Downlink (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>30</td>
<td>95.95%</td>
<td>0.07 + 1 RTT/2</td>
<td>3.55 + 16 RTT/2</td>
<td>0.92</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>77.31%</td>
<td>0.02 + 1 RTT/2</td>
<td>3.06 + 16 RTT/2</td>
<td>0.57</td>
</tr>
<tr>
<td>Parkinson’s</td>
<td>22</td>
<td>85.13%</td>
<td>0.04 + 1 RTT/2</td>
<td>3.35 + 16 RTT/2</td>
<td>0.79</td>
</tr>
<tr>
<td>Connectionist Bench</td>
<td>60</td>
<td>74.4%</td>
<td>0.11 + 1 RTT/2</td>
<td>4.16 + 16 RTT/2</td>
<td>1.39</td>
</tr>
<tr>
<td>Hill-Valley</td>
<td>100</td>
<td>60.07%</td>
<td>0.16 + 1 RTT/2</td>
<td>4.97 + 16 RTT/2</td>
<td>2.01</td>
</tr>
<tr>
<td>LSVT rehabilitation</td>
<td>310</td>
<td>53.17%</td>
<td>0.49 + 1 RTT/2</td>
<td>9.64 + 16 RTT/2</td>
<td>5.29</td>
</tr>
<tr>
<td>Spambase</td>
<td>57</td>
<td>92.7%</td>
<td>0.10 + 1 RTT/2</td>
<td>4.17 + 16 RTT/2</td>
<td>1.34</td>
</tr>
</tbody>
</table>

For each dataset the average was computed using more than 10000 scorings.

**Analysis and Comparisons to Previous Results**

**Decision Trees.** The computing time for running our protocol for the privacy-preserving evaluation of decision trees is at most 13 milliseconds for trees of depth up to 9. In Bost et al. [153], for evaluating a tree of depth 4, the computing time is in the order of a few seconds. Our protocol has 11 rounds of communication or less for trees with depth up to 9, while their number of interactions is always over 30, even for trees of depth 4. In the case of the protocols for computing decision trees of Wu et al. [94], the computing time for a tree with depth 4 is around 100 ms. The communication complexity of our protocol for a decision tree of depth 4 and 8 features is around 3KB, while the results in [94] are around 100KB and in [153] are around 3MB for trees of the same dimension. As stated in these previous works, solutions based on general purpose multiparty computation frameworks have a much poorer performance than their specific protocols (and hence than the solutions presented here as well).

**Support Vector Machines.** We run the protocols proposed in [126] with the building blocks presented in this paper. While there are no implementation times given in [126], it is clear that our implementations have a significant impact in the performance. The number of
rounds is usually the most important factor in determining the latency of these protocols and we reduce the round complexity from linear to logarithmic in the input length. Compared to the implementations described in Bost et al. [153] the computation times are about 50ms for 30 and 47 features. In our case for 30 features, the computing time is less than 4 ms. Our number of rounds is larger: our solution takes 16 rounds, while their solution takes 7 rounds. If the roundtrip time is the major factor in the total time their solution is preferable to ours. The main reason for the elevated round complexity in our solution is the bit decomposition protocol, which is not needed in their work.

**Logistic Regression.** The efficiency of the logistic regression protocol is the same as the support vector machine one.

*Other Approaches to Privacy-Preserving Evaluation of Machine Learning Classifiers*

There is a huge literature in training privacy-preserving machine learning models (see [206] for a survey). However, general (non-application specific) privacy-preserving protocols for privately scoring machine learning classifiers were proposed just recently in [153] for the case of hyperplane-based classifiers, Naive Bayes and decision trees and in [94] for decision trees and random forests. In [126] protocols for hyperplane-based and Naive Bayes classifiers were proposed.

De Hoogh et al. [96] introduced the most efficient protocol for privacy-preserving training of decision trees with categorical attributes only. They also presented a protocol for privacy-preserving scoring of decision trees. Their protocol is designed for categorical attributes. It does not scale well for fined-grained numerical attributes - the complexity of the protocol increases exponentially on the bit-length representation of a category.

Many classification problems are characterized by numerical attributes, such as age, temperature, or blood test results, or by a combination of numerical and categorical attributes. The well known top down algorithms to induce decision trees from data (ID3, CART) can easily be extended to include numerical attributes as well. This is typically
done with a binary split at internal nodes, e.g. instances with \( \text{cholesterol level} \leq p \) go down the left branch, and instances with \( \text{cholesterol level} > p \) go down the right. The threshold \( p \) is chosen dynamically at each node as the tree is grown, and, unlike with categorical attributes, a numerical attribute may appear more than once in the same tree branch, but with different thresholds. For instance, in the branch below the node \( \text{cholesterol level} \leq p \), a new node \( \text{cholesterol level} \leq p^* \) may appear, with \( p^* \) a smaller threshold than \( p \). The process of dynamically choosing and refining thresholds adds to the expressivity of decision trees with numerical values, making the hypothesis space of such trees far richer than that of decision trees with categorical values.

In [153], hyperplane-based classifiers were implemented by using a secure protocol for computing the inner product based on the Paillier encryption scheme and a comparison protocol that also relies heavily on the Paillier encryption scheme.

The decision tree protocol of Bost et al. [153] is divided in two phases. In a first stage Paillier-based comparison protocols are run with Alice inputting a vector containing her features and Bob inputting the threshold values of the decision tree. On a second stage, fully homomorphic encryption is used to process the outcomes of the comparison protocols run in the first stage. It is claimed that the protocol leaks nothing about the tree (we will show that in a more realistic attack scenario this is not true) and the second stage is round-optimal. However, the computations to be performed are heavy and the first stage involves many rounds (in total their protocol typically has more rounds than ours). In our solution, we allow the depth of the tree to be leaked, but avoid altogether using Paillier and fully homomorphic encryption. In our solution, the online phase for evaluating decision trees uses solely modular additions and multiplications.

In [94] protocols for decision trees and random forests were proposed. The protocols are based on an original comparison protocol also based on the Paillier encryption scheme and on oblivious transfer. The Paillier encryption scheme uses modular exponentiation and oblivious transfer protocols that are usually as expensive as public-key cryptographic
primitives. As pointed out in the introduction, our solutions use, in the online phase, solely additions and multiplications over a finite field or ring.

In [126], one can find protocols for hyperplane-based and Naive Bayes classifiers in the commodity-based model. By directly replacing some of the building blocks used in [126] (the comparison and bit decomposition protocols) by the ones used here, the communication and computing complexities can be decreased.

**How much information is leaked about the decision trees in [153] and in [94]?** In the protocol in [153], theoretically nothing is ever leaked about the tree. However, if an adversary can measure the time it takes for Bob to do the evaluation of the decision tree protocol, clearly the deeper the tree the longer the computation becomes. Therefore, some information about the depth of the tree is leaked if this side channel attack is considered. Therefore, in our solution we do not loose much by giving away the depth of the tree to an adversary. In [94], the depth of the tree is also leaked.

**Bit Decomposition Protocols.** The best solution for bit-decomposition, in terms of round complexity, is a constant-round solution by Toft [207], which has round complexity equal to 23. Veugen noted in [208] that for a certain range of practical parameters (number of input bits less than 20), a protocol with a linear number of rounds in the length of the input could outperform the solution presented by Toft [207]. Veugen proposed a protocol that has a linear number of rounds in $\ell$, where $\ell$ is the length of the input in bits. Veugen also proposed a way to reduce the number of rounds of this protocol by a factor of $\beta$, obtaining a round complexity equal to $\ell/\beta$ at the cost of performing an exponential (in $\beta$) number of multiplications in a pre-processing phase.

The bit-decomposition protocol used in our work is over binary fields and runs in $2 + \lceil \log \ell \rceil$ rounds. For practical values of $\ell$ (less than 100 typically), it is always better than Toft’s and Veugen’s solutions. The number of multiplications to be performed in our the online phase, $2\ell \lceil \log \ell \rceil + 3\ell$, is less than the $31\ell \lceil \log \ell \rceil + 71\ell + 30\lceil \sqrt{\ell} \rceil$ multiplications in the case of Toft’s protocol. While Veugen’s protocol can have a fast online phase, requiring
only $3\ell - 2\beta$ multiplications for $\ell/\beta$ rounds, it requires an exponential (in $\beta$) number of multiplications in the offline phase.

The protocol of Schoenmakers and Tuyls [209] has the same number of rounds and roughly half as many multiplications as the protocol used here. However the multiplication are in $\mathbb{Z}_q$ for big $q$ while our multiplications are in $\mathbb{Z}_2$. Hence our multiplications are faster and communicate less data. In addition, in our case OT extension can be directly used for the pre-computation if a trusted initializer is not available. For more details about Schoenmakers and Tuyls’ protocol see the original paper or Section 4.3.5 of De Hoogh’s PhD thesis [193].

A restriction of our protocol is that it only works for operations modulo a power of 2. As we need no modular inversions in our privacy-preserving machine learning protocols this imposes no problem at all. The bit-decomposition protocol of Laud and Randmets [194] for the case of three parties with at most one corruption is similar to one here. It first reduce the original problem to a new one between two-parties, and then uses the adder idea to obtain bitwise shares. Although the protocol is not fully specified in [194], we believe that the authors intended to use the same adder computation as here. We believe that Veugen’s and Toft’s protocols can also be improved in the case that the module is a power of 2.

3.2.4 Real World and System Considerations in Privacy-Preserving ML

Finally, we must also consider practical issues when discussing privacy-preserving machine learning. We specifically highlight the issues in computation cost, in variation in privacy policies, and the reality that no privacy framework exists in a vacuum but may be used in an ecosystem of solutions.

Computation Cost

Secure multiparty computation protocols routinely require expensive cryptographic operations, either with data encrypted with public key cryptography schemes [94, 153, 210,
211], or across a large finite fields [95, 212, 96]. Additionally, secure multiparty computation requires many back and forth exchanges between parties when compared to performing operations in the clear. Due to relatively high computation and communication costs, an open question is whether secure multiparty computation solutions are truly feasible for real world implementation. We argue that high performance SMC protocols can be a breakthrough for wide deployment of privacy-preserving inductive learning.

**Incorporating Different Trust and Sensitivity Levels**

Currently, solutions across randomization, differential privacy, and secure multiparty computation models treat all data samples and all participating organizations equally as homogeneous entities. However, this is not the case in practice. Quite often, different organizations may want to add different levels of noise based upon different levels of trust or sensitivity. One party may only agree to share information which is an aggregation across the data of many parties to protect against revealing or intruding the privacy of any individual. Another party may only agree to share information about those less sensitive data elements or model parameters. Some parties may abstain from sharing results to certain users. Each of these scenarios calls for variation in privacy treatment in order to match the varying sensitivity of different data elements and the varying trust in participating parties.

**Combining SMC with Differential Privacy**

There is, to date, limited work in combining secure multiparty computation with differential privacy. We argue that if both techniques were combined, then some previously mentioned pitfalls may be minimized. There are two potential ways to combine SMC and differential privacy: (1) develop an SMC protocol that produces a differentially private decision tree model or (2) use differential privacy techniques to accomplish secure multiparty computation tasks for some data exchanges within the secure multiparty computation protocol where the addition of noise may not impact the accuracy of the resulting decision tree.
too strongly. Choosing to employ noise injection techniques from the differential privacy domain for some portions of a secure multiparty computation protocol may improve the computational complexity problem which plagues many existing secure multiparty computation solutions. We conjecture that a patchwork implementation working across different privacy models may have the potential to produce solutions that address a broader range of security and privacy issues.

3.3 Inference Attacks Against Privacy-Preserving Machine Learning

While we have discussed at length the application of privacy-preserving techniques to ML tasks, we have not yet answered the question of how effective such techniques are in mitigating privacy attacks such as membership inference. We now look in detail at the effectiveness of differential privacy employed to deep learning models as a countermeasure for membership inference mitigation.

To define utility loss we follow [213] and consider $loss = 1 - \frac{dp - acc}{acc}$ where $acc$ represents model accuracy in a non-private setting and $dp - acc$ is the accuracy when differential privacy is employed for training the same model across the same training dataset.

We evaluate the effectiveness of privacy-preserving deep learning with differential privacy as a mitigation technique for membership inference attacks within the context of two important issues: (1) model and problem complexity and (2) skewed datasets.

3.3.1 Model and Problem Complexity

In Table 3.7 compares three different datasets (first column) and learning tasks (third column) using different model structures (second column). Recall from Section 2.2.1 that the MNIST dataset is a 10-class black and white image classification problem and Section 2.1.1 that the CIFAR-10 dataset is a 10-class color image classification problem. Also publicly available, CIFAR-100, similar to CIFAR-10, contains 60,000 color images [10] formatted to 32 x 32. 100 classes are represented ranging from various animals, pieces of house-
hold furniture, or types of vehicles. Each class has 600 available images. The problem is therefore a 100-class image classification problem. We measure utility loss incurred when differential privacy is introduced, including both the training loss and test loss, as well as the vulnerability when non-private training is used. For MNIST, we follow the architecture outlined in [168] by first applying PCA with 60 components to reduce data dimensionality before feeding the data into simple neural network with one hidden ReLU layer containing 1,000 units.

For CIFAR-10 and CIFAR-100 we test two separate training approaches and architectures. First, we implement the architecture used in [213] for both CIFAR-10 and CIFAR-100. This includes no transfer learning but does include a PCA transformation prior to training to reduce dimensionality to 50 features. The network used in this case consists of two fully connected layers with 256 units each and a softmax layer for the output.

We also follow the approach from [168] which uses two ReLU convolutional layers with 64 channels, $5 \times 5$ convolutions, and a stride of 1. Each convolutional layer is then followed by a $2 \times 2$ max pool. There are two fully connected layers with 384 units before the final softmax layer. Prior to training the network, transfer learning (TL) is used to initialize model weights. For CIFAR-10 transfer learning is done with the CIFAR-100 dataset while the reverse is done for the CIFAR-100 dataset. The convolutional layers are fixed so that only the fully connected and softmax layers are updated during training. Finally, the data is cropped to $24 \times 24$ to further reduce dimensionality.

For the non-private setting, we fix learning to 100 epochs and use the RMSPROP optimizer with learning rate set to 0.001, decay to $1e-6$, and the batch size set at 32. For the differential privacy setting we use some of the default parameter values from the tensorflow privacy repository for MNIST example [214]: setting the learning rate to 0.15, batch size to 256, microbatches to 256, and the clipping value to 1.0. We then set the noise multiplier to the correct scale according to the privacy budget $(\epsilon, \delta)$ and 100 epochs of learning. In Table 3.7, $(\epsilon, \delta) = (10, 10^{-5})$. We employ the Rényi accounting approach [215] and leverage
Table 3.7: Problem complexity vs membership inference vulnerability and differential privacy utility loss with a DP guarantee of \((\epsilon, \delta) = (10, 10^{-5})\).

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Trainable Parameters</th>
<th>Classes</th>
<th>Train Loss w/ DP (%)</th>
<th>Test Loss w/ DP (%)</th>
<th>Non-Private Vulnerability (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>71,010</td>
<td>10</td>
<td>5.09</td>
<td>3.51</td>
<td>53.11</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>83,978</td>
<td>10</td>
<td>55.90</td>
<td>14.34</td>
<td>72.58</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>107,108</td>
<td>100</td>
<td>82.66</td>
<td>49.34</td>
<td>74.04</td>
</tr>
<tr>
<td>CIFAR-10 (with TL)</td>
<td>1,036,810</td>
<td>10</td>
<td>51.00</td>
<td>26.59</td>
<td>72.94</td>
</tr>
<tr>
<td>CIFAR-100 (with TL)</td>
<td>1,071,460</td>
<td>100</td>
<td>85.60</td>
<td>58.39</td>
<td>89.08</td>
</tr>
</tbody>
</table>

the tensorflow DPGRADEIENTDESCENTGAUSSIANOPTIMIZER which adheres to the fixed \(\sigma\) technique detailed in Algorithm 1.

In Table 3.7 we compare the five scenarios (rows) with respect to utility loss and vulnerability of membership inference attacks. The results demonstrate that there is an unfortunate trade-off between the vulnerability and the usability of differential privacy as a mitigation technique. More complex models and classification problems display more significant utility loss when differential privacy is employed during the DNN model training. The MNIST dataset represents a less complex dataset than the CIFAR-10 and CIFAR-100 datasets. In MNIST the images are black and white with MNIST images of the same class being more similar than the colored object images in the CIFAR datasets. Therefore, the MNIST dataset is more easily learned by a small DNN model architecture with reasonable accuracy in the presence of a differentially private optimizer which adds noise perturbation into each training epoch. Not surprisingly, for the same reason, the MNIST model has reduced vulnerability to membership inference attacks as the training examples are more similar to one another.

The CIFAR-10 dataset, on the other hand, is more complex and require more complex models with a greater number of trainable parameters. In this case the models demonstrate a higher vulnerability to membership inference attack than the MNIST dataset. Unfortunately, this complexity also increases the loss in utility when a differentially private
optimizer is employed in the model training.

Additionally, comparing the CIFAR-10 dataset with the CIFAR-100 dataset, CIFAR-100 represents a more complex learning problem with 100 possible classes rather than 10. This again results in not only a more significant loss for the CIFAR-100 dataset in the presence of differentially private deep learning, but also higher vulnerability to membership inference attacks.

Compare the two models trained on the most complex dataset CIFAR-100, we observe the impact of the model complexity on the vulnerability of membership inference attack. The CIFAR-100 with transfer learning (last row) will have a (24, 24, 3) image input compared to the 50 feature input from the PCA approach using CIFAR-100 (third row). After processing the image through the fixed convolutional layers, the model with transfer learning still has significantly more parameters (compare row 5 with row 3). We note the more complex model structure (over 1 million parameters) has increased membership inference vulnerability for the CIFAR-100 dataset with 89.08% attack accuracy compared with the vulnerability of 74.04% when using the less complex model with reduced dimensionality of input (with 107,108 trainable parameters). However the mode complex model also reports a greater test loss of 58.39% compared to the loss of 49.34% for the less complex model.

Our experimental results from Table 3.7 underscore an important observation. Using differential privacy as effective mitigation for membership inference remains an open challenge. One primary reason is that even with differentially private deep learning, the most vulnerable DNN models (and datasets) are unfortunately also those which experience the greatest utility loss.

3.3.2 Implications for Skewed Datasets

An additional challenge in the deployment of differential privacy is the impact on skewed datasets. Using the approach and model structure from [213], we experiment with different
levels of data skewness in the CIFAR-10 dataset. We again gradually decrease the representation of the automobile class from 10% (even distribution) to 2% of the training data for a fixed $\epsilon$ of 1000. While this is a large $\epsilon$ value when considering the differential privacy theory, the results in [213] indicate that more common theoretical $\epsilon$ values, such as 1, result in no meaningful learning at all. Our experiments also validate this observation. We conduct this set of experiments by varying the data skewness in the presence of the largest $\epsilon$ reported in [213]. In addition, we also measure and report the results for smaller $\epsilon$ with the automobile class set to 2% of its training data. Figure 3.18 shows the results of these two sets of experiments, measured using overall model accuracy loss and automobile F-1 score loss. We compare the differentially private deep learning setting with the non-private setting using the same model architecture and data distribution. We report the percentage
of utility loss according to these scores.

In statistical analysis of binary classification, the F-1 score measures the accuracy by considering both the precision and the recall of the test to compute the score. The F-1 score is the harmonic mean of the precision and recall. An F-1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. Let $p$ be the number of correct positive results divided by the number of all positive results returned by the classifier, and $r$ be the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). We have $F - 1 = 2(p \times r)/(p + r)$. To isolate the utility loss for the automobile class we therefore report the loss in F-1 score calculated using the precision and recall of only the automobile class where automobile is considered positive and all other classes negative.

On the left sub-figure of Figure 3.18, we compare the accuracy loss of the overall model with the loss in F-1 score of the automobile class (class 1). We vary the data skewness of the automobile class in the training set, and report the results by the percentage of decrease in F-1 score when using differentially private deep learning setting compared to the non-private deep learning setting. This left sub-figure shows that as the skewness decreases from 2% to 10% (X-axis), the gap between the private setting and the non-private setting becomes closer. When the data skewness is high for the automobile class (class 1), the minority group is more vulnerable, as seen in Figure 2.2, but unfortunately also shows the greatest utility loss for both training and test when differential privacy is introduced.

On the right sub-figure of Figure 3.18, we again compare the accuracy loss of the overall model with the loss in F-1 score of the automobile class (class 1). This time we vary the privacy parameter $\epsilon$ from 10 to 1000, and report the measurement results by the percentage of utility loss when using a differentially private deep learning setting compared to a non-private deep learning setting. We observe that the minority group has much higher utility loss under the privacy setting, even when the privacy budget parameter $\epsilon$ is set to larger (and theoretically almost meaningless) values. This confirms the observation that differential
privacy as a mitigation technique presents a catch-22 as the minority class is more likely to be successfully targeted by a membership inference attack but also suffers the most under differential privacy.

In summary, Figure 3.18 highlights how differential privacy as a mitigation technique also leads to unfortunate outcomes for minority classes. That is, the data which is most vulnerable to attack is also the most likely to experience intolerable accuracy loss under differentially private setting compared to non-private setting. This mirrors the challenge wherein complex datasets and complex model architectures are more vulnerable to membership inference attacks, and at the same time, are also more likely to experience significant accuracy loss with differentially private deep learning.
CHAPTER 4
PRIVACY-PRESERVING FEDERATED LEARNING

In Chapter 3 we demonstrated how to apply privacy techniques to machine learning training and evaluation processes. However, modern systems are becoming more frequently distributed and users are becoming more hesitant to (a) share their personal data or (b) allow those who collect their data to share it with others. This has led to the growing prominence of federated learning (FL) systems. We therefore now turn our attention to the challenge of protecting privacy in FL.

4.1 Federated Learning Systems

4.1.1 Federated Machine Learning

Recall that FL systems allow global model training without the sharing of raw private data. Instead, individual participants only share model parameter updates. Let’s consider federated learning more closely from the perspective of training a deep neural network (DNN) model. Recall that DNNs consist of multiple layers of nodes where each node is a basic functional unit with a corresponding set of parameters. Nodes receive input from the immediately preceding layer and send output to the following layer; with the first layer nodes receiving input from the training data and the final layer nodes generating the predictive result.

In a traditional DNN learning scenario, there exists a training dataset \( D = (x_1, \ldots, x_n) \) and a loss function \( \mathcal{L} \). Each \( x_i \in D \) is defined as a set of features \( f_i \) and a class label \( c_i \in \mathcal{C} \) where \( \mathcal{C} \) is the set of all possible class values. The final layer of a DNN architecture for such a dataset will consequently contain \( |\mathcal{C}| \) nodes, each corresponding to a different class in \( \mathcal{C} \). The loss of this DNN given parameters \( \theta \) on \( D \) is denoted: \( \mathcal{L} = \frac{1}{n} \sum_i^n \mathcal{L}(\theta, x_i) \).
When \( f \) is fed through the DNN with model parameters \( \theta \), the output is a set of predicted probabilities \( p_i \). Each value \( p_{c,i} \in p_i \) is the predicted probability that \( x_i \) has a class value \( c \), and \( p_i \) contains a probability \( p_{c,i} \) for each class value \( c \in C \). Each predicted probability \( p_{c,i} \) is computed by a node \( n_c \) in the final layer of the DNN architecture using input received from the preceding layer and \( n_c \)’s corresponding parameters in \( \theta \). The predicted class for instance \( x_i \) given a model \( M \) with parameters \( \theta \) then becomes \( M_\theta(x_i) = \arg \max_{c \in C} p_{c,i} \). Given a cross entropy loss function, the loss on \( x_i \) can consequently can be calculated as 
\[
\mathcal{L}(\theta, x_i) = -\sum_{c \in C} y_{c,i} \log(p_{c,i}) \quad \text{where} \quad y_{c,i} = 1 \text{ if } c = c_i \text{ and } 0 \text{ otherwise.}
\]
The goal of training a DNN model then becomes to find the parameter values for \( \theta \) which minimize the chosen loss function \( \mathcal{L} \).

The process of minimizing this loss is typically done through an iterative process called stochastic gradient descent (SGD). At each step, the SGD algorithm (1) selects a batch of samples \( B \subseteq D \), (2) computes the corresponding gradient \( g_B = \frac{1}{|B|} \sum_{x \in B} \nabla_\theta \mathcal{L}(\theta, x) \), and (3) then updates \( \theta \) in the direction \( -g_B \). In practice, \( D \) is shuffled and then evenly divided into \(|B|\) sized batches such that each sample occurs in exactly one batch. Applying SGD iteratively to each of the pre-determined batches is then referred to as one epoch.

In FL environments however, the training dataset \( D \) is not wholly available at one location. Instead, \( N \) participants \( \mathcal{P} \) each hold their own private training dataset \( D_1, ..., D_N \). Rather than sharing their private raw data, participants instead execute the SGD training algorithm locally and then upload updated parameters to a centralized server (aggregator). Specifically, in the initialization phase (i.e., round 0), the aggregator generates a DNN architecture with parameters \( \theta_0 \) which is advertised to all participants. At each global training round \( r \), a subset \( \mathcal{P}_r \) consisting of \( k \leq N \) participants is selected based on availability. Each participant \( P_i \in \mathcal{P}_r \) executes one epoch of SGD locally on \( D_i \) to obtain updated parameters \( \theta_{r,i} \), which are sent to the aggregator. The aggregator sets the global parameters \( \theta_r = \frac{1}{k} \sum_i \theta_{r,i} \forall i \) where \( P_i \in \mathcal{P}_r \). The global parameters \( \theta_r \) are then advertised to all \( N \) participants. These global parameters at the end of round \( r \) are used in the next training
round $r + 1$. After $R$ total global training rounds, the model $M$ is finalized with parameters $\theta_R$.

### 4.1.2 Privacy in Federated Machine Learning

**Threat Model**

Let’s consider a FL system wherein $n$ participants use an ML service for FL. This service will serve as the *aggregator*. We consider three potential adversaries within the context of such as system: (1) the aggregator, (2) the participants, and (3) outsiders.

**Aggregator** As we saw in Chapter 3, the honest-but-curious or semi-honest adversarial model is commonly used in the field of SMC since its introduction in [216] and application to data mining in [151]. Honest-but-curious adversaries follow the protocol instructions correctly but will try to learn additional information. Therefore, we will discuss in this chapter an aggregator that *will not* vary from the predetermined ML algorithm but *will* attempt to infer private information using all data received throughout the protocol execution. Given the aggregator is likely, in practice, to be some MLaaS functionality, it is a practical assumption that the aggregator service would be motivated to, at a minimum, provide the promised aggregator functionality and therefore will follow the ML algorithm.

**Participants** As we showed in Section 2.4.1, participants in the FL system also present a privacy threat. Additionally, participants may not be as trusted as the aggregator or may be able to collude amongst one another. Therefore, in contrast to the aggregator, we want to sometimes consider scenarios in which, potentially colluding, parties in $\mathcal{P}$ may deviate from the protocol execution to achieve additional information on data held by honest parties.

**Outsiders** We also consider potential attacks from adversaries outside of the system. This includes adversaries monitoring communications during training looking to infer the pri-
vate data of the participants as well as adversarial users of the final model.

In addition to the aforementioned adversarial models, proposed FL systems may make a number of additional assumptions to more concretely formulate the threat model.

**Additional System Assumptions** One such assumption is secure channels between each party and the aggregator. This allows the aggregator to authenticate incoming messages and prevents an adversary, whether they be an outsider or malicious participant, from injecting their own responses.

Another critical assumption when cryptographic schemes are leveraged (as discussed in Section 3.1.4 is secure key distribution. There may also be assumptions made by the cryptosystem itself as well to ensure security. Within the context of FL systems, these assumptions can be essential in ensuring the privacy of individual messages sent to the aggregator.

**Inference Threats**

We must consider two potential threats to data privacy in FL environments: (1) inference during the learning process and (2) inference over the outputs. *Inference during the learning process* refers to any participant in the federation inferring information about another participant’s private dataset given the data exchanged during the execution of the federated learning algorithm. *Inference over the outputs* refers to the leakage of any participants’ data from intermediate outputs as well as the final, trained predictive model.

We consider two types of inference attacks: insider and outsider. Recall from Section 2.4.1 that *Insider attacks* include those launched by participants in the FL system. This includes both data holders as well as any third parties. *Outsider attacks* on the other hand include those launched both by eavesdroppers to the communication between participants and by users of the final predictive model when deployed as a service. Let’s consider each of these threats in more detail.
**Threats During Model Training**  Let us consider a federated learning algorithm $f_M$. We can view $f_M$ as the combination of computational operations and a set of queries $Q_1, Q_2, \ldots, Q_k$. That is, for each step $s$ in $f_M$ requiring knowledge of the parties’ data there is a query $Q_s$. In the execution of $f_M$ each party $P_i$ must respond to each such query $Q_s$ with appropriate information on $D_i$. The types of queries are highly dependent on $f_M$. For example, to build a decision tree, a query may request the number of instances in $D_i$ matching a certain criteria. In contrast, to train an SVM or neural network a query would request model parameters after a certain number of training iterations. Any privacy-preserving FL system must account for the risk of inference over the responses to these queries.

Privacy-preserving ML approaches addressing this risk often do so by using secure multiparty computation (SMC). Recall that, generally, SMC protocols allow $n$ parties to obtain the output of a function over their $n$ inputs while preventing knowledge of anything other than this output [217]. Unfortunately, approaches exclusively using secure multiparty computation remain vulnerable to inference over the output. As the function output remains unchanged from function execution without privacy, the output can reveal information about individual inputs. Therefore, we must also consider potential inference over outputs.

**Threats from Model Evaluation** This refers to intermediate outputs available to participants as well as the predictive model. As we demonstrated in Chapter 2, given only black-box access to the model through an ML as a service API, an attacker can still make training data inferences. An FL system should prevent such outsider attacks while also considering insiders. That is, participant $P_i$ should not be able to infer information about $D_j$ when $i \neq j$.

Solutions addressing privacy of output often make use of the DP framework. As a mechanism satisfying differential privacy guarantees that if an individual contained in a given dataset is removed, no outputs would become significantly more or less likely [104],
a learning algorithm $f_M$ which is theoretically proven to be $\epsilon$-differentially private is guaranteed to have a certain privacy of output quantified by the $\epsilon$ privacy parameter.

In the federated learning setting it is important to note that the definition of neighboring databases is consistent with the usual DP definition – that is, privacy is provided at the individual record level, not the party level (which may represent many individuals).

### 4.1.3 Deployment Considerations in Federated Learning Systems

In addition to privacy threats from inference attack, there are additional deployment considerations for the real-world use of federated learning systems including straggler participants and malicious participants launching data poisoning attacks.

**Straggler Participants**

The computing resources of individual participants in a FL system can be far less powerful than the computing nodes in conventional super computers. Additionally, cross-device FL where the participants are usually a massive number (e.g., up to 1010) of mobile or IoT devices with various computing and communication capacities [165, 158, 163] intrinsically pushes the heterogeneity of computing and communication resources. The data in FL is also owned by participants where the quantity and content can be quite different from each other, causing severe heterogeneity in data that usually does not appear in datacenter distributed learning, where data distribution is well controlled.

In [164] we conduct a case study to quantify how data and resource heterogeneity in participants impacts the performance of FL in terms of training performance and model accuracy. The key findings are as follows: (1) training throughput is usually bounded by slow participants (a.k.a. stragglers) with less computational capacity and/or slower communication, which we name as the resource heterogeneity. (2) Different participants may train on different quantity of samples per training round and results in different round time that is similar to the straggler effect, which impacts the training time and potentially also the ac-
accuracy. We name this observation the data quantity heterogeneity. (3) In FL the distribution of data classes and features depends on the data owners, thus resulting in a non-uniform data distribution, known as non-Identical Independent Distribution (non-IID data heterogeneity). Our experiments in [164] show that such heterogeneity can significantly impact the training time and accuracy.

[47] proposes a simple approach to handle the stragglers problem in FL, where the aggregator selects 130% of the target number of devices to initially participate, and discards stragglers during training process. With this method, the aggregator can get 30% tolerance for the stragglers by ignoring the updates from the slower edge devices. However, the 30% is set arbitrarily which requires further tuning. Furthermore simply dropping the slower participants might exclude certain data distributions available on the slower participants from contributing towards training the global model. FedProx [161] also tackles resource and data heterogeneity by making improvements on the aggregation algorithm. However, they also discard training data to make up for the systems heterogeneity.

[161] takes into account the resource heterogeneity. However, the proposed approach is mainly focused on only two types of participants - stragglers and non-stragglers. In a real FL environment there is a wide range of heterogeneity levels. The proposed approach performs well in case of high ration of stragglers vs non-stragglers (80-90%). Moreover, their proposed solution involves partial training on stragglers which can further lead to biasness in trained model and sub-optimal model accuracy. [218] proposes that adding local cache is an efficient and reliable technique to deal with stragglers in datacenter distributed Learning. However, FL is powerless in governing the resources on the participant side, so it’s impractical to implement similar mechanisms in FL. Stragglers therefore remain an important open issue in FL systems.

[160] proposes a general statistical model for Byzantine machines and participants with data heterogeneity that clusters based on data distribution. While data distribution may cause stragglers, [160] focuses on grouping edge devices such that their datasets are sim-
The authors do not consider the impact of clustering on training time or accuracy. [219] propose a novel design named RapidReassignment to handle stragglers by specializing work shedding. It uses P2P communication among workers to detect slowed workers, performs work re-assignment, and exploits iteration knowledge to further reduce how much data needs to be preloaded on helpers. However, migrating a user’s private data to other unknown users’ devices is strictly restricted in FL. An analogical approach named SpecSync is proposed in [220], where each worker speculates about the parameter updates from others, and if necessary, it aborts the ongoing computation, pulls fresher parameters to start over, so as to opportunistically improve the training quality. However, information sharing between participants is not allowed in FL. We will present our solution to addressing this issue in Section 4.2.2.

**Malicious Participants Launching Label Flipping Attacks**

Another issue in FL is the risk of adversarial ML attacks. Recall the label flipping data poisoning attack discussed in Section 2.4.2. We now investigate practical concerns relevant to FL system vulnerability to such attacks including the impact of attack timing and malicious participant availability. Again, recall from Section 2.4.2 that our data poisoning experiments are conducted using the popular Fashion-MNIST and CIFAR-10 image classification datasets.

**Impact of Timing on Attack Effectiveness**  While label flipping attacks can occur at any point in the learning process and last for arbitrary lengths, it is important to understand the capabilities of adversaries who are available for only part of the training process. For instance, Google’s Gboard application of FL requires all participant devices be plugged into power and connected to the internet via WiFi [47]. Such requirements create cyclic conditions where many participants are not available during the day, when phones are not plugged in and are actively in use. Adversaries can take advantage of this design choice,
Figure 4.1: Source class recall by round for experiments with “early round poisoning”, i.e., malicious participation only in the first 75 rounds ($r < 75$). The blue line indicates the round at which malicious participation is no longer allowed.

making themselves available at times when honest participants are unable to.

We consider two scenarios in which the adversary is restricted in the time in which they are able to make malicious participants available: one in which the adversary makes malicious participants available only before the 75th training round, and one in which malicious participants are available only after the 75th training round. As the rate of global model accuracy improvement decreases with both datasets by training round 75, we choose this point to highlight how pre-established model stability may effect an adversary’s ability to launch an effective label flipping attack. Results for the first scenario are given in Figure 4.1 whereas the results for the second scenario are given in Figure 4.2.

In Figure 4.1, we compare source class recall in a non-poisoned setting versus a setting with poisoning only before round 75. Results on both CIFAR-10 and Fashion-MNIST show that while there are observable drops in source class recall during the rounds with poisoning (1-75), the global model is able to recover quickly after poisoning finishes (after round 75). Furthermore, the final convergence of the models (towards the end of training) are not impacted, given the models with and without poisoning are converge with roughly the same recall values. We do note that some CIFAR-10 experiments exhibited delayed convergence by an additional 50-100 training rounds, but these circumstances were rare
Figure 4.2: Source class recall by round for experiments with “late round poisoning”, i.e., malicious participation only after round 75 ($r \geq 75$). The blue line indicates the round at which malicious participation starts.

and still eventually achieved the accuracy and recall levels of a non-poisoned model despite delayed convergence.

Table 4.1: Final source class recall when at least one malicious party conducting a label flipping data poisoning attack participates in the final round of training vs. when all participants in the final round of training are non-malicious.

<table>
<thead>
<tr>
<th>$c_{src} \rightarrow c_{target}$</th>
<th>CIFAR-10</th>
<th>Fashion-MNIST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m% \in P_R &gt; 0$</td>
<td>$m% \in P_R = 0$</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>0 $\rightarrow$ 2</td>
<td>73.90%</td>
<td>82.45%</td>
</tr>
<tr>
<td>1 $\rightarrow$ 9</td>
<td>77.30%</td>
<td>89.40%</td>
</tr>
<tr>
<td>5 $\rightarrow$ 3</td>
<td>57.50%</td>
<td>73.10%</td>
</tr>
<tr>
<td>1 $\rightarrow$ 3</td>
<td>84.32%</td>
<td>96.25%</td>
</tr>
<tr>
<td>4 $\rightarrow$ 6</td>
<td>51.50%</td>
<td>89.60%</td>
</tr>
<tr>
<td>6 $\rightarrow$ 0</td>
<td>49.80%</td>
<td>73.15%</td>
</tr>
</tbody>
</table>

In Figure 4.2, we compare source class recall in a non-poisoned setting versus with poisoning limited to the 75th and later training rounds. These results show the impact of such late poisoning demonstrating limited longevity; a phenomena which can be seen in the quick and dramatic changes in source class recall. Specifically, source class recall quickly returns to baseline levels once fewer malicious participants are selected in a training round even immediately following a round with a large number of malicious participants having
caused a dramatic drop. However, the final poisoned model in the late-round poisoning scenario may show substantial difference in accuracy or recall compared to a non-poisoned model. This is evidenced by the CIFAR-10 experiment in Figure 4.2, in which the source recall of the poisoned model is $\sim 10\%$ lower compared to a non-poisoned model.

Furthermore, we observe that model convergence on both datasets is negatively impacted, as evidenced by the large variances in recall values between consecutive rounds. Consider Table 4.1 where results are compared when either (1) at least one malicious participant is selected for $P_R$ or (2) $P_R$ is made entirely of honest participants. When at least one malicious participant is selected, the final source class recall is, on average, 12.08% lower with the CIFAR-10 dataset and 24.46% lower with the Fashion-MNIST dataset. The utility impact from the label flipping attack is therefore predominantly tied to the number of malicious participants selected in the last few rounds of training.

**Impact of Malicious Participant Availability on Attack Effectiveness**

Given the impact of malicious participation in late training rounds on attack effectiveness, we now introduce a malicious participant availability parameter $\alpha$. By varying $\alpha$ we can simulate the adversary’s ability to control compromised participants’ availability (i.e. ensuring connectivity or power access) at various points in training. Specifically, $\alpha$ represents malicious participants’ availability and therefore likeliness to be selected relative to honest participants. For example, if $\alpha = 0.6$, when selecting each participant $P_i \in P_r$ for round $r$, there is a 0.6 probability that $P_i$ will be one of the malicious participants. Larger $\alpha$ implies higher likeliness of malicious participation. In cases where $k > N \times m\%$, the number of malicious participants in $P_r$ is bounded by $N \times m\%$.

Figure 4.3 reports results for varying values of $\alpha$ in late round poisoning, i.e., malicious participation is limited to rounds $r \geq 75$. Specifically, we are interested in studying those scenarios where an adversary boosts the availability of the malicious participants enough that their selection becomes more likely than the non-malicious participants, hence in Fig-
Figure 4.3: Evaluation of impact from malicious participants’ availability $\alpha$ on source class recall. Results are averaged from 3 runs for each setting.

In Figure 5 we use $\alpha \geq 0.6$. The reported source class recalls in Figure 4.3 are averaged over the last 125 rounds (total 200 rounds minus first 75 rounds) to remove the impact of individual round variability; further, each experiment setting is repeated 3 times and results are averaged. The results show that, when the adversary maintains sufficient representation in the participant pool (i.e. $m \geq 10\%$), manipulating the availability of malicious participants can yield significantly higher impact on the global model utility with source class recall losses in excess of 20\%. On both datasets with $m \geq 10\%$, the negative impact on source class recall is highest with $\alpha = 0.9$, which is followed by $\alpha = 0.8$, $\alpha = 0.7$ and $\alpha = 0.6$, i.e., in decreasing order of malicious participant availability. Thus, in order to mount an impactful attack, it is in the best interests of the adversary to perform the attack with highest malicious participant availability in late rounds. We note that when $k$ is significantly larger than $N \times m\%$, increasing availability ($\alpha$) will be insufficient for meaningfully increasing malicious participant selection in individual training rounds. Therefore, experiments where $m < 10\%$ show little variation despite changes in $\alpha$.

To more acutely demonstrate the impact of $\alpha$, Figure 4.4 reports source class recall by round when $\alpha = 0.6$ and $\alpha = 0.9$ for both the CIFAR-10 and Fashion-MNIST datasets. In both datasets, when malicious participants are available more frequently, the source class recall is effectively shifted lower in the graph, i.e., source class recall values with
Figure 4.4: Source class recall by round when malicious participants’ availability is close to that of honest participants ($\alpha = 0.6$) vs significantly increased ($\alpha = 0.9$). The blue line indicates the round in which attack starts.

$\alpha = 0.9$ are often much smaller than those with $\alpha = 0.6$. We note that the high round-by-round variance in both graphs is due to the probabilistic variability in number of malicious participants in individual training rounds. When fewer malicious participants are selected in one training round relative to the previous round, source recall increases. When more malicious participants are selected in an individual round relative to the previous round, source recall falls.

We further explore and illustrate our last remark with respect to the impact of malicious parties’ participation in consecutive rounds in Figure 4.5. In this figure, the x-axis represents the change in the number of malicious participants participating in consecutive rounds, i.e., $(\# \text{ of malicious } \in P_r) - (\# \text{ of malicious } \in P_{r-1})$. The y-axis represents the change in source class recall between these consecutive rounds, i.e., $(r_{\text{recall}}^\text{src} @ \text{round } r) -$
\( c_{\text{src}}^{\text{recall}} \) @ round \( r - 1 \). The reported results are then averaged across multiple runs of FL and all cases in which each participation difference was observed. The results confirm our intuition that, when \( P_r \) contains more malicious participants than \( P_{r-1} \), there is a substantial drop in source class recall. For large differences (such as +3 or +4), the drop could be as high as 40% or 60%. In contrast, when \( P_r \) contains fewer malicious participants than \( P_{r-1} \), there is a substantial increase in source class recall, which can be as high as 60% or 40% when the difference is -4 or -3. Altogether, this demonstrates the possibility that the DNN could recover significantly even in few rounds of FL training, if a large enough decrease in malicious participation could be achieved.

4.2 Applying Privacy-Preserving Techniques to Federated Learning Systems

As we did in the centralized setting, we now consider the task for applying privacy preserving techniques to FL environments.

4.2.1 Developing Privacy-Preserving Federated Learning Algorithms

The task of generating and deploying privacy protection for any given model training algorithm is non-trivial. Consider differentially private model training. First, a DP version of the algorithm must be developed. Second, this must be written as a series of queries. Finally, each query must have an appropriate aggregation procedure. We are then able to develop model training with accurate, federated, private results.

![Figure 4.6: Decision tree training time in the presence of encryption.](image-url)
Algorithm 3 Private Decision Tree Learning

**Input:** Set of participants $\mathcal{P}$; min # of honest, non-colluding parties $t$; privacy guarantee $\epsilon$; attribute set $\mathcal{F}$; class attribute $C$; max tree depth $d$; public key $pk$

- $\bar{t} = n - t + 1$
- $\epsilon_1 = \frac{\epsilon}{2(d+1)}$

Define current splits, $S = \emptyset$, for root node

$M = \text{BuildTree}(S, \mathcal{P}, t, \epsilon_1, \mathcal{F}, C, d, pk)$

**return** $M$

**procedure** BuildTree($S, \mathcal{P}, t, \epsilon_1, \mathcal{F}, C, d, pk$)

- $f = \max_{F \in \mathcal{F}} |F|$
- Asynchronously query $\mathcal{P}$: $\text{counts}(S, \epsilon_1, t)$
- $N = $ decrypted aggregate of noisy counts

**if** $\mathcal{F} = \emptyset$ or $d = 0$ or $\frac{N_{|C|}}{\epsilon_1} < \sqrt{2}$ **then**
- Asynchronously query $\mathcal{P}$: $\text{class_counts}(S, \epsilon_1, t)$
- $N_c = \text{vector of decrypted, noisy class counts}$
- **return** node labeled with $\arg\max_c N_c$

**else**

- $\epsilon_2 = \frac{\epsilon_1}{2|\mathcal{F}|}$
- **for each** $F \in \mathcal{F}$ **do**

  - **for each** $f_i \in F$ **do**

    - Update set of split values to send to child node: $S_i = S + \{F = f_i\}$
    - Asynchronously query $\mathcal{P}$: $\text{counts}(S_i, \epsilon_2, t)$
    - and $\text{class_counts}(S_i, \epsilon_2, t)$
    - $N_{i,F} = $ aggregate of counts
    - $N_{i,c} = $ element-wise aggregate of class counts
    - Recover $N_{i,F}$ from $\bar{t}$ partial decryptions of $N_{i,F}$
    - Recover $N_{i,c}$ from $\bar{t}$ partial decryptions of $N_{i,c}$

  - **end for**

- $V_F = \sum_{i=1}^{|F|} \sum_{c=1}^{|C|} N_{i,c} \cdot \log \frac{N_{i,F}}{N_i}$

**end for**

- $F = \arg \max_F V_F$

Create root node $M$ with label $\bar{F}$

**for each** $f_i \in F$ **do**

- $S_i = S + \{F = f_i\}$
- $M_i = \text{BuildTree}(S_i, \mathcal{P}, t, \epsilon_1, \mathcal{F} \setminus \bar{F}, C)$
- Set $M_i$ as child of $M$ with edge $f_i$

**end for**

**return** $M$

**end if**
Additionally, when secure multiparty computation is used there may be notable overhead required to coordinate within the chosen cryptographic system. In [221], we leverage the threshold Paillier encryption system from [110] in developing our privacy-preserving federated learning system. We choose this approach rather than a complex SMC protocol run by the parties themselves to provide a streamlined interface between the aggregator and the parties. Parties need only answer data queries with encrypted, noisy responses and decryption queries with partial decryption values. Management of the global model and communication with all other parties falls to the aggregator, therefore decreasing the barrier to entry for parties to engage in our federated learning system.

Figure 4.6 demonstrates the impact of this choice as our approach is able to effectively handle the introduction of more parties into the federated learning system without the introduction of increased encryption overhead. The algorithm used for privacy-preserving federated decision tree learning with encryption from the aggregator perspective is detailed in Algorithm 3. Further details are available in [221]. The results in Figure 4.6 are for experiments using the approach in Algorithm 3 to privately train a decision tree model on the Nursery dataset. The Nursery dataset is publicly available as part of the UCI Machine Learning Repository [11] and contains 8 categorical attributes about 12,960 nursery school applications. The target attribute has five distinct classes with the following distribution: 33.333%, 0.015%, 2.531%, 32.917%, 31.204%.

Another issue in the deployment of new machine learning training algorithms is the choice of algorithmic parameters. Key decisions must be made when developing or using a private FL system and many are domain-specific. While experimental results may broadly inform such decisions through the analysis of trade-offs between privacy, trust and accuracy, but note that the impact will vary depending on the data and the training algorithm chosen. For example, questions surrounding what impact various data-specific features will have on the privacy budget are algorithm-specific. Consider Algorithm 3. Here we demonstrates how, in decision tree training, the number of features and classes impact the
Algorithm 4 Private DNN Learning: Participant $P_i$

**procedure** TRAIN_EPOCH($M, \eta, b, \mathcal{L}, c, \sigma, t$)  

$\theta = \text{parameters of } M$

**for** $j \in \{1, 2, \ldots, \frac{1}{b}\} \text{ do}$

Randomly sample $D_{i,j}$ from $D_i$ w/ probability $b$

**for each** $d \in D_{i,j} \text{ do}$

$g_{j}(d) \leftarrow \nabla \theta \mathcal{L}(\theta, d)$

$\Bar{g}_{j}(d) \leftarrow g_{j}(d) / \max(1, \frac{||g_{j}(d)||}{c})$

**end for**

$\Bar{g}_{j} \leftarrow \frac{1}{|D_{i,j}|} \left( \sum_{d} \Bar{g}_{j}(d) + \mathcal{N} \left( 0, c^2 \cdot \frac{\sigma^2}{t-1} \right) \right)$

$\theta \leftarrow \theta - \eta \Bar{g}_{j}$

$M \leftarrow \theta$

**end for**

**return** $Enc_{pk}(\theta)$

**end procedure**

privacy budget at each level. Similarly, Algorithms 4 and 5 for private DNN and SVM training respectively show the role of norm clipping in neural network and SVM learning.

In neural networks, this value not only impacts noise but will also have a different impact on learning depending on the size of the network and number of features.

**Extending Secure Two-Party Protocols to Multiple Parties**

If a task-specific SMC protocol such as those presented in Sections 3.2.2 and 3.2.3 are used rather than a cryptosystem there may be extensions required to convert two party protocols, such as those previously introduced, to multiple, $> 2$, parties. Let’s consider our approach to private training of linear regression models with two parties from Section 3.2.2. We want to generalize the previous protocol to the case in which the design matrix $\bar{X}$ and the response vector $\bar{y}$ are shared among more than two parties.

Let $A, B$ and $C$ be random matrices such that $C = AB$ and let the party $P_i$ have shares $A_i, B_i, C_i$ of these matrices. Such triples of shares will be called matrix multiplication triples. Given a pre-distributed matrix multiplication triple and two shared matrices $X$ and $Y$, it is possible to compute shares of $Z = XY$ without leaking any information about these values. The idea is for the parties to locally compute shares of $D = X - A$ and
Algorithm 5 Private SVM Learning: Participant $P_i$

procedure TRAIN_EPOCH($M, \eta, K, \mathcal{L}, c, \sigma, t$)
   \( w = \) parameters of $M$
   \textbf{for each} \((x_i, y_i) \in D\) \textbf{do}
   \( x_i \leftarrow x_i / \max\left(1, \frac{||x_i||_2}{c}\right) \)
   \textbf{end for}
   \textbf{for} \(k \in \{1, 2, ..., K\}\) \textbf{do}
   \( g(D) \leftarrow \nabla w \mathcal{L}(w, D) \)
   \( \bar{g} \leftarrow g(D) + N\left(0, \frac{\sigma^2}{t-1}\right) \)
   \( w \leftarrow w - \eta \bar{g} \)
   \( M \leftarrow w \)
   \textbf{end for}
   \textbf{return} \(\operatorname{Enc}_{pk}(w)\)
end procedure

\( E = Y - B \) and then open the values $D$ and $E$. The parties can compute the shares of $Z$ using the fact that

\[
Z = AE + BD + AB + DE = AE + BD + C + DE.
\]

For instance, $P_1$ can output $Z_1 = A_1E + B_1D + C_1 + DE$ and all the other parties $Z_i = A_iE + B_iD + C_i$. The correctness can trivially be verified by inspection. The security of this method follows from the fact that, when $D$ and $E$ are opened, the values of $X$ and $Y$ are masked by the random factors $A$ and $B$ from the matrix multiplication triple (and all other operations are local). For the truncation protocol, one of the parties, let's say $P_1$, learns the value to be truncated $w$ blinded by a factor $r$, the other parties only use their shares of $w$ and $r'$ (which represents the least significant bits of $r$) in order to get their outputs. The remaining protocols can be trivially generalized to the case of multiple parties.
4.2.2 Practical Considerations in Privacy-Preserving FL Systems

Variation in Privacy Policy

Preliminary results such as those discussed for skewed datasets in Section 2.3.2 would indicate that individual participants in a FL are likely to have different privacy risks as vulnerability to membership inference attacks is not uniform across all data. Also when taking into account the privacy policy of the participating institution, the sensitivity of the data held by that institution, and the distribution of their data, one institution may demand a higher degree of privacy than another participant.

In a real world collaborative learning environment, such variance in privacy risk tolerance should be accounted for. For example, not all parties can be or need to be online to participate in the entire work flow of the learning process. Whomever is available at any given time may share information or participate in some subset of protocols, representative of their level of trust in the participating parties or the level of privacy demanded by the kind of input data being used. Enabling such a dynamic collaborative learning environment, where each party may be a part of the learning process when it is convenient and at their own comfort level is a challenging approach but is more realistic for wide deployment of privacy-preserving federated learning solutions in real world applications.

Accounting for Straggler Participants

As discussed in Section 4.1.3, stragglers remain another open problem for practical federated learning. We now present our approach to this issue: the tier-based system for federated learning (TiFL).

The TiFL System for Federated Learning The key idea of a tier-based system is that given the global training time of a round is bounded by the slowest participant selected in that round, selecting participants with similar response latency in each round can significantly reduce the training time.
The overall system architecture of TiFL is present in Figure 4.7. TiFL follows the system design to the state-of-the-art FL system [157] and adds two new components: a tiering module (a profiler & tiering algorithms) and a tier scheduler. These newly added components can be incorporated into the coordinator of the existing FL system [47]. In TiFL, the first step is to collect the latency metrics of all the available participants through a lightweight profiling. The profiled data is further utilized by our tiering algorithm. This groups the participants into separate logical pools called tiers. Once the scheduler has the tiering information (i.e., tiers that the participants belong to and the tiers’ average response latencies), the training process begins. Different from vanilla FL that employs a random participant selection policy, in TiFL the scheduler selects a tier and then randomly selects targeted number of participants from that tier. After the selection of participants, the training proceeds as state-of-the-art FL system does. By design, TiFL is non-intrusive and can be easily plugged into any existing FL system in that the tiering and scheduler module simply regulate participant selection without intervening the underlying training process.
Compatibility of the TiFL System with Privacy-Preserving FL

Consider privacy-preserving FL based on participant level differential privacy, where the privacy guarantee is defined at each individual participant. This can be accomplished by each participant implementing a centralized private learning algorithm as their local training approach. For example, with neural networks this would be one or more epochs using the approach proposed in [168]. This requires each participant to add the appropriate noise into their local learning to protect the privacy of their individual datasets. We demonstrate that TiFL remains compatible with such privacy preserving approaches.

Assume that for participant $c_i$, one round of local training using a differentially private algorithm is $(\epsilon, \delta)$-differentially private, where $\epsilon$ bounds the impact any individual may have on the algorithm’s output and $\delta$ defines the probability that this bound is violated. Smaller $\epsilon$ values therefore signify tighter bounds and a stronger privacy guarantee. Enforcing smaller values of $\epsilon$ requires more noise to be added to the model updates sent by participants to the FL server which leads to less accurate models. Selecting participants at each round of FL has distinct privacy and accuracy implications for participant-level privacy-preserving FL approaches. For simplicity we assume that all participants are adhering to the same privacy budget and therefore same $(\epsilon, \delta)$ values. Let us first consider the scenario wherein $C$ participants are chosen uniformly at random each round. Compared with each participant selected at every round, the overall privacy guarantee, using random sampling amplification [172], improves from $(\epsilon, \delta)$ to $O(q \epsilon, q \delta)$ where $q = \frac{|C|}{|P|}$, where $P$ represents the total number of participants in the FL system. This means that there is a stronger privacy guarantee with the same noise scale. Participants may therefore add less noise per round or more rounds may be conducted without sacrificing privacy. For the tiered approach the guarantee also improves. Compared to $(\epsilon, \delta)$ in the all participant scenario, the tiered approach improves to an $O(q_{max} \epsilon, q_{max} \delta)$ privacy guarantee where the probability of selecting tier with weight $\theta_j$ is given by $\frac{1}{n_{tiers}} \times \theta_j$, $q_{max} = \max_{j=1\ldots|n_{tiers}|} q_j$ and $q_j = \left(\frac{1}{n_{tiers}} \times \theta_j\right) \frac{|C|}{|n_j|}$, where $n_{tiers}$ specifies the number of tiers in the TiFL system and
$n_j$ specifies the number of participants in tier $j$. 
In this chapter we introduce our system for trust and security enhanced customizable private federated learning: TSC-PFed. We first introduce the overall architecture of the TSC-PFed system. We then propose our trust and security enhancements for private federated learning. Finally, we detail our solutions for customizing TSC-PFed making it a realistic solution for a broad range of privacy-preserving settings.

5.1 Components in the Private Federated Learning Architecture

We consider the following scenario for differentially private federated learning. There exists a set of parties \( P = P_1, P_2, \ldots, P_n \), a set of disjoint datasets \( D_1, D_2, \ldots, D_n \) belonging to the respective parties and adhering to the same structure, and an aggregator \( A \). Our system takes as input at least four additional parameters: \( f_M, B, \Phi, \) and \( f_{AGG} \). \( f_M \) specifies the training algorithm. \( B \) is the schedule of privacy parameter values. For example, \( B \) may include the \( \epsilon \) or \((\epsilon, \delta)\) settings which would ensure that a pre-determined global differential privacy guarantee is adhered to. \( \Phi \) specifies the privacy mechanism. For example, in differentially private neural network training, \( \Phi \) would specify the Gaussian mechanism. Finally, \( f_{AGG} \) is the model and step specific aggregation procedure used by \( A \) to combine information received from individual participants in \( P \) so that a single global model \( M \) can be trained.

When TSC-PFed is run, the aggregator \( A \) runs the machine learning algorithm \( f_M \) consisting of \( k \) linear queries \( Q_1, Q_2, \ldots, Q_k \), each requiring information from the \( n \) datasets. This information may include model parameters after some local learning on each dataset or may be more traditionally queried information such as how many individuals match a set of criterion. For each algorithm deployed into TSC-PFed, any step \( s \) requiring such
information reflective of some analysis on the private, local datasets must be represented by a corresponding query $Q_s$. A model agnostic view of the TSC-PFed system is given in Algorithm 6 where any additional parameters are denoted as $\text{PARAMS}$. We note that we assume secure channels between $A$ and each of the parties. At each step $Q_s$ in $f_M$ the aggregator will send query $Q_s$. Each party will then calculate a response using their respective datasets.

**Algorithm 6** Federated Learning in TSC-PFed

**System Parameters:** $f_M$: ML algorithm, $\mathcal{P}$: set of participants of size $n$ with each $P_i \in \mathcal{P}$ holding a private dataset $D_i$, $\mathcal{B}$: schedule of privacy parameters, $\Phi$: privacy mechanism, $f_{\text{AGG}}$: model and step specific aggregation procedure, $\text{PARAMS}$

**procedure** TRAIN\_MODEL

$A$ initializes global model $M$

for each $Q_s \in f_M$ do

$\sigma_s \leftarrow B[s]$  
$R_s \leftarrow \emptyset$

for each $P_i \in \mathcal{P}$ do

$A$ asynchronously queries $P_i$ with $Q_s$

$P_i$ sends $r_{i,s} = \Phi(Q_s(D_i), \sigma_s)$

$A$ adds $r_{i,s}$ to set of responses $R_s$

end for

$A$ computes $r_s \leftarrow f_{\text{AGG}}(R_s, Q_s)$

$A$ updates $M$ according to $r_s$

for each $P_i \in \mathcal{P}$ do

$A$ asynchronously send $P_i$ updated global model $M$

$P_i$ updates local model $M_i = M$

end for

return $M$

end procedure

The participant will use the privacy mechanism $\Phi$ as specified by the privacy parameters in $\mathcal{B}$ allocated to step $s$. These parameters are denoted as $\sigma_s$ in Algorithm 6. The budget allocation is determined by a system level privacy scheduler. Consider a differentially private training approach. In this case, the total budget $\epsilon$ required to meet the desired level of differential privacy is divided at the system level amongst the $k$ queries. Let $\epsilon_s \leq \epsilon$ be the budget allocated to step $s$. Recall the sequential composition property of the differential privacy framework from Section 3.1.3 Definition 4. Applied to TSC-PFed with differential
privacy, this property tells us that releasing the outputs $Q_1, Q_2, \ldots, Q_k$ satisfies $\sum_{s=1}^{k} \epsilon_s$-DP. We note there are more recent proposals for accounting methods such as the moments accountant proposed in [168] which can additionally provide for tighter guarantees. The budget scheduler will then, on input of the current step $s$, return the allocated privacy budget $\epsilon_s$.

The aggregator $\mathcal{A}$ collects the query responses from the participants and then uses the pre-defined $f_{\text{AGG}}$ function to determine an aggregate value for the training algorithm step. The global model $M$ is then updated according to this aggregate value. The global model is distributed to each participant $P_i \in \mathcal{P}$ so that they may update their local model $M_i$. At the conclusion of $f_M$, the model $M$ is exposed to all participants.

Let us consider the specific example of training a differentially private DNN model in TSC-PFed. Recall from Section 3.2.1 where we introduced differentially private DNN model training in the centralized, or one data holder, setting. In TSC-PFed, each participant has a differentially private training module as shown in Figure 5.1. This module would consist of a private training algorithm, such as Algorithm 1 detailed in Section 3.2.1, which takes as input the current privacy budget as well as the local data and returns perturbed

![Figure 5.1: Overview of the workflow for differentially private federated DNN training with TSC-PFed.](image)

178
model updates. These updates are then returned to the aggregator. In the case of deep neural networks these updates can be directly averaged by $\mathcal{A}$ to determine an appropriate update for the global model $M$. The global update is then sent back to the individual participants who update their own local model instances before the next round of training.

Algorithm 7 Differentially Private Federated DNN Training in TSC-PFed

**System Parameters:** $f_M$: DNN federated training algorithm, $\mathcal{P}$: set of participants of size $n$ with each $P_i \in \mathcal{P}$ holding a private dataset $D_i$, $B$: schedule of differential private budgets, $\Phi$: Gaussian mechanism, $f_{AGG}(R_s, Q_s) : 1/n \sum R_s$, loss function $\mathcal{L}(\theta, D_i) = \frac{1}{|D|} \sum_i \mathcal{L}(\theta, x_i)$, learning rate $\eta_t$, batch size $L$, norm bound $C$

**procedure** $\text{TRAIN\_DP\_DNN}$

$\mathcal{A}$ initializes global model $M$

for each $Q_s \in f_M$ do

$\sigma_s \leftarrow B[s]$ \hspace{1cm} \{ $\sigma_s$ corresponds to budget $\epsilon_s$ for step $s$ \}

$R_s \leftarrow \emptyset$

$Q_s \leftarrow \{ \text{Algorithm 1, } L, \eta_t, L, C, \sigma_s, E = 1 \}$

for each $P_i \in \mathcal{P}$ do

$\mathcal{A}$ asynchronously queries $P_i$ with $Q_s$

$P_i$ sends $r_{i,s} = \text{Algorithm 1}(D = D_i, L, \eta_t, \sigma = \sigma_s, L, C, E = 1, N = \Phi)$

$\mathcal{A}$ adds $r_{i,s}$ to set of responses $R_s$

end for

$\mathcal{A}$ computes $r_s \leftarrow f_{AGG}(R_s, Q_s) = 1/n \sum R_s$

$\mathcal{A}$ updates $M$ according to $r_s$

for each $P_i \in \mathcal{P}$ do

$\mathcal{A}$ asynchronously send $P_i$ updated global model $M$

$P_i$ updates local model $M_i = M$

end for

return $M$

end procedure

This process for a federated implementation of the approach proposed by Abadi et al. in [168] is outlined in Algorithm 7 where we see that the TSC-PFed system is in charge at the aggregator level of maintaining the neural network training and therefore maintains the learning parameters. The $\sigma$ value from the Abadi work is retrieved from $B$ at each iteration and passed to the individual participants so that the onus of privacy accounting also falls to the aggregator service. Then, for each round of training each participant is queried to train one epoch on their local dataset. The local training will introduce noise according
to the privacy parameter $\sigma$ and therefore in response to each query participants will return noisy, privatized model parameters. The aggregator then conducts federated averaging and updates the model. After a pre-determined number of rounds or epochs, each represented in TSC-PFed by one of the $k$ queries in $f_M$, a globally trained model $M$ is returned.

We note that we can also easily extend this training to leverage the dynamic schedules from our work in [169]. When noise schedules are used, the values $\sigma_s$ determined at the aggregator level by $B$ at each step $s$ differ from one another according to the chosen schedule rather than remaining consistent across the $k$ queries as in the case of the fixed $\sigma$ approach of Abadi et. al. [168]. In this way TSC-PFed supports multiple training approaches.

We have thus far described the overall infrastructure of the TSC-PFed system. We next introduce our trust and security enhancements.

### 5.2 Trust and Security Enhancements in TSC-PFed

We first propose our approach to trust in federated learning in which we leverage secure multiparty computation to decrease noise and consequently increase the accuracy of any models trained with TSC-PFed without sacrificing formal DP guarantees.

#### 5.2.1 Introducing Trust in Private Federated Learning

Recall from 4.1.2 that a private FL system must address risk of inference during the learning process and risk of inference over the outputs. Our system, TSC-PFed, additionally supports the concept of trust among participants specifically with respect to the risk of collusion between malicious participants. We combine methods from SMC and DP to develop protocols that guarantee privacy without sacrificing accuracy.

Specifically we introduce a new system parameter $t$ which specifies the minimum number of honest, non-colluding parties in the federation. Now, when noise is introduced to adhere to differential privacy guarantees, the scale of the noise will be controlled by the current step’s privacy budget, the step sensitivity, and the system trust level. Additionally,
Figure 5.2: Overview of a step \( s \) in the TSC-PFed system with the trust enhancement.

\( f_{\text{AGG}} \) now consists of two functionalities \( f_{\text{AGG}}^0 \) and \( f_{\text{AGG}}^1 \) for aggregation of encrypted and partially decrypted updates respectively. Figure 5.2 shows an outline of a step \( s \) in \( f_M \) when the trust enhancement is used.

Using secure channels between \( A \) and each of the parties, the aggregator will send query \( Q_s \). Each party will then calculate a response using their respective datasets. The participant will use the privacy mechanism to add the appropriate amount of noise according to the privacy budget allocated to that step, the sensitivity of \( Q_s \), and the level of trust in the system. The noisy response is then encrypted using the threshold variant of the Paillier cryptosystem proposed in [110] and sent to \( A \). Homomorphic properties then allow \( A \) to aggregate the individual responses with \( f_{\text{AGG}}^0 \). \( A \) subsequently queries at least \( \bar{t} = n - t + 1 \) participants to decrypt the aggregate value. These decryptions are themselves aggregated with \( f_{\text{AGG}}^1 \) and used to update the model \( M \). At the conclusion of \( f_M \), the model \( M \) is exposed to all participants. This process is outlined in Algorithm 8.

We consider trust with respect to possible collusion in two steps: (1) in the addition of noise and (2) in the threshold setting of the encryption scheme. The more participants colluding, the more knowledge which is available to infer the data of an honest participant. Therefore, the noise introduced by an honest participant must account for colluion.
Algorithm 8 Federated Learning in TSC-PFed with Trust Enhancement

**System Parameters:** $f_M$: ML algorithm, $\mathcal{P}$: set of participants of size $n$ with each $P_i \in \mathcal{P}$ holding a private dataset $D_i$ and a secret key $sk_i$, $\mathcal{B}$: schedule of privacy parameters, $\Phi$: privacy mechanism, $f_{\text{AGG}} = \{f_0^{\text{AGG}}, f_1^{\text{AGG}}\}$ aggregation procedures for encrypted values and partial decryptions respectively, $pk$: public key, $t$: minimum number of honest, non-colluding parties, $\text{PARAMS}$

**procedure** TRAIN_MODEL

$A$ initializes global model $M$

$t = n - t + 1$

for each $Q_s \in f_M$

$\sigma_s \leftarrow B[s]$

$R_s \leftarrow \emptyset$

for each $P_i \in \mathcal{P}$

$A$ asynchronously queries $P_i$ with $Q_s$

$P_i$ sends $r_{i,s} = \text{Enc}_{pk}(Q_s(D_i), \sigma_s, t)$

$A$ adds $r_{i,s}$ to set of responses $R_s$

end for

$A$ computes $r_s' \leftarrow f_0^{\text{AGG}}(R_s, Q_s)$

$A$ selects $\mathcal{P}_{\text{dec}} \subseteq \mathcal{P}$ such that $|\mathcal{P}_{\text{dec}}| \geq t$

$R_s' \leftarrow \emptyset$

for each $P_i \in \mathcal{P}_{\text{dec}}$

$A$ asynchronously queries $P_i$ with $r_s'$

$A$ receives $r_{i,s}' = \text{Dec}_{sk_i}(r_s')$, a partial decryption of $r_s'$, from $P_i$

$A$ adds $r_{i,s}'$ to the set of partial decryptions $R_s'$

end for

$A$ computes $r_s = f_1^{\text{AGG}}(R_s', Q_s)$

$A$ updates $M$ according to $r_s$

for each $P_i \in \mathcal{P}$

$A$ asynchronously send $P_i$ updated global model $M$

$P_i$ updates local model $M_i = M$

end for

return $M$

end procedure
our implementation of trust enhanced private federated learning, our use of the additively homomorphic threshold Paillier encryption scheme allows us to account for the possibility of such collusion.

Reducing Noise with SMC

The key benefit of introducing the trust parameter $t$ into our TSC-PFed system is the ability to reduce the noise introduced by each individual participant by leveraging the SMC framework.

Specifically, let a step $s$, corresponding to a query $Q_s$ in $f_M$ with sensitivity $S_s$, be allocated a budget $\epsilon_s$. Let $\sigma_s$ be the noise parameter which guarantees $\epsilon_s$-differential privacy. In a traditional application of differential privacy to federated learning, each party will use the Gaussian mechanism to add $N(0, S_s^2 \sigma_s^2)$ noise to their response $r_{i,s}$ when queried with $Q_s$ by $A$ at step $s$. This guarantees the privacy of each $r_{i,s}$.

If, however, each $r_{i,s}$ is encrypted using the scheme proposed in [110] with a threshold setting of $\tilde{t} = n - t + 1$ and public key $pk$, the noise may be reduced by a factor of $t - 1$. Rather than returning $Q_s(D_i) + N(0, S_s^2 \sigma_s^2)$, each party may return $\text{Enc}_{pk}(Q_s(D_i) + N(0, S_s^2 \sigma_s^2/t^{t-1}))$.

Note that when $A$ aggregates these responses the value that is eventually decrypted and exposed will be $\sum_{i=1}^{n} Q_s(D_i) + Y_i$ where each $Y_i$ is drawn from the Gaussian distribution with standard deviation $S_s \sigma_s \sqrt{t^{-1}}$. This is equivalent to $N(0, S_s^2 \sigma_s^2/t^{t-1}) \sum_{i=1}^{n} Q_s(D_i)$. Since we know that $t - 1 < n$, the noise included in the decrypted value is strictly greater than that required to satisfy DP. Additionally, the encryption scheme guarantees that the maximum number of colluders, $\tilde{t}$, cannot decrypt values of honest parties.

Given this approach, we are able to maintain the customizable nature of our system with the trust parameter $t$ and the formal privacy guarantees of the DP framework while decreasing the amount of noise for each query response leading to more accurate ML models.
Experimental Evaluation

We now empirically demonstrate how to apply our trust enhancement to train three distinct learning models with differential privacy: decision trees (DT), deep neural networks (DNN) and linear Support Vector Machines (SVM). We additionally provide analysis on the impact of certain settings on performance.

Decision Trees  We first consider DT learning using the ID3 algorithm. In this scenario, each dataset $D_i$ owned by some $P_i \in \mathcal{P}$ contains a set of instances described by the same set of categorical features $\mathcal{F}$ and a class attribute $C$. The aggregator initializes the DT model with a root node. Then, the feature $F \in \mathcal{F}$ that maximizes information gain is chosen based on counts queried from each party in $\mathcal{P}$ and child nodes are generated for each possible value of $F$. The feature $F$ is then removed from $\mathcal{F}$. This process continues recursively for each child node until either (a) there are no more features in $\mathcal{F}$, (b) a pre-determined max-depth is reached, or (c) responses are too noisy to be deemed meaningful. This process is consistent with the algorithmic pseudocode in Algorithm 3 in Section 4.2.1.

There are two types of participant queries in private, federated DT learning: \textit{counts} and \textit{class_counts}. For executing these queries $A$ first divides the entire privacy budget $\epsilon$ equally between each layer of the tree. According to the composition property of differential privacy, because different nodes within the same layer are evaluated on disjoint subsets of the datasets, they do not accumulate privacy loss and therefore the budget allocated to a single layer is not divided further. Within each node, half of the budget ($\epsilon_1$) is allocated to determining total counts and half is allocated to either class counts (done at the leaf nodes) or evaluating attributes (done at internal nodes). For internal nodes, each feature is evaluated for potential splitting against the same dataset. The budget allocated to evaluating attributes must therefore be divided amongst each feature ($\epsilon_2$). In all experiments the max depth is set to $d = \frac{|\mathcal{F}|}{2}$.

We conduct a number of experiments using the Nursery dataset from the UCI Machine
Learning Repository [11]. Recall from Section 4.2.1 that this dataset contains 8 categorical attributes about 12,960 nursery school applications and has a target attribute with five distinct classes with an uneven distribution: 33.333%, 0.015%, 2.531%, 32.917%, 31.204%.

To put model performance into context, we compare with two different random baselines and two alternative FL approaches. Random baselines enable us to characterize when a particular approach is no longer learning meaningful information while the FL approaches visualize relative performance cost.

**Uniform Guess.** In this approach, class predictions are randomly sampled with a \( \frac{1}{|C|} \) chance for each class. **Random Guess.** Random guess improves upon Uniform Guess with consideration of class value distribution in the training data. At test time, each prediction is sampled from the set of training class labels. **Local DP.** In the local approach, parties add noise to protect the privacy of their own data in isolation. The amount of noise necessary to provide \( \epsilon \)-differential privacy to each dataset is defined in [91]. **No Privacy.** This is the result of running the federated learning algorithm without any privacy guarantee. We now look at how different settings impact results.

**Privacy Budget.** We first look at the impact of the privacy budget on performance in our system. To isolate the impact of the privacy budget we set the number of parties, \( |P| \), to 10 and assume no collusion. We consider budget values between 0.05 and 2.0. Recall that for a mechanism to be \( \epsilon \)-differentially private the amount of noise added will be inversely proportional to value of \( \epsilon \). In other words, the smaller the \( \epsilon \) value, the smaller the privacy budget, and the more noise added to each query.

We can see in Figure 5.3 that our approach maintains an F1-score above 0.8 for privacy budgets as small as 0.4. Once the budget dips below 0.4 we see the noise begins to overwhelm the information being provided which can have one of two outcomes: (1) learning pre-maturely halts or (2) learning become inaccurate. This results in degraded performance as the budget decreases, which is expected. It is clear that our approach maintains improved performance over the local DP approach for all budgets (until both approaches
Number of Parties. Another important consideration for FL systems is the ability to maintain accuracy in highly distributed scenarios. That is, when many parties, each with a small amount of data, such as in an IoT scenario, are contributing to the learning.

In Figures 5.4 we show the impact that $|P|$ has on performance. The results are for a fixed overall privacy budget of 0.5 and assume no collusion. For each experiment, the overall dataset was divided into $|P|$ equal sized partitions.

The results in Figure 5.4 demonstrate the viability of our system for FL in highly distributed environments while highlighting the shortcomings of the local DP approach. As $|P|$ increases, the noise in the local DP approach increases proportionally while our approach maintains consistent accuracy. We can see that with as few as 25 parties, the local DP results begin to approach the baseline and even dip below random guessing by 100 participants.

Another consideration relative to the scaling of participants is the overhead of encryption. Recall from Figure 4.6 in Section 4.2.1. These results highlight the scalability of using encryption in our system, showing the impact that encryption has on overall training.
time in our system as the number of parties increases from 1 to 10. While the entire system experiences a steady increase in cost as the number of participants increases, the impact of the encryption remains consistent as the interactions with the aggregator are done in parallel.

**Trust.** An important part of our system is the trust parameter. While the definition of a neighboring database within the context of the differential privacy framework considers privacy at the record level, the trust model for adversarial knowledge is considered within the context of the entire system. The trust parameter therefore represents the degree of adversarial knowledge by capturing the maximum number of colluding parties which the system may tolerate. Figure 5.5 demonstrates how the $\epsilon$ values used for both count and distribution queries in private, federated DT learning are impacted by the trust parameter setting when $|\mathcal{P}| = 50$.

In the worst case scenario where a party $P_i \in \mathcal{P}$ assumes that all other $P_j \in \mathcal{P}, i \neq j$ are colluding, our approach converges with existing local DP approaches. In all other scenarios the query $\epsilon$ values will be increased in our system leading to more accurate outcomes. Additionally, we believe the aforementioned scenario of no trust is unlikely to exist in real world instances. Let us consider smart phone users as an IoT example. Collusion of all but
one party is impractical not only due to scale but also since such a system is likely the be running without many users even knowing. Additionally, on a smaller scale, if there is a set of five parties in the system and one party is concerned that the other four are all colluding, there is no reason for the honest party to continue to participate. We therefore believe that realistic scenarios of FL will see accuracy gains when deploying the trust enhancement.

**Deep Neural Networks** We additionally demonstrate how to use our trust enhancement in the federated training of a differentially private DNN. In our approach, similarly to centrally trained DNNs, each party is sent a model with the same initial structure and randomly initialized parameters. Each party will then conduct one full epoch of learning locally. At the conclusion of each batch, Gaussian noise is introduced according to the norm clipping value $c$ and the privacy parameter $\sigma$. Recall that norm clipping allows us to put a bound on the sensitivity of the gradient update. We use the same privacy strategy used in the centralized training approach presented in [168] and outlined both in Algorithm 1 in Section 3.2.1 and applied to TSC-PFed in Algorithm 7 in Section 5.1. Once an entire epoch has com-
pleted the final parameters are sent back to $A$. $A$ then averages the parameters and sends back an updated model for another epoch of learning. After a pre-determined number of epochs, the final model $M$ is output.

Within our private, federated NN learning system, if $\sigma = \sqrt{2 \cdot \log \frac{bE}{\delta}} / \epsilon$ then by [105] our approach is $(\epsilon, \delta)$-differentially private with respect to each randomly sampled batch. Using the moments accountant in [168], our approach is $(\tilde{O}(b\epsilon \sqrt{E/b}), \delta)$-DP overall.

For our DNN experiments we use the publicly available MNIST dataset. Recall from Section 2.2.1 that this includes 60,000 training instances of handwritten digits and 10,000 testing instances. Each example is a 28x28 grey-scale image of a digit between 0 and 9 [222]. We use a model structure similar to that in [168]. Our model is a feedforward neural network with 2 internal layers of ReLU units and a softmax layer of 10 classes with cross-entropy loss. The first layer contains 60 units and the second layer contains 1000. We set the norm clipping to 4.0, learning rate to 0.1 and batch size of $0.01 |D_i|$. We use Keras with a Tensorflow backend.

We compare our results with the following baselines: **Central Data Holder, No Privacy.** In this approach all the data is centrally held by one party and no privacy is considered in the learning process. **Central Data Holder, With Privacy.** While all the data is still centrally held by one entity, this data holder now conducts privacy-preserving learning according to Algorithm 1. This is representative of the scenario in [168]. **Distributed, No Privacy.** In this approach the data is distributed to multiple parties, but the parties do not add noise during the learning process. **Local DP.** Parties add noise to protect the privacy of their own data in isolation, adapting from [168] and [156].

Figure 5.6 shows results with 10 parties conducting 100 epochs of training with the privacy parameter $\sigma$ set to 8.0, the “large noise” setting in [168] and equivalent to an $(\epsilon, \delta)$ guarantee of $(0.5, 10^{-5})$. Note that Central Data Holder, No Privacy and Distributed Data, No Privacy achieve similar results and thus overlap. Our model is able to achieve an F1-score in this setting of 0.9. While this is lower than the central data holder setting where
an F1-score of approximately 0.95 is achieved, however our approach again significantly out-performs the local approach which only reaches 0.723. Additionally, we see a drop off in the performance of the local approach early on as updates become overwhelmed by noise.

![Graph showing F1 score vs epoch](image)

Figure 5.6: Differentially private DNN training with MNIST data (10 parties and $\sigma = 8$, $(\epsilon, \delta) = (0.5, 10^{-5})$).

We additionally experiment with $\sigma = 4$ and $\sigma = 2$ as was done in [168]. When $\sigma = 4$ ($(\epsilon, \delta) = (2, 10^{-5})$) the central data holder with privacy is able to reach an F1 score of 0.960, the local approach reaches 0.864, and our approach results in an F1-score of 0.957. When $\sigma = 2$ ($(\epsilon, \delta) = (8, 10^{-5})$) those scores become 0.973, 0.937, and 0.963 respectively. We can see that our approach therefore demonstrates the most gain with larger $\sigma$ values which translates to tighter privacy guarantees.

Figure 5.7 again shows how the standard deviation of noise is significantly decreased in our system for most scenarios.

Our experiments demonstrate that the encryption time for one parameter at a party $P_i$ takes approximately 0.001095 sec while decryption between $A$ and $P_{dec}$ takes 0.007112 sec. While each parameter requires encryption and decryption, these processes can be done completely in parallel. Therefore, as previously discussed with the decision tree experiments, overhead remains relatively constant as both $|P|$ and the number of parameters
increase.

**Support Vector Machines (SVM)** We also demonstrate and assess our trust enhancement when solving a classic $\ell_2$-regularized binary linear SVM problem with hinge loss.

To train the linear SVM in a private distributed fashion, the aggregator distributes a model with the same weight vector $w$ to all parties. Each party then runs a predefined number of epochs to learn locally. To apply differential privacy in this setting, we first perform norm clipping on the feature vector $x$ to obtain a bound on the sensitivity of the gradient update. Then, Gaussian noise is added to the gradient according to [223]. After each party completes its local training, the final noisy encrypted weights are sent back to the aggregator. The aggregator averages the encrypted weights and sends back an updated model with a new weight vector for another epoch of learning. Training ends after a predefined number of epochs.

We use the publicly available ‘gisette’ dataset, which was used for NIPS 2003 Feature Selection Challenge [224]. This dataset has 6,000 training and 1,000 testing samples with 5,000 features. We again contrast the performance of our approach against other ways to train the model. **Central, No Privacy.** Centralized training without privacy. **Central DP.** Centralized training with DP. **Distributed, No Privacy.** Model trained through federated learning without privacy. **Local DP.** In this approach each party adds enough noise.
independently to protect their data according to [91].

Figure 5.8: Linear SVM training with differential privacy (10 parties and \(( \epsilon, \delta ) = (5, 0.0059) \)).

In these experiments, the learning rate was set to 0.01 for all settings. We used 100 epochs for all approaches. Additionally, for FL methods, each party runs 10 epochs locally. We used 10 non-colluding parties. Using \( \sigma = 4 \), we report findings to achieve \(( \epsilon, \delta ) = (5, 0.0059) \) according to [225].

Figure 5.8 shows F1-scores for the evaluated training methods. Central, No Privacy and Distributed, No Privacy perform similarly with F1-scores around 0.99 after fewer than 10 epochs due to the balanced nature of the dataset. Among the privacy-preserving approaches, Central DP introduces the least noise and achieves the highest F1-score. Among private FL methods, the TSC-PFed system using the trust enhancement achieves an F1-score over 0.87 which is almost equal to Central DP and significantly out-performs Local DP after 100 epochs.

We also evaluated our system in a lower trust setting with only half of the parties trusted as non-colluding. Our approach again out-performed Local DP. Specifically, after 100 epochs, our approach reached an F1-score of 0.85, while the Local DP achieves only 0.75.

These experimental results show that our approach consistently out-performs state of the art methods to train different ML models in a private FL fashion. We similarly showed
that our approach consistently out-performs baselines such as random guessing while remaining reasonably close to non-private settings.

**System Implementation**

The development and deployment of new machine learning training algorithms to the TSC-PFed system when using the trust enhancement requires the training process be first broken down into a set of queries in which meaningful aggregation may be done *via summation*. Each query must then be analyzed for its privacy impact and and designated a portion of the overall privacy budget. Additionally, support must exist at each party for each type of query required by the private training algorithm. We will now provide implementation details for each of the model types evaluated, Decision Trees, Neural Networks, and Support Vector Machines, with additional discussion on the applicability of our framework to machine learning algorithms for other model types.

**Application to Private Decision Tree Training**

DT learning follows these steps: (1) determine the feature which best splits the training data, (2) split the training data into subsets according to the chosen feature, (3) repeat steps (1) and (2) for each subset. This is repeated until the subsets have reached a pre-determined level of uniformity with respect to the target variable.

To conduct private decision tree learning in our system we first address step (1): determining the best feature on which to split the data. We define the “best” feature as the feature which maximizes information gain. This is the same metric used in the ID3 [226], C4.5 [227] and C5.0 [228] tree training algorithms. Information gain for a candidate feature $f$ quantifies the difference between the entropy of the current data with the weighted sum of the entropy values for each of the data subsets which would be generated if $f$ were to be chosen as the splitting feature. Entropy for a dataset (or subset) $D$ is computed via
the following equation:

\[
\text{Entropy}(D) = \sum_{i=1}^{\lvert C \rvert} p_i \log_2 p_i
\]

(5.1)

where \( C \) is the set of potential class values and \( p_i \) indicates the probability that a random instance in \( D \) is of class \( i \). Therefore, the selection of the “best” feature on which to switch can be chosen via determining class probabilities which in turn may be computed via counts. Queries to the parties from the aggregator are therefore counts and class counts, known to have a sensitivity of 1.

Given the ability to query parties for class counts the aggregator may then control the iterative learning process. To ensure this process is differentially private according to a pre-defined privacy budget, we follow the approach from [92] to divide the budget for each iteration and set a fixed number of iterations rather than a purity test as a stopping condition. The algorithm will also stop if counts appear too small relative to the degree of noise to provide meaningful information. The resulting private algorithm deployed in our system is detailed in Algorithm 3 in Section 4.2.1.

**Application to Private Neural Network Training**  The process of deploying our system for neural network learning is distinct from the process outlined in the previous section for decision tree learning. In central neural network training, after a randomly initialized model of pre-defined structure is created, the following process is used: (1) the dataset \( D \) is shuffled and then equally divided into batches, (2) each batch is passed through the model iteratively, (3) a loss function \( L \) is used to compute the error of the model on each batch, (4) errors are then propagated back through the network where an optimizer such as Stochastic Gradient Descent (SGD) is used to update network weights before processing the next batch. Steps (1) through (4) constitute one epoch of learning and are repeated until the model converges (stops demonstrating improved performance).

In our system we equate one query to the participants as one epoch of local learning. That is, each party conducts steps (1) through (4) for one iteration and then sends an updated
model to the aggregator. The aggregator then averages the new model weights provided by each party. An updated model is then sent along with a new query for another epoch of learning to each party.

Each epoch receives the noise parameter $\sigma$ and cost to the overall privacy budget is determined through a separate privacy accountant utility. Just as the decision tree stopping condition was replaced with a pre-set depth the neural network stopping condition of convergence is replaced with a pre-defined number of epochs.

At each participant we deploy code to support the process detailed in Algorithm 1 from Section 3.2.1. To conduct a complete epoch of learning we follow the approach proposed in [168] for private centralized neural network learning. Note that this algorithm itself requires a number of changes to the traditional learning approach. Rather than shuffling the dataset into equal sized batches, a batch is randomly sampled for processing. An epoch then becomes defined as the number of batch iterations required to process $|D_i|$ instances. Additionally, parameter updates determined through the loss function $\mathcal{L}$ are clipped to define the sensitivity of the neural network learning to individual training instances. Noise is then added to the weight updates. Once an entire epoch is completed the updated weights can be sent back to the aggregator.

**Application to Private Support Vector Machine Training** Finally, we focus on the classic $\ell_2$-regularized binary linear SVM problem with hinge loss, which is given in the following form:

$$
\mathcal{L}(w) := \frac{1}{|D|} \sum_{(x_i, y_i) \in D} \max\{0, 1 - y_i \langle w, x_i \rangle\} + \lambda \|w\|^2_2,
$$

(5.2)

where $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ is a feature vector, class label pair, $w \in \mathbb{R}^d$ is the model weight vector, and $\lambda$ is the regularized coefficient.

From the aggregator perspective the process of SVM training is similar to that of neural network training. However, where each query to the participants with neural networks was
defined as 1 epoch of training, the SVM model training was implemented with an additional parameter $K$ such that each query to the participants is for $K$ epochs of training. Once query responses are received, model parameters are averaged to generate a new support vector machine model. This new model is then sent to the participants for another $K$ epochs of training. We again specify a pre-determined number of epochs to control the number of training iterations.

To complete an epoch of learning at each participant, we iterate through each instance in the local training dataset $D_i$. We again deploy a clipping approach to constrain the sensitivity of the updates. The model parameters are then updated according to the loss function $\mathcal{L}$ as well as the noise parameter. The process conducted at each participant for $K$ epochs of training in response to an aggregator query.

### 5.2.2 Defending Against Label Flipping Attacks in TSC-PFed

We next introduce our security enhancements for federated learning in TSC-PFed. Given the highly effective model poisoning adversary discussed in Section 2.4.2, how can a FL system defend against label flipping attacks? To that end, we propose a defense in [80] which enables the aggregator to identify malicious participants which we outline in Algorithm 9.

After identifying malicious participants, the aggregator may blacklist them or ignore their updates in future rounds. We showed in Sections 4.1.3 and 4.1.3 that high-utility model convergence can be eventually achieved after eliminating malicious participation. The feasibility of such a recovery from early round attacks supports use of the proposed identification approach as a defense strategy.

Our defense is based on the following insight: The parameter updates sent from malicious participants have unique characteristics compared to honest participants’ updates for a subset of the parameter space. However, since DNNs have many parameters (i.e., updates are extremely high dimensional), it is non-trivial to analyze parameter updates by
Algorithm 9 Identifying Malicious Model Updates in FL

procedure EVALUATE_UPDATES($\mathcal{R}$ : set of vulnerable train rounds, $\mathcal{P}$ : participant set)

$\mathcal{U} = \emptyset$

for $r \in \mathcal{R}$ do

$\mathcal{P}_r \leftarrow$ participants $\in \mathcal{P}$ queried in training round $r$

$\theta_{r-1} \leftarrow$ global model parameters after training round $r - 1$

for $P_i \in \mathcal{P}_r$ do

$\theta_{r,i} \leftarrow$ updated parameters after $\text{TRAINDNN}(\theta_{r-1}, D_i)$

$\theta_{\Delta,i} \leftarrow \theta_{r,i} - \theta_r$

$\theta^\text{src}_{\Delta,i} \leftarrow$ parameters $\in \theta_{\Delta,i}$ connected to source class output node

Add $\theta^\text{src}_{\Delta,i}$ to $\mathcal{U}$

end for

end for

$\mathcal{U}' \leftarrow \text{STANDARDIZE}(\mathcal{U})$

$\mathcal{U}'' \leftarrow \text{PCA}(\mathcal{U}', \text{components}=2)$

$\text{PLOT}(\mathcal{U}'')$

end procedure

hand. Thus, we propose an automated strategy for identifying the relevant parameter subset and for studying participant updates using dimensionality reduction (PCA).

Psuedocode for our defense strategy is given in Algorithm 9. Let $\mathcal{R}$ denote the set of vulnerable FL training rounds and $c_{\text{src}}$ be the class that is suspected to be the source class of a poisoning attack. We note that if $c_{\text{src}}$ is unknown, the aggregator can defend against potential attacks such that $c_{\text{src}} = c \forall c \in \mathcal{C}$. We also note that for a given $c_{\text{src}}$, Algorithm 9 considers label flipping for all possible $c_{\text{target}}$. An aggregator therefore will conduct $|\mathcal{C}|$ independent iterations of Algorithm 9, which can be conducted in parallel. For each round $r \in \mathcal{R}$ and participant $P_i \in \mathcal{P}_r$, the aggregator computes the delta in participant’s model update compared to the global model, i.e., $\theta_{\Delta,i} \leftarrow \theta_{r,i} - \theta_r$. Recall from Section 4.1.1 that a predicted probability for any given class $c$ is computed by a specific node $n_c$ in the final layer DNN architecture. Given the aggregator’s goal of defending against the label flipping attack from $c_{\text{src}}$, only the subset of the parameters in $\theta_{\Delta,i}$ corresponding to $n_{c_{\text{src}}}$ is extracted. The outcome of the extraction is denoted by $\theta_{\Delta,i}^\text{src}$ and added to a global list $\mathcal{U}$ built by the aggregator. After $\mathcal{U}$ is constructed across multiple rounds and participant deltas, it is standardized by removing the mean and scaling to unit variance. The standardized list $\mathcal{U}'$ is
Figure 5.9: PCA plots with 2 components demonstrating the ability of Algorithm 9 to identify updates originating from a malicious versus honest participant. Blue Xs represent gradients from malicious participants while yellow Os represent gradients from honest participants.

fed into Principal Component Analysis (PCA), which is a popular ML technique used for dimensionality reduction and pattern visualization. For ease of visualization, we use and plot results with two dimensions (two components).

In Figure 5.9, we show the results of Algorithm 9 on CIFAR-10 and Fashion-MNIST across varying malicious participation rate \( m \), with \( \mathcal{R} = [10, 200] \). Recall from Section 2.1.1 that the CIFAR-10 dataset is a 10-class image classification problem with color images and from Section 2.4.2 that the Fashion-MNIST dataset contains black and white images also for 10 distinct classes. Even in scenarios with low \( m \), as is shown in Figures 5.9a and 5.9e, our defense is capable of differentiating between malicious and honest participants. In all graphs, the PCA outcome shows that malicious participants’ updates belong to a visibly different cluster compared to honest participants’ updates which form their own cluster. Another interesting observation is that our defense does not suffer from the “gradient drift” problem. Gradient drift is a potential challenge in designing a robust defense, since changes in model updates may be caused both by actual DNN learning and convergence (which is desirable) or malicious poisoning attempt (which our defense is trying to identify and prevent). Our results show that, even though the defense is tested with
a long period of rounds (190 training rounds since \( R = [10, 200] \)), it remains capable of separating malicious and honest participants, demonstrating its robustness to gradient drift.

The aggregator \( A \) in our TSC-PFed system can therefore effectively identify malicious participants, and consequently restrict their participation in mobile training, by conducting such gradient clustering prior to aggregating parameter updates at each round. Clustering model gradients for malicious participant identification presents a strong defense as it does not require access to any public validation dataset, as is required in [229], which is not necessarily possible to acquire.

### 5.3 Customizable Privacy in TSC-PFed

Finally, we propose customizable privacy solutions so that users of our TSC-PFed system may determine what type of privacy protection is appropriate for their needs.

#### 5.3.1 Customizing the Type of Privacy Protection

The first avenue of customization in TSC-PFed is the type of privacy protection. We have up until now exclusively discussed traditional differential privacy. Recall from Section 3.1.3 that the definition of differential privacy considers an entire entry in the database when formalizing privacy protection. In machine learning contexts this represents a complete row which may be, for example, a patient at a hospital. Recently however we have seen a boom of interest in a different flavor of differential privacy which considers single values rather than complete data entries. In the case of the patient this would represent protecting the confidentiality of a single sensitive attribute such as their cancer status. This flavor of differential privacy is broadly referred to as local differential privacy (LDP).

In our system TSC-PFed we propose adapting local differential privacy mechanisms for the federated training of predictive machine learning models. We adapt LDP mechanisms to perturb the high dimensional continuous values contained in model parameter updates. We additionally introduce parameter selection and filtering support into TSC-PFed to bal-
Local Differential Privacy Module

To combat inference attacks against shared data values, companies including Google, Apple, and Microsoft employ local differential privacy (LDP) ([230, 231, 232]). Rather than uploading raw data values, users in an LDP system perturb their data \( v \) using an algorithm \( \Phi \) and instead upload \( \Phi(v) \). This perturbed value \( \Phi(v) \) is then guaranteed to protect \( v \) from inference attacks according to a privacy parameter \( \epsilon \) where a lower \( \epsilon \) value indicates a higher level of privacy. This guarantee is formalized as follows.

**Definition 10.** \((\epsilon\text{-LDP})\). A randomized algorithm \( \Phi \) satisfies \(\epsilon\)-local differential privacy \((\epsilon\text{-LDP})\), where \( \epsilon > 0 \), if and only if for any inputs \( v_1, v_2 \) in universe \( U \), we have:

\[
\forall y \in \text{Range}(\Phi) : \frac{Pr[\Phi(v_1) = y]}{Pr[\Phi(v_2) = y]} \leq e^{\epsilon}
\]

where \( \text{Range}(\Phi) \) is the set of all possible outputs of algorithm \( \Phi \).

Condensed Local Differential Privacy  In [233], we propose a specialization of LDP, Condensed Local Differential Privacy (CLDP). CLDP ensures privacy according to a privacy parameter \( \alpha \). CLDP, however, also considers a distance metric \( d \) in its perturbation algorithm \( \Phi \). Specifically, let \( d : U \times U \rightarrow [0, \infty) \) be a distance function that measures the distance between any two items \( v_1, v_2 \in U \). CLDP is then formalized as follows.

**Definition 11.** \((\alpha\text{-CLDP})\). A randomized algorithm \( \Phi \) satisfies \(\alpha\)-condensed local differential privacy \((\alpha\text{-CLDP})\), where \( \alpha > 0 \), if and only if for any inputs \( v_1, v_2 \in U \):

\[
\forall y \in \text{Range}(\Phi) : \frac{Pr[\Phi(v_1) = y]}{Pr[\Phi(v_2) = y]} \leq e^{\alpha \cdot d(v_1, v_2)}
\]

where \( \text{Range}(\Phi) \) is the set of all possible outputs of algorithm \( \Phi \).
As $d$ increases, $\alpha$ must decrease to maintain a consistent privacy guarantee, making $\alpha \ll \epsilon$. Previous work in [233] provides details for converting $\epsilon$ to $\alpha$ from an adversarial perspective.

To guarantee $\alpha$-CLDP in practice, the Exponential Mechanism (EM) is applied to a raw user value $v$ as follows. Let $v \in \mathcal{U}$ be the raw user data, and let the Exponential Mechanism $\Phi_{EM}$ take as input $v$ and output a perturbed value in $\mathcal{U}$, i.e., $\Phi_{EM} : \mathcal{U} \rightarrow \mathcal{U}$. Then, $\Phi_{EM}$ that produces output $y$ with the following probability satisfies $\alpha$-CLDP:

$$\forall y \in \mathcal{U} : Pr[\Phi_{EM}(v) = y] = \frac{e^{-\alpha d(v, y)}}{\sum_{z \in \mathcal{U}} e^{-\alpha d(v, z)}}$$

Let’s consider how to apply LDP to a model parameter. First updates must be discretized as LDP mechanisms are designed for the one-time collection of discrete, raw data values. The LDP mechanism can then be applied to the discretized updates and the noisy model update is sent to the aggregator. Specifically we will accomplish this by introducing two new parameters: $c$ and $\rho$ into the LDP module. The $c$ parameter clips the values within a specified range while the $\rho$ parameter dictates how many digits after the decimal point that will be preserved.

For each participant, the LDP Module takes as input the high dimensional vector of model parameter updates and outputs a vector containing the privatized perturbed updates according to the participant’s chosen privacy budget. We set the default privacy definition in to be $\alpha$-CLDP-Fed, a variation of $\alpha$-CLDP. While the definition of $\alpha$-CLDP in [233] is provided for LDP perturbation on single integer values in finite spaces, parameter updates are instead real values with high precision (10s after decimal points). Therefore the $\alpha$-CLDP-Fed Module introduces a precision parameter $\rho$ and a clipping range parameter $c$. Each parameter update is then converted to an integer in the range $[-c \cdot 10^\rho, c \cdot 10^\rho]$. By transforming the parameter updates into integers, we can define the $\alpha$-CLDP-Fed system with $\Phi_{EM}$. Additionally, privacy cost is accumulated each time a parameter is updated by
a participant. Therefore, the iterative nature of model training must be considered in the privacy accounting with the LDP customization.

**Applying LDP in TSC-PFed**  As we saw previously in TSC-PFed, participants perform local training on their own private data and only share parameter updates to the server. When the LDP customization is leveraged, we additionally see that, prior to uploading updated parameter values to the server, updates are privatized locally according to each participant’s LDP module $\mathcal{K}$.

Let’s consider neural network model training in TSC-PFed with the LDP customization. In TSC-PFed this will look similar to Algorithm 7 with traditional DP with 3 unique changes.

First, prior to training in TSC-PFed with LDP, the aggregator directs each participant to initialize its local model with the same randomly generated parameters used to initialize the global model $M$ as well as to initialize its local $\mathcal{K}$ module. $A$ informs each $\mathcal{K}$ module initialization with relevant parameters such as the number of training rounds $E$ and the LDP mechanism $\Phi$. However, the privacy budget is specified at runtime. In Algorithms 10 and 11 this budget is denoted as $\epsilon$ but may also refer to $\alpha$ in $\alpha$-CLDP. Three parameters provided to each $\mathcal{K}$ module unique to TSC-PFed with LDP are $c$, $\rho$, and $\Psi$. In TSC-PFed, $c$ and $\rho$ are used to adapt LDP protocols designed for the one-time collection of discrete data values, while $\Psi$ is used to filter consequently decrease the dimensionality of the update. Therefore when using LDP in the TSC-PFed system, each parameter is not updated in response to each query from $A$. In our implementation of TSC-PFed with LDP we set the default behavior $\Psi$ to upload parameters associated with a single DNN model layer at any given round. The first rounds will update the output layer while subsequent rounds will move backward in the network. Larger layers are selected for updating in more rounds than smaller layers. In this way parameter privacy budgets may be divided amongst fewer rounds as they are not updated with each response to the aggregator.
Second, rather than conduct DP DNN training at each participant as is shown in Algorithm 7, local training is done without privacy protection.

Third, when using the LDP customization, participants will input their updates to their local privacy module prior to uploading to the aggregator. The privacy module first selects the parameters for updating based on the current step, discretizes the selected parameters according to the module parameters \( c \) and \( \rho \), and then applies the privacy mechanism \( \Phi \). Selected privatized updates are then returned to \( A \).

These changes are reflected in the participant pseudocode in Algorithms 10 and 11.

We experimentally validated our LDP approach with the Fashion MNIST dataset in Figure 5.10. We protect local parameters using the \( \alpha \)-CLDP approach for guaranteeing local differential privacy with \( \alpha = 1 \) and demonstrate that while ensuring privacy through this formal framework participants are still able to achieve model accuracy which outperforms non-private local training. Additionally, as LDP protocols were meant for data collection from edge devices, this approach leads to lightweight protocols which may be more appropriate in many settings rather than requiring the overhead of cryptographic methods. And, of notable importance, extends the customizability of the TSC-PFed system by introducing an alternative form of privacy protection.
Algorithm 11 TSC-PFed LDP Privacy Module

**System Parameters:** \( \Psi \): update filter, \( c \): LDP clipping value, \( \rho \): LDP precision value, \( \Phi \): privacy mechanism

**Procedure** PRIVATIZE\_UPDATE\( (\theta, s, \eta, \theta_0, \epsilon) \)

\[ I \leftarrow \Psi(s) \]
\[ \theta_u \leftarrow \theta_0 \]
\[ \tilde{\theta}_u \leftarrow \text{DISCRETIZE\_UPDATE}(\theta_u, c, \rho, \eta) \]
\[ \theta^0 \leftarrow \theta_0^0 + \Phi(\tilde{\theta}_u, \epsilon) \]

**Return** \( \theta \)

**Procedure** DISCRETIZE\_UPDATE\( (\theta, c, \rho, \eta) \)

\[ \tilde{c} \leftarrow \eta c \cdot 10^\rho \]
\[ \theta_d \leftarrow \text{ROUND}(\theta \cdot 10^\rho) \]

**For each** \( \omega \in \theta_d \) **do**

\[ \omega \leftarrow \text{MAX}(\tilde{c}, \text{MIN}(\tilde{c}, \omega)) \]

**End for**

**Return** \( \theta_d \)

**End Procedure**

5.3.2 Tiered Participant Selection for Individualized Privacy Budgets

Another area in which TSC-PFed supports customization is in the participant selection module. As was previously mentioned in the discussion on system latency, conventional federated learning systems do not query every participant at every round of learning. In TSC-PFed we determine which participants will be queried for model updates in our \( q \)-participant selection module. Where \( q \leq n \) participants are selected at each round. So how does this selection impact privacy? In differential privacy we can actually leverage sampling amplification to boost our privacy budgets according to the setting of \( q \) as participants will experience less leakage due to fewer rounds in which their data is collected and therefore potentially exposed to leakage. The privacy amplification theorem [234, 171] states that if random \( q \) samples are selected rather than the available \( n \) data, then each round satisfying \( \epsilon \)-DP incurs only a cost of \( \frac{q}{n} \epsilon \) against the privacy budget. Therefore, by only selecting a subset \( q \) of the \( n \) participants we can leverage this sampling amplification in our privacy accounting.
Privacy Agnostic Participant Selection

In TSC-PFed, prior to consideration of the customization elements, there are a number of options for selecting the \( q \) participants at each round. The first is to randomly select \( q \) participants from the pool at each round. The next is to choose participants based on their availability so that the system is not held up waiting for a selected participant that is not online or perhaps has limited power at the time of the learning round in question. This leads to participants with higher availability to be queried more frequently and therefore incur greater privacy costs. And finally there is the tiered approach from our TiFL system in which selection probability for a particular participant is based on the size of the tier assigned to that participant as well as the probabilities assigned to tiers in TiFL. Note that this will lead to different privacy guarantees for different participants.
Tiered Selection

As we introduced in our handling of stragglers in Section 4.2.2, we propose a tiered q-participant selection at each round for participants based on their privacy budgets. That is, participants with higher privacy risk tolerance and therefore higher privacy budgets are more likely to be selected similar to how participants with greater resources are more likely to be selected in the TiFL system.

Therefore, in TSC-PFed our participant selection further supports privacy customization. Specifically, we consider that participants are likely to have heterogeneity with respect to their privacy policies. We therefore support two different privacy-informed selection approaches. The first is privacy-based pools where TSC-PFed provides discrete privacy levels representing different risk levels. The privacy pool corresponding to higher risk tolerance with lower perturbation will be more likely to be selected at any given round than pools with lower risk tolerance. By setting discrete levels that broadly align with different privacy policies, TSC-PFed can support participants with less privacy expertise within their organizations. Our second privacy-aware approach considers a continuous scale of privacy settings and sets the probability of individual participants being selected in direct relation with the scale of their privacy budget. This option allows more control at each participant with respect to their privacy budget while still engaging a privacy-aware selection process. In this way through both customization of the type of privacy protection and the consideration of customization of privacy budgets in our selection protocol, TSC-PFed proposes a customizable private federated learning system.
CHAPTER 6
RELATED WORK

6.1 Machine Learning Vulnerabilities

6.1.1 Membership Inference Attacks

The membership inference attack against machine learning models was first presented in [13] where the authors proposed the shadow model attack. The authors in [26] then relax adversarial assumptions and to generate a model and data independent attacker. Melis et al. highlight the potential for feature leakage in collaborative learning environments [235]. Finally in [236] the authors articulate the power of black-box attacks compared with white-box attacks. We expand on the foundation from these works to offer new insights for the membership inference attack particularly focusing on skewed datasets and differentially private mitigation techniques.

Most of the existing proposals investigating the risk of membership inference attacks focus on deep learning models and are influenced by adversarial deep learning research such as [237], [238], [239]. For example, [34] identifies vulnerable instances for membership inference attacks exclusively relating to deep learning models while [36] seeks to define a measure of deep learning model vulnerability, with respect to the model’s encoding of a random secret within the training data, orthogonal to membership inference. [35] studies membership inference in generative adversarial networks (GANs), and shows that the level of generalization required to mitigate against membership inference in GANs will lead to worse results in accuracy and utility.

[38] proposed a measure of risk at the data instance level, and evaluated the measure on the Adult and Purchases-10 datasets with attack model, target model, and shadow model of the same type. The identified instance-level risks exemplify our analysis that membership
inference attacks are data-driven.

Alternative study on membership inference relates to the impact of overfitting based on the belief that the cause of membership inference is model overfitting. [240] investigates this belief and concludes that overfitting is not necessary for a model to show vulnerability to membership inference. Their investigation is limited to the role of overfitting and assumes a powerful adversary with prior knowledge of the average training loss for the targeted predictive model. This work does motivate that membership inference vulnerability is more complex than just the overfitting in the training data.

Application-specific membership inference, such as [108], studied membership inference vulnerability specific to location data under a powerful adversary with deep prior knowledge. Though this work aims at attacking aggregate data rather than a trained target model and its training data, it does demonstrate the risk of membership inference attacks in a privacy-conscious domain.

In this dissertation we extend the work done in the membership inference research space to a more general setting towards demystifying the adverse effect of membership inference across different types of models with both general and empirical characterization of why membership inference attacks are more effective in some scenarios than in others. We additionally provide a framework, MPLens, for the individualized evaluation of model vulnerability to membership inference attacks and an introduction to insider membership inference attacks specific to the growing area of distributed machine learning.

Membership Inference and Adversarial Machine Learning

There have been some recent works in analyzing the connection between membership inference and adversarial machine learning. Song et al. [241] and Mejia et al. [242] demonstrate that existing adversarial defense methods cause an increase in membership inference vulnerability. Our membership inference evaluation system highlights the orthogonal risk. That is, how membership inference can inform adversarial machine learning attackers.
6.1.2 Label Flipping Data Poisoning Attacks

Poisoning attacks are highly relevant in domains such as spam filtering [54, 55], malware and network anomaly detection [56, 57, 58], disease diagnosis [59], computer vision [60], and recommender systems [61, 62]. Several poisoning attacks were developed for popular ML models including SVM [63, 64, 51, 65, 52, 53], regression [66], dimensionality reduction [67], linear classifiers [64, 68, 69], unsupervised learning [70], and more recently, neural networks [64, 71, 72, 65, 73, 74]. However, most of the existing work is concerned with poisoning ML models in the traditional setting where training data is first collected by a centralized party. In contrast, our work studies poisoning attacks in the context of FL.

The rising popularity of FL has led to the investigation of different attacks in the context of FL, such as backdoor attacks [243, 244], gradient leakage attacks [245, 246, 247] and membership inference attacks [14, 16, 16]. Most closely related to our work are poisoning attacks in FL. There are two types of poisoning attacks in FL: data poisoning and model poisoning. Our work falls under the data poisoning category. In data poisoning, a malicious FL participant manipulates their training data, e.g., by adding poison instances or adversarially changing existing instances [75, 76]. The local learning process is otherwise not modified. In model poisoning, the malicious FL participant modifies its learning process in order to create adversarial gradients and parameter updates. [77] and [78] demonstrated the possibility of causing high model error rates through targeted and untargeted model poisoning attacks. While model poisoning is also effective, data poisoning may be preferable or more convenient in certain scenarios, since it does not require adversarial tampering of model learning software on participant devices, it is efficient, and it allows for non-expert poisoning participants.
6.2 Privacy-Preserving Machine Learning

There is a huge literature in training privacy-preserving machine learning models (see [206] for a survey). However, general (non-application specific) privacy-preserving protocols for privately scoring machine learning classifiers were proposed more recently such as in [153] for the case of hyperplane-based classifiers, Naive Bayes and decision trees and in [94] for decision trees and random forests. In [126] protocols for hyperplane-based and Naive Bayes classifiers were proposed.

De Hoogh et al. [96] introduced a more efficient protocol for privacy-preserving training of decision trees with categorical attributes only. They also presented a protocol for privacy-preserving scoring of decision trees. Their protocol is designed for categorical attributes. However, it does not scale well for fine-grained numerical attributes - the complexity of the protocol increases exponentially on the bit-length representation of a category.

Many classification problems are characterized by numerical attributes, such as age, temperature, or blood test results, or by a combination of numerical and categorical attributes. The well known top down algorithms to induce decision trees from data (ID3, CART) can easily be extended to include numerical attributes as well. This is typically done with a binary split at internal nodes, e.g. instances with “cholesterol level $\leq p$” go down the left branch, and instances with “cholesterol level $> p$” go down the right. The threshold $p$ is chosen dynamically at each node as the tree is grown, and, unlike with categorical attributes, a numerical attribute may appear more than once in the same tree branch, but with different thresholds. For instance, in the branch below the node “cholesterol level $\leq p$”, a new node “cholesterol level $\leq p^*$” may appear, with $p^*$ a smaller threshold than $p$. The process of dynamically choosing and refining thresholds adds to the expressivity of decision trees with numerical values, making the hypothesis space of such trees far richer than that of decision trees with categorical values.

In [153], hyperplane-based classifiers were implemented by using a secure protocol for
computing the inner product based on the Paillier encryption scheme and a comparison protocol that also relies heavily on the Paillier encryption scheme.

The decision tree protocol of Bost et al. [153] is divided in two phases. In a first stage Paillier-based comparison protocols are run with Alice inputting a vector containing her features and Bob inputting the threshold values of the decision tree. On a second stage, fully homomorphic encryption is used to process the outcomes of the comparison protocols run in the first stage. It is claimed that the protocol leaks nothing about the tree (we will show that in a more realistic attack scenario this is not true) and the second stage is round-optimal. However, the computations to be performed are heavy and the first stage involves many rounds (in total their protocol typically has more rounds than ours). In our solution, we allow the depth of the tree to be leaked, but avoid altogether using Paillier and fully homomorphic encryption. In our solution, the online phase for evaluating decision trees uses solely modular additions and multiplications.

In [94] protocols for decision trees and random forests were proposed based on an original comparison protocol also based on the Paillier encryption scheme and on oblivious transfer. The Paillier encryption scheme uses modular exponentiation and oblivious transfer protocols that are usually as expensive as public-key cryptographic primitives. Our solutions for private ML protocols with SMC use, in the online phase, solely additions and multiplications over a finite field or ring.

In [126], one can find protocols for hyperplane-based and Naive Bayes classifiers in the commodity-based model. By directly replacing some of the building blocks used in [126] (the comparison and bit decomposition protocols) by the ones used in this paper, the communication and computing complexities can be decreased.

6.2.1 Differential Privacy to Mitigate Membership Inference

Both [213] and [107] investigate differential privacy as a mitigation technique for membership inference attacks. Both indicate that existing differential privacy techniques do not
display viable accuracy and privacy protection trade-offs. That is, either models show large accuracy losses or display high membership inference vulnerability. We extend this line of work to investigate the implications of DNN model complexity, learning task complexity, and data skewness on membership inference vulnerability and on the effectiveness of differentially private learning as a mitigation strategy.

6.3 Privacy-Preserving Federated Learning

**Trusted Aggregator** Approaches in this area trust the aggregator to obtain data in plaintext or add noise. [168] and [248] propose differentially private ML systems, but do not consider a distributed data scenario, thus requiring a central party. In [249], the authors develop a distributed data mining system with DP but show significant accuracy loss and require a trusted aggregator to add noise.

More recently, [250] presented PATE, an ensemble approach to private learning wherein several “teacher” models are independently trained over local datasets. A trusted aggregator then provides a DP query interface to a “student” model that has unlabelled public data (but no direct access to private data) and obtains labels through queries to the teachers. While we have proposed a federated learning (FL) approach wherein one global model is learned over the aggregate of the parties’ datasets, the PATE method develops an ensemble model with independently trained base models using local datasets. Unlike the methods we evaluate, PATE assumes a fully trusted party to aggregate the teachers’ labels; focuses on scenarios wherein each party has enough data to train an accurate model, which might not hold, e.g., for cellphone users training a neural network; and assumes access to publicly available data, an assumption not made in our FL system. Models produced from our FL system learn from all available data, leading to more accurate models than the local models trained by each participant in PATE (Figure 4b in [250] demonstrates the need for a lot of parties to achieve reasonable accuracy in such a setting).
Cryptographic Approaches  [251] presents a protocol to privately aggregate sums over multiple time periods. Their protocol is designed to allow participants to periodically upload encrypted values to an oblivious aggregator with minimum communication costs. Their approach however has participants sending in a stream of statistics and does not address FL or propose an FL system. Additionally, their approach calls for each participant to add noise independently. As our experimental results show, allowing each participant to add noise in this fashion results in models with low accuracy, making this approach is unsuitable for FL. In contrast, our approach reduces the amount of noise injected by each participant by taking advantage of the additive properties of DP and the use of threshold-based homomorphic encryption to produce accurate models that protect individual parties’ privacy.

In [157, §B] the authors propose the use of multiparty computation to securely aggregate data for FL. The focus of the paper is to present suitable cryptographic techniques to ensure that the aggregation process can take place in mobile environments. While the authors propose FL as motivation, no complete system is developed with “a detailed study of the integration of differential privacy, secure aggregation, and deep learning” remaining beyond the scope.

[252] provides a theoretical analysis on how differentially private computations could be done in a federated setting for single instance operations using either secure function evaluation or the local model with a semi-trusted curator. By comparison, we consider multiple operations to conduct FL and provide empirical evaluation of the FL system. [253] proposes a system to perform differentially private database joins. This approach combines private set intersection with random padding, but cannot be generally applied to FL. In [254] the authors’ protocols are tailored to inner join tables and counting the number of values in an array. In contrast, we propose an accurate, private FL system for predictive model training.

Dwork et al. [255] present a distributed noise generation scheme and focus on methods
for generating noise from different distributions.

[256] proposes a method to train neural networks in a private collaborative fashion by combining MPC, DP and secret sharing assuming non-colluding honest parties. By contrast, in our system TSC-PFed, when using the trust enhancement, prevents privacy leakages even if parties actively collude.

Approaches for the private collection of streaming data, including [257, 258, 259, 260], aim to recover computation when one or more parties go down. Our system, however, enables private federated learning which allows for checkpoints in each epoch of training. The use of threshold cryptography also enables our system to decrypt values when only a subset of the participants is available.
CHAPTER 7
CONCLUSION

7.1 Summary

This dissertation research made a number of unique research contributions. We first took a holistic approach to create a structured and comprehensive analysis of privacy risks in machine learning including a characterization of privacy vulnerabilities in both centralized and decentralized settings and an in-depth study on inference-based privacy attacks, specifically membership inference, against machine learning models including demonstrating the importance of considering vulnerabilities at a more granular level when data is skewed.

Through our extensive experimentation we are able to characterize membership inference attack vulnerability as (a) data-driven, (b) transferable, and (c) impacted by model choice. We then reflect these factors in our privacy analysis and compliance evaluation system MPLens which allows machine learning practitioners to evaluate membership privacy through a multidimensional, customizable lens.

We also demonstrate that both membership inference and adversarial ML attacks can persist in distributed systems through malicious system participants.

In the second contribution we developed a number of privacy-preserving machine learning solutions and discuss formal and informal frameworks for privacy protection including attack-based techniques, randomization-based obfuscation, the differential privacy framework, and secure multiparty computation protocols. We then provide a comparison of these methods through representative approaches used to protect a specific, popular ML model: the decision tree.

We additionally provide privacy-preserving approaches to model training and evaluation which extend the state-of-the-art in privacy-preserving ML as well as discussion and
experimental evaluation to understand their usability in the context of real world machine learning as a service systems. We additionally demonstrate that there are important trade-offs in implementing differentially private deep learning strategies as a mitigation to membership inference attacks, particularly as vulnerability relates to model and problem complexity as well as the implications for skewed data.

Our third contribution recognized attack vulnerabilities in the federated learning landscape by identifying the stakeholders in a federated learning system and then formalizing the relevant privacy threats as well as deployment considerations unique to federated learning.

With the fourth contribution of this dissertation we proposed an architecture, TSC-PFed, for trust and security enhanced customizable private federated learning. We combine secure multiparty computation and differential privacy to allow participants to leverage known trust dynamics which allow for increased ML model accuracy while preserving privacy guarantees and introduce an update auditor to protect against malicious participants launching dangerous label flipping data poisoning. We additionally introduce customizable modules into the TSC-PFed ecosystem which (a) allow users to customize the type of privacy protection provided and (b) provide a tiered participant selection approach which considers variation in privacy budgets.

7.2 Open Issues and Future Work

In this dissertation we have developed a set of privacy-preserving techniques, attack analysis tools, and deployable machine learning systems which address the ecosystem of concerns in modern machine learning including data privacy, model security, and predictive accuracy. Moving forward, we would like to explore what it means to formalize the trade-offs between these concerns. Such formalization may include optimizing private learning systems, identifying and integrating entity relationships into system design, and formalizing privacy policies within the context of the growing body of privacy research. This
includes the following research questions.

7.2.1 Optimizing Private Systems

How can privacy settings be informed by vulnerability analyses? How can variation in resources and privacy tolerance be considered in privacy-preserving systems? In [164] we evaluated the impact of resource heterogeneity in federated learning, but what about heterogeneity with respect to privacy? Individuals and indeed companies have extremely variant perspectives on data privacy. Additionally, given the variation in vulnerability highlighted in [9], different data sources are likely to have different levels of vulnerability which in hand require different levels of mitigation. Some may then limit their participation in a machine learning system such as TSC-PFed more strictly than others. Developing systems which not only allow for the appropriate customization but also are designed to optimize model accuracy in the presence of such privacy heterogeneity will allow for a larger range of scenarios in which privacy-preserving systems can be deployed.

7.2.2 ML/AI Trust and Ethics

How can machine learning systems leverage trust? Can mitigation techniques be tuned to account for different players? In [221] we accounted for trust between participants according to a global trust parameter $t$ which allowed us, under a tailored threat model, to decrease noise without sacrificing privacy. We are therefore interested in exploring the impact of similar trust dynamics in other aspects of machine learning systems. For example, in mitigating data poisoning attacks in [80] we analyze gradient updates provided by federated learning participants to identify potentially malicious updates. The federated learning system may then discard these updates and stop receiving updates from the potentially malicious participant. What if, however, the participant was not malicious but rather had distinct data from the other participants, making their updates look notably different and resulting in the errant malicious flagging. A trust threshold could be used in conjunction
with the defense mechanism to better accommodate such cases. Specifically, entities with a good reputation or high trust value may be given more leniency by the defense algorithm than entities with lower or unknown trust values.

7.2.3 Multi-Disciplinary Applications for ML Systems

How do the settings for specific applications impact machine learning systems? How will parameter tuning decisions be made? Will the balance of privacy, accuracy, and security be tipped differently based on these settings? What are the new challenges presented by the domain? In realizing customizable solutions for private, secure, and effective machine learning systems that can have real-world impact, developing ML systems for multidisciplinary applications and exploring the technical challenges inherent in specific problem spaces is also of considerable interest.

7.2.4 Privacy Policy

Finally, an important issue in the privacy space is how privacy research in the computer science community can impact broader public policy. In particular, how can federal or international policy be developed in light of the known attacks to machine learning services? How can individual organizational policies be translated into parameter settings in machine learning systems? With the body of privacy legislation continuing to grow both internationally and within the US and individual companies honing their own privacy policies, there is a demand for the translation between privacy research and policy language as well as between policy language and available technologies.
REFERENCES


[85] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, “From throw-away traffic to bots: Detecting the rise of dga-based mal-


[210] M.-J. Xiao, L.-S. Huang, Y.-L. Luo, and H. Shen, “Privacy preserving id3 algorithm over horizontally partitioned data,” in Parallel and Distributed Computing, Appli-


239


VITA

Stacey Truex was born and grew up in the small Connecticut town of East Granby. After she received her BS in Computer Science and BA in Mathematics from Wake Forest University, she spent time in industry as a Software Developer for Epic Systems as well as an Applications Developer for CGI Federal. She then returned to graduate school in 2014 and earned her Masters in Computer Science and Systems from the University of Washington in Tacoma, WA. She started her Ph.D. in the College of Computing’s School of Computer Science at the Georgia Institute of Technology in 2016 where her research is broadly centered on data privacy in machine learning systems. Specifically, she is interested in understanding and formalizing the trade-offs between concerns around data privacy, predictive model security, and machine learning accuracy with pursuit of such crossing areas of machine learning, privacy, trust, and policy.