

# Practical Privacy-Preserving Federated Learning with Secure Multi-Party Computation

Benjamin Akhtar

Thesis submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of

Masters of Science  
in  
Computer Engineering

Wenjie Xiong, Chair

Linbo Shao

Thang Hoang

July 25, 2024

Blacksburg, Virginia

Keywords: Multi-Party Computation, Machine Learning, Federated Learning, Differential  
Privacy

Copyright 2024, Benjamin Akhtar

# Practical Privacy-Preserving Federated Learning with Secure Multi-Party Computation

Benjamin Akhtar

(ABSTRACT)

Federated Learning (FL) has become increasingly important for machine learning applications that do not allow user data to be removed from the device it exists on. Basic FL is still susceptible to attacks that compromise privacy through privacy inference and poisoning attacks. Through secure Multi-Party Computation (MPC) and Differential Privacy (DP) it is possible to eliminate more of these attacks.

Secure Multi-Party Computation is a promising efficient solution for privacy-preserving computation to ensure that the aggregation servers do not learn the information being protected. The implementation of MPC has occurred in many different formats and security threats. Prior work SAFEFL utilizes the use of MPC through MP-SPDZ, a secure MPC framework, in their secure federated learning framework, however, it suffered from numerous limitations in execution time that prevented any practical use.

In this study, SAFEFL's limitations are examined thoroughly and discussed how to be improved. A new design is proposed for secure federated learning with MPC. Additionally, differential privacy is added into the MPC aggregation for an extra layer of security. These new designs are compared with the original SAFEFL design and show the performance improvements and scalability for future.

# Practical Privacy-Preserving Federated Learning with Secure Multi-Party Computation

Benjamin Akhtar

(GENERAL AUDIENCE ABSTRACT)

In a world with ever greater need for machine learning and artificial intelligence, it has become increasingly important to offload computation intensive tasks to companies with the compute resources to perform training on potentially sensitive data. In applications such as finance or healthcare, the data providers may have a need to train large quantities of data, but cannot reveal the data to outside parties for legal or other reasons. Originally, using a decentralized training method known as Federated Learning (FL) was proposed to ensure data did not leave the client's device. This method still was susceptible to attacks and further security was needed. Multi-Party Computation (MPC) was proposed in conjunction with FL as it provides a way to securely compute with no leakage of data values. This was utilized in a framework called SAFEFL, however, it was extremely slow.

Reducing the computation overhead using programming tools at our disposal for this framework turns it from a unpractical to useful design. The design can now be used in industry with some overhead compared to non-MPC computing, however, it has been greatly improved.

# Dedication

*This is dedicated to my friends and family who have always been there with their love, support, and patience.*

# Acknowledgments

First, I want to thank my advisor Dr. Wenjie Xiong for her continued support and help throughout my journey here at Virginia Tech. She continually guided me through rough patches and provided endless patience, even if I made things more difficult. She provided me with expert knowledge and kindness that has allowed me to strive and achieve my best. I am truly grateful for everything she has provided to myself and other students.

I would like to also thank Dr. Linbo Shao and Dr. Thang Hoang for their valuable feedback and service as my thesis committee members.

I want to thank Jack Chandler for his immense help throughout this process, in particular with the fine-tuning models and differential privacy. He continually checked and verified my work and helped with the debugging process to ensure the best possible results. He also helped me with the equations that I did not understand.

I also want to thank Pedro Soto for being a Guinea pig on my thesis presentation. He also helped me be able to explain the concepts in a better and more understandable manner.

A huge thank you to my family for all of their support and love throughout the process. You have always believed in me and pushed me to continue my journey even at my lowest. You always helped me make the right decisions for myself and talked me through the toughest problems.

Finally, I want to thank all of my friends for your phenomenal support from start to finish. You have always provided a place to vent or come up with ideas. You made everything fun and helped ease the massive stress I have faced in the previous years. Even if I felt like giving up or couldn't imagine what to do, you always made me laugh and provided me a shoulder to lean on. Truly, from the bottom of my heart, thank you.

# Contents

- List of Figures** **xii**
  
- List of Tables** **xiv**
  
- 1 Introduction** **1**
  
- 2 Background** **4**
  - 2.1 Attacks on Machine Learning . . . . . 4
    - 2.1.1 Model and Dataset Leakage . . . . . 4
    - 2.1.2 Inference Attacks . . . . . 5
    - 2.1.3 Poisoning Attacks . . . . . 6
    - 2.1.4 Mitigation Techniques . . . . . 7
  - 2.2 Federated Learning . . . . . 9
  - 2.3 Multi-Party Computation . . . . . 10
    - 2.3.1 Security Setting . . . . . 10
    - 2.3.2 MPC Protocols . . . . . 13
  - 2.4 Differential Privacy . . . . . 15
    - 2.4.1 Inverse Transform Sampling . . . . . 16

<b>3</b>	<b>Profiling of Existing Frameworks</b>	<b>17</b>
3.1	MP-SPDZ Overview . . . . .	17
3.1.1	Use Cases of MP-SPDZ . . . . .	17
3.1.2	Related MPC Frameworks . . . . .	18
3.1.3	Compiler in MP-SPDZ . . . . .	19
3.1.4	Virtual Machine in MP-SPDZ . . . . .	20
3.2	MP-SPDZ Profiling . . . . .	21
3.2.1	Parallelization and Limitations . . . . .	21
3.2.2	Memory Bandwidth Usage . . . . .	23
3.3	SAFEFL Overview . . . . .	25
3.3.1	PyTorch Communicator: For Client-Server Communication . . . . .	25
3.3.2	Secure Aggregation with MP-SPDZ in SAFEFL . . . . .	26
3.3.3	Robust Aggregation . . . . .	27
3.3.4	Limitations . . . . .	27
<b>4</b>	<b>Proposed Framework</b>	<b>30</b>
4.1	Proposed Design Overview . . . . .	30
4.1.1	Threat Model . . . . .	30
4.1.2	System Overview . . . . .	31
4.2	Client Optimizations . . . . .	36

4.2.1	Cython	37
4.2.2	Array Conversion	39
4.2.3	No Input Masking	39
4.3	Server-Side Optimizations	40
4.3.1	Removing Memory Write	40
4.3.2	Probabilistic Truncation	41
4.3.3	Power of 2 Number of Clients	41
4.4	Deep Neural Networks Added to SAFEFL	41
4.4.1	Comparison of Models to MP-SPDZ	42
4.4.2	Model Information	43
4.5	Dataset Functionality	43
4.5.1	Added Datasets	44
4.5.2	Partitioning Data Among Clients	44
4.6	Differential Privacy	45
4.6.1	Use Case: Fine-Tuning the Last Layers in a Model	45
4.6.2	Aggregation Scheme for DP	46
<b>5</b>	<b>Evaluation</b>	<b>48</b>
5.1	Evaluation Setup	48
5.1.1	Hardware and Software Platform	49

5.1.2	Evaluation Metrics . . . . .	50
5.2	Standard Deviation of End-to-end Aggregation Execution Time . . . . .	51
5.3	End-to-End Latency . . . . .	52
5.4	Cython and Numpy Operations . . . . .	53
5.4.1	Converting Floats to Ring Integers . . . . .	54
5.4.2	Input Masking . . . . .	56
5.4.3	Array Conversion . . . . .	57
5.4.4	Unpacking Buffer Intermittently . . . . .	58
5.5	Server-Side Optimizations . . . . .	59
5.5.1	Removal of Memory Write . . . . .	59
5.5.2	Probabilistic Truncation and Modulo 2 Workers . . . . .	60
5.6	Latency for Different Settings . . . . .	61
5.6.1	Number of Clients . . . . .	62
5.6.2	Neural Network Size . . . . .	63
5.6.3	Network Latency . . . . .	64
5.7	Accuracy . . . . .	67
5.8	Differential Privacy Accuracy . . . . .	73
<b>6</b>	<b>Future Work</b>	<b>76</b>
<b>7</b>	<b>Conclusions</b>	<b>78</b>



# List of Figures

3.1	Local Parallelization Limitations in MP-SPDZ. . . . .	22
3.2	Network Parallelization Limitations in MP-SPDZ. . . . .	23
3.3	Memory Utilization in MP-SPDZ. . . . .	24
3.4	List Comprehension vs. Numpy in Python. . . . .	28
4.1	Overview of Proposed Design. . . . .	31
4.2	SAFEFL Original Design. . . . .	33
4.3	Detailed Proposed Design Flowchart with Optimizations and Speedups. . . . .	35
5.1	Number of Workers Latency for Secure Aggregation Scaling. . . . .	62
5.2	Number of Modulo 2 Workers Latency for Secure Aggregation Scaling. . . . .	63
5.3	Number of Parameters Latency for Secure Aggregation Scaling. . . . .	64
5.4	Network Latency with Packet Size 65536 for Secure Aggregation Scaling. . . . .	65
5.5	MPC Test Accuracy on CIFAR-10 for Different Nets. . . . .	69
5.6	MPC Test Accuracy on CIFAR-100 for Different Nets. . . . .	69
5.7	MPC Test Top 5 Accuracy on CIFAR-100 for Different Nets. . . . .	70
5.8	MPC Test Accuracy on CIFAR-10 for Variable Workers. . . . .	70
5.9	MPC Test Accuracy on CIFAR-100 for Variable Workers. . . . .	71

5.10 MPC Test Top 5 Accuracy on CIFAR-100 for Variable Workers. . . . .	71
5.11 MPC Test Accuracy on CIFAR-10 for Different Skews. . . . .	72
5.12 MPC Test Accuracy on CIFAR-100 for Different Skews. . . . .	72
5.13 MPC Test Top 5 Accuracy on CIFAR-100 for Different Skews. . . . .	73
5.14 MPC Accuracy of Vision Transformer on CIFAR-10 with Differential Privacy. . . . .	74
5.15 $\epsilon$ vs. Epochs for Differential Privacy. . . . .	74

# List of Tables

4.1	Semi-Honest vs. Malicious Protocols . . . . .	31
4.2	Parameters for Each Model in PPPFL with Secure MPC for CIFAR-10. . . . .	43
5.1	Standard Deviation by Phases as a Percentage. . . . .	52
5.2	End to End Latency of Secure Aggregation for Semi-Honest Adversary. . . . .	53
5.3	End to End Latency of Secure Aggregation for Malicious Adversary. . . . .	53
5.4	Convert Floats to 64-bit Integers Execution Time for Semi-Honest Adversary. . . . .	54
5.5	Convert Floats to 64-bit Integers Execution Time for Malicious Adversary. . . . .	55
5.6	Convert 64-bit Integers to Floats Execution Time for Semi-Honest Adversary. . . . .	55
5.7	Convert 64-bit Integers to Floats Execution Time for Malicious Adversary. . . . .	55
5.8	Server Random Time for Semi-Honest Adversary. . . . .	56
5.9	Server Random Time for Malicious Adversary. . . . .	56
5.10	Convert Masked Inputs to Byte String Execution Time for Semi-Honest Adversary. . . . .	57
5.11	Convert Masked Inputs to Byte String Execution Time for Malicious Adversary. . . . .	58
5.12	Receive Outputs Execution Time for Semi-Honest Adversary. . . . .	58
5.13	Receive Outputs Execution Time for Malicious Adversary. . . . .	59
5.14	Other Execution Time for Semi-Honest Adversary. . . . .	60

5.15 Other Client Execution Time for Malicious Adversary. . . . .	60
5.16 MPC Execution Time for Semi-Honest Adversary. . . . .	61
5.17 MPC Execution Time for Malicious Adversary. . . . .	61
5.18 Network Latency with Packet Size 128 for Secure Aggregation Scaling. . . . .	66
5.19 Network Latency with Packet Size 65536 for Secure Aggregation Scaling. . . . .	66
5.20 Network Latency with Packet Size 33554432 for Secure Aggregation Scaling. . . . .	66
5.21 Model Accuracy After 2500 Iterations on CIFAR-10. . . . .	67
5.22 Model Accuracy After 2500 Iterations on CIFAR-100. . . . .	68

# Chapter 1

## Introduction

Recently, Machine Learning (ML) has become more widespread with its potential for numerous applications in healthcare [28], finance [48], autonomous driving [52], computer vision [39] and other industries. This is combined with the increase in compute performance in modern Graphics Processing Units (GPUs) to help train and run these models. In the past, the entire dataset and model were collected and trained on a central server. This is not always possible due to new regulations that have stricter data protection collection such as the California Consumer Privacy Act and the General Data Protection Regulation.

Previously, multiple methods have been utilized to ensure that these regulations could be followed through Privacy Preserving ML (PPML). Secure Multi-Party Computation (MPC) or Homomorphic Encryption (HE) have been used for PPML [2, 40, 59, 69] as it kept user data secure throughout the computation, however, they have high computation overhead due to their secure continuous computing nature. Additionally, the communication costs are exceptionally high and a bottleneck in the performance.

Differential Privacy (DP) is a mathematical framework to ensure the privacy of individuals in datasets that has also been proposed for PPML [47, 76]. This allowed for extraction of the necessary data without revealing sensitive information about individuals [71]. While this was an excellent solution to avoid the computation overhead of MPC or HE [9], it is solving a different problem than MPC.

Google proposed Federated Learning [41] as a possible alternative that was tailored for PPML applications. This allowed all data to remain locally on client devices as the model was trained and to have a central server serve only as an aggregator between client updates [45]. This ensured that user’s data was not exposed to anyone except themselves. This approach has been extremely useful and has been implemented in real-world use cases [29, 73].

Although FL improves upon the computation and communication cost of MPC, FL’s privacy guarantees are weaker, which can make it more susceptible to malicious servers. FL removes the security threat of giving malicious servers direct access to sensitive data, but the gradients and model parameters are still exposed. In other words, FL is unable to protect the gradients and model throughout the process [25] compared to FL with MPC.

To solve the issues in FL, secure FL [11] was proposed that combined FL with either MPC or HE to ensure that data and all intermediate gradients are not exposed. This technique leveraged secure aggregation protocols that would not expose user data to the server. However, these designs were still vulnerable to poisoning attacks and often had drawbacks on their implementation [78], such as a set number of servers or a specific MPC protocols had to be used.

In recent years, frameworks have been proposed to expand upon the limitations from early secure FL frameworks [24, 66] by expanding the protocols allowed to be used, number of parties allowed, number of servers involved, or improving performance. However, these frameworks still lacked differential privacy to protect both user data from any possible attack or the server from malicious clients. Additionally, frameworks may often have lacked expanded machine learning functionality [13] and been built for only one portion of machine learning.

This research is essential as data privacy laws are subject to change and could become

more stringent. It is necessary to ensure both the clients and servers are not exposed to attacks, while being able to train models at an efficient pace. Additionally, having easy to utilize open-source frameworks that can be used by anyone, is extremely valuable as machine learning expands to mainstream usage.

The contributions are as follows:

1. Existing frameworks were profiled to determine limitations and possible areas of improvement.
2. Communication steps were optimized for secure aggregation between clients and servers using Numpy and Cython.
3. MPC server-side optimizations were provided to further decrease execution time.
4. Differential privacy was added as a feature in secure aggregation.
5. The proposed framework was evaluated for current limitations.

This thesis is organized as follows. Chapter 2 provides background and related protocols and security guarantees for MPC, FL, and DP. Chapter 3 provides an overview of the existing limitations of MP-SPDZ and SAFEFL. Chapter 4 discusses our proposed design and Chapter 5 presents the results for this design. Chapter 6 describes future work to be done and Chapter 7 concludes the thesis.

# Chapter 2

## Background

This chapter will discuss the relevant background related to the proposed Practical Privacy-Preserving Federated Learning (PPPFL) with Secure MPC framework. First, attacks on machine learning will be introduced along with mitigation techniques. From there, federated learning will be discussed. Next, a detailed description of Multi-Party Computation with its different security settings and protocols will be provided. Differential Privacy will also be introduced.

### 2.1 Attacks on Machine Learning

Machine Learning offers unique tools for computation, however, it open to a wide variety of attacks [4]. These attacks vary in the setting they execute and the scope of the attack.

#### 2.1.1 Model and Dataset Leakage

The most prominent attacks are to learn information about the model trained for machine learning [58] or data used for training [55, 64, 75]. These data attacks may be to gain information about a specific data point or where the data came from. Leaking the model may pose a loss of intellectual property for companies or individuals. It can also be used a precursor to future attacks. Leaking information about the data can expose sensitive

knowledge or personal details.

Protecting the model is necessary for servers or companies who may depend on this product for their business. Protections may not be as fundamental in a centralized learning environment where the model never leaves the server's device. In FL, the model gradients or weights must be passed back and forth between the clients and servers, potentially exposing the model parameters. Section 2.3 will expand on how MPC protects against this leakage of information.

Data and dataset information is highly sought after and can expose potentially sensitive information. Additionally, the data holder may not be in a position to share this data with a central server due to regulations. This requires updated methodology of how to train the model. Section 2.2 will discuss how client's do not expose their data to servers through Federated Learning by keeping data local.

### 2.1.2 Inference Attacks

While federated learning does offer more data privacy than centralized machine learning [3], it is still open to a wide range of attacks to gather information about the underlying data [16, 23, 34, 77]. These attacks to learn the training data indirectly from the model updates (e.g., gradients) are called inference attacks.

Inference attacks may have different goals, depending on the type of the attack and the information that the attacker wishes to learn. In feature inference attacks [43], the adversary attempts to learn the feature values of new samples for a given party or learn correlations between features throughout multiple iterations in vertical FL environment. The attacker can infer feature values that are unknown by testing and calculating the loss until a match is found. This poses a risk as the adversary can learn the underlying data of the client without

needing to have any of their original data.

Another type of inference attack is attempting to learn if a specific data point was part of the training set using membership inference attacks [65]. The attacker can utilize the labels of the targeted model in conjunction with the attack model to determine if that piece of data was part of the original client's dataset. If the model has a larger gradient change after training on that data point, then it is likely this piece of data has not been seen before. If over fitting from the model does not occur, then it likely this piece of data was already seen and the attacker can infer that the client used this data point.

A more extensive list of inference attacks was surveyed by [44] that discusses most of the possible inference attacks at the time and when and how they occur. New inference attacks are continually being shown and yields a need for more security. To prevent these attacks, secure aggregation techniques were developed to ensure the server could not gather information from the updates sent to them.

### 2.1.3 Poisoning Attacks

Users could cause fraudulent or inaccurate models by performing poisoning attacks [57, 61, 67, 72]. In poisoning attacks, the client can either provide a backdoor input or reduce model accuracy.

There are a wide variety of poisoning attacks that can be done by clients throughout the training process. An example is the label flipping attack [63] where the clients may flip the labels of certain pieces of data from the original value to another class, usually some target value. This increases the loss of the model and prevents it from learning optimally throughout the training process.

Another possible example is through scaling attacks [14] where the client creates a backdoor

in the model. This is done by adding a trigger to certain data that will then output a target label. This can reduce overall accuracy and skew certain results to a poor matching label.

Although these are a few examples of poisoning attacks, many more exist [72]. Various tools can be used against different poisoning attacks as not every defense covers all attacks. Robust aggregation schemes have been proposed to stop various poisoning attacks in federated learning.

### 2.1.4 Mitigation Techniques

Different mitigation techniques can be used against inference and poisoning attacks. As previously mentioned, inference attacks can be mitigated through secure aggregation and poisoning attacks through robust aggregation. These two aggregation methods can be combined in certain cases to protect both the servers and clients from attacks [8] that the other may attempt. While robust aggregation schemes generally add a minor overhead to the overall computation cost, secure aggregation is far more expensive [15] due to the cryptography schemes behind them.

#### 2.1.4.1 Secure Aggregation Techniques

Secure aggregation techniques can use homomorphic encryption (HE) [30] to ensure that the server would be unable to learn any information from clients user data. Instead of passing the plain text models back and forth, the clients would have a shared secret key that could be used to encrypt their trained model. This encrypted model could be passed to the server where it would perform aggregation on the encrypted model. The encrypted updated model would then be sent back to the clients who could decrypt it using their secret key to perform plain text computation.

As an alternative, MPC was proposed to be used for secure aggregation [11] due to its lower computational costs. Although MPC aggregation is secure, it varied slightly in how it would function compared to the HE based models. Whereas the HE based models would be for a central server aggregating the updates together, MPC would require at least two servers to aggregate, and optimally more for best execution time. Originally designed for Shamir’s secret sharing, MPC secure aggregation was then theorized for various different protocols [6, 24, 66] that could leveraged in different threat models.

Either of these secure aggregation techniques protect the clients from inference attacks, since the servers do not have access to the plain text gradients passed in by the clients. These do not protect the servers from poisoning attacks from the clients, however, there are other aggregation schemes that can be used in conjunction with secure aggregation to ensure both the clients and servers are protected throughout the training process.

#### 2.1.4.2 Robust Aggregation Techniques

Various robust aggregation schemes have been proposed to defend against poisoning attacks [50], [27, 53]. Generally, a robust aggregation scheme focuses on a specific attack and tries to defend against it. Sometimes, it may prevent various different attacks, however, it usually cannot defend against all poisoning attacks that a client could utilize.

Some of the techniques used by robust aggregation schemes are differential privacy [79], comparison of various updates [68], and holding a small amount of data to train and comparing the local updates to the server updates [14]. These can have varying level of results, however, they also may suffer in the case that the data is not independently and identically distributed. In this scenario, client model updates may deviate compared to other clients on the server. So, it becomes increasingly hard to determine if the updates are malicious or the

data is just poorly sampled.

If the servers have to throw out a large portion of the training gradients throughout the process, the model may suffer a loss of accuracy and skew towards specific labels or classes. Research is still being developed for robust aggregation schemes [49] to handle more attacks and best deal with poorly distributed data.

## 2.2 Federated Learning

Normally, machine learning occurs in a centralized manner [20], however, with the EU and California passing data privacy laws that went into effect in 2020, the use of centralized machine learning has become harder to leverage with limitations to data access. It may now not be possible for the data to leave the client's device, which prevents the centralized model from training on that necessary data. Many various techniques have been proposed to train models centrally or in a distributed manner [10, 32, 38]. These techniques have varying levels of utilization, however, the one that has become most common is federated learning.

Federated Learning (FL) [41] is a newer way to train machine learning or artificial intelligence models that utilizes a de-centralized approach to harness localized data for training. It can be used to ensure user data never leaves a phone, laptop, or other personal device and the a well-equipped machine learning model for that task can be created.

In regular federated learning, a server shares a global model with some number of clients. These clients can then run their private data on the global model. Then, the clients will send the model back to the server. The server then aggregates these models together to get a new updated model. This model is then sent back to the clients and the cycle continues until there are not enough parties or the model has been sufficiently trained.

## 2.3 Multi-Party Computation

Multi-Party Computation (MPC) is a series of cryptographic protocols that allows multiple parties to share data and perform a computation based on that data. In some cases, one party may provide all of the data or one party may perform all of the computation. It depends on the protocol set-up what role or roles each party will assume during a process.

### 2.3.1 Security Setting

In MPC, there are two factors for the system settings that the protocol must assume to ensure no corrupted party learns more information than necessary. The first is if the protocol assumes the computation occurs in an **honest majority setting or dishonest majority setting**. To have a honest majority setting, strictly more than half the parties must be honest or not corrupted. The benefit of using honest majority is that computation is significantly faster.

The second important factor is what the corrupted parties will do to extract information during the computation. There are two main types of corrupted parties, i.e., **semi-honest or malicious**. In the semi-honest setting, the party may not deviate from the protocol used in the computation, but are allowed to gather information from the data that is communicated to them during the process.

In the malicious setting, the corrupted party is allowed to deviate from the protocol in order to create an incorrect output or gain more information. In the case of active security, there is a need for more considerations as the honest parties may need to abort the protocol if deviates. If they do not, it is possible an incorrect result will be output. Additionally, if the computation reaches the output stage, then the output needs to be delivered to all parties.

While active security provides stronger security guarantees, it comes at the expense of greater computation overhead to ensure that information is not leaked to corrupted parties.

### 2.3.1.1 Semi-Honest Setting

In semi-honest setting, the assumption is that the corrupted parties will not deviate from the protocol [31]. This means the corrupted parties are gathering information via the communication from other parties. Additionally, they may try to pool their information with other corrupted parties to learn more information. Since these parties are not performing any deviation from the protocol, they are often called passive or honest-but-curious.

Although this does not pose a threat of ruining the output from a program, such an adversary could learn valuable information about one's personal data or inputs. Especially for federated learning, this is exactly what is needed to be prevented.

The semi-honest case could even be a scenario for when a server may be honest, but later have some data stolen or taken. Stolen data is fairly common [51], so protecting against a semi-honest adversary, while it may seem rather needless, serves an important purpose and is not trivial.

### 2.3.1.2 Malicious Setting

Unlike semi-honest setting, malicious setting assumes that the attacker may deviate from the protocol [31] or is an active attacker. The corrupted party has at its disposal all of the tools of a semi-honest corrupted party and can take any actions it wants to learn more information during the protocol. This can include manipulating or creating false messages or values to throw off the protocol and learn valuable information.

Since an malicious attacker can now invent or manipulate values, there is no guarantee their

output will be correct. This is important because the honest parties outputs may be affected by the corrupted parties. Therefore, our only assumption is that the honest parties will follow the protocol and attempt to produce the correct output.

By assuming this, checks have to be put in place to ensure that the output throughout the computation is correct from all parties. This tends to be fairly expensive [25] when compared with the semi-honest case. If a party does not produce the correct output, in MPC it is usual to abort and cancel any further communication.

### 2.3.1.3 Honest Majority

The honest majority setting assumes that more than half of the parties involved in the computation are honest [22]. This only works with more than two parties. In the two party case, dishonest majority is always used. The honest majority setting allows for secret sharing protocols without the use of any added security. This is because all protocols used in MPC are secure when less than half the parties are corrupt.

The honest majority setting also has some of the fastest protocols in the specific three party setting [22]. This is because the communication costs are relatively low compared to settings with more parties.

### 2.3.1.4 Dishonest Majority

In comparison to honest majority, dishonest majority does require elevated cryptography on top of any protocol already chosen. This comes in the form of oblivious transfer or homomorphic encryption [36]. This extra layer of cryptography is added on top of any scheme, such as secret sharing, that provides the underlying protocol for computation. This ensures during communication between parties, that no information will be leaked since the

data is properly encrypted and cannot be learned without the key to the encrypted data.

This extra security is usually extremely costly, as public-key cryptography is computationally expensive. To avoid this problem, it could be possible to add more known trusted parties to ensure that an honest majority is met.

### 2.3.2 MPC Protocols

Various protocols can be utilized for MPC, depending on the threat model.

**Secret sharing** is one of the most commonly used and splits values between the parties in the computation where each party receives a share of the original data. The shares can be reconstructed to form the original value. Throughout the protocol, operations on performed on the shares that equate to the plain text operations [62]. The benefit to secret sharing is relatively fast computation. Beaver's triples [5] is a way to perform multiplication computation in secret sharing protocol. It requires a pre-processing phase. In the pre-processing phase, correlated values are created that can be later be used in the online phase. The Beaver triple is a special trio of values where the parties hold shares of  $[a]$ ,  $[b]$ , and  $[c=ab]$ . The triples can be used to perform multiplication operations in a faster manner.

**Oblivious Transfer (OT)** [17] is a two-party protocol that allows for each party to learn nothing about the other party's input or data during communication phases while performing the necessary computation with other protocols. OT supports secret sharing and garbled circuit protocols. The receiver uses one bit  $b$  and the sender will input two bits  $s_0$  and  $s_1$ . The receiver will get some value from the output in the form of  $s_b$ , but the sender does not learn the value of  $s$ . This scheme is used for dishonest majority.

**Garbled Circuits (GC)** [7] is another two-party protocol that utilizes OT for the communication portion of the protocol. The goal of GC is to evaluate some function without leaking

information, usually the garbler needs to offload some heavy computation to the evaluator, who has more processing power. The main portion of GC consists of using truth tables for various input gates, such as XOR, AND, OR, and their associated output. The tables are then encrypted and passed via OT and then receiver can decrypt the tables to receive the corresponding output.

There are more MPC protocols such as Homomorphic Encryption (HE) and Beaver's triples. HE uses public-key cryptography to encrypt inputs before computation is performed on them. The outputs are then returned and decrypted and the result is the same as it would be in plain text [1].

### 2.3.2.1 Alternative Cryptographic Methods

Although MPC provides numerous options to use for secure computation depending on the threat level, there are other options available as well to maintain data security [54] in machine learning. This list is not comprehensive, but provides other methods that could be used to provide security for individual's data for computation.

1. HE as previously discussed allows computation on encrypted data. This can be used outside of MPC .
2. Zero-knowledge proofs provide a way to verify the accuracy of information without revealing anything.
3. Obfuscation adds misleading data to potentially mask any sensitive information.

The main reason to use MPC over other methods is that it provides enough security without extremely high computation costs. For example, differential privacy by itself may inject enough randomness to prevent individuals data from being learnt or exposed, however, it

does not actually mask that data fully during computation and may not provide enough security guarantees for a healthcare application. On the other end of the spectrum, while homomorphic encryption will guarantee enough security for the data, it may be abnormally expensive to use. MPC is a nice middle ground that does not expose the data outside of the final result and is computationally acceptable.

## 2.4 Differential Privacy

Differential Privacy (DP) [35] is a way to mask individual data points using randomly generated noise, either Gaussian or Laplacian, and adding to the data. The distance of two datasets  $D$  and  $D'$ ,  $d(D, D')$ , is defined as how much the samples must change to convert  $D$  into  $D'$  [21]. Additionally, the function  $f$  takes in some dataset  $D$  and outputs  $f(D)$ . So, to satisfy differential privacy, for some function  $f$ , and positive variables  $(\epsilon, \delta)$ ,  $d(D, D') = 1$  as long as for every subset  $S$  of  $f$  equation Equation 2.1 holds.

$$P(f(D) \in S) \leq \delta + e^\epsilon P(f(D') \in S) \quad (2.1)$$

For best privacy,  $\epsilon$  should be lower. This is the privacy guarantee or privacy budget that is measured throughout the course of machine learning [35]. Every time the dataset is accessed or trained on for a batch, epsilon will increase.

There are two main types of privacy [19],  $\epsilon$ -differential privacy with Laplacian noise and  $(\epsilon, \delta)$ -differential privacy with Gaussian noise.  $\epsilon$ -differential privacy is usually considered stronger since if  $\delta > 0$  there is still some possibility information is leaked.

### 2.4.1 Inverse Transform Sampling

Inverse transform sampling [33] is a technique to generate a distribution from a uniformly random distribution. First, a uniform distribution can be generated from 0 to 1. Then, the inverse Cumulative Distribution Function (CDF) is found for the function that is needed to be generated. The inverse CDF is applied to the uniform distribution and a Gaussian distribution with a standard deviation of 1 is created. The distribution can then be multiplied by the standard deviation that is desired,  $\sigma$ , to finalize the Gaussian distribution that is close to a similar plain text distribution.

This can be used to generate as Gaussian distribution necessary for DP. The reason techniques such as this may be used is that MPC frameworks do not always have the ability to generate a Gaussian distribution natively, since there is no way to securely generate it randomly. Therefore, as long as a uniform distribution can be generated, so can a Gaussian. Additionally, the inverse CDF requires an integral, so it must be approximated using a Taylor series with Taylor coefficients that can be applied to the uniform distribution. This is not exact, but is very near the ideal inverse CDF.

# Chapter 3

## Profiling of Existing Frameworks

This chapter profiles the two frameworks that will be utilized for PPPFL with Secure MPC and their current limitations.

### 3.1 MP-SPDZ Overview

MP-SPDZ [36] is a MPC framework that allows for secure continuous computation. It is an open-source code base, that can be used for MPC in various applications including machine learning. The code base provides a compiler to convert any program written in a high-level Python language to bytecode. From here, the bytecode runs off of a virtual machine that leverages the computation and communication phases of MPC.

#### 3.1.1 Use Cases of MP-SPDZ

MP-SPDZ was built as an MPC framework to use for everyone since MPC protocols are non-trivial to understand and implement. By providing a framework that takes a program written in Python, it allows more people to utilize MPC without having to research and implement custom programs built specifically for MPC. This use case can be used for industry or research.

MP-SPDZ has already been used frequently in existing works that leverage MPC protocols

due to its open source nature and usability. Additionally, the code is well-maintained, receiving frequent updates, and has extensive documentation.

Another feature of MP-SPDZ is its numerous built-in protocols. While many MPC frameworks may support one or a few protocols, MP-SPDZ offers extensive protocols that support honest-majority, dishonest-majority, passive or active security, and more. This allows the program to be compiled at a high-level and then run for the specific security guarantees that are necessary for the situation.

MP-SPDZ is also one of the few MPC frameworks that supports machine learning with Pytorch built-in. It supports logistic regression, linear regression and more advanced machine learning techniques such as neural networks. This support is only allowed through PyTorch and neural networks are limited to sequential only. This limits the use of networks such as ResNet which has some operations occur in parallel. Essentially, more complex neural networks may not be possible using sequential only.

As machine learning often requires data from multiple sources, it might be necessary that multiple parties provide the input of their data to train a model. MP-SPDZ supports splitting data input between parties in multiple ways. Features or labels can be split between any number of parties or inputted from a single party.

Still, these benefits make MP-SPDZ an extremely useful and easy to use MPC framework. This allows it to take on most of the necessary challenges or tasks that may be presented for industry or research.

### 3.1.2 Related MPC Frameworks

There are many other MPC frameworks outside of MP-SPDZ, each with their own advantages and use cases. While not an exhaustive list, the awesome-mpc [56] is an overview of open-

source MPC frameworks and introductory material. Some of these are more specialized with optimizations for specific protocols to more general that allow greater usability. Here are some specific highly utilized MPC frameworks apart from MP-SPDZ:

1. MOTION [12]: MOTION is an open-source MPC framework that can be leveraged as a tool in research and ideally extend the adoption of MPC protocols in practice. It supports full-threshold boolean and arithmetic GMW and BMR and is secure for semi-honest adversaries. It supports more than two parties up to an unlimited amount of parties and can be used in PPML.
2. EMP-Toolkit [70]: EMP-Toolkit is a open-source MPC framework that supports the special use case of secure two party computation as well as MPC. The framework specifically is tailored towards optimized computation for GC and depending on the sub-framework can be utilized for semi-honest or active security guarantees.
3. CrypTen [40]: CrypTen is a open-source MPC framework built specifically for PPML built for PyTorch applications. Its main focus currently is in research as it is not production ready. The framework is secure against semi-honest adversaries and utilizes secret sharing techniques supporting more than two parties.

### 3.1.3 Compiler in MP-SPDZ

The compiler in MP-SPDZ can take high-level written Python code and transform it into bytecode to be run by the virtual machine. By having a clear way of how secret information is revealed at the virtual machine level, dynamic typing through Python will not reveal any secret information accidentally as it will be processed through the compiler. The compiler leverages the use of a common compiling technique called basic blocks to perform round-

minimization within each block. Round minimization is not done between blocks as it requires re-ordering of instructions.

The main optimization conducted by the compiler is to merge multiple independent instructions of the same type within a basic block. It first creates a dependency graph of all instructions within a basic block. Then it will merge any instructions that can be merged of the same type. The compiler then will handle which round each instruction is assigned and the output is the topological order in the dependency graph.

The compiler also ensures that only if the same exact memory address is called will the instructions not be merged. In this case, there could be a memory conflict and merging is not possible. Additionally, any code that has an unused result may be removed.

### 3.1.4 Virtual Machine in MP-SPDZ

The virtual machine in MP-SPDZ takes in pre-compiled bytecode from the MP-SPDZ compiler and will run the bytecode according to the scheduler file that is provided when a program is compiled. The virtual machine handles all of the specific protocol information for that program as well. The main benefit of the virtual machine in MP-SPDZ is that it allows for communication instructions to have an unlimited number of arguments in the bytecode. If this did not occur, more computation would stall due to waiting for slower communication instructions to complete. This helps reduce the number of communication rounds by allowing parallelization of communication instructions, which tend to be the bottleneck due to network latency.

The virtual machine offers unlimited registers for all basic data types. Registers are hard-coded into the bytecode and typically are storing inputs or outputs of instructions. This is in contrast to the memory which is allocated at compile-time and can communicate information

between threads. The memory provides handling the more complex data types such as matrices compared to the simple basic data types registers hold.

Multi-threading is supported for the virtual machine. It allows for the allocation of a set number of threads when executing the program. One thread is allocated to the main tape to run the program. Tapes are a series of bytecode instructions. The other threads are free to run other tapes as called from the main tape, however, it is limited to the number of threads allocated. This allows for some parallelization of operations.

## 3.2 MP-SPDZ Profiling

MP-SPDZ provides built-in programs for the user to be able to run out of the box. These include some simple neural networks, as MP-SPDZ only supports sequential neural networks, and other secure computation programs. These programs can be run with any protocol supported by MP-SPDZ, assuming the program does not require a specific type of operation not supported by the protocol. For example, a program that requires arithmetic operations cannot be run using GC. The programs also support different optimization techniques, depending on the protocol.

Various programs were run and tested to determine possible limitations of the MP-SPDZ framework. The results display that MP-SPDZ has improvements that can be made to speed-up execution time, especially in certain programs.

### 3.2.1 Parallelization and Limitations

One of the optimizations provided by MP-SPDZ that can be added to any program is the ability to utilize multi-threading and the number of available threads that can be used.

Since MP-SPDZ does not support the use of GPUs in the training of neural networks, multi-threading is essential for increasing performance for operations such as matrix multiplication, which have high parallelization potential.

Our findings display that although these neural networks do benefit from a higher thread count, there is a bottleneck and limit to the number of threads that provide a performance benefit. Below in Figure 3.1 displays the limitations of multi-threading within MP-SPDZ. The setup was the performance of the dense neural network architecture provided by MP-SPDZ [37] running on the MNIST dataset. The figure displays the neural network run for various thread counts locally. These were run using the replicated ring protocol, in a three party setting. This is a semi-honest, honest-majority protocol.

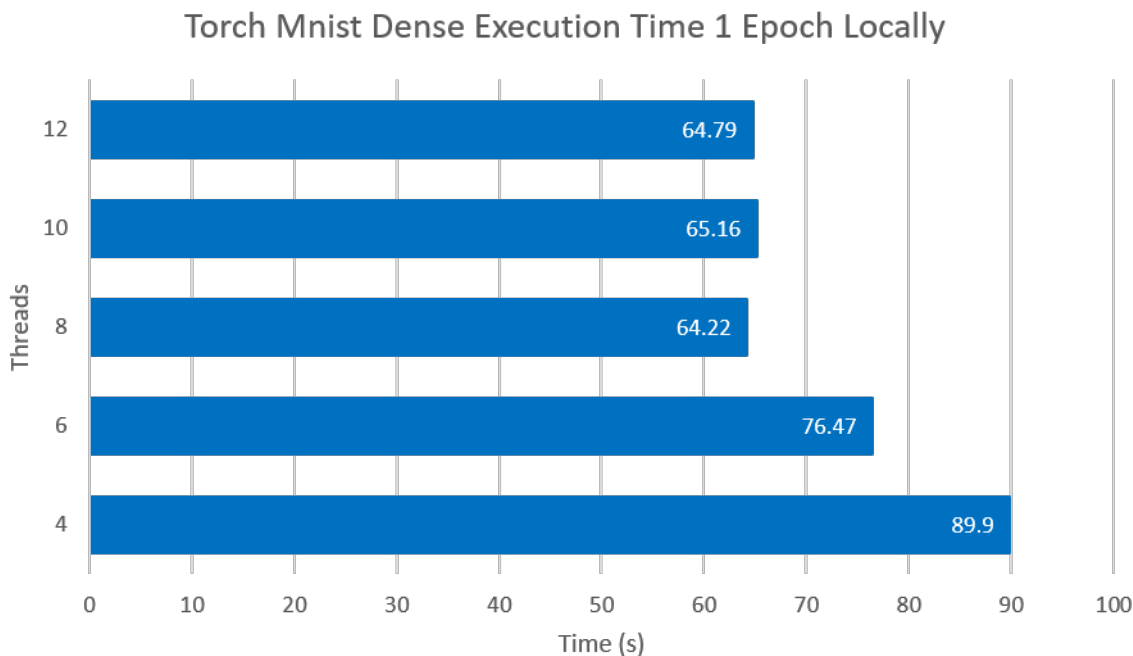


Figure 3.1: Local Parallelization Limitations in MP-SPDZ.

The execution time begins to increase after 8 threads, displaying a distinct lack of high enough multi-threading for certain applications. There are possible explanations for this

that will be discussed in the next section. Below displays the figure for the same setup, just performed over a network connection via TCP. This is shown in Figure 3.2.

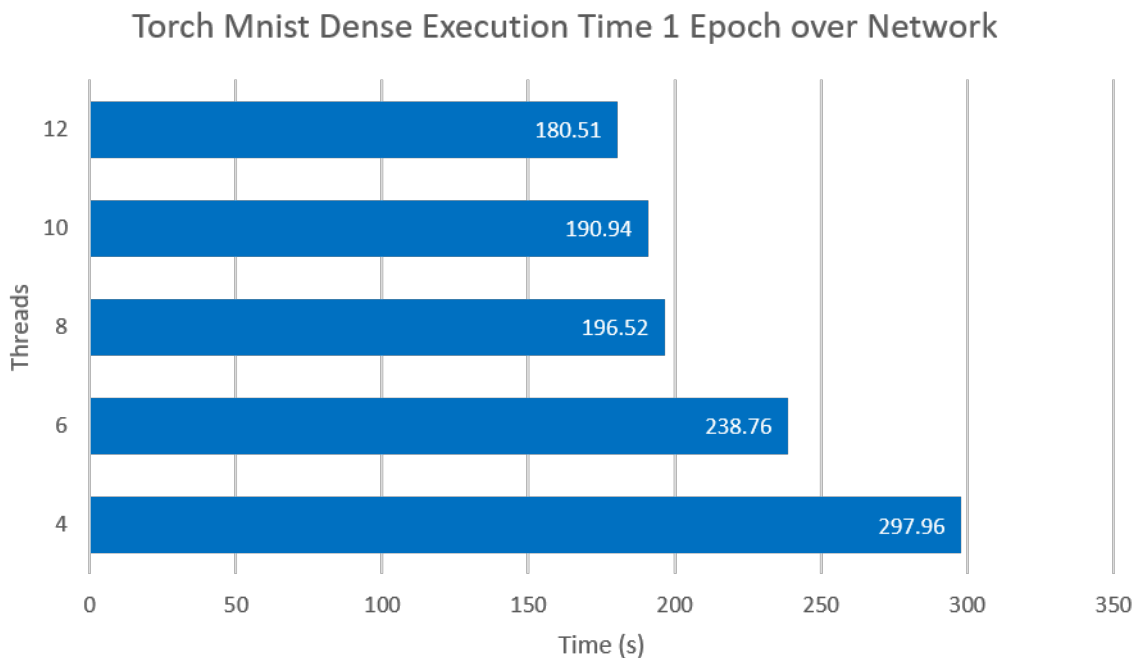


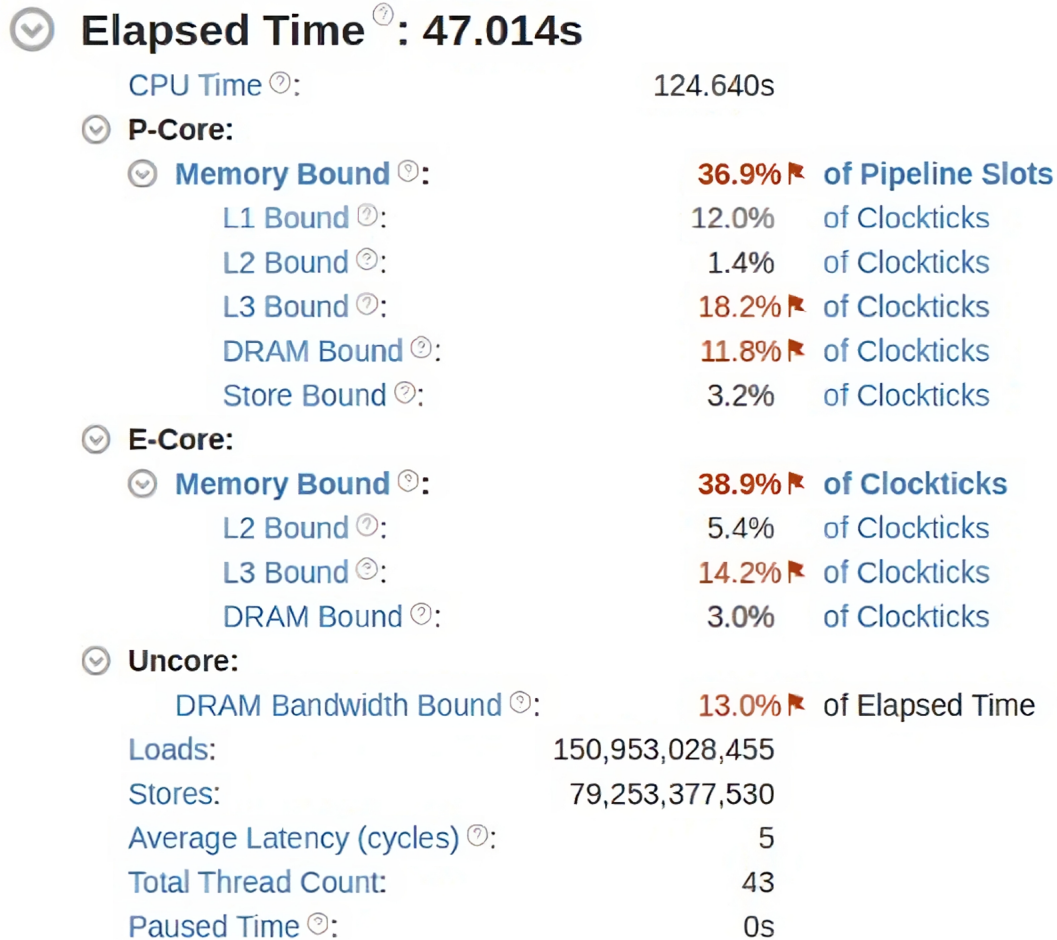
Figure 3.2: Network Parallelization Limitations in MP-SPDZ.

This time more threads does seem to improve execution time. Still, the performance improvements do decrease beyond 8 threads. This makes it harder to properly utilize the machine learning tools in MP-SPDZ due to high execution costs that could be reduced with better parallelization.

### 3.2.2 Memory Bandwidth Usage

The reason for the possible lack of speedup using higher thread counts in MP-SPDZ could be attributed to high memory overheads. When profiling using Intel’s VTune while running the default MP-SPDZ programs, the results yielded essential information. In Figure 3.3, the VTune profiler indicates that the threads are in deed limited by lack of memory and not due

to poor MP-SPDZ multi-threading capabilities. The red highlighted values along with the flags indicate possible large bottlenecked areas for the program. Lower values indicate that the program is efficient and not hampered by a specific issue.



*\*N/A is applied to metrics with undefined value. There is no data to calculate the metric.*

Figure 3.3: Memory Utilization in MP-SPDZ.

The figure shows it is possible the lack of increase in execution time for higher thread counts in MP-SPDZ is due to memory limitations. Although this at least indicates MP-SPDZ multi-threading could be better harnessed, being capped for memory usage in a 128 GB RAM setting, is rather disappointing. A program such as this, while expected to have high

memory usage, should not need server levels of RAM to boost performance on a relatively simple neural network even when occurring locally.

### 3.3 SAFEFL Overview

SAFEFL [26] is a federated learning framework that combines federated learning with MPC to provide secure federated learning, protecting against privacy inference attacks and poisoning attacks. It leverages the MP-SPDZ library to provide MPC for secure aggregation and utilizes a PyTorch communicator between the high-level models and MP-SPDZ.

#### 3.3.1 PyTorch Communicator: For Client-Server Communication

For secure FL to connect the client to the secure MPC aggregation process, the clients and servers must communicate. The SAFEFL PyTorch communicator provides this design functionality. The first module is the training module, which takes place locally on the client’s device using the neural network architecture. The second is the aggregation module, which allows for the secure aggregation to take place inside of MP-SPDZ.

The communicator allows for the models to be run locally by the clients using PyTorch in plain text, removing the expensive costs associated with MPC if running the setup fully over MPC. The gradients from the clients are then passed to the MPC servers who perform MPC aggregation on the gradients to receive the newly updated model. The updates are then securely shared back to the clients and further rounds will occur until all gradients have been passed through the secure aggregation scheme and the model can be updated. Additionally, having clients train locally and then aggregate the gradients allows for non-sequential neural networks to take place, which MP-SPDZ does not natively allow and requires specific

workarounds to implement something such as ResNet.

### 3.3.2 Secure Aggregation with MP-SPDZ in SAFEFL

While MP-SPDZ normally allows for a massive range of protocols options, SAFEFL has currently limited the number of protocols that may be used in MP-SPDZ to four. This is not a limit with the protocols, rather the engineering work behind SAFEFL's main function. These four protocols are all run using rings in 64-bit computation and allow for two or three servers to be run, depending on the protocol. Additionally, they cover all bases of honest-majority, dishonest-majority, semi-honest adversaries, and malicious adversaries.

MP-SPDZ has been modified to allow for the specific network requirements necessary for SAFEFL and their secure aggregation scheme. Because MP-SPDZ assumes that all computation occurring is in the secure framework, there is no need for dedicated MPC servers and clients. SAFEFL has modified this to allow for MPC clients who will secret share their gradients among the servers and then at the end receive the updates to the model. It should be noted once the servers receive their shares, the computation will occur as normal as specified by the MPC protocol.

Secure aggregation in SAFEFL occurs using the PyTorch communicator with the MP-SPDZ framework. Although many aggregation schemes are available for plain text use, only two schemes are available in secure aggregation, the federated averaging [46] and FLTrust [14]. FLTrust offers further protection as it is a robust aggregation scheme, which will be discussed in the next section.

The aggregation schemes implemented in MP-SPDZ do not fully mimic their plain text counterparts one to one. For example, the federated averaging scheme multiplies the gradients by the amount of training data each worker holds and adds all of these values together. It then

divides by the total data held by all workers. In contrast, the federated averaging scheme in MP-SPDZ just adds up the gradients and divides by the number of workers. This is not exact as the plain text example, however, it reduces computation costs and be relatively close. In the case that each worker holds the same amount of data, it would be one to one.

### 3.3.3 Robust Aggregation

SAFEFL provides numerous robust aggregation schemes that protect the server from potential attacks by the clients. Most of these schemes are implemented only for the plain text setting, however, FLTrust can be used with secure aggregation. They use a variety of methods ranging from differential privacy to comparing the similarity between the updates.

There are a few reasons that the robust aggregation schemes have likely not all been implemented in MP-SPDZ. One is that MP-SPDZ does not support all operations that may be necessary for the aggregation schemes. For example, there is no built-in way to get the median from an array in MP-SPDZ. This is essential to the median aggregation scheme [74] and would require a workaround to use with secure aggregation. The second reason is that it requires understanding the MP-SPDZ protocols and functions that can be utilized during a program. This takes time and is not always clear. Finally, not all robust aggregation schemes need to be implemented. By implementing FLTrust, it does provide the clients and servers a choice if they wish to engage in both secure and robust aggregation.

### 3.3.4 Limitations

SAFEFL does provide a strong starting point for secure aggregation with MPC, however, it suffers from a few limitations. For starters, it is using native Python lists and list comprehension compared to Numpy arrays for holding the values that are involved in the client

interface. Lists are fine for smaller amounts of values as in the original SAFEFL design, however, when using millions of values for neural networks, lists become much slower than Numpy. This is displayed in the bar chart in Figure 3.4 [60]. Additionally, as list comprehension becomes larger and larger, it will consume much more RAM. This is once again fine for a small list, but when dealing with millions of values, this is large enough to crash client computers who may not have as powerful machines.

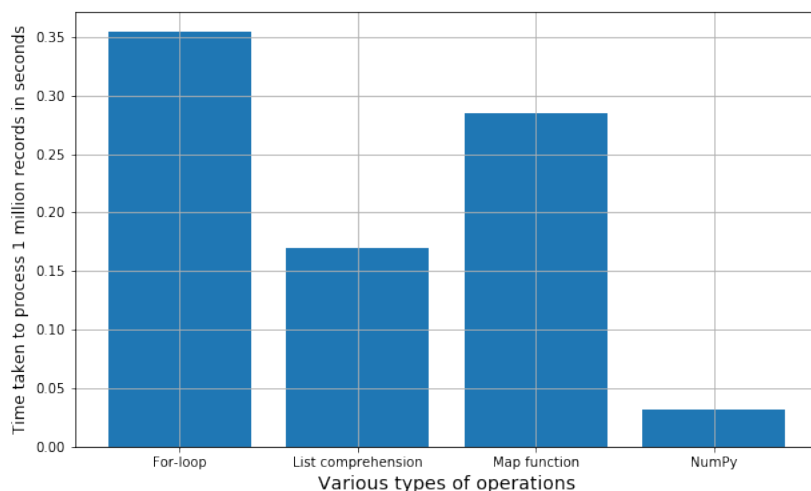


Figure 3.4: List Comprehension vs. Numpy in Python.

The use of custom ring defined objects to represent the MP-SPDZ input and output is also extremely slow. It does allow range of use beyond 64-bit ring integers, however, it will cause slowdowns when compared with using built-in data types. Using an unsigned long long data type works the exact same as custom 64-bit ring object and is built-in to Numpy and other data objects.

SAFEFL also does not use fast operations on the lists of custom ring objects to convert them to and from byte strings, which are necessary for communication over the network. The design packs and unpacks these values one at a time, which is incredibly inefficient. Further, appending to the byte string becomes much slower as it piles up into millions of characters. Reducing these inefficiencies is central to fixing some of SAFEFL's execution

time limitations.

# Chapter 4

## Proposed Framework

In this chapter, the proposed design is discussed in detail and how it differs from the original design. The chapter first gives an overview of the proposed design. Then, a detailed design will highlight the areas of improvement for PPPFL with Secure MPC compared with the original SAFEFL design. Following the overviews, specific optimizations will be described and how they were implemented. The design is improved through the usage of Numpy, Cython, some buffer unpacking, and server optimizations.

### 4.1 Proposed Design Overview

#### 4.1.1 Threat Model

The threat model was evaluated for two different scenarios, active and passive adversaries. In both cases, three parties were used with honest majority. Replicated secret sharing was the primary protocol and the clients were assumed to be honest, whereas the servers may have corruptions. Any speedups listed in the proposed design section are relative to the semi-honest case, however, when the evaluation and results section is shown, both the malicious and semi-honest cases will be presented. Table 4.1 displays the differences for semi-honest versus malicious security. Input masking will be explained in subsection 4.2.3, triple verification in subsection 4.2.1.2, and truncation in subsection 4.3.2.

	Semi-Honest	Malicious
MP-SPDZ Protocol	Replicated Ring Sharing	Post-Processing Replicated Ring Sharing
Input Masking	Optional	Necessary to prevent gradient leakage
Verification Triple	Not Needed	Detect malicious modification of results
Truncation	Can apply	Cannot apply

Table 4.1: Semi-Honest vs. Malicious Protocols

### 4.1.2 System Overview

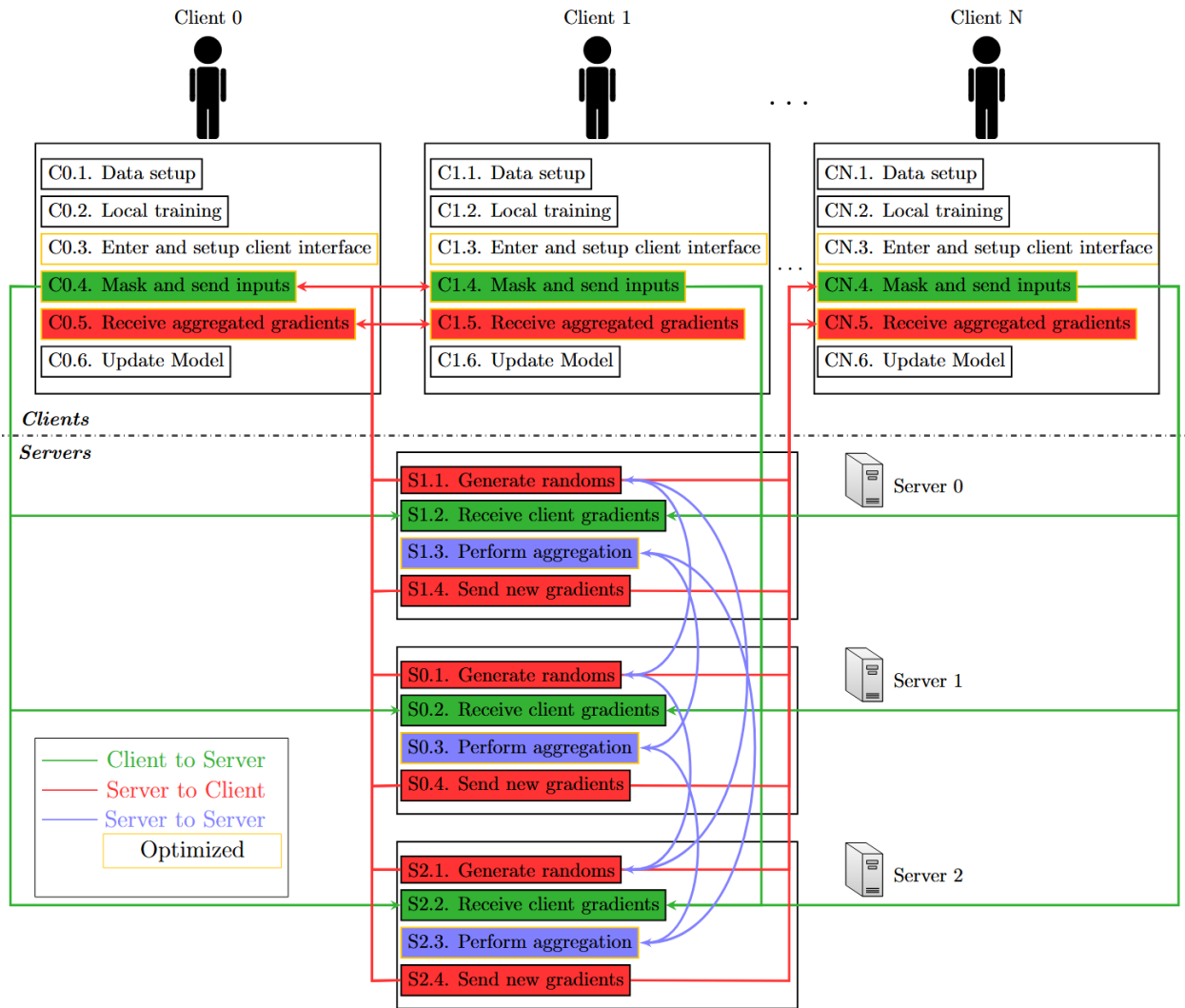


Figure 4.1: Overview of Proposed Design.

In Figure 4.1, a general overview of the proposed design is shown. There are phases that

display which areas will be optimized for execution time, along with all portions of communication between the aggregating MPC servers and the clients. The numbers in each box for the figure begin with C for client or S for server and their number and then the step in their specific process. For example, CN.5. corresponds to client N's step 5.

The list below describes what each step does and why if necessary. It also lists if communication occurs and if so who it is between.

1. C.1: Performs any data pre-processing for the model for better accuracy. This step does not require communication.
2. C.2: Performs model training. This step requires no communication.
3. C.3: Gets the gradients from local training and enters the client interface and performs any setup such as converting floats to integers as is necessary for MPC. This step requires no communication.
4. S.1: Generates random values that are required for the client to input mask their gradients and sends them to the client. This step requires communication between servers and then servers to clients.
5. C.4: Performs input masking to ensure the gradients are not learned by the servers and then sends them to the servers. This step requires communication from the servers to clients and then clients to servers.
6. S.2: Receives the gradients from every client. This step requires communication from clients to servers.
7. S.3: Performs the aggregation of the gradients. This step requires communication between the servers.

8. S.4: Returns the updated gradients to every client. This step requires communication from the servers to clients.
9. C.5: Receives the new aggregated gradients from the servers. This step requires communication from servers to clients.
10. C.6: Updates the model using the gradients and then would continue again at C.2. This step requires no communication.

The areas most subject to improvements in execution time generally come in steps with communication between the clients and servers, which will be shown in the evaluation chapter 5. In SAFEFL’s original design, the servers or clients may be waiting on one another to finish certain operations during the networking phases. This is not ideal as networking should be the most limiting factor to the design.



Figure 4.2: SAFEFL Original Design.

Figure 4.1 shows end-to-end federated learning process. To further discuss the proposed design for secure aggregation with MPC, the design of SAFEFL for the secure aggregation phase, C.4-C.5 in Figure 4.1 is displayed in Figure 4.2.

1. End-to-end Aggregation (Orange). This is the overall secure aggregation process.
2. MPC Server-Side (Blue). This is the server-side MPC portion of the aggregation.
3. Other (Light Blue). This is miscellaneous items such as setting up variables or waiting for the server to finish the program after gradients have been returned.
4. Convert Inputs (Red). This is the conversion of the gradients from floats to integers as is required for input into MPC.
5. Establish Connection (Green). This the TCP and secure handshake setup between each client and server.
6. Input Masking (Brown). This is the masking of the client gradients using the server generated randoms.
7. Pack Inputs to Bytes (Purple). This is converting the inputs from integer values to a byte string.
8. Send Inputs (Pink). This is the client sending the byte string to the servers.
9. Receive Outputs (Gray). This is the client receiving the aggregated gradients from the servers.
10. Convert Outputs (Light Yellow). This is the conversion of the gradients from integers back to floats.

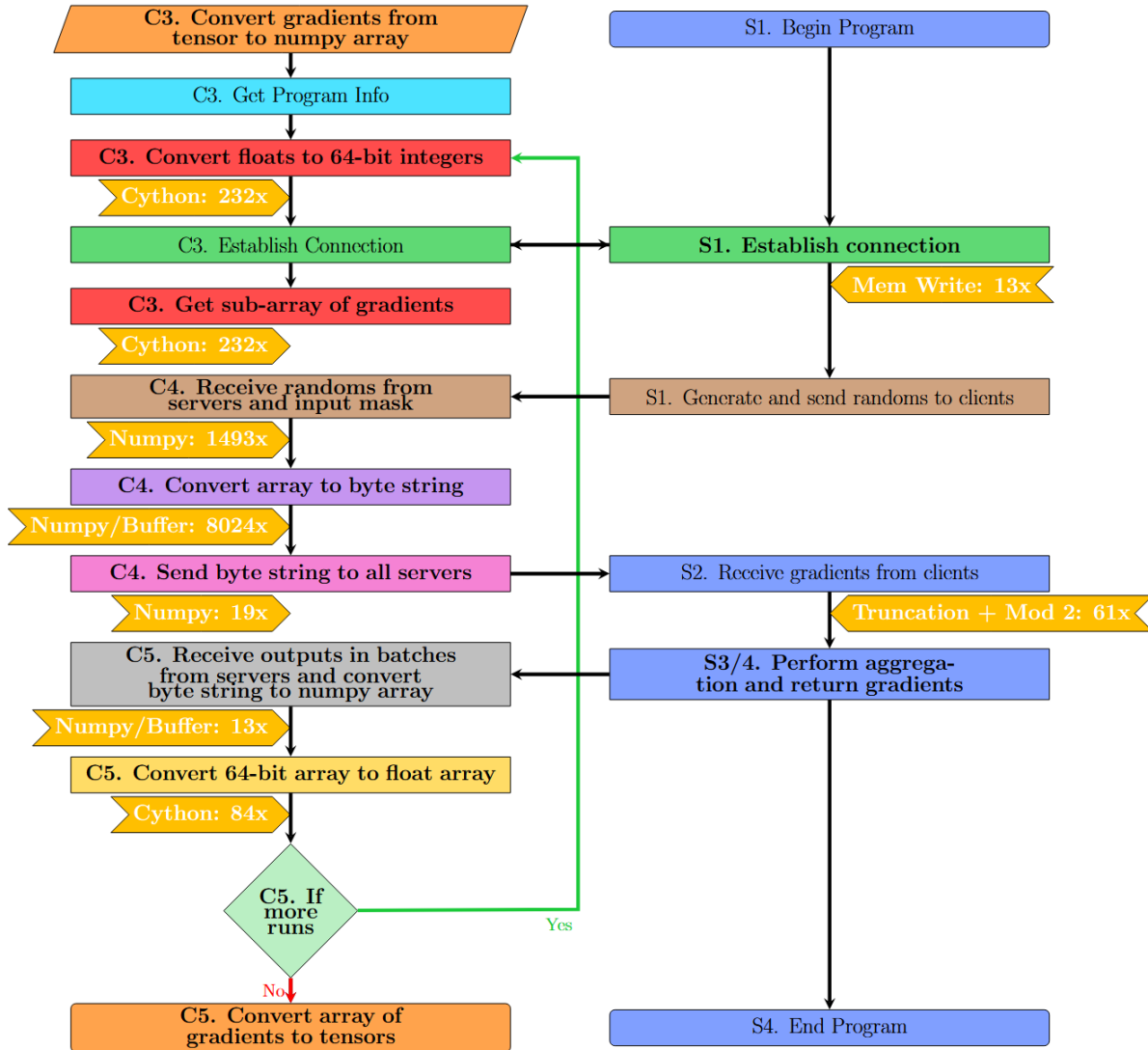


Figure 4.3: Detailed Proposed Design Flowchart with Optimizations and Speedups.

Some do not appear in Figure 4.2, but will appear in Figure 4.3

As mentioned previously, SAFEFL suffers from poor data structure choices and data object choices that hamper overall execution time for the secure aggregation protocol. Particularly in the random phase, colored brown, the client interface is the bottleneck and not the MPC framework or networking. This is true for almost all steps in the SAFEFL design; however, it is most prominent for the client and MPC setup, excluding the conversion from a tensor

to list and establishing the connection with the servers.

The detailed flowchart of the secure aggregation process for PPPFL with Secure MPC is shown in Figure 4.3. The colors match the same timed areas between SAFEFL and the proposed design. The bold steps represent an area that has been optimized in some way. Those bold steps also have a banner below them with the listed optimization and improved speedup compared to SAFEFL.

The speedup for various steps is rather remarkable. By utilizing Numpy in particular, speedups of over 1000x can be achieved. The Cython portions also perform remarkably well. Overall, by utilizing improved data structures and objects, the speedup can turn SAFEFL into a more practical framework.

## 4.2 Client Optimizations

SAFEFL's original design for transforming and inputting data to the servers in MP-SPDZ required list comprehension and custom objects for each integer. This was highly inefficient as list comprehension may work for the Linear Regression case where the number of model parameters is fairly low. This becomes an issue when trying to run list comprehension over millions of values as that will cost a high amount of memory and computation power.

Additionally, since each integer was represented by a custom object, any operations would be extremely slow. This, once again, was fine for the case of a small number of parameters, but does not work well for larger models. Since the ring being used was of 64-bits, this allows for easier and better optimizations than would be possible with a 128-bit ring. It would be possible to create a method of handling a larger than 64-bit ring, however, it would not be able to leverage some of the specific optimizations provided here. With the 64-bit

ring, Cython and NumPy arrays can be used to vastly improve performance compared to the old methods in SAFEFL. Additionally, unloading the buffer when receiving values from the servers more often can also provide an uplift in the latency.

### 4.2.1 Cython

Cython is a programming language that allows the use of C and Python code in one file. This provides performance gains as C code can run much faster than Python, due to it compiling prior to runtime. Cython files similarly also have to be compiled prior to runtime. The advantage of Cython is it can interact with regular Python code. Input can be provided from a Python file into a Cython file and output from a Cython file to a Python file. This makes it easy to integrate Cython into already built Python frameworks, such as SAFEFL.

#### 4.2.1.1 Converting Floats to Ring Integers

The proposed design leverages Cython to convert the gradients, which are 64-bit floats, to 64-bit integers. The client input into MP-SPDZ must be in integers over a ring or field, so this conversion is necessary. In the proposed design, the 64-bit floats exist in a NumPy array instead of a list. This is done because it can be passed in properly to a Cython function and indexed or modified as needed. The NumPy array is passed into a Cython function that converts the floats to 64-bit ring integers by multiplying the floats by  $2^{16}$ . 16 is used here since that is our precision, and the default precision provided by MP-SPDZ. Any operations on these new values will function as a ring, if overflow occurs, the integer will loop back around to start at 0. This value is then cast to a type of unsigned long long. This will remove the remaining fractional part and correctly convert the integer portion to a positive 64-bit integer. If the original integer part is negative, it will correctly cast it to the upper

half of the 64-bit range, just like a ring. This works since modern CPUs work off of 64-bit architecture, which is what unsigned long long is tied to. In future, if processors expand to larger than 64-bit, this would need to be modified, however, it would not require much change.

#### 4.2.1.2 Verifying Triples

If malicious security is assumed for the protocol, the clients will need to verify the triples they receive back from the servers. The triples are in the form of [a], [b], [ab]. The value of [a] will be used for input masking or is the final returned value depending on the stage. If the triple does not match the expected output, the clients will still detect the issue and abort with any continued communication or computation with the servers. This function can be utilized for both the input masking and verifying the output correctness.

In the new design the clients after receiving the triples will use a Cython function to index the Numpy array. The previous way of indexing used a list within a list using the custom ring objects created in Python. This required a significant memory footprint for the list as list comprehension will require loading the entire object into the RAM. Comparatively, Cython and Numpy will only load values one at a time and not require loading the entire array. The clients will also perform the multiplication of the first two numbers in the triple and comparison of that output and the third number in the triple in Cython as well. This allows for faster operations to take place than compared to Python because C arrays are able to be indexed more efficiently than Python.

### 4.2.2 Array Conversion

In order to pass the values to MP-SPDZ, a byte string must be sent by the clients over the socket. Similarly, when receiving the aggregated values back, a byte string will be received by the clients from over the socket. So, it is necessary after the final values are ready to be sent, that the data structure must be converted to a byte string. The original design used Python's `struct.pack` method. While this does work, it is slow and inefficient.

The proposed design has the data in a NumPy array, so it is possible to utilize NumPy's built in functions. These are `NumPy.tobytes()` and `NumPy.frombuffer()`. The first will convert a NumPy array to a byte string. The second will convert a byte string to a NumPy array. These methods are much faster than the previous design and allow the continued use of NumPy arrays throughout the process.

Additionally, throughout the process of receiving values from the servers, byte strings are created and continually concatenated until all values are received from one server. Concatenating the byte string becomes slower as it becomes longer, especially for a byte string millions of characters long. To improve upon this, it is possible to clear the byte string or buffer intermittently and transform it to a small NumPy array. This alleviates the slowness of the concatenation process.

### 4.2.3 No Input Masking

For the semi-honest case, input masking of the client gradients is not necessary. The client can directly pass the values into the servers secretly without revealing any information. This is done by passing in each gradient as a secret, therefore not allowing the servers to access it, and then secret sharing this value and giving a share to each server. This reduces a step in the process that is not necessary. Both designs will be evaluated, however, this is the fastest

one.

Input masking is necessary in the active case to ensure that the servers have not already acted maliciously for two reasons. One, so that no information is learned about the client's inputs since the malicious server may not properly engage in the secret sharing protocol. Two, the client should know before providing input if the server will attempt to act in a malicious way and be able to abort.

### 4.3 Server-Side Optimizations

In addition to the optimizations made on the client side, the servers can optimize their program to alleviate any potential bottlenecks. Some of these optimizations may not work for all protocols. The servers would know prior to running the program which optimizations would be possible.

#### 4.3.1 Removing Memory Write

At the end of the program, the servers write out the memory to a file. This in theory allows the program to use that file for next time the program is run. This, however, is a rather slow process and this will be displayed in subsection [5.5.1](#). that occurs at the end of the program loop while the client is waiting to start the next program. By removing this write section, the next iteration in the aggregation loop between the clients and servers can occur immediately. This does not provide any less security guarantees or potential issues in the computation of the program for any protocol.

### 4.3.2 Probabilistic Truncation

Since MPC only supports integers, division will round to the nearest integer. Depending on the protocol, in order to speed up any division inside the program, probabilistic truncation can be used. This allows the use of randomized rounding based on the input. The probability a value of float remainder of  $x$  after some division is rounded to 0 is  $1-x$ . This allows a relatively accurate method of rounding that improves computation speed based on the input as proposed by [18].

Although this will slightly decrease the accuracy, over the course of training, this error will have a marginal effect on the overall accuracy compared to the performance uplift in execution time.

### 4.3.3 Power of 2 Number of Clients

In addition to using probabilistic truncation, having a power of 2 number of clients will also decrease the execution time of the protocol. This is because the division can be performed using a right shift (i.e., truncation) instead of having to divide. The realistic scenario likely would not have a power of 2 number of clients, however, a power of 2 number could be selected each time aggregation occurs to utilize this speedup. Any clients not chosen in the previous aggregation would have to be chosen in the next round to ensure that the weights do not become overly skewed.

## 4.4 Deep Neural Networks Added to SAFEFL

Originally, SAFEFL only had linear regression as a model available for training. Resnet-18, vgg-11, SimpleDLA, and DenseNet-121 have all been added as models available to train

from scratch with. Since these neural networks are built using some sort of blocks, these models would be easily adaptable to add or remove layers. For example, Resnet-50 could be created fairly easily from the architecture already defined. The base vision transformer from Google using a patch size of 16, 224x224 images, and trained on ImageNet-21k is available for fine-tuning use only and the only current model that can be used for fine-tuning.

#### 4.4.1 Comparison of Models to MP-SPDZ

MP-SPDZ provides some pre-built neural networks that can be trained from scratch or are pre-trained for testing purposes. There are a few key differences between PPPFL with Scure MPC and what is already in MP-SPDZ.

First, MP-SPDZ natively only supports sequential neural networks. Although ResNet has been provided as an implementation for inference only by downloading additional packages. This means that no non-sequential neural networks are provided in MP-SPDZ to train from scratch. The models proposed do have non-sequential elements and are currently unable to be implemented in MP-SPDZ.

Second, the models are set up to be trained in MP-SPDZ itself. There is no interaction with clients in these models. This is different than the SAFEFL framework that does training locally by the clients and then provides aggregation through MP-SPDZ. So, the system architectures are different.

Finally, MP-SPDZ only allows for the use of CPU and does not support the use of a GPU. This means that there is no way to use CUDA and the benefits it provides when training a neural network. So, the training phase will be significantly slower than a similar plain text phase that occurs in SAFEFL locally by the clients on top of the overhead of MPC itself.

### 4.4.2 Model Information

The models that have been added for training from scratch are deep neural networks. For fine-tuning, a transformer model was used. The vision transformer is the base vision transformer from google pre-trained on ImageNet 21k with a patch size of 16 and input size of 224. The number of parameters for each model can be found below. The number of parameters will be multiplied by the number of workers to yield the total number of gradients that will be passed to the servers. For example, if there are 16 workers and the model is VGG-11, then the total number of gradients passed to the servers is 147,697,824.

Model	Parameters
Vision Transformer Base	7690
Linear Regression	30730
DenseNet-121	6956298
VGG-11	9231114
ResNet-18	11173962
Simple DLA	15142970

Table 4.2: Parameters for Each Model in PPPFL with Secure MPC for CIFAR-10.

## 4.5 Dataset Functionality

Previously, SAFEFL only supported loading in one dataset, Human Activity Recognition (HAR). The proposed design allows for datasets from PyTorch or HuggingFace to be used. They also do not require manual annotation of which client holds each piece of data, effective for simulated settings, allowing the user to focus on further advancements for federated learning, differential privacy, multi-party computation or something else.

### 4.5.1 Added Datasets

The reason for adding these datasets was because they are commonly used to test the viability of computer vision models. Currently, PPPFL with Secure MPC supports loading CIFAR-10 from both PyTorch and Hugging Face. In the latter case, this is used only for fine-tuning. Hugging Face was used in this case because the data was formatted in a better way to perform the necessary operations to the dataset prior to training. The dataset is still the same. CIFAR-100 can be loaded in from PyTorch. Although more datasets have not at this time been provided, the method of loading the datasets in has been generalized to make it easier to add future datasets. For example, ImageNet could be added in future with some simple modifications to the CIFAR-10 or CIFAR-100 dataset loaders.

### 4.5.2 Partitioning Data Among Clients

In a more practical setting, each client would have their own individual data and would either be able to easily create a list of their ID for each piece of data or it would not be necessary at all. It is necessary for testing purposes to simulate an environment in which clients data is split non-independently and identically distributed (non-IID). It is most likely that clients would have more of one label than others and possibly a few other randomly distributed data points. So, it was necessary to create a method of partitioning the data for each client that allocates a certain amount of their data to be one label. This value is set as a percentage of the clients total data and can be set as appropriate.

This does require the use of two distinct cases. One where there are more or equal number of clients than labels and one where there are more labels than clients. In the first case, the clients will receive one label where they hold a majority stake. The last few clients may not have a label that has enough to allocate a majority share and will be given as much of

their given label left and the rest randomly distributed. In the second case, each client will receive multiple labels where they have a majority share with the same percentage. This will at least skew the data towards these given data points when each client trains.

This does not provide a perfect replication of a setting where clients may have non-IID, but it does try to mimic a real-world scenario. This partitioning can be done for either the training from scratch or fine-tuning settings and could easily be used with any future datasets added to the framework.

## 4.6 Differential Privacy

Security can also be tightened in other ways outside of the protocol used by the servers. Differential Privacy can be used to help protect the client's data without adding expensive protocol overheads. This can be used in conjunction with the secure computation framework to help ensure the servers do not learn any one individual data point that clients may hold.

### 4.6.1 Use Case: Fine-Tuning the Last Layers in a Model

Differential Privacy can be used when training models from scratch, however, the noise may cause a unacceptable loss in accuracy of the model. The best approach then is to utilize DP when fine-tuning a pre-trained model. This allows the clients to securely mask their data, while retaining high accuracy.

For the vision transformer model that has been implemented for fine-tuning, the CIFAR-10 dataset was loaded and modified for use. Since the vision transformer takes as input images of size 224x224, the CIFAR-10 dataset must be transformed to match this. The images are transformed from 32x32 to 224x224 and normalized using the image mean and standard

deviation of the pre-trained model. This ensures the data is set up in the best way to match the original training done on the vision transformer.

The vision transformer is then loaded in as the model to be used and has all layers frozen, except the linear layer which is modified to match the correct number of class outputs for the CIFAR-10 dataset as compared to the ImageNet-21k. This guarantees that only the last layer will have the gradients modified throughout the process, reducing computation power and still receive high accuracy.

### 4.6.2 Aggregation Scheme for DP

The aggregation scheme for the fine-tuning process is similar to the aggregation done for a model that uses the secure federated learning framework. The gradients will be passed from the clients to the servers using input masking to not reveal the original gradient values held by the clients. Different from training from scratch, the gradients will not encompass the whole model, just the last linear layer that has been added. The gradients then will be added together and divided by the number of workers. The other difference for fine-tuning is that the aggregation scheme then will use differential privacy after this step.

Although gradient clipping is not currently used, it could be a possible additional step before adding the differential privacy. In our case, Gaussian noise was used for differential privacy. Since MP-SPDZ cannot create a Gaussian distribution natively, it requires some tools to create the distribution to be added.

The background for the theory behind implementing differential privacy with inverse transform sampling in MP-SPDZ was previously mentioned in subsection 2.4.1. First, a uniform distribution can be generated in MP-SPDZ. In this case, a uniform distribution from 0 to 1 was used as it is required to use inverse transform sampling. The inverse transform sampling

function is then initialized with the degree of coefficients in the Taylor series for part of the generation of the approximate inverse CDF. The inverse CDF using the Taylor coefficients is then applied to the uniform distribution to generate a very close approximation of the Gaussian distribution with a standard deviation of 1. The distribution is then multiplied by the  $\sigma$  value that has been chosen to finalize the Gaussian distribution matching a similar plain text generation.

# Chapter 5

## Evaluation

This chapter discusses the evaluation setup for both hardware and software platforms as well as the results for testing. The evaluation compares the original and proposed design, how Cython and other optimizations affect the performance and how they fare under different circumstances. The accuracy of the ML model is also compared for the models tested. The differential privacy accuracy is evaluated as well with the privacy guarantee.

### 5.1 Evaluation Setup

For the evaluation, unless otherwise stated, the learning rate was set to 0.1, the batch size set to 64, the number of threads and parallel computations per thread were both set to 8 for MP-SPDZ. Additionally, the dataset was CIFAR-10.

The protocol used in MP-SPDZ was replicated ring sharing for 3 parties in the semi-honest case. For the malicious case, post processing replicated ring sharing was used for 3 parties. To note, the malicious case does not support probabilistic truncation as an optimization and this optimization would fall under the server optimizations category in the latency tables. The other differences are between the two are that input masking is necessary for the malicious case, whereas it is not in the semi-honest version. If input masking is active for semi-honest, it only receives the random number to mask, whereas the malicious case receives triples and has to verify they are correct before masking. The verification would

occur on output as well for the triples in the malicious adversary threat model.

The aggregation in both plain text and MP-SPDZ was federated averaging. The default number of workers used was 16. Generally, VGG-11 will be used as the baseline discussion for latency. For accuracy, all nets will be presented throughout, however, the focus will be on ResNet-18. The speedup will be listed to the nearest integer.

### 5.1.1 Hardware and Software Platform

For the original design and proposed design comparisons, testing occurred on a Intel i9-13900K with 24 physical cores (8 performance cores and 16 efficient cores) per socket and 32 total threads. The computer was equipped with 128 GB of memory. The software platform was Ubuntu 22.04.4. The MP-SPDZ version used was 0.3.8 and the Python version used was 3.7.9. The compilers were gcc 11.4.0 and clang 14.0.0. The Linux Kernel was 6.5.0-27-generic.

For accuracy testing, a Intel Xeon Gold 6348 with 28 physical cores per socket and a total of 56 threads was used. The computer had 128 GB of memory. There were at least two Nvidia GeForce RTX 2080Ti's with 11 GB of VRAM each and for some portions of the testing, three were installed. Only one was used for accuracy testing. This would not affect accuracy. The software platform was Ubuntu 20.04.5. The MP-SPDZ version used was 0.3.8 and the Python version used was 3.8.10. The compilers were gcc 9.4.0 and clang 10.0.1. The Linux Kernel was 5.15.0-56-generic.

The proposed design testing for Section 5.6 occurred entirely on a AMD Ryzen 3900x with 12 physical cores per socket and a total of 24 threads. The computer was equipped with 32 GB of memory and a Nvidia GeForce RTX 3080 with 10 GB of VRAM. The software platform was Ubuntu 20.04.1. The MP-SPDZ version used was 0.3.8 and the Python version used was 3.8.10. The compilers were gcc 11.4.0 and clang 14.0.0. The Linux Kernel was

5.15.0-107-generic.

## 5.1.2 Evaluation Metrics

Two different evaluation metrics were used throughout the testing. The first was the measurement of the latency. The second was the measure of the accuracy.

For the measure of the latency, the execution time was measured for one iteration of training. Generally, the assumption is each iteration has the exact same code to be executed and thus should take roughly the same amount of time for aggregation with some acceptable level of noise.

The accuracy was measured after a number of training iterations. The accuracy was determined by testing the test set against the current model and evaluating how many correct predictions the model yielded. The datasets used were CIFAR-10 and CIFAR-100 and for CIFAR-100, a top 5 class accuracy was also measured.

### 5.1.2.1 Latency

To measure the latency, multiple timers were used. The MPC programs have a built-in timer that measures the time from the moment the program starts to the finish. The default timer package from `timeit` was used on the Python side to measure any client interactions.

Since the server-side MPC program and client interface overlap in their execution at times, some inference was used to determine the actual server-side MPC time for certain scenarios. The connection setup, conversion of inputs to a byte string, sending inputs, and server random time were subtracted from the raw MPC time. This gives a good idea of how much time the MPC program actually takes. The new MPC time subtracted by the raw receive

outputs time yields a relatively good measurement of the actual time this takes on the client side. These adjustments help give a more accurate picture of the MPC execution time and each client execution time phase by eliminating double counting or waiting periods.

These are not perfect estimates of how long the MPC program is actually the bottleneck or the receive outputs takes on the client side. It would require some more testing to get fully accurate for each phase and determine which side is the bottleneck in that phase.

Additionally, as mentioned previously input masking can be entirely removed and replaced with direct secret sharing of inputs from the clients to the servers for the semi-honest case. This will be presented in every semi-honest table to compare the gains made without input masking. Additionally, the row of the preferred design in every table will be bolded.

## 5.2 Standard Deviation of End-to-end Aggregation Execution Time

Throughout testing, the tests were only run once, however, a standard deviation was taken running the same test five times to determine possible variations in numbers. All the latency measurements may see some noise, however, it falls within the acceptable standard deviation. The results are displayed in Table 5.1. These were run for the Cython and optimized version of semi-honest with input masking. Similar results would be expected across the board. The specific steps listed here can be corresponded with the itemized list in subsection 4.1.2.

Overall, standard deviation generally falls within 1-2%, however, a few exceptions do occur. Most are likely due to the very small execution times of those steps or possible dependency on MPC that may be slower from fluctuations in available compute resources.

Standard Deviation of Latency Measurements	
Phase	Standard Deviation as Percentage
Setup	1.173
Training	2.799
End-to-end Aggregation	1.455
MPC server	1.705
Establish Connection	.0775
Convert Inputs	2.866
Pack Inputs to Bytes	2.049
Input Masking	1.219
Send Inputs	2.402
Receive Outputs	3.316
Convert Outputs	4.999

Table 5.1: Standard Deviation by Phases as a Percentage.

### 5.3 End-to-End Latency

The setup latency is a one-time occurrence at the beginning of the protocol. This is primarily data setup, however, the setup of the MPC program also occurs during this time. The training phase includes the training portion, minus the aggregation time. The end-to-end aggregation time is how long the total end to end time from when the workers enter the client interface and begin the aggregation process.

The proposed time is with all optimizations present. The difference between the second and third row is if input masking is used. If it is not, the input from the client is directly secret shared instead of requiring input masking. This was previously explained in subsection 4.2.3. The first table Table 5.2 displays the results for a passive adversary and Table 5.3 displays the results for a active adversary.

The difference in setup time is primarily due to the increased cost of the MPC program setup in the original design. The training time can be equated to the proposed running off of a GPU vs a CPU in the original. The main focus is on the end-to-end aggregation time. This

End to End Latency (s) for Semi-Honest Adversary					
Design	Masking	Setup	Training	End-to-end Aggregation	Speedup
Original	Yes	125.765	7.218	41166.890	1
Proposed	Yes	6.892	.850	20.278	2030
<b>Proposed</b>	<b>No</b>	<b>7.066</b>	<b>0.478</b>	<b>13.700</b>	<b>3005</b>

Table 5.2: End to End Latency of Secure Aggregation for Semi-Honest Adversary.

is the major difference between the two and was the bottleneck in the original design. In the proposed design, the MPC aggregation still is the bottleneck, however, it is now practical to use. Removing input masking yields further gains. The active case displays similar results below.

End to End Latency (s) for Malicious Adversary				
Design	Setup	Training	End-to-end Aggregation	Speedup
Original	35	2.368	46805.728	1
<b>Proposed</b>	<b>7.004</b>	<b>0.477</b>	<b>338.662</b>	<b>138</b>

Table 5.3: End to End Latency of Secure Aggregation for Malicious Adversary.

Overall, the active case sees less performance gains compared to the semi-honest format, however, that is primarily due to MPC program time. Since, the bottleneck has shifted from the client to the servers, it is expected that the servers will take far longer in an active case than a semi-honest case. Still, these overall performance gains are very promising considering probabilistic truncation is not available to further speed the program up. One possibility is to offload the final division to the clients, as that would not leak any secret information and could improve performance.

## 5.4 Cython and Numpy Operations

The improvement in performance by using Cython or Numpy operations ranges depends on the specific task. The speedups in these steps tend to result in the highest overall decrease in

execution time. The specific use of Cython or Numpy will be mentioned in each sub-section.

### 5.4.1 Converting Floats to Ring Integers

The step evaluated in this subsection is described in subsection 4.2.1.1. The speedup in performance from allocating the portion of array to be used to converting the floats to ring integers is triple digits. In the same setting as the original SAFEFL framework, with no other optimizations outside of Cython, the speedup is 181x for input masking. When using all other optimizations, the speedup for random input masking is 232x. The direct secret sharing does see some improved speedup, however, this is likely due to decreased computation resources being used. These are shown below in Table 5.4.

Convert Inputs Time (s) for Semi-Honest Adversary				
Masking	Cython	Server Optimizations	Convert Inputs Time (s)	Speedup
Yes	No	No	179.019	1
Yes	Yes	No	0.989	181
Yes	Yes	Yes	0.772	232
<b>No</b>	<b>Yes</b>	<b>Yes</b>	<b>0.575</b>	<b>311</b>

Table 5.4: Convert Floats to 64-bit Integers Execution Time for Semi-Honest Adversary.

The increase in Table 5.4 using server optimizations is almost entirely due to the decrease in the number of gradients. Since the server optimizations uses 16 workers compared to 20 in the other two cases, there will be roughly 20% less data to convert. This matches up closely with the decrease in execution time.

For active security, a similar result is shown below in Table 5.5, however, larger gains are seen. This is primarily because of slower conversion in SAFEFL for malicious when compared to semi-honest.

The conversion of the outputs back to floats from the 64-bit ring integers does not see as large increase in speedup, however, this may be due to list size being significantly smaller

Convert Inputs Time (s) for Malicious Adversary			
Cython	Server Optimizations	Convert Inputs Time (s)	Speedup
No	No	929.912	1
Yes	No	0.745	1248
<b>Yes</b>	<b>Yes</b>	<b>0.618</b>	<b>1505</b>

Table 5.5: Convert Floats to 64-bit Integers Execution Time for Malicious Adversary.

since only the aggregated gradients are needed to be converted, not every clients'. As the list grows, the list comprehension used in the original design likely increases exponentially. Still, the speedup is over 70x for semi-honest and over 85x in the malicious scenario.. The semi-honest results are presented in Table 5.6

Convert Outputs Time (s) for Semi-Honest Adversary				
Masking	Cython	Server Optimizations	Convert Outputs Time (s)	Speedup
Yes	No	No	1.851	1
Yes	Yes	No	0.026	71
Yes	Yes	Yes	0.022	84
<b>No</b>	<b>Yes</b>	<b>Yes</b>	<b>0.023</b>	<b>80</b>

Table 5.6: Convert 64-bit Integers to Floats Execution Time for Semi-Honest Adversary.

There is a decrease in the speedup when the optimizations are present. The differences between the runs do fall within the standard deviation. Relatively similar results are presented for the active scenario below in Table 5.7.

Convert Outputs Time (s) for Malicious Adversary			
Cython	Server Optimizations	Convert Outputs Time (s)	Speedup
No	No	3.1	1
Yes	No	0.033	94
<b>Yes</b>	<b>Yes</b>	<b>0.035</b>	<b>89</b>

Table 5.7: Convert 64-bit Integers to Floats Execution Time for Malicious Adversary.

The speedup here is greater than the semi-honest case, however, that is primarily due to slower execution time for the original design. The execution times for converting the outputs

match up almost identically for semi-honest and malicious security when optimized, which makes sense.

## 5.4.2 Input Masking

The step evaluated in this subsection is described in subsection 4.2.3. For the server random generation execution time, only the results with input masking are presented. This is because if there is no input masking, no server random generation occurs and the execution time will be 0. The server random time was affected by multiple Numpy optimizations as well as the buffer optimization.

Server Random Time (s) for Semi-Honest Adversary			
Cython	Server Optimizations	Server Random Time (s)	Speedup
No	No	40350.583	1
Yes	No	6.338	6366
<b>Yes</b>	<b>Yes</b>	<b>5.029</b>	<b>8024</b>

Table 5.8: Server Random Time for Semi-Honest Adversary.

The results for a semi-honest adversary are astounding. Over 8000x speedup in the fully optimized design displays the practicality that PPPFL with Secure MPC brings to the table. The active results are less startlingly, however, still remarkably good. The one addition with the malicious adversary is that the verification of the triples is now performed in Cython. The active results are presented below in Table 5.9.

Server Random Time (s) for Malicious Adversary			
Cython	Server Optimizations	Server Random Time (s)	Speedup
No	No	43885.664	1
Yes	No	52.973	828
<b>Yes</b>	<b>Yes</b>	<b>42.290</b>	<b>1038</b>

Table 5.9: Server Random Time for Malicious Adversary.

The overall results in both cases are quite an improvement over the design in SAFEFL. The likely bottleneck in the active case is now the MPC program since triple generation is slow.

### 5.4.3 Array Conversion

The step evaluated in this subsection is described in subsection 4.2.2. The execution time speedup by using the built-in NumPy array operations to convert arrays to byte strings provide the single largest uplift in terms of ratio. The results for semi-honest adversaries can be found in Table 5.10 below. The speedup is over 1000x for all cases, which is a remarkable gain. Once again, the increase using optimizations likely comes from a decreased total number of gradients since there are less workers.

Convert Masked Inputs to Byte String Time (s) for Semi-Honest Adversary				
Masking	Cython	Server Optimizations	Convert to Bytes Time (s)	Speedup
Yes	No	No	398.702	1
Yes	Yes	No	0.334	1194
Yes	Yes	Yes	0.267	1493
<b>No</b>	<b>Yes</b>	<b>Yes</b>	<b>0.268</b>	<b>1488</b>

Table 5.10: Convert Masked Inputs to Byte String Execution Time for Semi-Honest Adversary.

The active results in Table 5.11 show a little more modest gains, however, can see over 1000x improvement in execution time. The likely increase in execution time compared to the semi-honest scenario is because of decreased compute resources available as the MP-SPDZ program is still running in the background. In a situation where each server was on a different machine from the clients as intended, these gains could possibly be higher.

The conversion from bytes to 64-bit integers occurs within the unpacking of the buffer, so it will be included in the section below.

Convert Masked Inputs to Byte String Time (s) for Malicious Adversary			
Cython	Server Optimizations	Convert to Bytes Time (s)	Speedup
No	No	539.279	1
Yes	No	0.336	1605
<b>Yes</b>	<b>Yes</b>	<b>0.270</b>	<b>1997</b>

Table 5.11: Convert Masked Inputs to Byte String Execution Time for Malicious Adversary.

#### 5.4.4 Unpacking Buffer Intermittently

The step evaluated in this subsection is described in subsection 4.2.2. Unpacking the buffer does see a noticeable improvement, however, it is more modest when compared to some other improvements. This improvement also encompasses the conversion from bytes to a NumPy array. The results for the semi-honest adversary are shown in the table Table 5.12 below. This is the receive outputs execution time that encompasses both of these improvements and the actual receiving of the outputs from the servers.

Receive Outputs Execution Time for Semi-Honest Adversary				
Masking	Cython	Server Optimizations	Receive Outputs Time (s)	Speedup
Yes	No	No	47.010	1
Yes	Yes	No	6.528	7
Yes	Yes	Yes	3.635	13
<b>No</b>	<b>Yes</b>	<b>Yes</b>	<b>3.101</b>	<b>15</b>

Table 5.12: Receive Outputs Execution Time for Semi-Honest Adversary.

The active results see more gains compared to the semi-honest results as displayed in Table 5.13. The reason for this is primarily from the increased execution time in SAFEFL, since there are triple the amount of values being received by the clients.

Overall, the decrease in the optimized versus server optimized versions is likely due to the decrease in workers and hence total parameters passed to the MPC program. The gains are likely more modest here since the outputs are a fraction of the inputs. Additionally, since this does include the networking time, this is likely the biggest bottleneck. The other

Receive Outputs Execution Time (s) for Malicious Adversary			
Cython	Server Optimizations	Receive Outputs Time (s)	Speedup
No	No	225.13	1
Yes	No	4.495	50
<b>Yes</b>	<b>Yes</b>	<b>2.066</b>	<b>109</b>

Table 5.13: Receive Outputs Execution Time for Malicious Adversary.

possible note is that due to the modification from the raw numbers, these may have some more noise and not be fully representative of the gains in efficiency.

## 5.5 Server-Side Optimizations

The server optimizations provide varying levels of execution time improvement. Some of these optimizations do not fall under the specific individual sectioned timers, however, they can be inferred as there are very few steps that are not timed. These are what have been previously referred to as server optimizations. All of these are compared directly after Cython has already been applied. The reason for this is, these optimizations are not as relevant for the original design since that bottleneck is by the client. After Cython, the proposed design has a bottleneck by the MPC program.

### 5.5.1 Removal of Memory Write

The step evaluated in this subsection is described in subsection 4.3.1. The removal of the memory write at the end does not fall under a direct timer, however, since it does not affect any of the other times, it was grouped into other. The increase is not massive, however, sees a decent speedup considering this is one of the last steps to be optimized. The results for the semi-honest adversary are seen in the table Table 5.14 below.

Other Execution Time (s) for Semi-Honest Adversary			
Masking	Memory Write Removal	Other Time (s)	Speedup
Yes	No	22.26	1
Yes	Yes	1.723	13
<b>No</b>	<b>Yes</b>	<b>1.444</b>	<b>15</b>

Table 5.14: Other Execution Time for Semi-Honest Adversary.

Slower improvements are seen in the active results below in Table 5.15, but still a gain in execution time. The difference between the un-optimized execution times in semi-honest and malicious can be attributed to a higher wait time for the virtual machine to finish writing the memory, since it has to write more memory for the active program. Afterwards, the higher execution time is likely due to MPC setup and resolving any necessary instructions at the end of the program that take longer in an active environment.

Other Client Execution Time (s) for Malicious Adversary		
Memory Write Removal	Other Client Time (s)	Speedup
No	23.232	1
<b>Yes</b>	<b>3.895</b>	<b>6</b>

Table 5.15: Other Client Execution Time for Malicious Adversary.

This improvement does impact performance by a relatively large amount considering the ease of removing it from the virtual machine. It is possible that further optimizations inside the virtual machine may be made for this specific use case, however, that is left to future work.

### 5.5.2 Probabilistic Truncation and Modulo 2 Workers

The step evaluated in this subsection is described in subsection 4.3.2. The results for using probabilistic truncation and a modulo 2 number of workers for semi-honest are shown in Table 5.16. The improvements display the necessary tools to be used to further decrease the

execution time of the secure aggregation protocol after the client side has been removed as the bottleneck.

MPC Server Time (s) for Semi-Honest Adversary			
Masking	Truncation & Mod 2 Workers	MPC Server Time (s)	Speedup
Yes	No	232.690	1
Yes	Yes	3.812	61
<b>No</b>	<b>Yes</b>	<b>3.466</b>	<b>67</b>

Table 5.16: MPC Execution Time for Semi-Honest Adversary.

The active results are displayed in Table 5.17 with more limited gains. This is likely a result of the lack of probabilistic truncation that can be used. Still, the performance increase does help significantly compared to the original SAFEFL design.

MPC Server Time (s) for Malicious Adversary		
Mod 2 Workers	MPC Server Time (s)	Speedup
No	1003.448	1
<b>Yes</b>	<b>283.454</b>	<b>4</b>

Table 5.17: MPC Execution Time for Malicious Adversary.

Overall, the gains in the semi-honest case are large enough to yield a highly practical secure aggregation scheme within MP-SPDZ. The active case does see an improvement, however, is still relatively slow compared to what would be ideal. One possibility as mentioned previously is to move the division to the client side as that does not compromise security. Since the active protocol does not have probabilistic truncation, this may be a necessity for practical use cases.

## 5.6 Latency for Different Settings

A few different scenarios were tested to gather information on how they affect the execution time for the secure aggregation. The number of workers, model size, and networking settings

were deemed the 3 most important to determine their effect. In these cases, it was tested for semi-honest security only and similar results would likely be displayed for the malicious case.

### 5.6.1 Number of Clients

The number of workers (i.e., clients) will affect the execution time of the aggregation protocol, however, this result is fairly linear. There are two cases, one for the non-modulo 2 number of workers and one for the modulo 2 number of workers. Non-modulo 2 number of workers is presented in Figure 5.1 and a modulo 2 number of workers is presented in Figure 5.2.



Figure 5.1: Number of Workers Latency for Secure Aggregation Scaling.

The scaling does appear to get better as the number of workers increases. This is likely because there is some overhead in the setup of the servers. This is also the case for a modulo 2 number of workers. There is a slight dip in that may be due to some noise, however, the

trend-line follows a generally linear pattern.

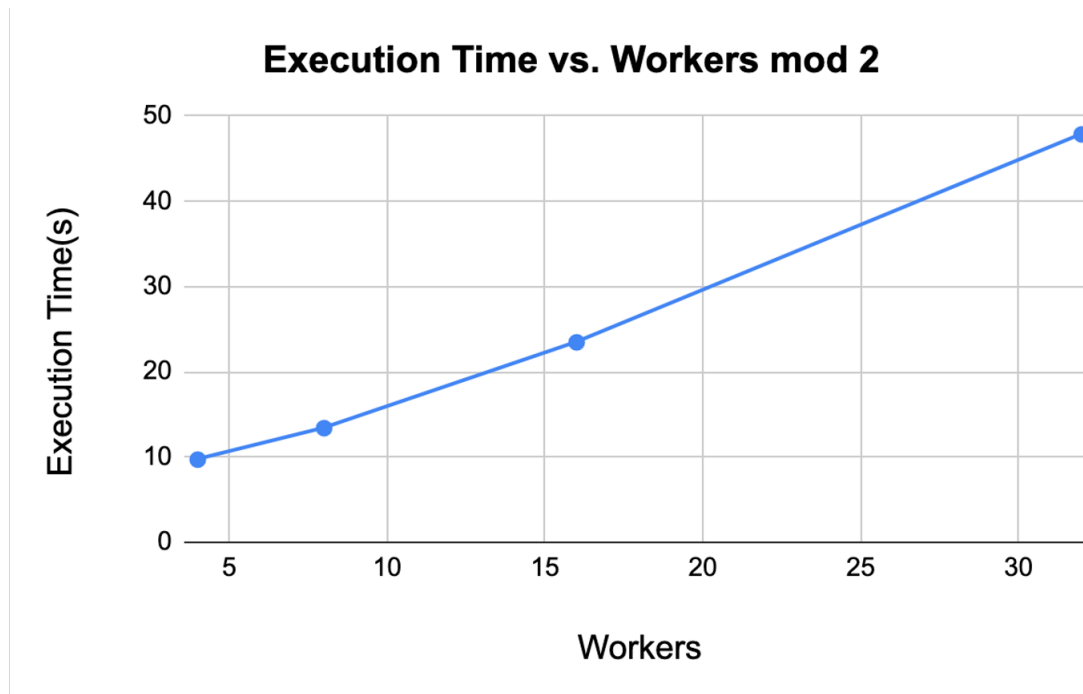


Figure 5.2: Number of Modulo 2 Workers Latency for Secure Aggregation Scaling.

Overall, the results display that increasing the number of workers does not cause an exponential increase in the execution time. Generally, the execution time increases close to linear or slightly better. There is some noticeable overhead with a lower number of workers due to setup time.

### 5.6.2 Neural Network Size

The size of the neural network can be simulated using randomly generated 64-bit floats to pass into the client interface for testing the scalability for the number of parameters. The Figure 5.3 below displays the latency for the number of parameters that would be passed into MPC. This would be equivalent to the number of parameters in the neural network multiplied by the number of workers. All optimizations were used in this scenario.

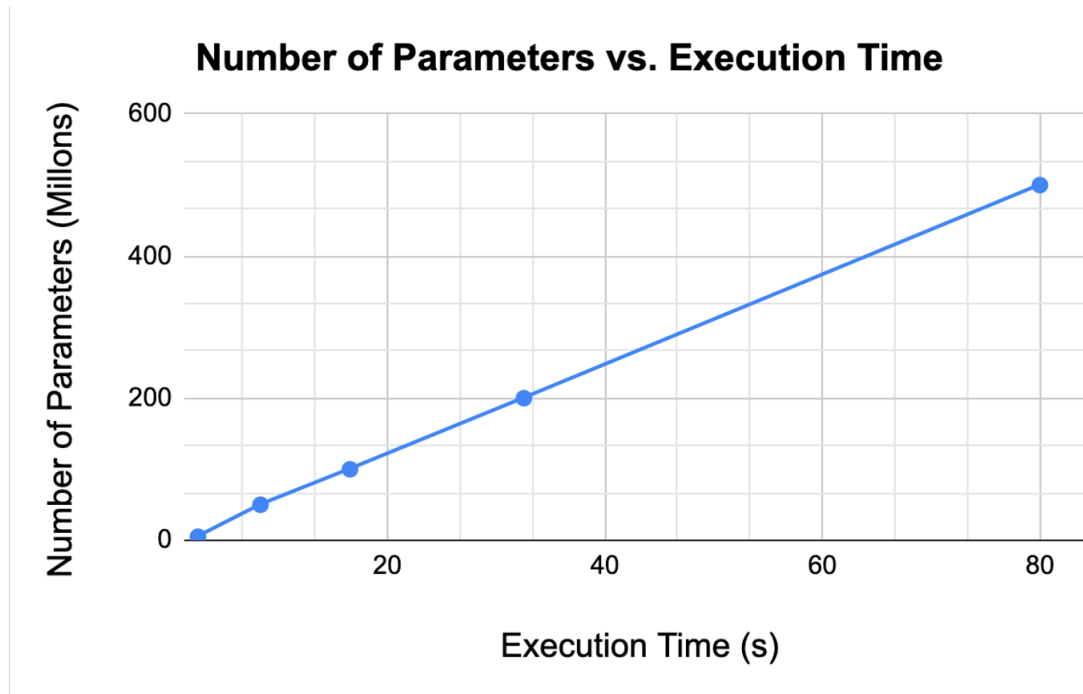


Figure 5.3: Number of Parameters Latency for Secure Aggregation Scaling.

As seen in the graph, the scaling is almost perfectly linear based on the number of parameters. The one possible outlier is at the very bottom-left when there are very few parameters, however, this is likely due to necessary overhead in setting up the connections between the clients and servers.

### 5.6.3 Network Latency

The networking latency results can be simulated using Linux's traffic controller and this is applied to the traffic between the servers and between clients and servers. This allows to set the latency for per packet, allow for changing of packet size, dropped packets, and more. It emulates a real-world scenario when testing in a local environment. All other values in this test remained the same outside of the latency. The packet size is indicated in each figure and the workers were always 16.

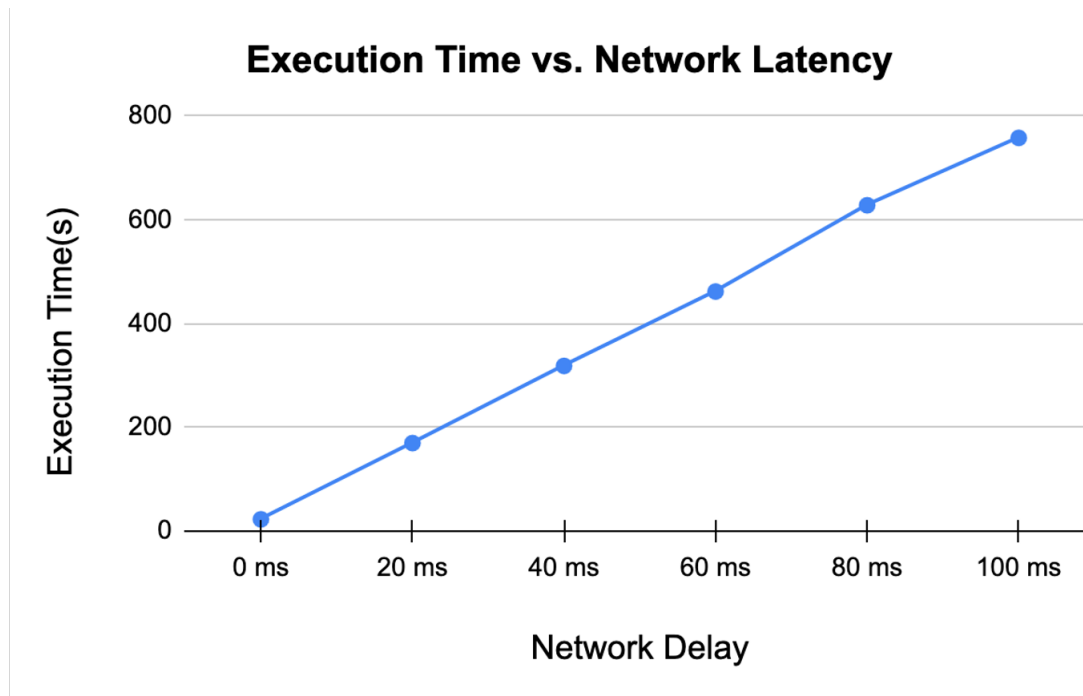


Figure 5.4: Network Latency with Packet Size 65536 for Secure Aggregation Scaling.

The Figure 5.4 highlights the linear scaling in execution time for the secure aggregation process for 6 different packet delay latencies starting from 0 ms and increasing on 20 ms intervals to 100 ms. Although the execution time is rather disappointing for higher packet delay latency values, this likely could be looked into and improved as in the next few tables, it will be shown that packet size does not have as big of an impact as it could for the execution time over a network.

The three tables below Table 5.18, Table 5.19, Table 5.20 highlight the various secure aggregation execution times with varying packet delays.

Although the packet size does see some impact on the execution time, it does not have as out-sized of an impact as expected considering the network latency per packet. This is likely due to other OS settings or MP-SPDZ framework settings that would need to be further tweaked to see an improvement for the overall networking latency. Being able to send fewer

Network Latency with Packet Size 128	
Packet Latency (ms)	End-to-end Aggregation Time (ms)
0	23.212
20	175.094
40	362.451
60	547.102
80	700.824
100	890.674

Table 5.18: Network Latency with Packet Size 128 for Secure Aggregation Scaling.

Network Latency with Packet Size 65536	
Packet Latency (ms)	End-to-end Aggregation Time (ms)
0	22.759
20	169.496
40	318.511
60	461.813
80	627.979
100	757.899

Table 5.19: Network Latency with Packet Size 65536 for Secure Aggregation Scaling.

Network Latency with Packet Size 33554432	
Packet Latency (ms)	End-to-end Aggregation Time (ms)
0	23.268
20	173.168
40	311.806
60	459.438
80	618.875
100	753.827

Table 5.20: Network Latency with Packet Size 33554432 for Secure Aggregation Scaling.

and larger packets so less latency is added from networking is a necessity for best practical purposes.

## 5.7 Accuracy

Accuracy was evaluated for all models with 16 workers for roughly 50 epochs. For example, most accuracy results work with 2500 iterations as a baseline, which is relatively close to 50 epochs. The data has been normalized to best match the epoch value since if there are more workers, more iterations must be run to iterate over the same amount of data. The models were evaluated on both CIFAR-10 and CIFAR-100.

The way the data was split will be referenced as the skew. For CIFAR-10 the skew is with each worker having at least 90% of their data be in one label, outside of the last few, who may have slightly more evenly distributed data depending on what remained. For CIFAR-100, each worker had multiple classes, but held at least 90% of the data in those classes. These are both skews of 0.9 which is the default unless otherwise specified. In a real-world setting, it is unlikely to have perfectly distributed data, which is why the accuracy was tested with a skew.

The CIFAR-10 results for loss in secure aggregation compared to the plain text results are shown in Table 5.21. The difference was calculated by subtracting the MPC version minus the plain text version.

Model Accuracy on CIFAR-10			
Model	MPC Accuracy	Plain Text Accuracy	Difference vs. Plain Text
DenseNet-121	0.5995	0.5947	0.0048
VGG-11	0.5823	0.5834	-0.0011
ResNet-18	0.5918	0.5528	0.0390
Simple DLA	0.5516	0.5192	0.0324

Table 5.21: Model Accuracy After 2500 Iterations on CIFAR-10.

Overall, there generally is a gain in accuracy which is surprising. Even in the case of loss, the accuracy drop is acceptable considering the gains in protection from inference attacks made by using secure aggregation. In the semi-honest case, this loss could vary slightly due to the

probabilistic truncation. A similar table is displayed for CIFAR-100 below in Table 5.22.

Model Accuracy on CIFAR-100			
Model	MPC Accuracy	Plain Text Accuracy	Difference vs. Plain Text
DenseNet-121	0.5151	0.4844	0.0307
VGG-11	0.4776	0.4897	-0.0121
ResNet-18	0.4770	0.5070	-0.0300
Simple DLA	0.4386	0.4769	-0.0383

Table 5.22: Model Accuracy After 2500 Iterations on CIFAR-100.

This time, the loss is relatively high and generally across the board. This is possibly due to the way the data was generated for runs that the random flipping of images and random cropping was not as good in one run compared to the other. Overall, the loss would not be expected to be this high unless the data has differences as is possible for the CIFAR-100 data.

The MPC accuracy for the different nets on CIFAR-10 in PPPFL with Secure MPC are shown in Figure 5.5. The accuracy is relatively poor when comparing to centralized learning, however, this is likely explained by the extreme skew of the data in this scenario. Later on, the skew will be evaluated and show that it does have a large impact on the accuracy. There are some rather large drops in accuracy at certain testing times, this occurs in various different tests and then the accuracy jumps back up the next time it is tested. It is possible the operation was partially interrupted by another process.

In Figure 5.6 and Figure 5.7 the same nets are evaluated on CIFAR-100. The former displays the top 1 accuracy whereas the latter displays the top 5 accuracy. While there is a drop in accuracy, it is not as significant as the CIFAR-10 results. This is perhaps due to the way the data is split in the CIFAR-10 versus CIFAR-100 case. Overall, considering the skewed data, these results are more promising.

When looking at Figure 5.8 it displays that the number of workers does possibly have an

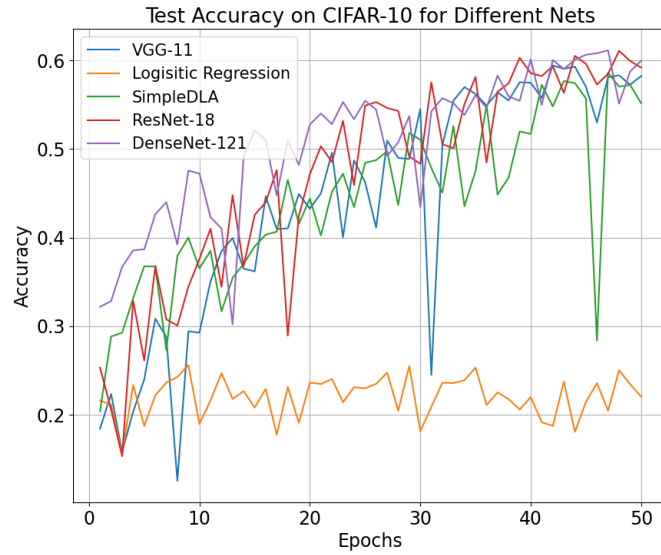


Figure 5.5: MPC Test Accuracy on CIFAR-10 for Different Nets.

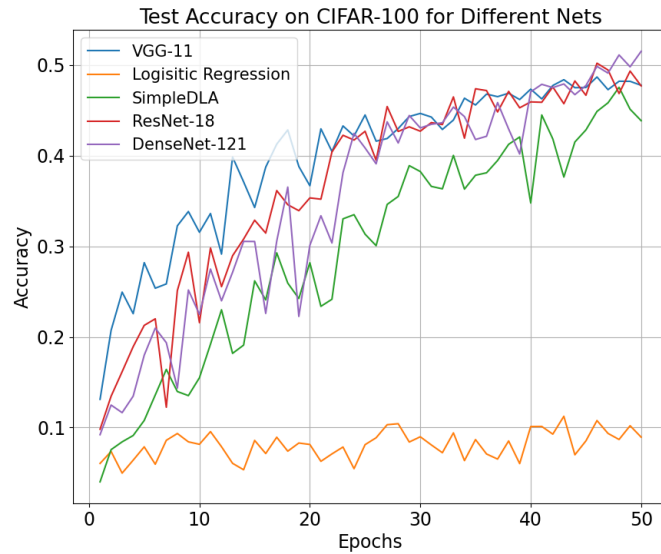


Figure 5.6: MPC Test Accuracy on CIFAR-100 for Different Nets.

impact on the accuracy for CIFAR-10. This could be attributed to better overall split of the data with more workers since there are so few labels.

In the CIFAR-100 case, it is shown that the number of workers does not have an impact on

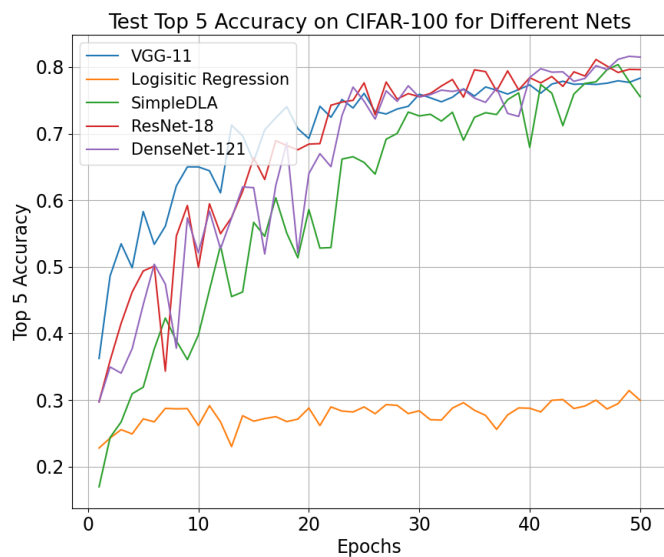


Figure 5.7: MPC Test Top 5 Accuracy on CIFAR-100 for Different Nets.

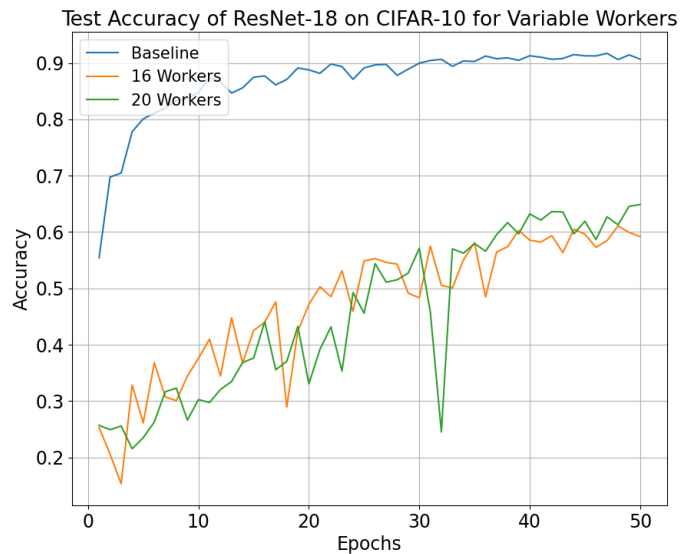


Figure 5.8: MPC Test Accuracy on CIFAR-10 for Variable Workers.

the accuracy, which is more expected. This is presented in Figure 5.9 for the top 1 accuracy and Figure 5.10 for the top 5 accuracy.

As mentioned earlier, the skew would be presented to display the impact it has on the

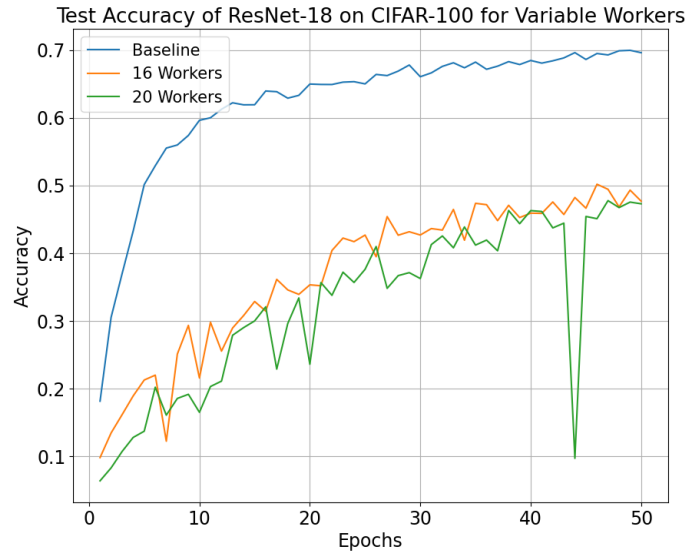


Figure 5.9: MPC Test Accuracy on CIFAR-100 for Variable Workers.

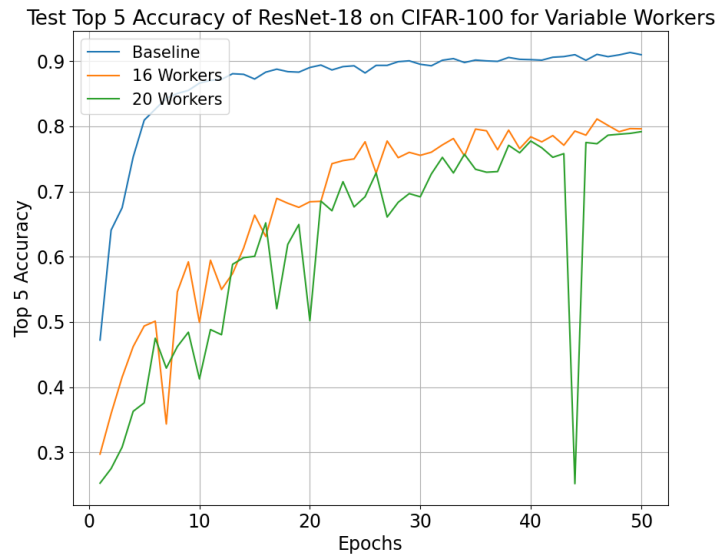


Figure 5.10: MPC Test Top 5 Accuracy on CIFAR-100 for Variable Workers.

accuracy. In Figure 5.11 the results are shown for CIFAR-10. Even with a skew of 0.5 the accuracy is still not close to the plain text result, but it is higher than tests with increased skews. This suggests that datasets with a few number of labels will suffer if the data is not

well distributed, which does make some sense.

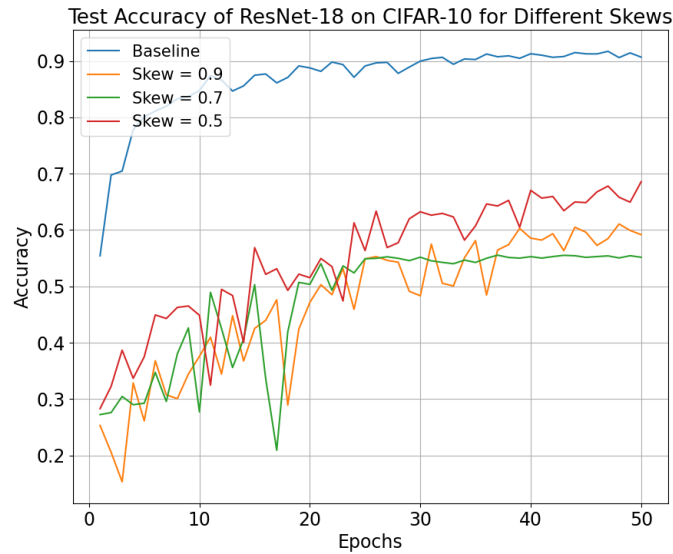


Figure 5.11: MPC Test Accuracy on CIFAR-10 for Different Skews.

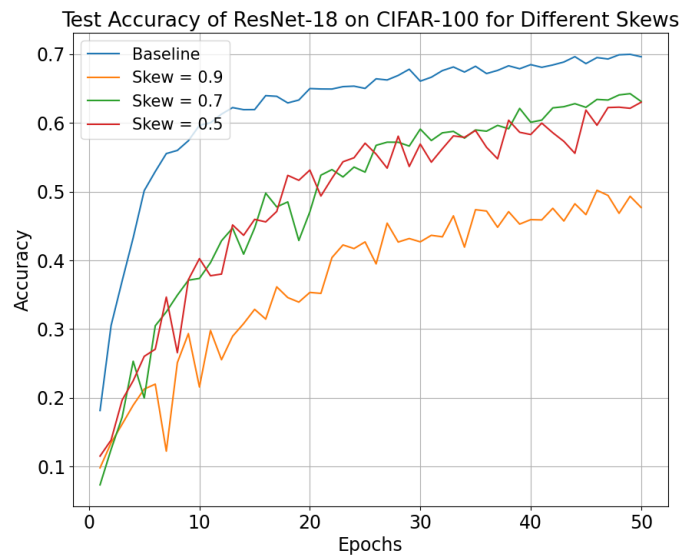


Figure 5.12: MPC Test Accuracy on CIFAR-100 for Different Skews.

In Figure 5.12 the top 1 accuracy for CIFAR-100 with different skews is shown. The top 5 accuracy is displayed in Figure 5.13. Overall, CIFAR-100 begins to become quite close to

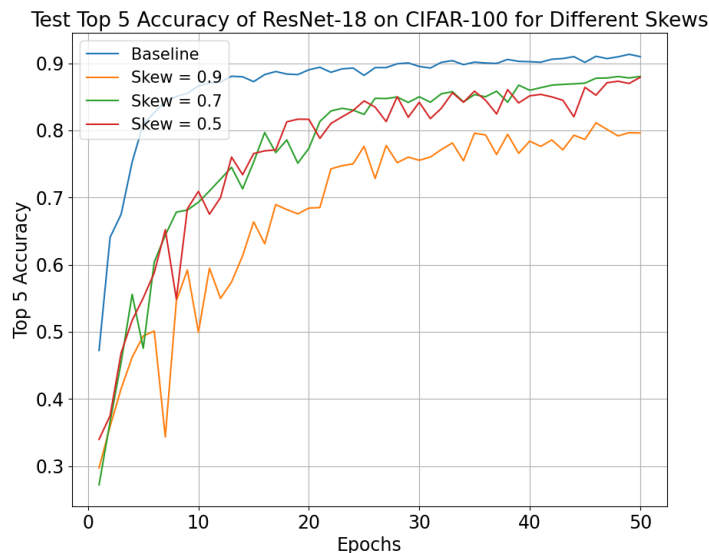


Figure 5.13: MPC Test Top 5 Accuracy on CIFAR-100 for Different Skews.

the plain text results with a skew of 0.5. Even a skew of 0.7 yields relatively high results. This suggests that in many label datasets, somewhat poorly distributed data will not suffer a large loss of accuracy.

## 5.8 Differential Privacy Accuracy

The use of differential privacy still yielded high accuracy results when training well-equipped models. In Figure 5.14, it is shown that the accuracy of the vision transformer is in the mid-90s once trained. This was for two different  $\sigma$  values and as expected, the higher  $\sigma$  value did see a little bit of a loss of accuracy, but not much.

$$\epsilon = \frac{\text{batch size} * \text{workers}}{\text{total dataset size}} * \frac{\sqrt{2 * \ln \frac{1.25}{\delta}}}{\sigma} * \sqrt{\text{epoch}} \quad (5.1)$$

In Figure 5.15 the privacy guarantee is shown throughout the 10 epochs of testing for the

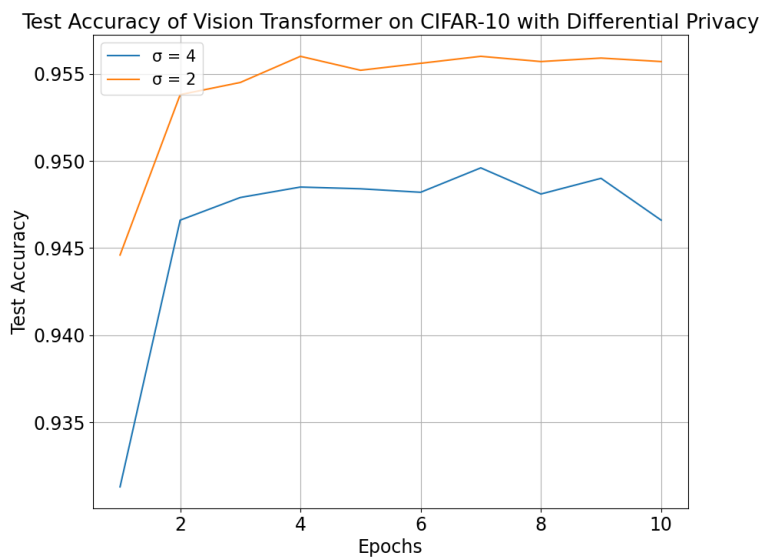


Figure 5.14: MPC Accuracy of Vision Transformer on CIFAR-10 with Differential Privacy.

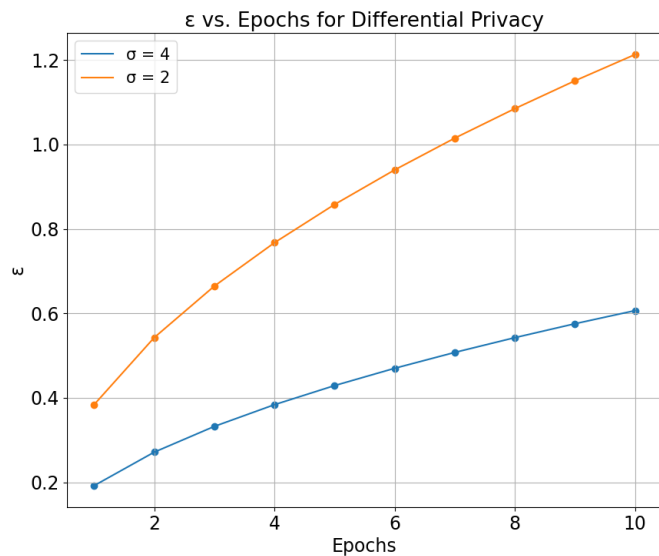


Figure 5.15:  $\epsilon$  vs. Epochs for Differential Privacy.

various  $\sigma$  values. The equation to calculate  $\epsilon$  is presented in Equation 5.1  $\epsilon$  does begin to get a bit high for  $\sigma = 2$ , however, the accuracy is not seeing an increase beyond 4 epochs, so this is not a concern. For  $\sigma = 4$ , the privacy guarantee is still intact after 10 epochs and

could keep training. Overall, the results for differential privacy are quite encouraging and suggest that it can be effectively used for fine-tuning models.

# Chapter 6

## Future Work

There is still much future work that can occur within the Privacy Preserving Federated Learning with Secure Multi-Party Computation framework to decrease execution time and improve accuracy of the models. Additionally, much functionality could be improved for future use.

For starters, the current client interface only works with a ring of 64-bits. Anything larger and the client interface will not properly be able to handle the values needed. One possibility for this is to use other libraries capable of handling these large integers. Some possibilities are the boost multiprecision library or the GNU multiple precision library. Boost is built for C++, however, C++ can be used within the Cython framework.

The second portion that could be expanded upon is an increase in accuracy. Currently, since the federated learning algorithms are relatively rudimentary for secure aggregation, more of the aggregation schemes could be implemented. Further, an addition to help model accuracy such as MOON [42] could be added since it does not alter the server aggregation phase. It has shown to increase performance and would be relatively straightforward to implement without requiring more extensive knowledge of MP-SPDZ.

Another issue is the current latency increase when using a network. There are a few different areas that could be improved upon in this area. One is to parallelize network communication between the clients and servers if possible. The second is to transform most of the direct net-

working communication from Python to Cython. This could possibly reduce some overhead from Python.

The final key area is to re-work the current training loop for the secure aggregation. To limit any possible overhead of waiting for the servers to start, the servers could be started at the beginning of each iteration. Since the client interface runs for multiple loops, the servers could start each program on a different port for as many client interface loops as necessary. This would then eliminate any possible overhead, however, it may be necessary to add extra security in this event.

Other possible areas of future work are increasing the number of models, expand fine-tuning functionality, expand the differential privacy functionality and general clean-up and organization of the framework.

# Chapter 7

## Conclusions

Data privacy is highly important for a wide variety of applications and has become more necessary as laws have been passed. While federated learning allows for data not to leave a client's machine, it does not guarantee enough security of their data. SAFEFL expanded the security benefits, but did not provide a reasonable enough execution time to utilize the multi-party computation portion of the framework. By leveraging Cython, Numpy, and other tools, it is possible to decrease execution time to a practical amount that could be used in real-world setting in PPPFL with Secure MPC. Further, fine-tuning and differential privacy have been added to allow for more functionality and privacy in various application settings. Further work can be done in improving accuracy and execution time of framework to ensure its practicality.

# Bibliography

- [1] Abbas Acar, Hidayet Aksu, A. Selcuk Ulugac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation, 2017. URL <https://arxiv.org/abs/1704.03578>.
- [2] Ehud Aharoni, Moran Baruch, Pradip Bose, Alper Buyuktosunoglu, Nir Drucker, Subhankar Pal, Tomer Pelleg, Kanthi Sarpatwar, Hayim Shaul, Omri Soceanu, and Roman Vaculin. He-pex: Efficient machine learning under homomorphic encryption using pruning, permutation and expansion, 2022.
- [3] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Federated learning versus classical machine learning: A convergence comparison, 2021. URL <https://arxiv.org/abs/2107.10976>.
- [4] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, Nov 2010. ISSN 1573-0565. doi: 10.1007/s10994-010-5188-5. URL <https://doi.org/10.1007/s10994-010-5188-5>.
- [5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991. doi: 10.1007/3-540-46766-1\_34.
- [6] James Bell, K. A. Bonawitz, Adrià Gascón, Tancrede Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. Cryptology ePrint Archive, Paper 2020/704, 2020. URL <https://eprint.iacr.org/2020/704>. <https://eprint.iacr.org/2020/704>.

- [7] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12*, page 784–796, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450316514. doi: 10.1145/2382196.2382279. URL <https://doi.org/10.1145/2382196.2382279>.
- [8] Yaniv Ben-Itzhak, Helen Möllering, Benny Pinkas, Thomas Schneider, Ajith Suresh, Oleksandr Tkachenko, Shay Vargaftik, Christian Weinert, Hossein Yalame, and Avishay Yanai. ScionFL: Efficient and robust secure quantized aggregation. *Cryptology ePrint Archive*, Paper 2023/652, 2023. URL <https://eprint.iacr.org/2023/652>. <https://eprint.iacr.org/2023/652>.
- [9] Alberto Blanco-Justicia, David Sánchez, Josep Domingo-Ferrer, and Krishnamurthy Muralidhar. A critical review on the use (and misuse) of differential privacy in machine learning. *ACM Comput. Surv.*, 55(8), dec 2022. ISSN 0360-0300. doi: 10.1145/3547139. URL <https://doi.org/10.1145/3547139>.
- [10] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. MP2ML: A mixed-protocol machine learning framework for private inference. *Cryptology ePrint Archive*, Paper 2020/721, 2020. URL <https://eprint.iacr.org/2020/721>. <https://eprint.iacr.org/2020/721>.
- [11] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, page 1175–1191, New York, NY, USA, 2017. Association for Computing Machinery. ISBN

9781450349468. doi: 10.1145/3133956.3133982. URL <https://doi.org/10.1145/3133956.3133982>.
- [12] Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. Motion - a framework for mixed-protocol multi-party computation. Cryptology ePrint Archive, Paper 2020/1137, 2020. URL <https://eprint.iacr.org/2020/1137>. <https://eprint.iacr.org/2020/1137>.
- [13] David Byrd and Antigoni Polychroniadou. Differentially private secure multi-party computation for federated learning in financial applications, 2020.
- [14] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping, 2022. URL <https://arxiv.org/abs/2012.13995>.
- [15] Wei-Ning Chen, Christopher A. Choquette-Choo, Peter Kairouz, and Ananda Theertha Suresh. The fundamental price of secure aggregation in differentially private federated learning, 2022. URL <https://arxiv.org/abs/2203.03761>.
- [16] Yao Chen, Yijie Gui, Hong Lin, Wensheng Gan, and Yongdong Wu. Federated learning attacks and defenses: A survey, 2022. URL <https://arxiv.org/abs/2211.14952>.
- [17] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology, CRYPTO '87*, page 350–354, Berlin, Heidelberg, 1987. Springer-Verlag. ISBN 3540187960.
- [18] Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. Cryptology ePrint Archive, Paper 2019/131, 2019. URL <https://eprint.iacr.org/2019/131>. <https://eprint.iacr.org/2019/131>.

- [19] Anindya De. Lower bounds in differential privacy, 2011. URL <https://arxiv.org/abs/1107.2183>.
- [20] Georgios Drainakis, Konstantinos V. Katsaros, Panagiotis Pantazopoulos, Vasilis Sourlas, and Angelos Amditis. Federated vs. centralized machine learning under privacy-elastic users: A comparative analysis. In *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, pages 1–8, 2020. doi: 10.1109/NCA51143.2020.9306745.
- [21] Cynthia Dwork. *Differential Privacy*, pages 338–340. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5\_752. URL [https://doi.org/10.1007/978-1-4419-5906-5\\_752](https://doi.org/10.1007/978-1-4419-5906-5_752).
- [22] David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. 2018. doi: 10.1561/33000000019.
- [23] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to Byzantine-Robust federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1605–1622. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL <https://www.usenix.org/conference/usenixsecurity20/presentation/fang>.
- [24] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möl-  
lering, Thien Duc Nguyen, Phillip Rieger, Ahmad Reza Sadeghi, Thomas Schnei-  
der, Hossein Yalame, and Shaza Zeitouni. Safelearn: Secure aggregation for pri-  
vate federated learning. Cryptology ePrint Archive, Paper 2021/386, 2021. URL <https://eprint.iacr.org/2021/386>. <https://eprint.iacr.org/2021/386>.
- [25] Jun Furukawa and Yehuda Lindell. Two-thirds honest-majority mpc for malicious ad-  
versaries at almost the cost of semi-honest. In *Proceedings of the 2019 ACM SIGSAC*

- Conference on Computer and Communications Security, CCS '19*, page 1557–1571, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367479. doi: 10.1145/3319535.3339811. URL <https://doi.org/10.1145/3319535.3339811>.
- [26] Till Gehlhar, Felix Marx, Thomas Schneider, Ajith Suresh, Tobias Wehrle, and Hossein Yalame. Safefl: Mpc-friendly framework for private and robust federated learning. Cryptology ePrint Archive, Paper 2023/555, 2023. URL <https://eprint.iacr.org/2023/555>. <https://eprint.iacr.org/2023/555>.
- [27] Matei Grama, Maria Musat, Luis Muñoz-González, Jonathan Passerat-Palmbach, Daniel Rueckert, and Amir Alansary. Robust aggregation for adaptive privacy preserving federated learning in healthcare, 2020. URL <https://arxiv.org/abs/2009.08294>.
- [28] Gohel S Habebh H. Machine learning in healthcare. *Curr Genomics*, 22(4):291–300, 2021. doi: <https://doi.org/10.2174/1389202922666210705124359>.
- [29] Andrew Hard, Chloé M Kiddon, Daniel Ramage, Françoise Beaufays, Hubert Eichner, Kanishka Rao, Rajiv Mathews, and Sean Augenstein. Federated learning for mobile keyboard prediction, 2018. URL <https://arxiv.org/abs/1811.03604>.
- [30] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption, 2017. URL <https://arxiv.org/abs/1711.10677>.
- [31] Carmit Hazay and Yehuda Lindell. A note on the relation between the definitions of security for semi-honest and malicious adversaries. Cryptology ePrint Archive, Paper 2010/551, 2010. URL <https://eprint.iacr.org/2010/551>. <https://eprint.iacr.org/2010/551>.

- [32] Aditya Hegde, Helen Möllering, Thomas Schneider, and Hossein Yalame. SoK: Efficient privacy-preserving clustering. *Cryptology ePrint Archive*, Paper 2021/809, 2021. URL <https://eprint.iacr.org/2021/809>. <https://eprint.iacr.org/2021/809>.
- [33] Wolfgang Hörmann and Josef Leydold. Continuous random variate generation by fast numerical inversion. *ACM Trans. Model. Comput. Simul.*, 13(4):347–362, oct 2003. ISSN 1049-3301. doi: 10.1145/945511.945517. URL <https://doi.org/10.1145/945511.945517>.
- [34] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, and Xuyun Zhang. Source inference attacks in federated learning, 2021.
- [35] Zhanglong Ji, Zachary C. Lipton, and Charles Elkan. Differential privacy and machine learning: a survey and review, 2014. URL <https://arxiv.org/abs/1412.7584>.
- [36] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. *Cryptology ePrint Archive*, Paper 2020/521, 2020. URL <https://eprint.iacr.org/2020/521>. <https://eprint.iacr.org/2020/521>.
- [37] Marcel Keller. Mp-spdz torch mnist dense, 2024. URL [https://github.com/data61/MP-SPDZ/blob/master/Programs/Source/torch\\_mnist\\_dense.mpc](https://github.com/data61/MP-SPDZ/blob/master/Programs/Source/torch_mnist_dense.mpc).
- [38] Marcel Keller and Ke Sun. Secure quantized training for deep learning, 2022. URL <https://arxiv.org/abs/2107.00501>.
- [39] Asharul Islam Khan and Salim Al-Habsi. Machine learning in computer vision. *Procedia Computer Science*, 167:1444–1451, 2020. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2020.03.355>. International Conference on Computational Intelligence and Data Science.

- [40] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning, 2022.
- [41] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency, 2017.
- [42] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning, 2021. URL <https://arxiv.org/abs/2103.16257>.
- [43] Xinjian Luo, Yuncheng Wu, Xiaokui Xiao, and Beng Chin Ooi. Feature inference attack on model predictions in vertical federated learning. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 181–192, 2021. doi: 10.1109/ICDE51399.2021.00023.
- [44] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey, 2020. URL <https://arxiv.org/abs/2003.02133>.
- [45] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023.
- [46] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023. URL <https://arxiv.org/abs/1602.05629>.
- [47] Kato Mivule, Claude Turner, and Soo-Yeon Ji. Towards a differential privacy and utility preserving machine learning classifier. *Procedia Computer Science*, 12:176–181, 2012. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2012.09.050>. URL <https://www.>

- [sciencedirect.com/science/article/pii/S1877050912006412](https://www.sciencedirect.com/science/article/pii/S1877050912006412). Complex Adaptive Systems 2012.
- [48] Noella Nazareth and Yeruva Venkata Ramana Reddy. Financial applications of machine learning: A literature review. *Expert Systems with Applications*, 219:119640, 2023. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2023.119640>.
- [49] Do-Van Nguyen, Anh-Khoa Tran, and Koji Zettsu. Fedprob: An aggregation method based on feature probability distribution for federated learning on non-iid data. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2875–2881, 2022. doi: 10.1109/BigData55660.2022.10020923.
- [50] Thien Duc Nguyen, Phillip Rieger, Huili Chen, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Shaza Zeitouni, Farinaz Koushanfar, Ahmad-Reza Sadeghi, and Thomas Schneider. FLAME: Taming backdoors in federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1415–1432, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-31-1. URL <https://www.usenix.org/conference/usenixsecurity22/presentation/nguyen>.
- [51] Kenneth Olmstead and Aaron Smith. Americans’ experiences with data security, 2017. URL <https://www.pewresearch.org/internet/2017/01/26/1-americans-experiences-with-data-security/>.
- [52] Someswari Perla, Naga Nimesh K, and Srinidhi Potta. Implementation of autonomous cars using machine learning. In *2022 International Conference on Edge Computing and Applications (ICECAA)*, pages 1444–1451, 2022. doi: 10.1109/ICECAA55415.2022.9936102.

- [53] Krishna Pillutla, Sham M. Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing*, 70:1142–1154, 2022. ISSN 1941-0476. doi: 10.1109/tsp.2022.3153135. URL <http://dx.doi.org/10.1109/TSP.2022.3153135>.
- [54] IEEE Digital Privacy. What is multiparty computation? URL <https://digitalprivacy.ieee.org/publications/topics/what-is-multiparty-computation>.
- [55] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. Adversarial attacks and defenses in deep learning. *Engineering*, 6(3):346–360, 2020. ISSN 2095-8099. doi: <https://doi.org/10.1016/j.eng.2019.12.012>. URL <https://www.sciencedirect.com/science/article/pii/S209580991930503X>.
- [56] Dragoş Rotaru. awesome-mpc. Github, 2017. URL <https://github.com/rdragos/awesome-mpc>.
- [57] Subhash Sagar, Chang-Sun Li, Seng W. Loke, and Jinho Choi. Poisoning attacks and defenses in federated learning: A survey, 2023. URL <https://arxiv.org/abs/2301.05795>.
- [58] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. MI-leaks: Model and data independent membership inference attacks and defenses on machine learning models, 2018. URL <https://arxiv.org/abs/1806.01246>.
- [59] Tim Salzmann, Elia Kaufmann, Jon Arrizabalaga, Marco Pavone, Davide Scaramuzza, and Markus Ryll. Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. *IEEE Robotics and Automation Letters*, 8(4):

- 2397–2404, April 2023. ISSN 2377-3774. doi: 10.1109/lra.2023.3246839. URL <http://dx.doi.org/10.1109/LRA.2023.3246839>.
- [60] Tirthajyoti Sarkar. Why you should forget ‘for-loop’ for data science code and embrace vectorization, 2017. URL <https://www.kdnuggets.com/2017/11/forget-for-loop-data-science-code-vectorization.html>.
- [61] Or Shalom, Amir Leshem, and Waheed U. Bajwa. Mitigating data injection attacks on federated learning, 2024. URL <https://arxiv.org/abs/2312.02102>.
- [62] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979. ISSN 0001-0782. doi: 10.1145/359168.359176. URL <https://doi.org/10.1145/359168.359176>.
- [63] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning, 2021. URL <https://arxiv.org/abs/2108.10241>.
- [64] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017. doi: 10.1109/SP.2017.41.
- [65] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models, 2017. URL <https://arxiv.org/abs/1610.05820>.
- [66] Jinhyun So, Basak Guler, and A. Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning, 2021.
- [67] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning

- attacks against federated learning systems, 2020. URL <https://arxiv.org/abs/2007.08432>.
- [68] Randall Wald, Taghi M. Khoshgoftaar, and David Dittman. Mean aggregation versus robust rank aggregation for ensemble gene selection. In *2012 11th International Conference on Machine Learning and Applications*, volume 1, pages 63–69, 2012. doi: 10.1109/ICMLA.2012.20.
- [69] Qifan Wang, Lei Zhou, Jianli Bai, Yun Sing Koh, Shujie Cui, and Giovanni Russo. Ht2ml: An efficient hybrid framework for privacy-preserving machine learning using he and tee. *Computers & Security*, 135:103509, 2023. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2023.103509>. URL <https://www.sciencedirect.com/science/article/pii/S0167404823004194>.
- [70] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. Cryptology ePrint Archive, Paper 2017/189, 2017. URL <https://eprint.iacr.org/2017/189>. <https://eprint.iacr.org/2017/189>.
- [71] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David R. O'Brien, Thomas Steinke, and Salil Vadhan. Differential privacy: A primer for a non-technical audience. *Vanderbilt Journal of Entertainment & Technology Law*, 21(1):209–275, 2018. URL <http://www.jetlaw.org/journal-archives/volume-21/volume-21-issue-1/differential-privacy-a-primer-for-a-non-technical-audience/>.
- [72] Geming Xia, Jian Chen, Chaodong Yu, and Jun Ma. Poisoning attacks in federated learning: A survey. *IEEE Access*, 11:10708–10722, 2023. doi: 10.1109/ACCESS.2023.3238823.

- [73] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions, 2018. URL <https://arxiv.org/abs/1812.02903>.
- [74] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates, 2021. URL <https://arxiv.org/abs/1803.01498>.
- [75] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2805–2824, 2019. doi: 10.1109/TNNLS.2018.2886017.
- [76] Sergey Zapechnikov. Privacy-preserving machine learning as a tool for secure personalized information services. *Procedia Computer Science*, 169:393–399, 2020. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2020.02.235>. URL <https://www.sciencedirect.com/science/article/pii/S1877050920303598>. Postproceedings of the 10th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2019 (Tenth Annual Meeting of the BICA Society), held August 15-19, 2019 in Seattle, Washington, USA.
- [77] Gongxi Zhu, Donghao Li, Hanlin Gu, Yuxing Han, Yuan Yao, Lixin Fan, and Qiang Yang. Evaluating membership inference attacks and defenses in federated learning, 2024. URL <https://arxiv.org/abs/2402.06289>.
- [78] Hangyu Zhu, Rui Wang, Yaochu Jin, Kaitai Liang, and Jianting Ning. Distributed additive encryption and quantization for privacy preserving federated deep learning, 2020.
- [79] Heng Zhu and Qing Ling. Bridging differential privacy and byzantine-robustness via model aggregation, 2022. URL <https://arxiv.org/abs/2205.00107>.