ENSURING INTEGRITY, PRIVACY, AND FAIRNESS FOR MACHINE LEARNING USING TRUSTED EXECUTION ENVIRONMENTS

by

Aref Asvadishirehjini

APPROVED BY SUPERVISORY COMMITTEE:

Murat Kantarcioglu, Chair

Bhavani Thuraisingham

Latifur Khan

Rishabh Krishnan Iyer

Dedicated to my family and friends.

ENSURING INTEGRITY, PRIVACY, AND FAIRNESS FOR MACHINE LEARNING USING TRUSTED EXECUTION ENVIRONMENTS

by

AREF ASVADISHIREHJINI, BS,MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2021

ACKNOWLEDGMENTS

I would like to thank Dr. Murat Kantarcioglu for his continuous support in guiding me during the PhD years. In particular, I appreciate his unbounded patience during ups and downs of my PhD career. I am also thankful to end with the support and collaboration I received from Dr. Bradley Malin.

I am thankful to Dr. Bhavani Thuraisingham, Dr. Latifur Khan, and Dr. Rishabh Krishnan Iyer for their interest and spectacular support serving on my PhD committee.

I would like to thank Dr. Yan Zhou and Dr. Cuneyt Gurcan Akcora who helped me graciously, by reviewing my papers before submitting and giving me numerous precious feedback on effectively relaying my work.

I also want to thank Computer Science departmental staff, Mrs. Rhonda Walls, Mr. Doug Hyde, and Mr. Shyam Karrah for helping me meet departmental requirements.

During these years, I have had an amazing opportunity to know a lot of great friends. I am extremely thankful to Dr. Nazmiye Ceren Abay, Dr. Bulut Coskun, Dr. Maryam Imani, Dr. Sarah Rouhani, Dr. Hamed Alemansour, Dr. Hafez Eslami, Dr. Ashkan Yousefpour, Mustafa Ozday, Vibha Belavadi, Dr. Fahad Shaon, Dr. Imrul Chowdhury, Navid Hashemi, Behzad Mirkhanzadeh, Nima Taherkhani, Azadeh Samadian, Afshin Alipour, Mohammadreza Haghpanah, Mousa, and all the great friends who helped and made me happy.

Finally, I thank my parents, and brother whom I have not seen in almost nine years, but always have kept my spirit up.

The research reported herein, was supported by NIH award 1R01HG006844, NSF awards, CNS-1633331,CNS-1837627, OAC-1828467, IIS-1939728, DMS-1925346, CNS-2029661, and ARO award W911NF-17-1-0356.

November 2021

ENSURING INTEGRITY, PRIVACY, AND FAIRNESS FOR MACHINE LEARNING USING TRUSTED EXECUTION ENVIRONMENTS

Aref Asvadishirehjini, PhD
The University of Texas at Dallas, 2021

Supervising Professor: Murat Kantarcioglu, Chair

In this day and age, numerous decision-making systems increasingly rely on machine learning (ML) and deep learning to deliver cutting-edge technologies to the members of society. Due to potential security, privacy and bias issues with respect to these ML methods, currently, end users cannot fully trust these systems with their private data, and their prediction outcome. For instance, in many cases, it is not clear how an individual's medical record is being used for building tools for medical diagnosis? Is the data always encrypted at rest? When they are decrypted, is there a guarantee that only a trusted application can have access to the private data to eliminate potential misuse? Throughout this dissertation, solutions that leverage various security and integrity capabilities provided by hardware assisted Trusted Execution Environments (TEE) are proposed to make these ML based systems more reliable and *trustworthy* so that end users can have a greater trust in these systems.

As a starting point, we first address the privacy and integrity issues in ML model learning in the cloud setting. Training of a deep learning model that only relies on a TEE is not very attractive to businesses that need to continuously train their models in a remote cloud setting. This is due to the fact that special hardware such as Graphical Processing Units (GPU) are much more efficient in training ML models compared to CPU based TEEs. In this dissertation, we propose an *integrity*-preserving solution that combines TEEs, and GPUs in

order to provide an efficient solution. In this solution, we focus on the ML model training task using the efficient GPU while ensuring the detect any deviation from the ML model learning protocol with a high probability using the TEE capabilities. Using our solution, we can ascertain (with high probability) the model is trained with the correct training dataset using the correct training hyperparameters, and correct code execution flow.

Once we provide an integrity preserving ML model training solution, we focus on how to use the learned ML model privately and securely in practice. To provide privacy-preserving inference on sensitive data, wherein ML model owner and data owner do not trust each other, the dissertation proposes a solution that the inference task is run inside a TEE and the result is sent to the data owner(s). The most important benefit of our solution is that the data owner can ensure their data will not be used for any other purposes in the future and no information other than the agreed model inference result is disclosed. Furthermore, we show the efficacy of our solution in the context of genomic data analysis.

Next we focus on the bias and unfairness embedded in certain ML models. It is has been reported that the ML models can unfairly treat certain subgroups, and it is hard to test for such issues in application deployment settings where both the ML model and the input data to the ML model is sensitive (i.e., both the model and the data cannot be disclosed to public for auditing directly). This dissertation proposes a privacy-preserving solution for fairness analytics using TEEs. In this setting, the model owner and the fairness test set owner do not trust each other, therefore they do not want their input to be disclosed. The end goal is for the fairness analyst to conduct tests about the quality and fairness of the model's outcome with respect to a set of predefined minority groups or subgroups and compare and contrast them with privileged group(s). This way, models can be analyzed, and the analyst can shed light on the potential latent biases in the ML model in a privacy-preserving manner.

Even if the ML model is trained, and deployed securely, due to data poisoning, the final model may still contain hidden backdoors (which in the literature is referred to as *trojan*

attacks). Finally, in this dissertation, we develop novel techniques to detect such attacks. We design experiments that first creates a multitude of models that carry a trojan, and another set that does not have any trojan. Then, we build classifiers to see if we can tell them apart. Our results show that ML models could be used to detect trojan attacks against other ML models.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Numerous decisions impacting our lives are byproduct of automated systems that heavily rely on Machine Learning (ML), and Deep Learning (DL). Decisions such as advertisements we are exposed to via social media, deals we receive from online stores, movies recommended to us through streaming services, credit worthiness when applying for bank loans, etc. are only a few examples of how we are impacted on a daily basis through these automated systems. Building such giant, and heterogeneous ML systems demands an extraordinary amount of data (for training) that include privacy-sensitive content, that utilize significant computational resource on cloud platforms. Therefore, individuals are worried how their data (or meta-data) are being used on such remote environments. Also, another point that cast doubts on ML, is that the outcome of these systems can have life-lasting effects on individual's quality of life. Therefore, ensuring that the decisions are not biased is an important promise to the users affected by ML decisions.

Throughout this dissertation, our main objective is to provide an end-to-end ML (more specifically DL) infrastructure to increase public trust in ML with the help of the security and privacy capabilities provided by trusted execution environments (TEE) such as Intel SGX. First, we explore the training of models that guarantees the integrity of the training process is preserved. Next, we propose a privacy-preserving solution to use the trained models for the inference task. Later, we propose a solution for privacy-preserving fairness analytics to detect the potential biases of the ML models. Finally, we explore an analysis on neural network backdoor attacks (trojan attacks), and explore whether we can create a meta-model that decide whether an ML model has a backdoor. Below, we provide an overview of these solutions.

1.1 Integrity-Preserving Deep Learning Training Using A Hybrid Approach Via. GPU & TEE

Machine learning models based on Deep Neural Networks (DNNs) are increasingly deployed in a wide variety of applications, ranging from self-driving cars to COVID-19 diagnosis (Panwar et al., 2020). To support the computational power necessary to train a DNN, cloud environments with dedicated Graphical Processing Unit (GPU) hardware support have emerged as critical infrastructure. However, there are many integrity challenges associated with outsourcing the computation to use GPU power, due to its inherent lack of safeguards to ensure computational integrity. Various approaches have been developed to address these challenges, building on trusted execution environments (TEE). Yet, no existing approach scales up to support realistic integrity-preserving DNN model training for heavy workloads (e.g., deep architectures and millions of training examples) without sustaining a significant performance hit. To mitigate the running time difference between pure TEE (i.e., full integrity) and pure GPU (i.e., no integrity), in this dissertation, we combine random verification of selected computation steps with systematic adjustments of DNN hyperparameters (e.g., a narrow gradient clipping range), which limits the attacker's ability to shift the model parameters arbitrarily.

1.2 Deep Learning & Genomic Statistical Test Inference Analytics Via. TEE

As the quantity and quality of genomic records continue to grow, geneticists at disparate institutions are increasingly motivated to integrate and analyze genomic data collections. While these endeavors are often hampered by various privacy concerns, recently developed techniques can address privacy concerns via trusted execution environments (TEEs) (e.g., hardware-enabled secure enclaves). TEEs provide confidentiality for sensitive data and means to attest to programs' correctness and their corresponding inputs. However, due

to the profound consequences of private genomic data leakage for individuals, integrating TEEs for genomic analytics requires extra assurances to ensure that the program's execution does not leak participants' private information in a direct or indirect form. One major drawback of state-of-the-art TEEs is related to their shortcomings in hiding memory access patterns due to performance ramifications. As a result, the TEE's computation must be data oblivious, i.e., independent of how it accesses the sensitive private data through its course. In Chapter 5, we present our data oblivious and privacy-preserving genomic data analysis framework using two popular genomic tasks. The first application involves securely performing a multi-institution test for the most significant single nucleotide polymorphism (SNP) in an association study. The second application focuses on the growing trend in utilizing Deep Learning (DL) based inference for genomics data. In this application, we assume that genomic data and the model are private. In both of these applications, we show that our framework can enable efficient processing and analysis of encrypted genomic data without disclosing sensitive memory access patterns. We also discuss the future considerations needed to use TEE based privacy-preserving genomic data analysis in practice.

1.3 Machine Learning Fairness Analytics Via. TEE

Machine learning models are increasingly used in various critical applications, ranging from health care to finance. At the same time, there are many concerns whether these ML models discriminate against different subgroups. For example, an ML model that assigns credit scores may be biased against black Americans and may assign lower scores for them. To detect such biases and unfairness embedded in ML models, different ML fairness criteria have been proposed. In many cases, the appropriate fairness criteria for a given ML model depends on the task and the groups involved. Ideally, the ML models used for critical tasks could be publicly disclosed (e.g., credit-scoring model) and different researchers can run tests on the ML models using their own test data under chosen fairness criteria to see

whether the model has any fairness issues. Unfortunately, testing whether the given ML model satisfies a specified fairness criteria is challenging due to security and privacy issues with respect to ML models and test data. In many cases, organizations that spent lots of effort in developing the ML model may want to keep it private. On the other hand, the test data (e.g., credit application information) belonging to individuals must be kept private to protect individual privacy. Therefore, there is a need for enabling secure and privacy-preserving fairness auditing of ML models. To address the above research challenge, in Chapter 6, we present **AISolace** - a fairness analytics framework for ML models that relies on the security guarantees provided by the Trusted Execution Environments (TEE) such as Intel SGX. At a high level, AISolace allows ML model owners to share their private ML model with the general public (owners of private test data) for fairness analytics securely (i.e., only fairness audit information will be revealed). Further, AISolace supports complex privacy-aware grouping schemes (e.g., any subset of demographic features), ranging from conditionals based on attributes to clustering based on arbitrary distance metrics for complex fairness analytics to support any of the existing fairness criteria. Furthermore, our system could be easily extended to support other grouping based fairness criteria. In addition, AISolace protects against memory access pattern leaks that are not available out of the box TEE implementations. Furthermore, using extensive experimental evaluation, we show that using TEEs, we can allow fairness auditing on private data with acceptable overhead compared to pure CPU based non-secure fairness auditing.

1.4 Investigation of Trojaning Attacks On Neural Networks

Deep Learning (DL) is becoming a popular paradigm in a broad category of decision systems that are crucial to the well-being of our society. Self-driving vehicles, online dating, social network content recommendation, chest X-Ray screening, etc. are all examples that show how the quality of our lives is tied to the decisions of these systems. Therefore, it is not

enough to only ensure the health of those systems based on classical notions (accuracy, precision, recall, etc.) of performance. Given the complexity of DL-based models, we must take into account that these systems may be *gamed* to make favorable decisions for unqualified instances by malicious actors. For instance, researchers have shown that a vision classification model can conveniently be made to classify every input to a target class provided a special pixel pattern is present. Imagine if a self-driving car's traffic-sign detection model can classify a traffic stop sign as speed-limit if the pattern that triggers the faulty behavior is present. Currently, serious weaknesses similar to the aforementioned example are very difficult to detect efficiently and will significantly reduce the technology sector's aptitude and desire for continuous innovation (or at least the development speed) in fear of hefty law-suites. In this work, we investigated a cryptanalysis approach to analyze the models' behavior on random inputs from unrelated datasets. There were two main pools of models. 1) Clean, and 2) Poisoned. Poisoned models all have backdoors with different attack configurations. Given the simple construction of our framework, our initial investigation result show, given we can generate/access a rich and high-quality dataset of random images, we may be able to build meta-models that can distinguish the poisoned/clean models with acceptable performance.

1.5 Dissertation Outline

The upcoming chapters of this dissertation are as follows. In Chapter 2, we go over the related works, and current solutions. In Chapter 3, we iterate through the necessary backgrounds. In Chapter 4 we propose an integrity-preserving solution for deep learning training. In Chapter 5, we propose a privacy-preserving solution for genomic analytics and inference. Next, in Chapter 6, we present a privacy-preserving fairness analytics framework for machine learning. Afterwards, in Chapter 7 we conduct an analysis on detection methods for neural network trojaning attacks. Finally, in Chapter 8 we conclude this dissertation.

CHAPTER 2

RELATED WORKS

In this chapter, we cover the relevant related works in the literature, that have tried to bridge the gap between different disciplines to realize the goal of end-to-end secure machine learning operations using TEE.

2.1 Oblivious Random Access Memory

Oblivious Random Access Memory (ORAM) first introduced in (Goldreich and Ostrovsky, 1996) which enables arbitrary software programs to access/modify RAM locations independent of the data fed to the program [1]. An application that accesses memory addresses in an oblivious manner eliminates chances of being exploited by malicious adversaries that have access to the machine that hosts the application. For instance, in case of sorting an array which stores patients' diseases, when an adversary is able to see the memory access locations, they can infer the ordering of elements in the array even if they may not have access tho the contents of the array. Further, if the cancer patients have the lowest ordering, then all the records can easily be tied to the elements of the array that comes first during merge comparisons (merge-sort (Bron, 1971)).

So far, there has been a fair amount of focus on improving the performance of ORAM algorithms, and ORAMs with tree layout (Stefanov et al., 2013; Wang et al., 2015; Devadas et al., 2016) have become the most popular candidate. Additionally, there have been efforts to propose efficient oblivious data-structures (Wang et al., 2014), and data-oblivious compilers (Liu et al., 2015; Zahur and Evans, 2015). Further, there have been previous work in porting the ORAM architectures for secure processor and TEEs (Sasy et al., 2017; Maas

[1] It is acceptable to have different access pattern when data size is different, however for a fixed data size the algorithm memory access pattern must remain indistinguishable.

et al., 2013), because the oblivious primitives are not readily available in their architectural design. However, AISolace do not rely on ORAMs customized for TEEs, and builds upon a set of data-oblivious primitives which are explained further below. Finally, (Shaon et al., 2017; Zheng et al., 2017) are some prominent examples of data analytics tools developed for TEEs that are data oblivious.

2.2 Privacy & Integrity Preserving Computation Using Trusted Execution Environments

The line of research((Mohassel and Zhang, 2017; Phong et al., 2018; Juvekar et al., 2018; Mohassel and Rindal, 2018)) purely relying on cryptographic primitives has shown promising progress. In addition, there has been efforts (Knott et al., 2021) to utilize accelerated processors such as GPU to perform the computationally-heavy cryptographic logic. However, the computational overhead is nowhere near acceptable, particularly in case of training of very deep networks.

In the cross-road of TEE, ML, and DL, there has been a good deal of research work that mostly focus on bringing privacy, and integrity for training and inference. (Hunt et al., 2018; Kunkel et al., 2019; Hynes et al., 2018; Hanzlik et al., 2021; Ohrimenko et al., 2016) provide privacy for training/inference models solely inside SGX. In addition, there have been other efforts (Ng et al., 2021; Tramer and Boneh, 2018; Asvadishirehjini et al., 2020; Ozga et al., 2021; Hashemi et al., 2021) to integrate TEE for ML/DL stack with GPU to mitigate the extraordinary performance gap of the CPU (SGX is only available on the CPU).

CHAPTER 3

BACKGROUND AND RELATED WORKS

In this chapter, we give an overview of the background knowledge needed in the remaining chapters.

3.1 Trusted Execution Environments (Intel SGX)

TEE is a hardware-level feature of modern processors that provides a confidential and tampering-resistant area of main memory for code and data of software applications. This area is commonly referred to as *enclave*. An application that runs in the enclave has an important guarantee; all the run-time data stored in the heap (or stack) are not to be accessed or modified by any privileged programs, such as the operating system (OS). Therefore, the execution flow cannot be altered in a way that jeopardizes the computational integrity of the application.

Intel Software Guard Extensions (SGX) (Costan and Devadas, 2016) is an example of a common TEE that is available in many modern-day computers and *existing cloud infrastructure such as Microsoft Confidential Computing Cloud* (Russinovich, 2016). It provides a secluded hardware reserved area, namely, processor reserved memory (PRM), that is kept private (i.e., it is not readable in plaintext) from the host, or any privileged processes, and is free from *direct* undetected tampering. In addition, SGX enables applications that run within the enclave to attest to the outside world that they are being run on a legitimate hardware. With *remote attestation* (Johnson et al., 2016), users can attest the running code within the enclave before provisioning their secrets to a remote server. This feature enables a TEE to be used in cloud environments (Tian et al., 2019; Russinovich, 2016) where it can receive or send secrets remotely.

Calls from routines that should transition to/from enclave are handled through predefined entry points that are called Ecall/Ocall that must be defined in advance, before

building the enclave image. While it provides security and privacy for applications, (e.g., (Priebe et al., 2018; Shaon et al., 2017; Kunkel et al., 2019)), directly running unmodified applications inside SGX can induce a significant hit on performance because the memory and computational capacity are limited. This is especially the case for applications that require large amounts of memory, such as training deep neural networks. Specifically, SGX partitions an application into two components: 1) the Untrusted Application (UApp) and 2) the Trusted Application (TApp) that is loaded inside the enclave. TApp is responsible for managing data-sensitive tasks that are not safe for UApp to perform. To materialize the communication between TApp and UApp, special bridge functions need to be described in Enclave Definition Language (EDL) files. Without loss of generality, there are two types of bridge calls: 1) Calls from UApp to TApp – ECall and 2) Calls from TApp to UApp – OCall. At the moment, Intel supports a SDK in C/C++ language with limitations that supports a subset of the standard library for those languages. There are some ongoing efforts for making development more convenient by supporting other programming languages such as Rust (Wang et al., 2019) with better memory safety guarantees. Moreover, to facilitate porting of big projects (e.g., TensorFlow), Graphene (Tsai et al., 2017) minimizes the need for source code modifications and provides a wrapper container for the program to run on the SGX enclave. Nonetheless, these ongoing efforts do not mitigate potential leakages through the *memory access pattern* side channel.

Finally, there are a set of side-channel attacks that exploit micro-architectural capabilities (vulnerabilities) of modern TEE processors (e.g., out of order execution) in recent years, which are out of the scope of this dissertation. However, the reader can refer to (Murdock et al., 2020; Qui et al., 2020; Chen et al., 2019; Bulck et al., 2018; Götzfried et al., 2017; van Schaik et al., 2021; Brasser et al., 2017) for a list of important example such attacks.

```rust
// courtesy to github.com/mobilecoinofficial/mc-oblivious
fn cond_assign(cond: bool, src: &f32,dest: &mut f32)
{
    let mut temp = *dest;
    unsafe {
        llvm_asm!(
            "
            test $1, $1
            cmovnz $0, $2
            "
            :"+&r"(temp)
            : "r"(cond), "rm"(*src)
            : "cc"
            : "volatile", "intel");
    }
    *dest = temp;
}
fn cond_swap(cond: bool, a: &mut f32, b: &mut f32)
{
    let temp = *a;
    cond_assign(cond,b,a);
    cond_assign(cond,&temp,b);
}
fn cond_select(cond: bool,a: &f32, b:&f32, c:&mut f32)
{
    cond_assign(cond,a,c);
    cond_assign(!cond,b,c);
}
```

Listing 1: Data-oblivious conditional (branch-less) routines

3.2 Data-Oblivious Execution

Here, we go over a number of the fundamental data-oblivious primitives that are used extensively throughout this dissertation. Basic conditional logic, sorting and filtering records are important operations. By ensuring its data-oblivious execution, applications increase the privacy guarantee for the users.

3.2.1 Oblivious Conditional Assignments

There are numerous cases where data oblivious primitives deal with conditional logic and branching. To handle those cases we need to rely on CPU registers so that an adversary is not able to observe the move operation based on the conditionals derived from the contents of the data. Refer to (Rane et al., 2015) for a detailed usage of in-register conditional move instructions. Listing 1 shows a simple way of deriving conditional routines based on CMOV instruction.

3.2.2 Oblivious Sort

Commonplace sort methods such as Quick-Sort (Hoare, 1961), are prone to leaving the traces of comparison during their execution. This gives a malicious adversary, who has the knowledge of the sorting criteria, the ability to infer sensitive information about the contents of the rows even when they are encrypted. AISolace uses Bitonic-Sort (Batcher, 1968) that has a fixed order of comparison given the dataset size. However, the running time is $O(n(lg(n)^2)$ which is less efficient. Another, particular part that needs a special care, is swapping the rows in the dataset, which we rely on the oblivious conditional swap primitive.

3.2.3 Oblivious Filter

For filtering the rows based on a predicate (Zheng et al., 2017; Arasu and Kaushik, 2013), a new column is amended to the dataset, which indicates if the row should be selected. The dataset is traversed and the value of the new column is assigned obliviously. In the end, the data is obliviously sorted so that the passing rows are on top, and the non-matching rows are dropped from further processing.

3.2.4 Oblivious Grouping

In this dissertation we use the primitive introduced in (Arasu and Kaushik, 2013) for generic grouping operations. In short, the dataset is sorted obliviously, based on the grouping columns. Next, an incremental ID is assigned to each row in the group using oblivious conditional assignment routines. (Listing 1). Afterwards, by traversing the sorted dataset backwards, running aggregates are computed, and when passing the boundary of one group to the other, AISolace obliviously resets the running aggregates. Finally, the rows are obliviously filtered and only row s with $ID = 1$ are kept (because the final aggregation result are stored in rows with $ID = 1$).

3.3 Machine Learning & Deep Learning

Machine Learning (ML) offers a systematic solution for decision-making problems based on data gathered from past events to predict *future* events. The most popular style of problems that fall into the realm of ML are *classification*, and *clustering* problems. Classification problems deal with ML algorithms wherein the association of each entry in the past data (training set) is known to belong to a specific category (usually called class) and the model solves the association for unseen data (future events). SVM (Boser et al., 1992), and Random Forest (Breiman, 2001) are two of the most popular traditional machine learning classification algorithms. Unlike classification, clustering problems, build models wherein there is no pre-specified association for the entries in the training set, and the goal is to put the entries in the training set into clusters (groups) where they are closest compared to other clusters. K-Means (Hartigan and Wong, 1979), K-Modes (Chaturvedi et al., 2001), and K-Prototype (Huang, 1997) are some renowned examples of the most used clustering algorithms in ML.

In the past decade, with the popularization of special hardware such as Graphical Processing Unit (GPU), and Tensor Processing Unit (TPU), a new paradigm for solving ML

problems has emerged which is usually referred to as Deep Learning (Goodfellow et al., 2016c) (DL). DL based models have been extensively used to solve problems in computer vision, speech recognition, etc. DL is currently dominating the field of ML, particularly when the training set is humongous (traditional ML cannot scale), or the nature of the underlying data fits best for DL (e.g., video stream classification). AISolace only supports DL model inference where it is necessary to run an inference to obtain the prediction.

3.3.1 Deep Learning Training

Over the past decade, Deep Neural Networks (DNN) have become popular for solving problems related to computer vision and natural language processing (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; He et al., 2016; Szegedy et al., 2015, 2017). In practice, these networks are stacks of layers, each of which perform a transformation $\mathcal{F}^l_{\mathcal{W}}(\cdot)\ \forall l \in |L|$ where $\mathcal{X}^{l+1} = \mathcal{F}^l_{\mathcal{W}}(\mathcal{X}^l)$ and $|L|$ is the number of layers. The training task is to learn the correct parameters (point-estimates) $\mathcal{W}^*$ that optimizes (commonly minimizes) a task-specific (e.g., classification) loss function $\mathcal{L}$. The most common approach to training the parameters of a DNN is through mini-batch Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951). A randomly selected mini-batch of a dataset is fed to the DNN and the value of objective function $\mathcal{L}$ is calculated. This is usually referred to as the *forward* pass. Next, to derive the partial gradients of $\mathcal{L}$ with respect to $\mathcal{W}$ ($\nabla^{\mathcal{L}}_{\mathcal{W}}$), a *backward* pass is performed (Goodfellow et al., 2016b). Finally, the parameters are then updated according to Equation 3.1, where $0 < \alpha < 1$ is referred to as the learning rate. Depending on the complexity of the dataset and the task, this process might require hundreds of passes (called *epoch*) over the input dataset to achieve convergence.

$$\mathcal{W}^{t+1} = \mathcal{W}^t - \alpha \nabla^{\mathcal{L}^t}_{\mathcal{W}^t} \tag{3.1}$$

3.3.2 Gradient Clipping

Gradient Clipping (GC) is a method that has been shown to help mitigate the problem of exploding gradients during training (Goodfellow et al., 2016a). Simply, GC forces the gradients into a narrow interval to prevent very large updates during the SGD step. There have been some efforts to analyze GC with respect to convergence. For instance, Zhang and colleagues (Zhang et al., 2019) prove (assuming a fixed step size) that training with GC can be faster than training without it. Moreover, their theoretical analysis suggests that too small clipping values can reduce the training performance (i.e., require more steps for convergence). However, in practice, this performance reduction is rarely observed. (Chen et al., 2020) has an interesting theoretical analysis coupled with empirical evidence (symmetry of gradients distribution with respect to the SGD trajectory) that answers the gap between previous theoretical and practical observations. These results suggest that GC could be leveraged in practical DNN training without significant performance issues.

3.4 Fairness In Machine Learning

As delineated in the previous section, especially in the case of a *classification* task, the outcome associates a piece of data to a category (class). When that piece of data is related to individuals, then the prediction category of the model for individuals becomes of utmost importance, because the ramifications of the prediction can impose lasting consequences. For instance, the system Correctional Offender Management Profiling for Alternative Sanctions (COMPAS), allows the criminal justice system to evaluate how risky it is to release the offenders and it has been shown that COMPAS continually present a high degree of bias against African-American offenders relative to Caucasian offenders (ProPublica, 2016). There have been other instances of systems that utilize ML to make decisions significantly that are more favorable to the majority population over the minority population (majority

vs. minority is context-dependent) (Chouldechova et al., 2018; Lambrecht and Tucker, 2019; Raji and Buolamwini, 2019).

Fairness is not a new concept, however, upholding fairness in algorithms (particularly ML) is a brand-new direction of research that has consumed a great deal of attention. There are dozens of definitions for fairness, however most of them fall into three categories of group-based, individual-based, and subgroup-based fairness. Equality of odds (Hardt et al., 2016), equality of opportunity (Hardt et al., 2016), demographic parity (Dwork et al., 2012), and (conditional) statistical parity (Corbett-Davies et al., 2017) are a few famous examples of fairness definition metrics. Finally, we advise the reader to read the following surveys (Mehrabi et al., 2021; Chouldechova and Roth, 2018) to gain more insights on the current state of affairs in fairness for ML.

3.5 Trojan Attacks On Deep Learning Training

Attacks on DNN models can be realized during both *training* or *test* phases. However, GINN is concerned with integrity issues during the training phase of DNN models, such that attacks related to testing are out of the scope of this dissertation since test time attacks have been addressed before (e.g., *Slalom* (Tramer and Boneh, 2018)). In the literature, particularly in the computer vision community, targeted trojan attacks on DNN classification models have become a real concern as deep learning has grown in its adoption. These attacks tend to alter the prediction of models when a specific condition in the input is met. These conditions may be *feature-based* (Gu et al., 2017a,b) or *instance-based* (Shafahi et al., 2018). As an instance, consider a traffic sign classification model in an autonomous vehicle. For a feature-based attack, it can be anything as simple as having a small yellow square stamped on a stop sign which will lead to misclassification to "speed limit 75". Recently, trojan attacks were extended to Reinforcement Learning (RL) and text classification models (Kiourti et al., 2019; Sun, 2020).

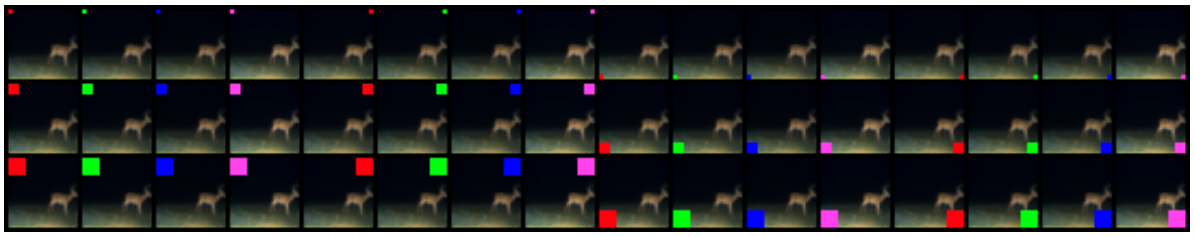


Figure 3.1: All backdoor trigger patterns

In practice, these attacks are implemented by manipulating samples during training through data poisoning. This could be achieved, for instance, by stamping images with a certain pattern and modifying the label of the image (e.g., changing "dog" to "cat"). Notably, these models provide similar competitive classification test accuracy when compared to clean models (i.e., models that have not been attacked). As a consequence, it is non-trivial to distinguish trojaned models from non-trojaned models based on model accuracy alone. To make matters worse, even if the model owner was aware of examples of the trojan trigger pattern, the owner would need to patch the model through a retraining process to dampen the efficacy of the trojan trigger pattern. Retraining does not always guarantee complete removal of the trojan behavior from the model. Various techniques have been proposed to diagnose and mitigate the effect of trojaned models. However, all approaches developed to date are either 1) based on unrealistic assumptions or 2) are excessively costly. For instance, Neural Cleanse (Wang et al., 2019) requires access to a sizable sample of clean inputs to reverse-engineer the backdoor and has shown to be successful only for trigger patterns with a relatively small size. ABS (Liu et al., 2019) improves upon Neural Cleanse in that requires a significantly smaller number of samples; however, it assumes that the neurons responsible for the trojan can activate trojan behavior independently of each other, which is unlikely to be true in practice.

Most common-place backdoor attacks (Liu et al., 2017; Gu et al., 2017b) involve training/retraining (for a small number of epochs (Liu et al., 2017)) and usually are instrumented through *data poisoning*. Usually, if the mini-batches contain 0.5%-5% of poisoned samples

that both have a backdoor trigger pattern, and also supplied with the attacker's desired label c^t. These backdoor attacks are very successful while incurring a trivial performance degradation, which makes it a cumbersome task for the users to doubt the healthiness of the model. In figure 3.1, the patterns we used for our attack is illustrated. For a comprehensive list of the attack, and configurations refer to Table 7.1. Current mitigation techniques (Liu et al., 2019; Wang et al., 2019; Gao et al., 2019) either are resource-consuming or have assumptions that are not realistically modeling the enormous vector of possibilities that are at the behest of the attackers.

3.6 Integrity Protection for DNN Training

(Tramer and Boneh, 2018) took the first step towards achieving both *fast* and *reliable* execution in the *test* phase, but neglected the training phase. The training phase is far more computationally demanding than the test phase, such that verification of all steps in training requires a substantially longer time because 1) parameters continuously changing, and 2) the backward pass involves computing gradients for both the inputs and the parameters, which requires a larger amount of time than the forward pass alone. Moreover, attacking the training pipeline to inject a trojan in the final model is quite simple (i.e., replace the actual model with the desired attack model by modifying the last SGD update) and, thus, is likely the most desirable form of attack for real-world adversaries to launch. Altogether, throughout this work, we mainly focus on showing the effectiveness of our method at preventing this type of attack from being successful. The main objective of GINN is to enable a high-integrity training pipeline so that the users are assured that the model 1) is built on the correct dataset and 2) uses the correct parameters without modification. Thus, the final model users know who built the model, what dataset was used for training, and what algorithms were put in place for building the model. If, at any point during the training,

GINN detects a deviation from the specified execution, it will not approve the final model to ascertain its validity.

GINN relies upon the *proactive* training as opposed to the post-training or deployment-time methods to protect the health of a DNN model. It should be noted that our approach is *independent of the attack strategy* and is sufficiently *generic* to catch any continuous attack during the training of a DNN model. As we explain later in Chapter 4, we assume that the initial training dataset is provided by an honest user and is free of manipulation. With this as a basis,

GINN limits the amount of change an adversary can inflict on a model through a single SGD step. As a consequence, the adversary is forced to keep attacking while being verified at random by the TEE. In particular, in an era where vectors of new attacks on DNNs are unfolding, a majority of them are via data-poisoning in training phase. Unfortunately, it is nearly impossible to detect them during deployment unless a special input is fed to the network that would trigger the anomalous behavior. (Gu et al., 2017a,b; Chen et al., 2017; Wang et al., 2020).

CHAPTER 4

GINN: FAST GPU-TEE BASED INTEGRITY FOR NEURAL NETWORK TRAINING

4.1 Introduction

Deep learning (DL) is radically changing how machine learning interacts with, and supports, society. Numerous industries increasingly rely on DL models to make decisions, ranging from computer vision to natural language processing. One example is, Tesla that relies on DL for its *autopilot* mode of operation (Csongor, Csongor) and it seems that, in the near future, DL will be a routine technique for enabling autonomous vehicles to commute on roads. Another prominent application of DL is Covid-19 diagnosis (Panwar et al., 2020), where wrong decisions can have irreversible consequences. The training process for these DL models requires a substantial quantity of computational resources (often in a distributed fashion) for training, which traditional CPUs are unable to fulfill. Hence, special hardware, with massive parallel computing capabilities, (e.g., GPUs, and TPUs) are often utilized. At the same time, the DL model training process is increasingly outsourced to the public cloud. This is natural, as applying cloud services (e.g., Amazon EC2, Microsoft Azure, or Google Cloud) for DL training can be more fiscally palatable for companies, while also enabling them to focus on the software product without worrying about the hardware maintenance and quality service level agreements of the hardware. Nevertheless, such outsourcing raises numerous concerns with respect to the privacy and integrity of the learned models. In recognition of the privacy and integrity concerns around DL (and machine learning (ML) in general), a considerable amount of research has been dedicated to applied cryptography, in three general areas: 1) *Multi-Party Computation (MPC)* (e.g., (Mohassel and Zhang, 2017)), 2) *Homomorphic Encryption (HE)* (e.g., (Gilad-Bachrach et al., 2016)), and 3) *Trusted Execution Environment (TEE)* (e.g., (Hunt et al., 2018; Hynes et al., 2018)). However, the

majority of these investigations are limited in that: 1) they are only applicable to simple shallow network models, 2) they are evaluated with datasets that have a few records (such as MNIST (LeCun and Cortes, 2010) and CIFAR10 (Krizhevsky, Nair, and Hinton, Krizhevsky et al.)), and 3) they incur an amount of overhead that is unacceptable for real-life DL training workloads.

In an effort to mitigate some of these problems, and securely move from CPUs to GPUs, the *Slalom* (Tramer and Boneh, 2018) system was developed to focus on the computational integrity at the *test* phase while depending on the application context. It can also support enhanced data privacy. However, at a much greater performance cost. Similarly, *Goten* (Ng et al., 2021) introduced a privacy-preserving training and inference solution based on TEEs and GPUs. However, the empirical analysis suggested it is still not feasible to both enable privacy, and integrity with an acceptable performance comparable to the non-private setting (using GPU).

To address some of these limitations, we introduce **GINN** (see Figure 4.1); a framework for *integrity-preserving* learning as a service that provides integrity guarantees in outsourced DL model training in TEEs. We assume that only the TEE running in the cloud is to be trusted, while all the other resources such as GPUs can be controlled by an attacker to launch an attack (e.g., insert a trojan). In this context, our goal is to support realistic DL training workloads while *ensuring data and model integrity*. To achieve this goal, we focus on the settings where maintaining the learning process's integrity is critical, while the training data may not contain privacy-sensitive information. For example, for a traffic sign detection model on public traffic sign images, it is desirable to prevent malicious behaviors that can put the pedestrians or the driver in harm's way. This is because safety is of paramount importance in the above scenario and trainers must ensure the integrity of the training process to avoid unintended consequences.

The trivial approach of executing the entire learning process inside a TEE is not scalable. This is mainly due to the fact that TEEs based on CPU extensions are substantially slower

compared to GPUs. It should be noted that performance improvement techniques, such as random matrix verification (Tramer and Boneh, 2018)), have been proposed, but the gains achieved are insufficient to scale up to large DL model learning settings. To significantly improve the performance of pure TEE based approach (i.e., running entire DNN training inside the TEE), we introduce an approach that incorporates *randomized verification* into the DNN training process. This strategy is based on the observation that it is unnecessary to verify all computation steps of the GPU during the DNN training. Rather, we only need to occasionally verify to ensure a very high likelihood of catching any deviation.

Unfortunately, naive randomized verification of the DNN training steps may not be enough. Because, if the DNN training steps is not adjusted properly, even an attack in one step could have a devastating impact. For example, DNN training usually require stochastic gradient descent (SGD) based updates using the current batch of the data. If there are no limits on the SGD update step, the attacker can arbitrarily modify the model even in a single update. Given that randomized verification may itself be insufficient, we further show how parts of the DNN hyperparameter setting process, such as *clipping rate* should be modified to prevent single step attacks, and require a larger number of malicious updates by an attacker that controls the GPU. In other words, **GINN** limits the amount of change an adversary can inflict on a model through a single SGD update step. As a consequence, the adversary is forced to keep attacking while, randomly, being verified by the TEE. Using state-of-the-art backdoor attacks, we illustrate that a randomized verification technique can detect attacks with a high probability (e.g., 0.999) while enabling 2x-20x performance gains compared to pure TEE based solutions.

The specific contributions of this chapter are as follows:

- We introduce the first approach to support integrity-preserving DNN training by randomized verification of stochastic gradient (SGD) steps inside TEE. This approach has an extremely high probability of ensuring the integrity the DNN training.

- We illustrate how gradient clipping can be used as a defensive measure against single (or infrequent) step attack in combination with randomized verification.

- We show the effectiveness of our TEE randomized verification and gradient clipping through extensive experimentation on DNN backdoor attacks.

4.2 Threat Model

Attacks on the integrity of DNNs can be orchestrated at different stages of the model learning pipeline (e.g., data collection or training). We assume the TEE node in GINN is trusted, and the bytes stored on the *Processor Reserved Memory* (PRM) are always encrypted and authenticated before they are fetched inside the CPU. We assume that the data sent to GINN is provided by honest users via a secure/authenticated channel and is devoid of malicious samples.[1] For the training phase, we assume that the adversary has complete knowledge about the network structure, learning algorithm, and inputs (after TEE performs an initial pre-processing) to the model. In our threat model, the adversary is in complete control of the host system's software stack, and hardware (unprotected RAM, GPU), except for the CPU package and its internals. Therefore, the code that runs inside the enclave is free from tampering, and the data that are accessed inside the cache-lines or registers are not accessible to the adversary. For the inputs supplied to DNN tasks, the adversary is capable of performing insertion, modification, and deletion to influence the final model towards her advantage. Finally, the adversary controls the communication between TEE and the user, but cannot impose denial of service attacks. As a result, an attacker may report falsified gradients at any time.

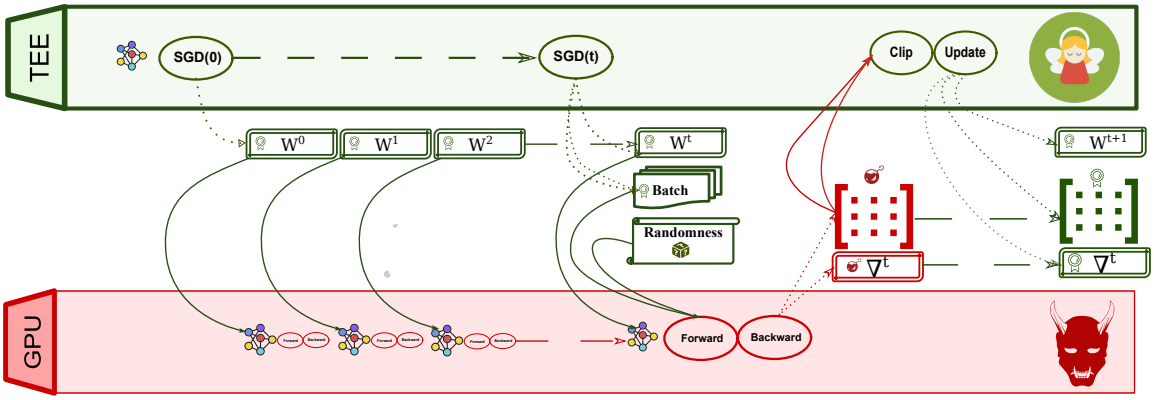


Figure 4.1: The main architecture of **GINN**. The TEE handles mini-batch selection, layer-specific randomness, and parameter initialization. The GPU performs forward and backward passes over the mini-batch (items selected by SGX provided seed) and reports the computed gradients to the TEE. TEE then clips the gradients and performs the weight update. Also, TEE preserves the MAC-authenticated intermediate gradient reports. During verification, TEE performs the forward and backward passes with the batch items along with layer-specific randomness (regenerated) and compares the gradients with the GPU's report.

4.3 System Design

GINN offers integrity for the training phase of a DNN model while inducing limited computational overhead. An overview of GINN is illustrated in Figure 4.1.

Training Setup Before the training phase initiates, the training dataset is decrypted and validated inside the TEE. We assume an honest and authenticated user will send her data encryption key K_{client} (after remote-attestation) to the TEE. Next, the TEE decrypts/verifies the initial encrypted dataset using the K_{client} and supplies the trainer (GPU) the plain-text of the training set. Lastly, the TEE allocates the necessary resources for the model and initializes the parameters with random values for the GPU to initiate the first SGD step.

Training with GINN 1) At the beginning of mini-batch SGD step **i**, the TEE supplies the untrusted GPU with the pseudorandom number generator (PRNG) seeds for the mini-batch selection and the per-layer PRNG seeds (e.g., Dropout (Srivastava et al., 2014)) that

[1]Detecting malicious samples is beyond the scope of this work.

are derived within the TEE and supplied to the GPU. As a result, the adversary will have to use the provided PRNGs for random choices. TEE generated randomness is applied to populate the data buffers of the GPU with the correct batch. Depending on the number of layers, other PRNG seeds will be generated for each layer to generate random values for operations of each layer. 2) After completion of the forward and backward passes over the mini-batch, the computed gradients are sent back to the TEE. In our design, the GPU always performs the forward and the backward passes and reports the computed gradients to the TEE. 3) GINN always clips the reported gradients and ensures that they are within a narrow range before performing the update *so that evolving the model towards the attacker's intended model requires a prolonged malicious intervention by the attacker.* 4) GINN updates the parameters ($O(N)$ complexity) with the clipped gradients and saves authenticated snapshots of the state outside the TEE. 5) If the computation at this step is randomly selected for verification, then the faulty behavior, if any exists, can be detected. Otherwise, the chance that the model has evolved towards the attacker's desired optima likely requires multiple rounds, which provides ample opportunities for detection. Additionally, the verification is performed randomly to prevent the attacker from guessing which step is likely to be verified. In the end, if the TEE does not detect any violation, it will certify the final model by digitally signing the model hash. As we stated, an honest and authenticated user will send her data encryption key K_{client} (after remote-attestation) to the TEE. Next, the TEE decrypts and verifies the initial encrypted dataset using the K_{client} and supplies the trainer (GPU) the plain-text of the training set. If the TEE fails to detect any violations of the protocol during training, it will sign the following message that certifies the final model where $\mathcal{W}$ is the model parameters, ds is training dataset, v_num is the model version number, $SHA256$ is Sha 256 bit cryptographic hash function,

$$SHA256(SHA256(\mathcal{W})||v\_num||SHA256(ds)||Sig^{v}_{client})$$

with signature key Sig^{s}_{SGX} of the enclave.

Probabilistic Verification with GINN The TEE randomly decides whether or not to verify the computation over each mini-batch. If the mini-batch is selected for verification, then the intermediate results are saved and the verification task is pushed into a verification queue. *Verification by the TEE can take place asynchronously* and it does not halt the computation for future iterations on the GPU. The authenticity of snapshots is always verified with a key that is derived from a combination of the TEE's session key, $SK^{session}_{SGX}$ and the corresponding iteration. When the TEE verifies the step $\mathbf{i}$, it populates the network parameters with the snapshot it created for the step $\mathbf{i-1}$. It then regenerates the randomness designated to step i to obtain the batch indices and correctly sets up the per-layer randomness. Given that the TEE's goal is to verify that the reported gradients for step i are correctly computed, GINN does not keep track of the activation results. Rather, it only requires the computed gradients, batch mean/std (for BatchNorm layer), and the matrix multiplication(MM) outcomes (in case random MM verification is chosen).

Randomized Matrix Multiplication Verification with GINN Matrix Multiplications (MMs) take up the bulk of the resource-heavy computations in DNNs. In modern DNN frameworks, convolutional and connected layers computation are implemented in the form of a matrix multiplication in both of the forward and backward passes. Table 4.1 depicts the computations in the forward pass and backward gradient with respect to the weights and previous layers' outputs in the form of a rank 2 tensor multiplication. Fortunately, there exists an efficient verification algorithm (i.e., Freivalds's randomized MM verification algorithm) for matrix multiplication((Freivalds, 1977)) when the elements of matrices belong to a field. In this work, we leverage the Freivalds's randomized MM verification algorithms as well.

Table 4.1: Matrix Multiplication Operations

Layer Type	Pass	Computation	Verification	(Sub)Batched/ Precomp.
Fully Connected	Forward	$\mathcal{O}_{[B][O]} = \mathcal{I}_{[B][I]} \times (\mathcal{W}_{[O][I]})^\intercal$	$\Upsilon_{[I][1]} = (\mathcal{W}_{[O][I]})^\intercal \times \mathcal{R}_{[O][1]}$ $\mathcal{Z}_{[B][1]} = \mathcal{I}_{[B][I]} \times \Upsilon_{[I][1]}$ $\mathcal{Z}'_{[B][1]} = \mathcal{O}_{[B][O]} \times \mathcal{R}_{[O][1]}$	YES / YES
	Backward Parameters Gradient	$\nabla^{\mathcal{W}}_{[O][I]} = (\nabla^{\mathcal{O}}_{[B][O]})^\intercal \times \mathcal{I}_{[B][I]}$	$\Upsilon_{[B][1]} = \mathcal{I}_{[B][I]} \times \mathcal{R}_{[I][1]}$ $\mathcal{Z}_{[O][1]} = (\nabla^{\mathcal{O}}_{[B][O]})^\intercal \times \Upsilon_{[B][1]}$ $\mathcal{Z}'_{[O][1]} = \nabla^{\mathcal{W}}_{[O][I]} \times \mathcal{R}_{[I][1]}$	YES / NO
	Backward Inputs Gradient	$\nabla^{\mathcal{I}}_{[B][I]} = \nabla^{\mathcal{O}}_{[B][O]} \times \mathcal{W}_{[O][I]}$	$\Upsilon_{[O][1]} = \mathcal{W}_{[O][I]} \times \mathcal{R}_{[I][1]}$ $\mathcal{Z}_{[B][1]} = \nabla^{\mathcal{O}}_{[B][O]} \times \Upsilon_{[O][1]}$ $\mathcal{Z}'_{[B][1]} = \nabla^{\mathcal{I}}_{[B][I]} \times \mathcal{R}_{[I][1]}$	YES / YES
Convolutional	Forward	$\mathcal{O}_{[f][w_o.h_o]} = \mathcal{W}_{[f][k^2.C_i]} \times \mathcal{I}_{[k^2.C_i][w_o.h_o]}$	$\Upsilon_{[1][k^2.C_i]} = \mathcal{R}_{[1][f]} \times \mathcal{W}_{[f][k^2.C_i]}$ $\mathcal{Z}_{[1][w_o.h_o]} = \Upsilon_{[1][k^2.C_i]} \times \mathcal{I}_{[k^2.C_i][w_o.h_o]}$ $\mathcal{Z}'_{[1][w_o.h_o]} = \mathcal{R}_{[1][f]} \times \mathcal{O}_{[f][w_o.h_o]}$	NO / YES
	Backward Parameters Gradient	$\nabla^{\mathcal{W}}_{[f][k^2.C_i]} = \nabla^{\mathcal{O}}_{[f][w_o.h_o]} \times (\mathcal{I}_{[k^2.C_i][w_o.h_o]})^\intercal$	$\Upsilon_{[w_o.h_o][1]} = (\mathcal{I}_{[k^2.C_i][w_o.h_o]})^\intercal \times \mathcal{R}_{[k^2.C_i][1]}$ $\mathcal{Z}_{[f][1]} = \nabla^{\mathcal{O}}_{[f][w_o.h_o]} \times \Upsilon_{[w_o.h_o][1]}$ $\mathcal{Z}'_{[f][1]} = \nabla^{\mathcal{W}}_{[f][k^2.C_i]} \times \mathcal{R}_{[k^2.C_i][1]}$	NO / NO
	Backward Inputs Gradient	$\nabla^{\mathcal{I}}_{[k^2.C_i][w_o.h_o]} = (\mathcal{W}_{[f][k^2.C_i]})^\intercal \times \nabla^{\mathcal{O}}_{[f][w_o.h_o]}$	$\Upsilon_{[1][f]} = \mathcal{R}_{[1][k^2.C_i]} \times (\mathcal{W}_{[f][k^2.C_i]})^\intercal$ $\mathcal{Z}_{[1][w_o.h_o]} = \Upsilon_{[1][f]} \times \nabla^{\mathcal{O}}_{[f][w_o.h_o]}$ $\mathcal{Z}'_{[1][w_o.h_o]} = \mathcal{R}_{[1][k^2.C_i]} \times \nabla^{\mathcal{I}}_{[k^2.C_i][w_o.h_o]}$	NO / YES

4.4 Integrity Analysis

To achieve our integrity goal p_i (i.e., the probability that an attacker can modify the result without being detected is less than $1 - p_i$), we need to derive the p_v (i.e., the probability that TEE verifies an SGD step).

4.4.1 Random Mini-Batch Verification

We define the total number of DNN training steps as B. For each step, report R_b ($\forall b \in [1, B] \wedge R_b \in \{0, 1\}$) has a probability p_c for being corrupted (i.e., $R_b = 1$) and the overall integrity probability goal of p_i (for example $p_i = 0.999$).

We define $V_b = 1$ if a batch is chosen by the TEE for verification, and the verification result indicates a malicious SGD step. If the verification passes, then $V_b = 0$.

We define random variable $X = \sum_{b=1}^{\lceil B \times p_v \rceil} V_b$ to be the total number of random verifications performed that resulted in catching a malicious SGD step. It should be recognized that the TEE needs to catch at least one deviation (i.e., $X \geq 1$) with probability greater than p_i to invalidate the overall model learning.

Theorem 1. *Given a total of B steps during SGD training, a P_c probability of an SGD step being malicious, and P_i probability of detecting at least one malicious SGD step, the required probability of choosing a step to verify (p_v) should be greater than* $B^{-1}(\frac{\log(1-p_i)}{\log(1-p_c)} - 1)$.

Proof. We need to ensure that $P(X \geq 1)$ (i.e., the probability that at least one deviation is caught) is greater or equal to p_i.

$$\begin{aligned}
P(X \geq 1) = 1 - P(X = 0) &\geq p_i \\
1 - p_i &\geq P(X = 0) \\
1 - p_i &\geq \binom{\lceil B \times p_v \rceil}{0} p_c^0 (1 - p_c)^{\lceil B \times p_v \rceil} \\
1 - p_i &\geq (1 - p_c)^{\lceil B \times p_v \rceil} \\
\log(1 - p_i) &\geq \lceil B \times p_v \rceil \log(1 - p_c) \\
p_v &> B^{-1}(\frac{\log(1 - p_i)}{\log(1 - p_c)} - 1) \qquad (4.1)
\end{aligned}$$

As shown in Figure 4.2 in 4.4.3, we only need to verify a small subset of the batch computations inside the TEE to ensure a high probability of correct computation. For example, for large datasets such as Imagenet (Deng et al., 2009), when corruption probability is %0.5, we need to randomly verify %1 of computation to achieve *0.9999* correctness probability on the computation outsourced to the GPU .

4.4.2 Random Mini-Batch Verification with Randomized Matrix Multiplication

Since Freivalds's randomized matrix multiplication verification scheme (Freivalds, 1977) is a randomized algorithm, it is possible that the scheme will falsely attest to the validity of the MM operations performed by the GPU. Thus, in addition to the previous configuration, the verification can be replicated k times to decrease the chance of encountering a false negative. The scheme has no false positive, so if the matrix multiplication is correct then the

verification succeeds. Given that each SGD step contains m independent MM operations, which are to be repeated k times (independently), and random values are uniformly sampled from a field of size $|S|$, the probability of error (attesting to the validity of a malicious MM operation) is less than $\alpha = \frac{1}{|S|^{mk}}$. We define the random variable $V'_b = 1$ if $R_b = 1$ (i.e., corrupt report of a malicious step) and $MM\_verify(b) = 0$ (Freivalds's verification rejects the equality), otherwise $V'_b = 0$

Also, define $X = \sum_{b=1}^{\lceil B \times p_v \rceil} V'_b$. We need to detect at least one deviation with probability greater than p_i while conducting random matrix multiplication verification.

Theorem 2. *If random matrix multiplication verification is applied, then, given the configuration of Theorem 1, the required probability of choosing a step to verify (p_v) should be greater than*

$B^{-1}(\frac{\log(1-p_i)}{\log((1+(\alpha-1)p_c)} - 1)$.

Proof. Again, we need to ensure that $P(X \geq 1)$ (i.e., the probability that at least one deviation is caught) is greater or equal to p_i.

$$
\begin{aligned}
P(X \geq 1) &\geq p_i \\
1 - P(X = 0) &\geq p_i \\
1 - p_i &\geq \binom{\lceil B \times p_v \rceil}{0} (p_c(1-\alpha))^0 ((1-p_c) + p_c\alpha)^{\lceil B \times p_v \rceil} \\
1 - p_i &\geq ((1-p_c) + p_c\alpha)^{\lceil B \times p_v \rceil} \\
\log(1 - p_i) &\geq \lceil B \times p_v \rceil \log((1-p_c) + p_c\alpha) \\
p_v &> B^{-1}(\frac{\log(1-p_i)}{\log((1+(\alpha-1)p_c)} - 1)
\end{aligned} \tag{4.2}
$$

The threshold in Theorem 2 yields approximately the same values as Theorem 1 when $\alpha \to 0$. However, the randomized MM verification requires a $O(N^2)$ operations for a $N \times N$ matrix, compared to an iterative algorithm that requires $O(N^3)$ operations.

4.4.3 Verification Probability Growth with Respect to Detection Probability

Fig. 4.2 shows how verification probability changes with respect to the probability that a batch step is maliciously manipulated by the attacker. The first row shows the verification probability for a dataset with 60K samples. The second row depicts the required for much bigger dataset (1M samples) over different mini-batch sizes. The smaller the mini-batch size is, there is a higher chance for detecting malicious behavior.

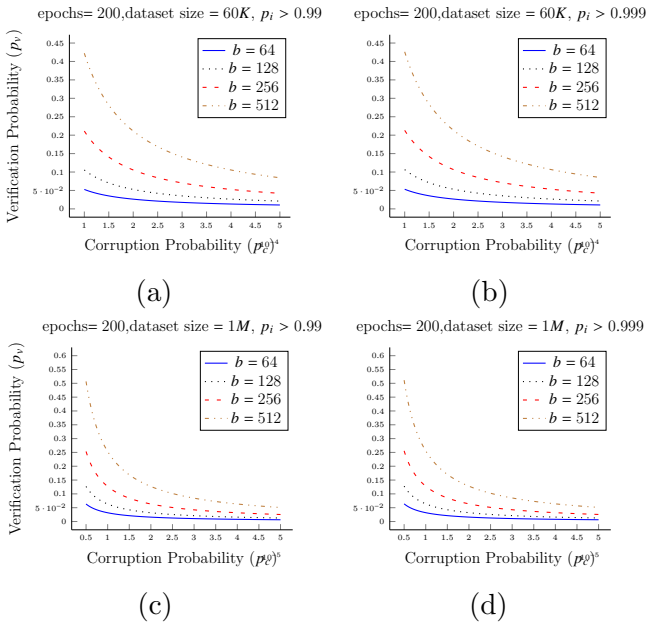


Figure 4.2: Required verification probability with respect to batch corruption probability and the desired integrity probability for a fixed 200 epochs and different SGD batch size.

4.5 Experiments

All of our experiments were run on a server with a Linux OS, Intel Xeon CPU E3-1275 v6@3.80GHz, 64GB of RAM and an NVIDIA Quadro P5000 GPU with 16GB of memory. Our attack code is implemented in python 3.6 using the PyTorch library. We use Intel SGX for the TEE plaform. For SGX proof-of-concept implementation, we significantly modified the DarkNet (Redmon, 2016) library to run the experiments.

Our SGX code has been tested with SDK 2.9 and the code runs inside a Docker container in hardware mode. Our experiments are designed to investigate efficiency and effectiveness. First, we evaluate the degree to which integrating randomized matrix verification influences the computational efficiency of the process. Second, we analyze the effectiveness of gradient clipping in forcing the attacker to deviate from the honest protocol in higher number of mini-batch steps. Various attack hyper-parameters (e.g., poisoning rate) were evaluated in determining their importance towards a successful attack with minimal deviation. This is important because, if the attacker needs to deviate in more mini-batch steps, then the TEE can detect such deviations with a smaller number of random verification steps (i.e., p_c is higher in equation 4.1).

The following is an enumeration of experiments in the order they are discussed. In section 4.5.1, we analyze the performance of our system for a large dataset. For this experiment we use ImageNet (Deng et al., 2009), which consists of RGB images of 1000 categories. Next in section 4.5.2 we analyze the performance for a much smaller dataset, CIFAR10 (Krizhevsky, Nair, and Hinton, Krizhevsky et al.). CIFAR10 dataset contains RGB images of 32 by 32 pixels in 10 categories of objects or animals. Then in section 4.5.3 we analyze the impact of the heap allocated to the enclave. For this experiment we used images from ImageNet dataset. In section 4.5.4 we seek to evaluate the impact of gradient clipping, in an attack scenario for three different datasets and contemplate about the potential verification rate necessary for a TEE to catch the malicious execution. CIFAR10, MNIST (LeCun and Cortes, 2010) (black-white digit images from 0 to 9), and GTSRB (Stallkamp et al., 2012) (RGB images of traffic signs) are used to conduct this analysis. Afterwards, in section 4.5.5, we investigate whether training with gradient clippings can have an unusually high impact on the quality of the models, especially if the chance of the training being attacked is not high. Finally, in section 4.5.6, we conducted a similar analysis on ImageNet (an instance of a very large dataset) to observe the potential impact of gradient clipping on big datasets.

4.5.1 TEE Performance On ImageNet With Common Architectures

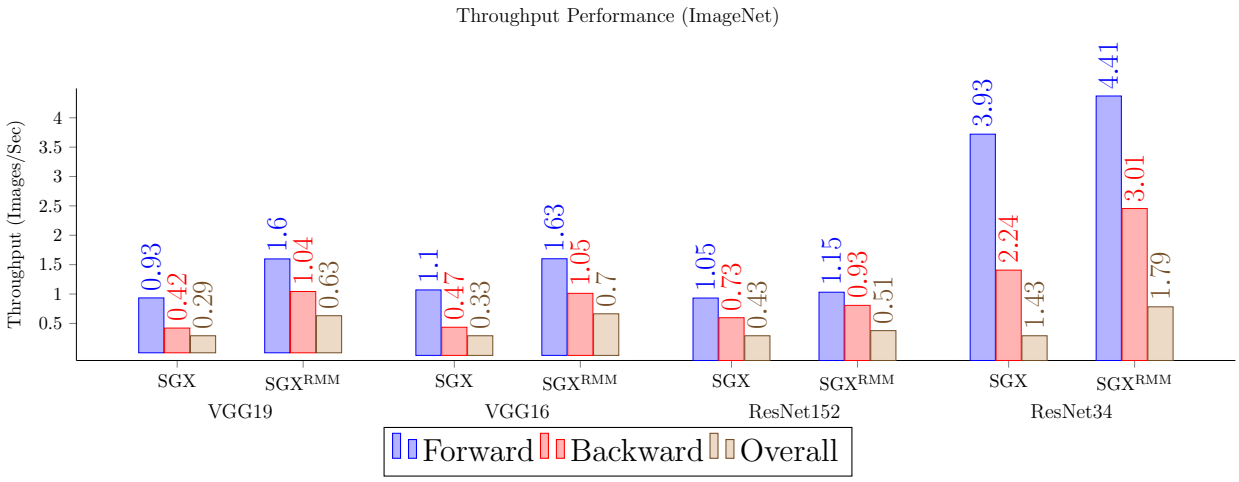


Figure 4.3: Throughput of the SGD training step for VGG19,VGG16, ResNet152, and Resnet34 on ImageNet dataset with respect to forward and backward passes. "SGX$^{\text{RMM}}$" refers to Freivalds' MM verification scheme, while "SGX" refers to the baseline case of fully computing the MM operation. RMM can lead to verification that is twice as fast as full MM verification in case of a VGG architecture.

We tested TEE performance on popular DNN architectures such as VGG16, VGG19 ((Simonyan and Zisserman, 2014)), ResNet152, and ResNet34 ((He et al., 2016)) with the ImageNet ((Deng et al., 2009)) dataset. Figure 4.3 illustrates the throughput of the deep networks (i.e., the number of images processed per second during the DNN training). Usually, most of the computation takes place in the convolution and fully-connected layers of the network. However, the backward pass yields a smaller throughput, on average, because it involves one more MM (weight gradients and input gradients) than the forward pass, which only invokes MM once (i.e., output of convolution or fully-connected layers). The implementation is quite efficient in terms of MM operations, which uses both vectorized instructions along with multi-threading. We note that the baseline GPU, which lacks TEE support and integrity protection, significantly outperforms the pure TEE baseline. For instance, the overall throughput for ResNet34 is about 60 images per second, whereas the pure SGX baseline solution is about 1.5 images per second.

Table 4.2: TEE Architectures Used

Arch	FC1	FC2	FC3
VGG11	(128,10)	(128,64,10)	(256,128,10)
VGG13	(128,10)	(128,64,10)	(256,128,10)
VGG16	(128,10)	(128,64,10)	(256,128,10)

Both VGG networks gives better improvements ($\approx 2.2\mathbf{X}$) than ResNet ($\approx 1.2\mathbf{X}$). Considering that the TEE randomly decides to verify an SGD step with probability p_v, the overall improvement is approximately multiplied by $\frac{1}{p_v}\mathbf{X}$. For instance, if we assume the attacker only needs to deviate with probability $7E{-}5$ (i.e., deviating only 70 steps out of 1M steps), then we can detect such deviation with $p_v \approx 0.1$ (See Figure 4.2d). For VGG networks, this means that approximately $22\mathbf{X}$ performance improvement in throughput compared to a pure TEE-based solution that verifies every step. Nonetheless, as our experiment results suggest, we believe that attacking deep models that are trained on very large datasets should require a significantly larger number of deviations. This, in return, may result in a much greater performance gain.

4.5.2 TEE Performance on CIFAR10

Figure 4.4 depicts the throughput performance for the CIFAR10 dataset and 9 different VGG architectures . We chose three popular VGG(11,13,16) architectures adapted for CIFAR10 image inputs with custom fully connected layers attached to its end. FC-1 is one layer (128×10), FC-2 is two layers (128×64, 64×10), and FC-3 is two layers (256×128, 128×10). For CIFAR10, our verification technique generally do not benefit from randomized matrix multiplication scheme as it did for ImageNet. This is mainly because most of the operations and network layers fit well within the hardware memory limit. Therefore, since the dimensions of MM operations are not too large, using randomized MM verification does not improve performance significantly.

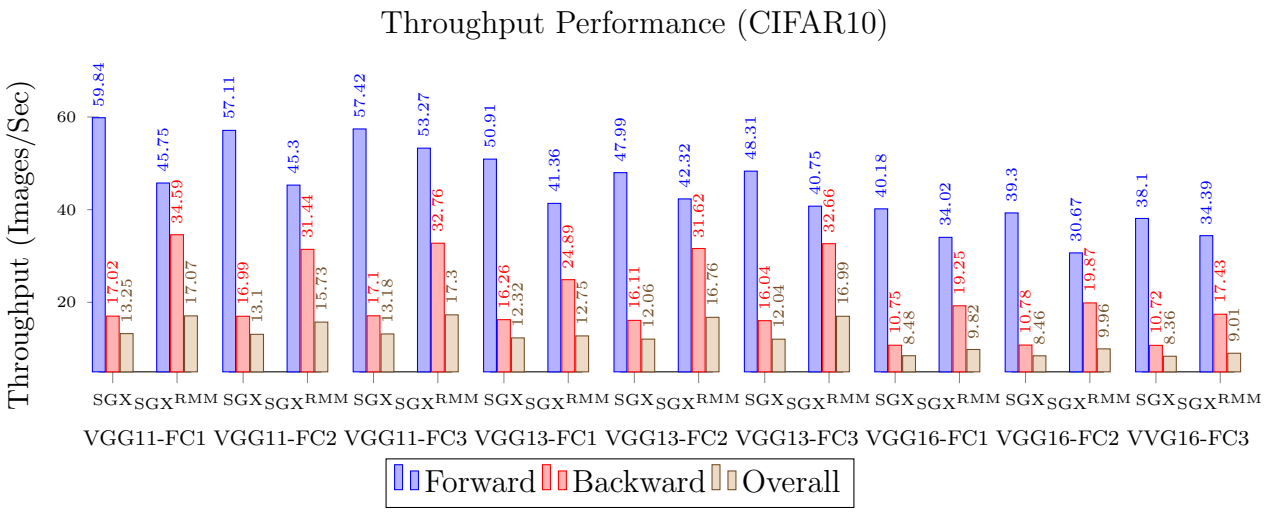


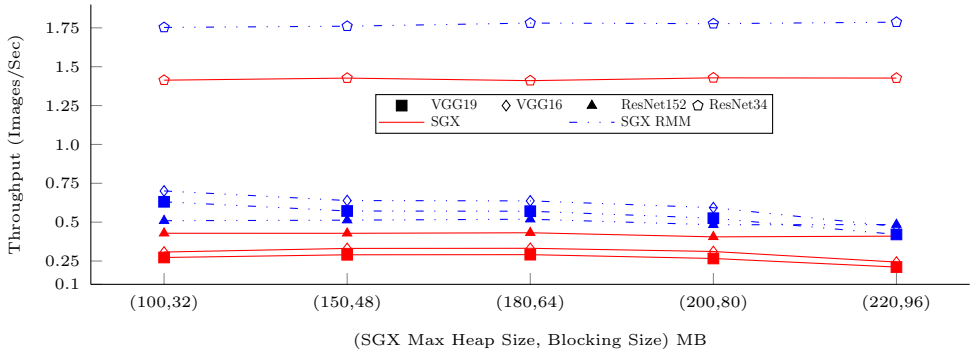
Figure 4.4: Throughput of SGD training step for VGG19,VGG16, ResNet152, and Resnet34 on CIFAR10 dataset. Randomized Matrix Multiplication (Freivalds' scheme) can make verification twice faster in case of VGG architecture.

4.5.3 Enclave Heap Size Impact on TEE Performance

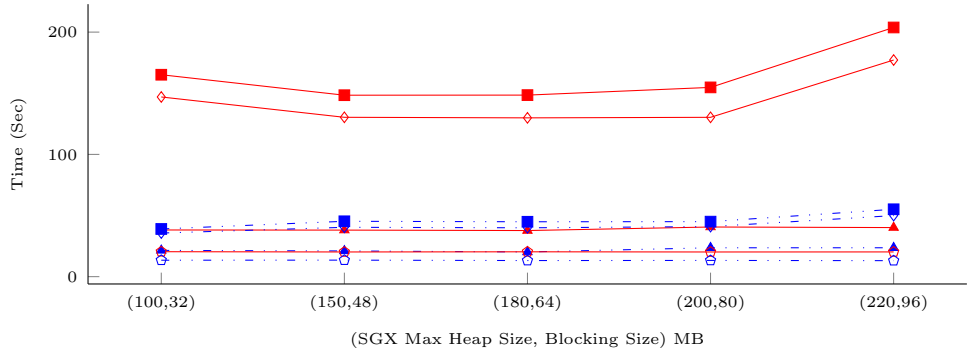
Figure 4.5 depicts the impact of the heap size on the performance of DNNs for a single SGD step. It can be seen that increasing the heap size substantially beyond the available hardware limit (to around 92MB) can induce a negative impact on the performance. This is especially the case for the VGG architecture. This result is mainly due to two factors. First, it causes driver level paging, which must evict enclave pages that require an extra level of encryption/decryption. Second, there is a non-trivial amount of extra bookkeeping required for the evicted pages.

4.5.4 Combined Impact of Gradient Clipping and Learning Rate on Attack Success

As we discussed, if the SGD updates are not bounded, the attacker can launch a successful attack by deviating from the correct training in a single update step. To prevent this, we require each update to be clipped using the gradient clipping approach. We conducted series



(a) Available heap with respect to throughput



(b) Available heap with respect to time spent on matrix-matrix(vector) multiplication

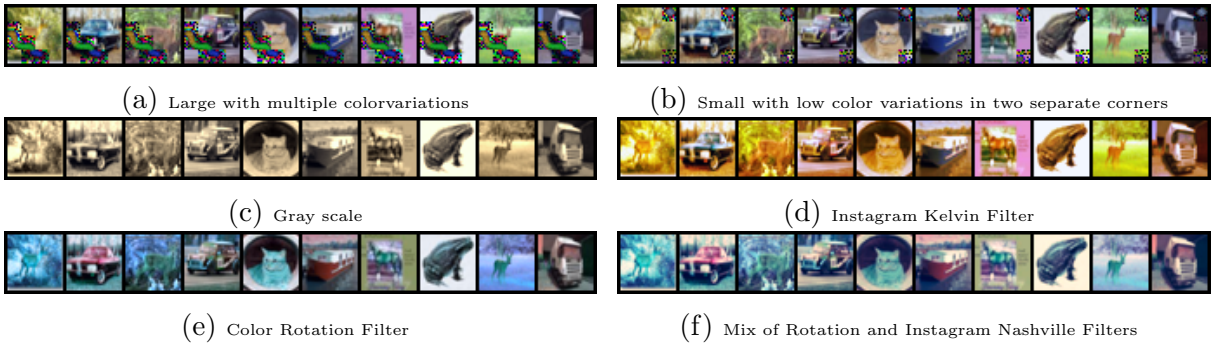
Figure 4.5: The impact of increasing TEE heap size on (a) overall throughput and (b) the time spent in matrix multiplication routine. VGG shows significant reduction in performance as opposed to ResNet.

Table 4.3: Trained models with restricted clipping and learning rate

Dataset	**clip**	**lr**	**%clean**	**%attack**	**total**
GTSRB	10^{-4}	10^{-4}	$\geq$ %95	$\geq$ %85	124
MNIST	10^{-4}	5×10^{-5}	$\geq$ %95	$\geq$ %85	49
CIFAR10	5×10^{-4}	10^{-1}	$\geq$ %90	$\geq$ %85	94

of experiments to understand how gradient clipping impacts the attacker success in poisoning the model during training. As shown in Table 4.3, we selected the models that achieved high performance on both clean and poisoned test samples (267 out of 600+). [2] First, during

[2]We also performed a series of experiment for the scenario where the attack is performed on steps chosen at random. However, this type of attack was not successful, such that we do not report those results here.



(a) Large with multiple colorvariations

(b) Small with low color variations in two separate corners

(c) Gray scale

(d) Instagram Kelvin Filter

(e) Color Rotation Filter

(f) Mix of Rotation and Instagram Nashville Filters

Figure 4.6: All examples of triggers on CIFAR10 images

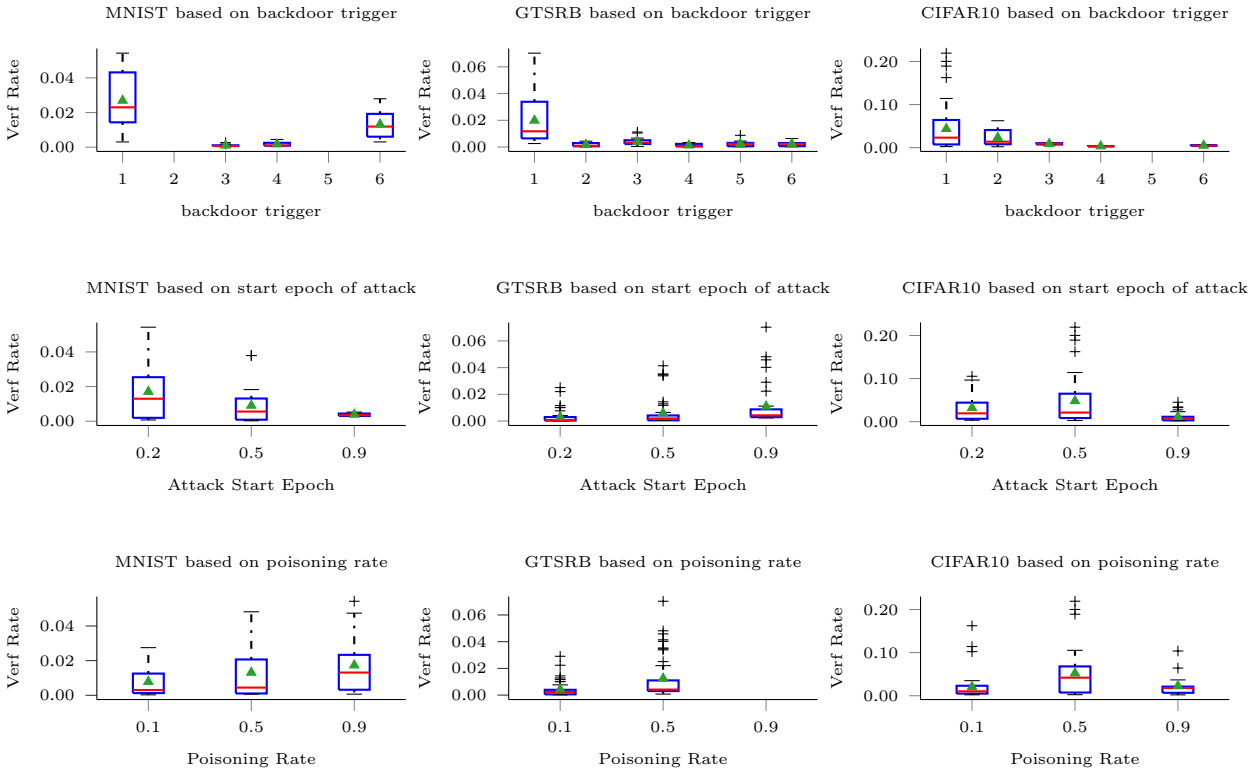


Figure 4.7: A boxplot representation of the verification rates required by TEE for detection failure $\leq 10^{-3}$. Based on the attack hyper-parameters: 1) backdoor trigger type (first row). 2) epoch that attack starts (second row, 20%×,50%×, and 90%×|epochs|). 3) backdoor poisoning ratio

the attack, attacker follows the correct protocol (mini-batch SGD) until $\boldsymbol{epoch}_{attack}$. At this epoch, the attacker starts attacking by injecting a certain number of poisoned samples

($pois_{rate} \times batch\_size$) from every class into the training batch and labels it as the target label. The attacker continues to attack until they achieve a desired threshold in terms of success rate (correctly classifying backdoored samples as the attacker's target label). Once it passes the threshold, the attacker halts the attack and returns to the honest protocol, while observing the decay in attack success rate. If the success rate falls below a desired lower threshold, the attacker transitions back to attack mode and repeats the aforementioned strategy. The CIFAR10 ((Krizhevsky, Nair, and Hinton, Krizhevsky et al.)), GTSRB ((Stallkamp et al., 2012)) and MNIST ((LeCun and Cortes, 2010)) datasets were used to analyze the impact of multiple factors imposed by the attacker. For MNIST, and GTSRB, the *Adam* (Kingma and Ba, 2014) optimizer (which requires a smaller initial learning rate), and for the CIFAR10 dataset, *SGD* with momentum are used. Additionally, for MNIST, and GTSRB 10% of the training set was chosen for the validation set to help adjust the learning rate based on the validation set loss. Same for the CIFAR10 dataset, the learning rate was set to decay (by tenfold) at fixed epochs $(40, 70, 100)$.

Backdoor Trigger Pattern

We applied 6 different backdoor triggers (Figure 4.6). The MNIST dataset only has single channel images, we converted it to a three channel image to apply the triggers 3 to 6. As shown in Figure 4.7 (first row) the trigger pattern can significantly influence the effectiveness of the attack. The red lines show the median, while the green dots correspond to the mean. In all of the datasets, the first trigger pattern (Figure 4.6a) was the most effective. This pattern covers a wider range of pixels compared to the second trigger type (Figure 4.6b). As a consequence, it is more likely that the model can remember the trigger pattern across longer periods of SGD steps. Moreover, since photo filters (e.g. *Instagram*) have become popular, we investigated the potential for conducting attacks using some of the filters (or transformations) as the trigger pattern. However, covering a very wide range of pixels does

not lead to a stealthy attack, as illustrated by the last four patterns. These patterns cover the whole input space and transform it to a new one that they share a lot of spacial similarities while only different in tone or scale (e.g. Figure 4.6d). Learning to distinguish inputs that are similar, and only different in their tone, is demanding in terms of continuity of the attack. In this case, both of the images (that is, with and without the trigger) are influencing most of the parameters and filters in a contradictory manner (different classification label). Thus, it takes a significant number of steps for the network to learn to distinguish them when gradient clipping is applied.

Attack Start Epoch

Another major factor influencing the evasiveness of the attack is when the attacker initiates the attack. For instance, early in the training phase the learning rate will be high, such that a savvy attacker might believe that they can avoid low clipping values by initiating their attack. However, if the attack begins too early, then it is unlikely that the model has yet converged. As a result, the attack may need to commit a substantially higher number of (unnecessarily) poisoned batches, which, in turn, would raise the probability of detection. Yet even if the attacker was successful, once they halt the attack, the model will likely evolve the parameters back to a clean state relatively quickly, such that, once again, the attacker would need to re-initiate their attack. Additionally, because of a low clipping value, if the attacker waits until later in the training process, the attack is again unlikely to be effective. In this case, this would mainly be due to the considerably smaller learning rate. As shown in Figure 4.7, the best time to attack is when 1) the model has a relatively low loss on clean training inputs and 2) the combination of learning rate and clipping value (effective attainable update) allows the model to move toward attacker's desired optima. For MNIST, which is a trivial learning task, attacking early endows the attacker with a better chance to launch a stealthier attack. We believe that this is due, in part, to the faster convergence

of the model. After a few epochs, the system quickly reaches a stable, low training loss for clean images. As a result, after reaching the desired attack success threshold, the attack success is generally preserved far longer than the other two datasets.

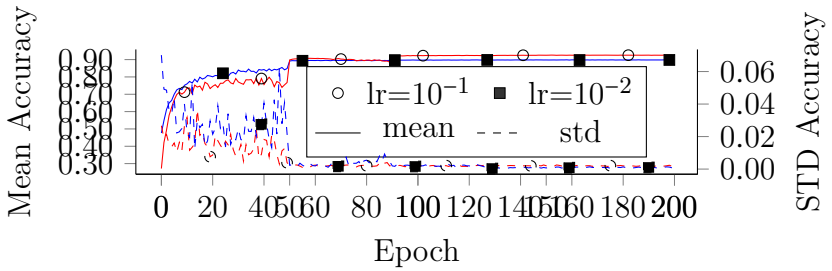
Mini-Batch Poisoning Ratio

Another critical factor is $pois_{rate}$, the ratio of the number of poisoned samples in the batch to the batch size. This is particularly the case when gradient clipping is applied. Setting $pois_{rate}$ appropriately can help the attacker by moving more parameters toward the desired optima. However, going beyond a ratio of 0.9 (i.e., $pois_{rate} > 0.9$) can impact the training negatively for both clean inputs and poisoned inputs. As depicted in Figure 4.7, our experiments suggest that filling more than half the batch with poisoned samples seems to be effective across all datasets. For the MNIST dataset, it appears that higher values can achieve slightly better performance, however, this finding is not replicated in more complex datasets. For example, for the GTSRB dataset, we did not observe a successful attack on the model where clean input accuracy is close to the clean input accuracy where there is no attack.

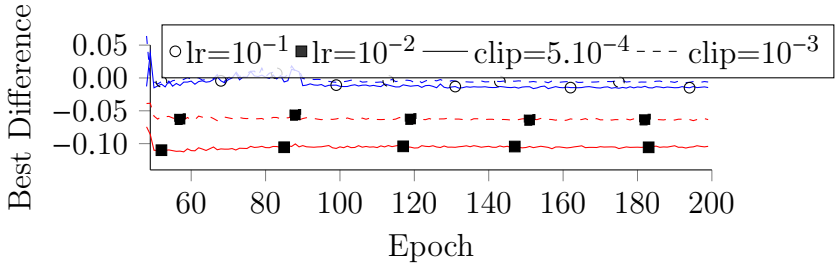
4.5.5 Impact of Gradient Clipping for Honest Trainers

One important question is whether the gradient clipping tat applied to prevent the attacker from changing parameters in a given mini-batch update can have a performance impact on training when there is no attack. We ran six experiment configurations, repeated 5 times, each with different randomness (initialization, batch order, etc.). Initial learning rates are set $\in (0.1, 0.01)$ and clipping thresholds are set $\in (nil, 0.001, 0.0005)$ (nil stands for no gradient clipping). In total, there were 30 ResNet56 (complex architecture with state-of-the-art performance) models trained on the CIFAR10 dataset (with no attack) for 200 epochs. Usually, for the *SGD* (Robbins and Monro, 1951) optimizer with momentum, a learning rate value of 0.1 is chosen, (and for *Adam* optimizer a value less than 0.001). The reader can refer

(a) Mean and STD of reference accuracy across 5 repeats for each epoch



(b) Highest test accuracy gap across 5 repeats



(c) Lowest test accuracy gap across 5 repeats

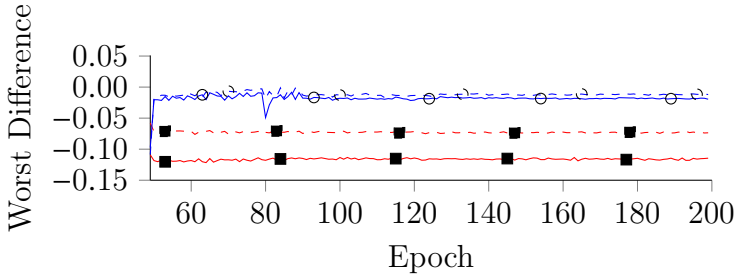


Figure 4.8: 4.8a Reference Models (no gradient clipping) mean/std on test accuracy of 5 repeats for two different learning rates. Each configuration had 5 repeats and a reference model (no attack and unbounded updates). 4.8b For each run configuration the test accuracy difference $diff_{lr,clip}$ is defined as $max\left(acc^{rep}_{lr,clip} - acc^{rep}_{ref}\right) \quad \forall rep \in [1,5]$. 4.8c $min\left(acc^{rep}_{lr,clip} - acc^{rep}_{ref}\right) \quad \forall rep \in [1,5]$

to (A., A.) for an introductory background on deep learning optimization paradigms. For these experiments, we used the configuration with unbounded gradient updates as the main reference point. For learning decay schedule, we used a fixed decay by tenfold at epochs (50, 90, 130, 160).

Figure 4.8a shows the mean and standard deviation (dashed lines) of test accuracy taken for 5 runs at each epoch for the two learning rate configurations. As it can be seen, both models start to take giant leaps toward convergence at the first two learning decays enforced by the scheduler. Note that these reference runs have no gradient clipping enforcement during the update step. Toward the end of training, the setting with the higher initial learning rate slightly outperforms in terms of accuracy ratio.

In Figure 4.8b, the largest difference (accuracy rate) with respect to the reference run is plotted for each combination of learning rate and clipping value. The plot shows that test accuracy is minimally influenced by the clipping value. Rather, the test accuracy is highly dependent on the learning rate value. When $lr = 0.1$, both clipping values can achieve values that are close to the reference runs that have no gradient clipping. In Figure 4.8c, the opposite - that is, the smallest difference - of the previous measure is plotted. Again, by the end of the training, the gaps are significantly reduced for the case where a better learning rate is chosen. Therefore, having a smaller clipping value has minimal impact on the performance, which is notable because it is crucial to set a reasonable learning rate.

Overall, Figure 4.8 shows that clipping has negligible impact on the learning task once an appropriate learning rate is chosen. It can be seen that, if the trainer chooses an acceptable learning rate for the task, small clipping values (e.g., 0.001 or 0.0005) do not impede the learning task. Once the model passes the first learning rate decay schedule, all of the configurations behaves the same in terms of their test performance compared to their reference model (no gradient clipping limit).

4.5.6 Gradient Clipping Impact Analysis On Large Dataset

We investigated our attack on the ImageNet dataset with the ResNet34 architecture. The state-of-the-art reference achieves an accuracy (rate) of 0.73. However, we only reinitialized and trained the fully-connected plus the last four convolutional layers (10M out of 21M

parameters). We used an SGD optimizer for 30 epochs, and the initial learning rate was set to 0.05. Additionally, the learning rate was set to decay (by tenfold) at fixed epochs $(4, 7, 13)$. Also, $pois_{rate}$ was 0.5, and the first trigger pattern (Figure 4.6a) was used as the only pattern (resized to 256x256). Finally, at the beginning of the third epoch (one epoch before the first decay) the attack started.

Gradient Clipping Impact on Poisoning Backdoor Attack Evasiveness

Table 4.4: ImageNet (Re-)Training on ResNet34's Last Five Layers with $epochs = 30$, $epoch\_attack = 3$, $pois\_rate = 0.5$

#class targets	GC	clean accr	trojan accr	# steps	verf. rate
10	5	0.71	1.0	280	0.024
10	2	0.71	1.0	280	0.024
10	0.0005	0.57	0.99	2540	0.0027
50	5	0.71	0.93	181	0.038
50	2	0.71	0.93	181	0.038
50	0.0005	0.57	0.92	1380	0.005

We conducted the attack for a subset of source labels (10, and 50 out of 1000 classes) to a single target class. Table 4.4 shows that small clippings (2 and 5) forces the attacker to continue for a longer period of time, which results in a small verification rate while supporting a reasonable performance loss (around 0.02). It is evident that the total number of malicious steps to launch successful attacks (achieved within 30 epochs of training) demands a very low verification rate, given the detection failure $< 10^{-3}$. It is common for models that are built on the ImageNet to perform more than 1**M** SGD steps. To require anything beyond 0.1 verification rate, the total number of steps that the attacker intervenes should be below 70, which is unlikely given the strict gradient clippings.

Gradient Clipping Impact On Training W.O Attack

To investigate if the optimal accuracy that can be achieved for the current setting (partial retraining) , we selected four clipping values: unbounded, mild (5.0), low (5.0), and tiny (0.0005). The training with unbounded clipping value was unstable and the model barely achieved an accuracy of 0.01. We believe that this is an example of how gradient clipping can help large DNN models converge faster.[3] However, the training with the low clipping values converged to an acceptable optima of ≈0.72 accuracy. It should be noted that we only trained the model for far fewer iterations than what is usual for ImageNet (1M-2M steps). Additionally, for the tiny clipping value, the model converged to an optimum accuracy rate of ≈0.61. We wish to highlight that, for large datasets (millions of inputs), if the trainer can choose a suitable learning rate, gradient clipping values that are not significantly smaller than the learning rate, are unlikely to incur an unacceptable performance overhead, and empirically, we observe that gradient clipping can also help the model converge faster.

4.6 Conclusion

This chapter introduced the GINN system, which provides integrity in outsourced DNN training using TEEs. As our experimental investigation illustrates, GINN scales up to realistic workloads by randomizing both mini-batch verification and matrix multiplication to achieve integrity guarantees with a high probability. We have further shown that random verification, in combination with hyperparameter adjustment (e.g., setting low clipping rates), can achieve 2**X**-20**X** performance improvements in comparison to pure TEE-based solutions while catching potential integrity violations with a very high probability.

[3]Training from scratch with unbounded clipping can potentially take at least a week using a single GPU of 16GB.

CHAPTER 5

A FRAMEWORK FOR PRIVACY-PRESERVING GENOMIC DATA ANALYSIS USING TRUSTED EXECUTION ENVIRONMENTS [1]

5.1 Introduction

Ever since the completion of the Human Genome Project (Guttmacher and Collins, 2003), numerous research resources have been allocated to improve sequencing workflow efficiency. With the advent of moderately cheap Next Generation Sequencing (NGS) technology, (Xu, 2014) and its massive parallelism capability, companies and governments alike are heavily investing in platforms to support gathering, processing, and querying genomic data. Given its potential in various settings (e.g., personalized medicine, ancestry characterization, and kinship analysis), numerous endeavors have been oriented to use such information. One notable example is Genome Wide Association Studies (GWAS), which aim to discover the genomic variations that are most correlated with diseases and phenotypic traits. At the same time, the sharing and analysis of genomic data raises significant privacy concerns. For instance, while genomic data can shed light on predisposition to various diseases, which may be exploited for various purposes. In the United States, though the Genetic Information Nondiscrimination Act of 2008 prohibits the use of such data for health insurance eligibility and premium pricing, it does not address its use in the context of life, long-term care, or disability insurance. Moreover, concerns extend beyond the individual, as genomic data reveals information about biological relatives (Edge et al., 2017; Humbert et al., 2013). Most recently, this was highlighted by the Golden State serial killer case, who was identified three decades later, after it was found that a public genealogy database contained the genomic

[1] ©2020 IEEE Reprinted. with permission, from A. Asvadishirehjini, M. Kantarcioglu and B. Malin, "A Framework for Privacy-Preserving Genomic Data Analysis Using Trusted Execution Environments", 2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), 2020, pp. 138-147

markers of the suspect's relative (G. and H., 2018). This issue has sparked substantial controversy in the scientific community, and public discourse (Court, 2018). This is only one example of various ways by which privacy intrusions may be committed against genomic data, as documented in various review articles (see (Mittos et al., 2017; Naveed et al., 2015; Akgün et al., 2015)).

In this respect, there are concerns that as the diversity of attacks against genomic data grow, individuals (and the institutions managing genomic data on their behalf) can not be afforded privacy protections. As such, they will either need to accept that significant privacy risks abound or choose not to participate in research (or at least the sharing of research data beyond the borders of the institution that collected and analyzed it initially). However, technological advancements may provide an alternative solution. To mitigate privacy risks for individuals and data custodians, techniques borrowed from applied cryptography have been adapted to process genomic data. These methods are designed to provide confidentiality and integrity for an institution's private inputs at the expense of increased computational resources and, at times, loss of precision for outputs (e.g., GWAS). In practice, these techniques can be broadly categorized into two classes. The first class incorporates a cryptographic technique, Secure Multi-party Computation (SMC), that transforms the genomic data analysis function into a boolean or an arithmetic circuit (e.g., (Constable et al., 2015; Tkachenko et al., 2018)). SMC protects private genomic data, such as single nucleotide polymorphism (SNP) variants, for all collaborating institutions. However, these circuits are limited in that they 1) are unable to support the simultaneous processing of a large amount of genomic data, and 2) induce a substantial increase in bandwidth and computation time. The second class of multi-party solutions leverage trusted execution environments (TEEs). Using hardware capabilities, one can derive guarantees that the execution of source code and the data under investigation remain in an isolated tamper-resistant chip. The Intel Software Guard Extensions (SGX) has become a popular TEE choice for the genomic setting (Chen et al., 2016,

2017, 2016; Carpov and Tortech, 2018; Mandal et al., 2018). Unlike SMC solutions, a TEE can avoid running a boolean (or arithmetic) circuit. The genomic data is kept encrypted during storage, and it is only decrypted inside a special region of memory called enclave. Furthermore, this enclave's content is not accessible to operating system or the programs running on the machine. Unfortunately, TEEs such as SGX leak memory access patterns (Costan and Devadas, 2016; Intel, Intel). As a consequence, a naive implementation can leak information, such as variant counts of institutions, in the case of GWAS analysis. To enable SGX resistance for such leaks, algorithms need to be implemented in a *data oblivious* fashion. For instance, in matrix multiplication, the standard algorithm will always access the same memory locations irrespective of the matrix values. Conversely, a matrix multiplication that is optimized for sparse matrices may reveal the number of zeros due to memory access patterns. A number of data-oblivious frameworks have been developed for SGX in various domains, including distributed databases and machine learning (Zheng et al., 2017; Shaon et al., 2017; Ohrimenko et al., 2016).

This chapter introduces ObliVGene, a data-oblivious, privacy-preserving, and multi-party (non-trusting organizations or entities) genomic data analysis framework for geneticists. ObliVGene always keeps the sensitive data encrypted and authenticated when it resides outside the TEE boundary, and the data only is decrypted within the TEE when necessary. Moreover, depending on the type of data (e.g., VCF) and the computation, it may duplicate and preprocess the genomic data for optimization and data-obliviousness purposes. Throughout the chapter, two common applications of genomic data analysis using individuals' private data are investigated. For GWAS analytics, institutions initially preprocess their private data, which are encrypted and submitted to a central repository endowed with Intel SGX hardware (ObliVGene). In a second phase, an aggregation server will run an attested program to compute a pooled GWAS.

We evaluate the IDASH 2017 competition (Track 2) dataset, which consists of 5×10^6 unique SNPs. We illustrate that, while the state-of-the-art TEEs solution leaks memory

access patterns, our approach leaks no such information. This investigation shows that multi-institution genome association testing with privacy and integrity guarantees is not out of reach. Additionally, stronger protections against attacks based on memory access patterns are possible with reasonable overhead. Notably, the proposed techniques presented throughout this chapter apply to a range of tests for association studies based on variant counts, such as χ^2 tests. For the second application, we considered Deep Learning for genomic data analysis. The reason is that given the success and continuous improvements of Deep Neural Networks (DNN) in making reliable decision models in a variety of domains (e.g., vision, natural language, audio processing), there is also a growing trend within the genomic community to incorporate Deep Learning (DL) for an array of tasks (Yue and Wang, 2018). ObliVGene supports commonly used layers in DL, such as Convolution, Pooling, Softmax, etc., and augmented their standard implementations with data-oblivious versions to run within a TEE. Achieving data-obliviousness for DNNs is relatively cheap because most of the standard layers and activation functions are data-oblivious. However, there are cases that conditional branching may leak information about the private genomic information, which are discussed at Section 5.2.3. For evaluating the secure DL framework using TEEs, we tested a model inspired by IDASH 2019 workshop/competition for a binary classification task. The model consisted of more than 80 layers and 10.8 million parameters containing specialized one-dimensional convolutions, max-pool, and residual shortcuts (He et al., 2016).

The specific contributions of this investigation are:

- For GWAS tasks that rely on variant counts in a pooled dataset, ObliVGene transforms the input data from subject-oriented into SNP-oriented genotype data, which is suitable for executing such tests efficiently. Additionally, we introduce a bit-level compression strategy for genomic data. We apply this representation to realize substantial computing efficiency gains by leveraging the hardware data processing mechanisms. To

provide enhanced security, we evaluate a data-oblivious solution where no information, such as the number of SNPs, is leaked during the computation.

- We show that our framework can securely support other tasks such as DL classification for genomic data. Several new layers (one-dimensional extensions) have been implemented that are not in the baseline implementation (DarkNet (Redmon, 2016)) that are popular in genomic applications. We further augment the implementations of layers that leak memory access patterns via *assembly in-register* implementations. Finally, we show the trade-off between memory footprint vs. batch inference (number of instances that are fed to the network simultaneously), due to negative impact of driver-level paging.

5.2 Architecture & Methodology

This section begins with a formalization of the system. Next, we introduce the core building blocks of the genomic data protection model. We then combine the blocks to engineer ObliVGene.

5.2.1 Security Considerations and Threat Model

The institutions will choose an encryption scheme that guarantees the confidentiality and integrity of the data. In this work, we use AES-GCM. We assume that institutions will not collude with the server or each other. This is a common assumption in multiparty computation settings. In terms of trust assumptions for server, we use an honest-but-curious (HBC) model (Goldreich, 2009). In HBC, the server is expected to follow the designated protocol but may learn as much as possible from the protocol's trace. In this setting, we assume that the memory locations accessed by the program is observed. (Islam et al., 2012) showed that the memory access pattern involuntarily leaks sensitive information. SGX only

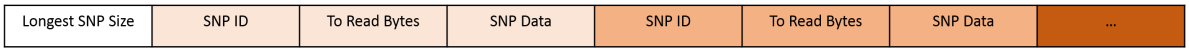


Figure 5.1: A depiction of how an institution stores its SNPs.

encrypts when memory pages are evicted outside of the enclave; however, it does not hide the memory access pattern. In other words, the addresses of requested memory pages are transparent to the OS (Intel., 2015).

5.2.2 GWAS Analysis

Problem Definition

This research focuses on a multi-institution GWAS scenario as it was defined in the 2017 iDASH Genome Privacy Competition (Ida, 2017). In this setting, the institutions are tasked with computing the *k most significant SNPs in a case/control study*, which we refer to as top-K. There are P participating institutions, each of which provides a dataset SNP^i where $i \in \{1, \cdots, P\}$. Each record in the dataset is structured as $SNP^i_j \subseteq U$, where $j \in \{1, \cdots, |SNP^i|\}$ represents a specific (human) subject and U is the universe of all known SNPs covered for a χ^2 test.

Architecture

We assume there is a machine with Intel SGX hardware support that is accessible to the institutions, which we refer to as the *server*. A pertinent candidate server can be obtained from a cloud computing vendor such as *Azure Confidential Computing* (Microsoft, 2021) by Microsoft. As illustrated by Fig. 6.1, the institutions will submit their private encrypted data to the server to conduct the Top-K computation.

Compression and Storage Format

The size of the enclave memory is limited by the SGX hardware to approximately 96MB. As such, it is not feasible to incorporate off-the-shelf software to perform GWAS analysis

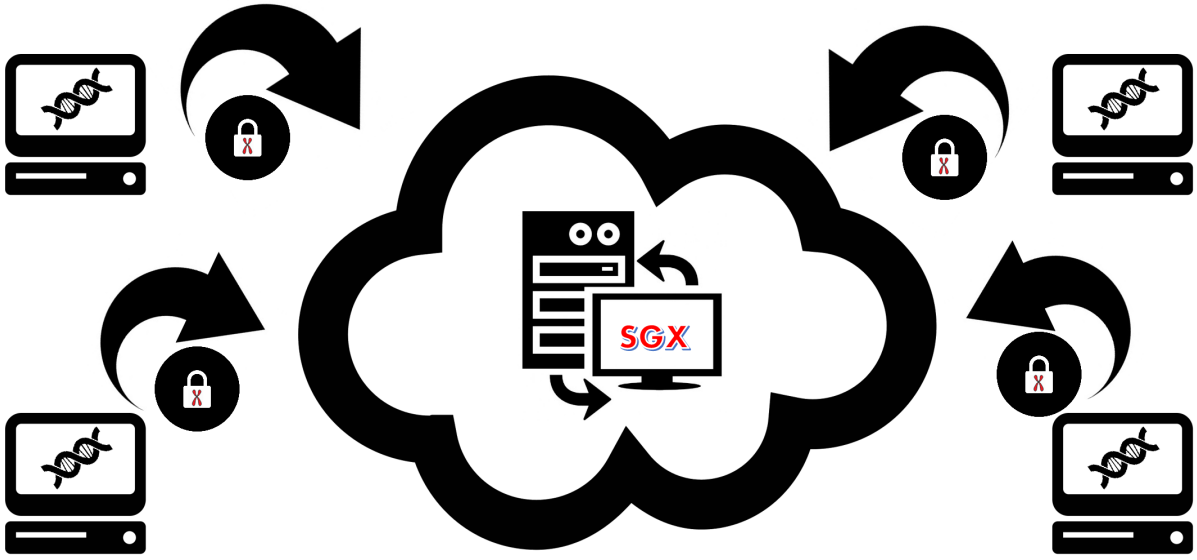


Figure 5.2: Architecture. Institutions submit genomic records, in an encrypted and compressed format, to the centralized SGX server. The server then computes the K most significant SNPs in a secure manner or the deep learning model result on the input.

inside the enclave. For example, the dataset from Task 2 of 2017 iDASH competition (Ida, 2017) contained 27 GB of Variant Call Format (VCF) (SAMTools, 2021) files. Moreover, VCF files are subject-oriented, such that for each subject there is an associated file that contains their genomic data. By contrast, GWAS is a SNP-oriented (based on variants) analysis. It is more efficient to realign such a dataset to have all subjects with the same SNP in one file. As a matter of fact, this method is a well-known transformation that enables supporting aggregate queries, and in the database community, often referred to as *vertical storage* (Harrison, 2015; Ramakrishnan and Gehrke, 2000). To conduct a GWAS, we need to learn the counts for the alleles of each SNP in the case and control groups. The most important information in this regard is the number of heterozygous and homozygous cases in the dataset. We store this information in two bits as 10 and 11 for the heterozygous and homozygous case, respectively.

In light of these considerations, we introduce a new compression format for institutions conducting GWAS analytics, as illustrated in Fig. 5.1. First, a fixed-size field (1 byte in our

implementation) stores the length of the longest SNP ID in the institution's dataset. All of the SNP IDs in the dataset will be padded to this length. Next, for each SNP that is present in the file, the following information is appended:

a) *SNP ID*: A unique identifier. It corresponds to a string that is the concatenation of the CHROM, POS, ID, REF, and ALT of a VCF entry. SNPs are written in ascending sorted order based on this field.

b) *To Read Bytes*: The length of the SNP Data field.

c) *SNP Data*: A bit vector representing information on subjects possessing the SNP.

When an SNP is present in both study groups, We give precedence to the case group, such that it will be written before the control group.

Lossless Compression

Reconstruction of the VCF files from the compressed file is achieved by producing an auxiliary file that has an almost identical format to that shown Fig. 5.1. The only difference is that it contains subject IDs in the same order as the bit vector elements in the compressed file instead of the SNP bit vector. Afterwards, for each SNP in both files there is a one-to-one correspondence between every two-bit group and patient ID.[2]

Concurrent Preprocessing

We developed a concurrent approach that relies on parallel execution (processing multiple VCFs at the same time) to transform VCF files into a file that adopts the format in Fig. 5.1. Specifically, we utilize a producer-consumer paradigm to speed up the data compression. A

[2]We did not pursue implementing this recovery scheme. Nevertheless, it is a simple procedure that institutions can use.

single thread is allocated for a producer (one entity can read from the disk at any time), while multiple consumer threads are used to read and parse the data from buffers that are filled by the producer. After all of the threads complete, we aggregate the outputs and write them into one final output file.

Algorithm 1 provides a procedural walkthrough. There are three inputs to the algorithm. $read\_buffer$ queue is a shared medium that is continuously filled, up to a prespecified limit, by the producer. The consumer reads from the buffer and extracts the relevant SNP information. out_map is updated after the consumer local operation completes using the global mutual exclusion input $mutex$. It should be noted that $local\_map$ in line 1 has the same structure as out_map in that it stores the SNP identifier as a key while maintaining a bit vector for it as a value. In lines 7 and 9, two bits, descriptive of the type of SNP, are appended to the bit vector associated with the currently processed SNP. Eventually, when there is no more data for the consumer, it appends its local map items to the corresponding items of the main program (lines 12 to 20).

GWAS Oblivious Computation

Algorithm 2 achieves data obliviousness in that it always accesses the same memory locations as long as different inputs are of the same size. Specifically, the institutions run the same compression algorithm, with two differences. First, we assume there is an agreement on some predefined set of SNPs that covers each institution's set of SNPs. Consequently, each institution will cover all of the genomic regions under investigation. Second, each SNP's data field (Figure 5.1) is padded with zero-valued bits to ensure that the data associated with each SNP is of the same size. As a result, the data provided by each institution has the same size and the same field-wise alignment. It should be noted that, while this approach is oblivious, it incurs an increase in the storage space required by the remote service, an issue we evaluate empirically below.

Algorithm 1: Consumer Compression Task

Input: read_buffer, out_map, mutex
Result: The final output map based on consumer's local data

```
1  local_map.initialize()
2  while read_buffer.hasdata() do
3  |   data ← read_buffer.get_next_snp()
4  |   snp_id ← data.get_snp_id()
5  |   type ← data.get_type()
6  |   if type = heterozygous then
7  |   |   local_map[snp_id].push_back(b10)
8  |   else
9  |   |   local_map[snp_id].push_back(b11)
10 |   end
11 end
12 mutex.get_lock()
13 for temp_snp_id ∈ local_map do
14 |   if out_map.find(temp_snp_id) then
15 |   |   out_map.push_back(local_map[temp_snp_id])
16 |   else
17 |   |   out_map[temp_snp_id] ← (local_map[temp_snp_id])
18 |   end
19 end
20 mutex.release_lock()
```

Algorithm 2 describes how the cloud service processes the chunks provided by the institutions. In lines 2 to 5, the server ingresses a chunk from each provider into the enclave. From line 7 to 14, a SNP from each provider is read and its corresponding counts are processed using the *popcnt* instruction. Next, the computed counts are sent to an oblivious test statistic computation-insertion algorithm in line 15. Note that we no longer use max-heap because its processing is data-dependent. Finally, the Top-K significant SNPs are encrypted and sent back to the untrusted memory (line 18).

We use Algorithm 3 to process the list insertion obliviously. When a new SNP is ready to be tested against the list, it is inserted as the last element of the current list (line 1). Next, it is compared with the preceding element. If it is smaller, then the two SNPs are swapped. This comparison and swapping process continues until a complete pass is made over the list

Algorithm 2: Oblivious Top-K Significant SNPs

```
   Input: institutes, top_k_list, num_chunks
   Result: Top-K Significant SNPs
 1 for i ∈ (1, 2, ··· , num_chunks) do
 2 |   chunks ← new chunk[institutes.size()]
 3 |   for j ∈ (1, 2, ··· , institutes.size()) do
 4 |   |   chunks[j] ← institttutes[j].get_chunk(i)
 5 |   end
 6 |   while chunks.has_snp() do
 7 |   |   snps ← new snp[institutes.size()]
 8 |   |   case_counts ← 0
 9 |   |   control_counts ← 0
10 |   |   for s ∈ (1, 2, ··· , institutes.size()) do
11 |   |   |   snps[s] ← chunk[s].next_snp()
12 |   |   |   case_count ← case_count+ popcnt(snps[s].case_bitset)
13 |   |   |   control_count ← control_count+ popcnt(snps[s].control_bitset)
14 |   |   end
15 |   |   obliv_calc_append(case_count, control_count, top_k_list)
16 |   end
17 end
18 ocall_sendout_top_k()
```

(in reverse order - see lines 5 to 16). Finally, if the total number of elements in the list is greater than K, we remove the final element from the list in line 18. Each time an insertion to the list is invoked, we begin with a sorted (or, in the beginning, empty) list, and we finish with a sorted list that may or may not have the new SNP in it.

5.2.3 Deep Learning Inference Analytics

Problem Definition

A private model $\mathcal{F}$ with parameters $\mathcal{W}$ is provided to the TEE by the model owner. Users of this model (institutions or individuals) want to feed their private data $\mathcal{X}$ to this model and obtain the predicted outcome $\mathcal{Y}$. However, none of these entities trust each other. Model owner wants the model to be private, and data providers also want their data kept private,

Algorithm 3: Oblivious Append to Top-K List

Input: chi_snp, top_k_list,k
Result: Oblivious χ^2 append
/* new item is added to the end of the list */
1 top_k_list.append(chi_snp)
2 $list\_size \leftarrow top\_k\_list.size()$
3 $type\_size \leftarrow$ sizeof(chi_snp)
4 $swap \leftarrow$ char[$type\_size$]
/* iterate over list in reverse order */
5 **for** $i \in (list\_size, \cdots, 2)$ **do**
6 $temp\_snp \leftarrow top\_k\_list[i]$
7 **if** $top\_k\_list[i] < top\_k\_list[i-1]$ **then**
/* indices must be swapped */
8 memset($swap$,1,$type\_size$)
9 **else**
10 memset($swap$,0,$type\_size$)
11 **end**
12 **for** $j \in (1, \cdots, type\_size)$ **do**
13 $top\_k\_list[i][j] \leftarrow$
$(1 - swap[j]) \times temp\_snp[j]) + (swap[j]) \times top\_k\_list[i-1][j])$
14 $top\_k\_list[i-1][j] \leftarrow$
$(swap[j]) \times temp\_snp[j]) + (1 - swap[j]) \times top\_k\_list[i-1][j])$
15 **end**
16 **end**
17 **if** $list\_size > k$ **then**
18 delete $top\_k\_listp[list\_size]$
19 **end**

and the computation itself should not leak information regarding the $\mathcal{W}$, $\mathcal{X}$, and $\mathcal{Y}$.

$$\mathcal{Y} = \mathcal{F}_{\mathcal{W}}(\mathcal{X})$$

Architecture

First, the model owner will send its encrypted and authenticated parameters to the ObliVGene. After the model is loaded and initialized, data providers submit their genomic inputs to ObliVGene server for inference task. In the end, the encrypted (with institutions' keys) results are sent back to the data providers.

```
float relu_activate(float x){
  if (x > 0) return x;
  else return 0;
}
```

Listing 2: Conditional branching in ReLU activation depends on the sensitive value

```
float relu_activate(float x){
  const float ZERO = 0.;
  __asm__ volatile(
  "cmp %1,%0\n\t"
  "cmovl %1,%0\n\t"
  :"+r"(x)
  :"r"(ZERO),"0"(x)
  :"cc"
  );
  return x;
}
```

Listing 3: Values are moved and compared inside CPU registers

Data Oblivious Inference

The bulk of the computation for a deep learning model arises from matrix multiplication (MM) operation, which is inherently data-oblivious. For the most mainstream layers and activation functions, the computation is not data-dependent; however, there are a few instances that the memory locations accessed by the program can differ depending on the values of the inputs. MaxPool is an instance of layers with such privacy pitfalls that may lead to full or partial recovery of the inputs/parameters. Additionally, ReLU is an example of activation functions that are very common and leaks memory access patterns. Listing 2 shows a sample code for ReLU activation, which is not safe and likely will leak if the input is greater than zero (conditional branch accesses different addresses). In contrast, Listing 3 performs all the conditional checking and moving within the registers, which is not accessible to the adversary. In our augmented implementation, for all the data-dependent operations, we make similar modifications that are mainly based on CMOVcc instructions (Intel, 64)

that operate on registers and values of the registers are flushed in memory in the end regardless of the values of sensitive inputs. Such operations prevent the leakage of sensitive information due to branching information.

Breaking Apart Big Layers

In case a convolutional layer's auxiliary space (reshaped input feature-maps) or fully-connected layer's parameters are well beyond the hardware imposed memory limit, ObliVGene breaks the computation into smaller chunks and performs matrix multiplication in multiple rounds. For a convolutional layer, ObliVGene breaks the input feature-maps' on the channel dimension. Likewise, for a fully-connected layer of shape $outputs \times inputs$, it will break the parameter's matrix on the row dimension. After these steps, each chunk will be able to fit into the memory.

5.3 Experiments

To assess the ObliVGene's performance, we conducted a series of tests on a machine running the Linux 64-bit operating system with an Intel(R) Xeon(R) CPU E3-1275 v6 @ 3.80GHz (with SGX hardware) and 64GB of memory. All files were stored on a solid state drive.

First experiment shows the result for GWAS analysis (χ^2) over 5.6 million SNPs for 2,000 individuals that are randomly divided into five subsets (representing five non-trusting institutions). Therefore, each institution has 200 case and 200 control individuals in their pool. We used the dataset released for the 2017 iDASH competition, Task 2 (Ida, 2017). The client (we assume clients will send column-layout) performs the compression and server SGX program performs the secure aggregation. Compression and aggregation modules are developed in C++ and the Intel SGX SDK was relied upon for encryption and decryption inside the enclave.

Second experiment shows ObliVGene's performance for DL tasks. We chose Darknet (Redmon, 2016) as the baseline because of its simplicity. We patched several lines of custom code to support new layers (e.g. 1D layers for genomic data), plus *data-oblivious implementations* for private data processing. For matrix multiplication operations we used oneDNN (Evarist Fomenko, 2020) multiplication kernel implementation (by default is data-oblivious).

5.3.1 GWAS Experiment

To obliviously perform the test each institution must cover the same genomic regions. As such, each institute's file is perfectly aligned with the rest of the providers. Our approach ingresses the chunks into the enclave and process SNPs one-by-one from each institution. For this test, we set the chunk size to 10MB. Due to an extended region coverage of this scheme (none of the case or control individuals may have a particular SNP, but the region should be padded), the induced final file size for each institution is about 905MB. To examine the performance of our oblivious aggregation on the SGX server, we conducted five GWAS runs. On average, this scheme completes the test in less than 40.6 seconds with standard error of 0.156. Additionally, note that we evaluate different values (1M, 4MB, and 10MB) for chunk size as expressed in Algorithm 2, however, there was no significant difference in terms of time performance.

In summary, the results show that bit-wise compression and columnar storage reduces the size from 27GB to less than 5GB (each institution has 905MB). In addition, server aggregation is fast, and memory usage is reasonably low.

5.3.2 Deep Learning Experiment

Inspired by IDASH 2019 competition/workshop (Ida, 2019) Task III, we ported a similar DNN architecture (more than 10.8 million parameters) that performs a binary classification.

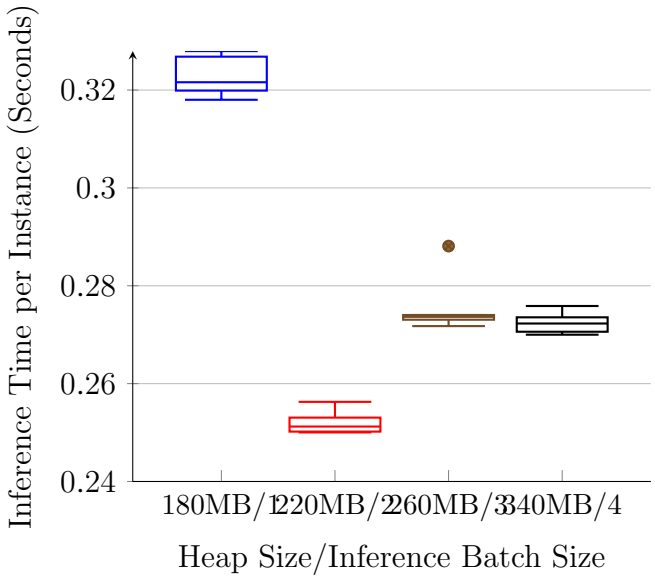


Figure 5.3: For this particular architecture, processing 2 items at once yields the best performance (about 4 inferences per second).

The dataset contained 102 test examples, and we investigated the trade-off between minimum heap size vs. the maximum number of test items that can be processed simultaneously (batch inference). As shown in Figure 5.3, consuming the items one by one does not yield the best performance, albeit it consumes less memory footprint. As can be seen, processing two items at once gives the best performance, roughly four predictions per second. In contrast, if the heap footprint is increased to accommodate the inference of three or four items simultaneously, performance is dropped but performs better than the first case. Note that **oneDNN** (Evarist Fomenko, 2020) port for SGX requires many threads allocated for successful compilation and run. These threads significantly increase the memory footprint, making it easy to pass the hardware limit (93MB). Overall, since the bulk of the work is matrix multiplication operations, we benefit from dispatching to oneDNN's implementation (it is a backend for popular DL frameworks) instead of relying on simple cubic-time implementation that is used by DarkNet (Redmon, 2016).

5.4 Discussions

In this section, we begin the discussion on the data-oblivious computation, particularly for TEE-based applications. Then, current known SGX vulnerabilities are enumerated. Next, we talk about other genomic analytics or optimizations that can utilize the SGX enclave as future directions for ObliVGene.

5.4.1 Data-Oblivious Computation

In our setting, data obliviousness can also be achieved by using oblivious RAM, or ORAM (Goldreich and Ostrovsky, 1996). ORAM accesses memory locations in ways that the sequence of accesses appears as though it is random. As a result, it hides the real memory access pattern. Unfortunately, current state-of-the-art ORAMs (Stefanov et al., 2013; Devadas et al., 2016; Wang et al., 2015) incur a heavy performance bottleneck when applied to large scale computations. First and foremost, these schemes require at least twice storage space on server side and logarithmic space on client side. Yet another drawback is their significant network cost. However, network cost can be mitigated in case of having access to a TEE on server, so that the client's work can be shifted to the enclave. Finally, another performance weakness of these schemes is that they require reshuffling, which basically rearranges all of the blocks to maintain randomness with respect to the access pattern. During this phase all other read/write accesses are halted, which has a significant negative impact on the concurrency.

For the case of a GWAS task, another approach that allows oblivious processing of SNPs is based on generic union and sorting operations. First, each data provider's chunks are brought into the enclave, so that the necessary information is extracted. The enclave will emit intermediate outputs of the form $\{snp\_id, case\_count, control\_count\}$ after each chunk is processed (before requesting the next chunk). Next, a sorting operation must be performed based on the snp_id. After this sorting operation, all related SNPs are next to each other

and the test statistic can be computed by one pass over the sorted list. In the literature, the bitonic sort (Batcher, 1968) is usually incorporated for oblivious sorting. It always exhibits the same access pattern independent of the input array with $n \times log^2(n)$ comparisons. This approach will not scale well providing that the region covered for the test has SNPs in the order of millions because all the intermediate outputs will not fit inside SGX's enclave. Instead, we tried to devise specific oblivious computation approach that processes the data and goes over the data intelligently without revealing any access pattern.

5.4.2 Protection Against Stronger Adversaries with Timing and Cache Side Channel Leaks

Our trust assumptions are a commonplace practice in the security community (Goldreich, 2009). We assumed all data providers are semi-honest, and they will not provide malicious or incorrect data to the computation server. In addition, the central SGX server is semi-honest; it always follows the protocol but tries to infer information regarding sensitive data during protocol execution. Ideally, we do not want the server to learn anything other than what can be learned from the final output size. The adversary should only see the encrypted result. Likewise, no party should learn anything about the other parties except what can be inferred from the final output.

Even with memory-access obliviousness, that is the central theme of this chapter; the adversary can exploit other side-channels. The recent Spectre (Kocher et al., 2018) and Meltdown attacks (Lipp et al., 2018) demonstrated that secrets can be read from memory by malicious processes. Briefly, these attacks take advantage of speculative execution and out of order execution (which most modern CPUs almost unanimously support) to facilitate leaking sensitive data through CPU cache. Intel SGX is susceptible to these types of attacks but can be remedied using specific techniques as outlined by Chen et. al. (Chen et al., 2019). Note that there is a glossary of attacks on shared cache, page table and other vulnerabilities

with SGX (Brasser et al., 2017; Schwarz et al., 2017; Van Bulck et al., 2017; Bulck et al., 2018; Wang et al., 2017; Gras et al., 2018). We leave the development of defenses against such side channel and time leaking attacks to future work.

5.4.3 Distributed Genomic Analytics and Other Genomic Tasks

Most genetic tests, such as Cochran-Armitage (statgen.org, 2012), Chi-Square and Fisher's Exact (Kim, 2017) tests, are dependent on variant counts. These tests require computing the frequencies at which allele or genotypes appear. The framework we described in this chapter can be applied in these test settings. As our results indicate, storing data in a vertical fashion enables fast processing. Moreover, we implemented ObliVGene using a single SGX enabled server. A secure genomic analysis task could be distributed to multiple servers (with TEEs) to increase the performance. We believe such an extension combined with tools such as (Shaon et al., 2017; Zheng et al., 2017) would be needed to scale to other genomic data analysis workloads.

5.4.4 Support for Update, Insert and Delete

Currently, our solution does not support the update, insert and delete operations. If an institution wanted to perform any of the aforementioned operations, they must perform compression again and re-submit the new file so that our solution can be applied. We strongly believe any genomic data storage-processing service should offer these capabilities whilst guaranteeing a high level of privacy. We leave this extension as a future work.

5.5 Conclusion

This chapter introduced ObliVGene; a framework for privacy-preserving data-oblivious genomic data analysis using Intel SGX. We show our framework's applicability using two example applications: χ^2 association test and DNN inference. To perform an association

test, genomic data is collected from multiple institutions, and counting is performed within the TEE. We developed new techniques to achieve high-performance solutions for Intel SGX technology. First, we showed how to transform genomic data into a compressed form via a bit vector representation. Second, we illustrated that SNP-oriented storage allowed the placement of inputs for efficient computation. Third, we showed how to apply hardware instructions to perform the counts needed for computing the required statistics. Finally, we showed that the server-side the solution can run in a data-oblivious manner without leaving a memory access pattern trace. For the DL task, we showed that ObliVGene allows private DNN model owner and users of the model to engage in a private computation without revealing their secrets to other parties. Moreover, our implementation performs inference in a data-oblivious fashion, mostly relying on CMOVcc (Intel, 64) in-register instructions that are not accessible to the adversary.

CHAPTER 6

AISOLACE: SECURE FAIRNESS ANALYTICS FOR MACHINE LEARNING USING TEES

6.1 Introduction

ML models are increasingly used for many critical tasks, ranging from education to healthcare. At the same time, recent scrutiny of ML models have found disturbing cases of bias built into the ML models. One important example of such bias has been observed in an ML model that predict recidivism. Analysis using the publicly available data and the ML model predictions has shown that "black defendants were far more likely than white defendants to be incorrectly judged to be at a higher risk of recidivism, while white defendants were more likely than black defendants to be incorrectly flagged as low risk." (ProPublica, 2016).

Ideally, to prevent bias and enable public auditing of critical ML models, ML models could be disclosed, and all interested parties could test the model to find out potential biases embedded in the model. Unfortunately, this is not possible in many cases. The organizations who have invested in the ML model building may be reluctant to publicly disclose their models. In addition, the test data required for fairness assessment may be private as well. For example, individuals may not be willing to disclose sensitive information related to their credit card application. Therefore, in many cases, security and privacy issues may prevent the public auditing of the ML models.

Another complicating factor is how to define the appropriate fairness criteria required for auditing the ML models. Researchers over the years come up with many, often conflicting fairness criteria. For example, IBM AI Fair360 toolbox (Bellamy et al., 2018) has tens of different fairness definitions.

In addition to choosing fairness criteria, in many cases ML model performance needs to be measure across "different" groups, and group definition may change based on the

application. For example, in Compass, the ML model performance was measured based on race (e.g., black vs white defendant). In some cases, single or multiple demographic features may be selected (e.g., gender, gender and sexual orientation).

The above observations show that we need a secure system where the ML models could be audited for fairness without disclosing sensitive information while supporting large number of fairness and potentially changing group information.

In this work, we address this research challenge by developing our system named AISolace. In our solution, ML model owner sends their encrypted ML model to AISolace running on a Trusted Execution Environment (TEE) in the cloud. Similarly, ML model auditor sends encrypted test data and their desired fairness definition to AISolace. AISolace securely runs the given model inside the TEE based on the given fairness criteria and grouping definition and only disclose the fairness analytics/audit results. The grouping operations include two forms. First, *regular* groupings, based upon a set of columns, where each variant of columns together constitutes a group. Second, a *clustering* based grouping where user provides the number of clusters, together with the columns to be used for cluster distance calculation. To encompass real-world settings, the clustering methodology allows for both numerical and categorical columns (Huang, 1997). Furthermore, to prevent potential side channel attacks, all the execution is data oblivious. Oblivious execution of algorithms eliminates the dependency of code paths and memory accesses to the inputs of algorithms. Thus, in addition to the data being only decrypted when inside the TEE (outside the TEE it is always encrypted), the traces of memory access does not leak any information about the content of the inputs, beyond the size of the input, the size of the outputs and public parameters of the algorithm (e.g. K number of clusters in K-Means clustering (Hartigan and Wong, 1979)).

Our contributions can be summarized as follows:

- We are the first to propose a privacy-preserving deep learning fairness analytics/auditing framework using TEEs.

- Our system enables clustering approach for groupings, wherein researchers can provide arbitrary distance metrics for numerical columns in the test set.
- Our system hides memory access pattern so all the operations are data-oblivious.

6.2 Security Model

In this section, we present the technical details on the assumptions and security guarantees of AISolace .

6.2.1 Assumptions

AISolace has two categories of users; 1) model providers (MP), and 2) fairness analysts (FA). Both MP and FA establish a secure channel (e.g. TLS) after *remote attestation* of the server which runs the AISolace. Through this channel they will send the model, test data and fairness algorithms of their choice encrypted. We assume the enclave region of memory is secure and not directly accessible for any privileged software, such as the OS admin, to tamper with. In addition, the timing or cache side channel attacks are out of the scope of this chaper. Moreover, we assume the contents of registers within the CPU package is not accessible to the adversary. Finally, we assume the adversary is honest-but-curious; meaning they will not deviate from the algorithms, however they will try to follow the memory traces of the algorithms to gain insights about the underlying data provided by the users (test set, model parameters, predictions, etc.).

6.2.2 Security Guarantees

The MP has the guarantee that no information about the model parameters is leaked to the FA or any adversary that maybe residing on the server. For the FA, no information about the test dataset, is leaked to the FA or any adversary monitoring the computation

of the fairness metrics. AISolace will also protect against memory access pattern leakage via data oblivious primitives. Most of the literature on data oblivious designs with TEEs, assume a memory oblivious buffer availability, which is far from reality. In short, assuming the security guarantees provided by the TEE, any adversary will not be able to jeopardize the privacy requirements of the users of AISolace.

6.3 System Design

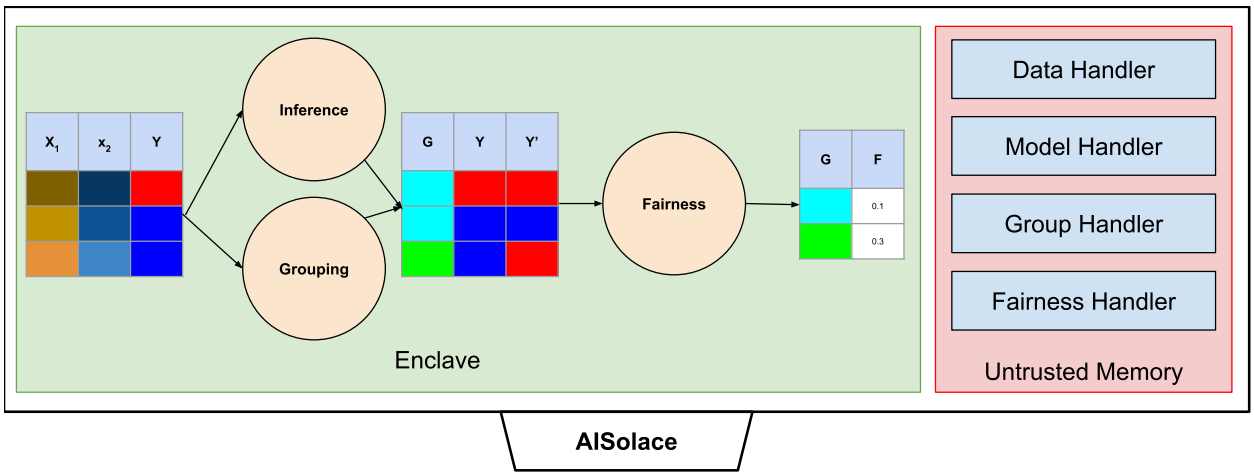


Figure 6.1: Main architecture of AISolace. Inference →Grouping →Fairness

In this section, we go over different pieces of AISolace. First, the general architecture of the system is introduced (6.3.1). Next, the initialization and communication steps are discussed. Finally, we iterate through the inference, groupings, and fairness analytics components of the system.

6.3.1 System Architecture

Here, we explain the different aspects of the system as depicted by Figure 6.1. AISolace receives the test set, and the model, from users through a secure channel after remote attestation. For the test set provided by FA, the system expects for the label to be in the

dataset. To conduct fairness analysis, the system *does not* rely on user provided predictions. Once the predictions on the test set are gathered (by running secure inference), the system uses secure grouping primitives to assign the rows to groups in a semantically valid fashion predicated on the conditions provided by the FA. After the grouping stage is over, the system uses the grouping(s) to conduct fairness test(s) on subgroups and further compare or contrast across groups, so the FA can reason about the relative difference between different groups and how they potentially behave differently for the protected and privileged segments. All the operations mentioned above are done securely inside the TEE, and if needs be, the buffers taken out of the TEE (handler components) will be encrypted using AES-GCM (McGrew and Viega, 2004), with time-stamps to guarantee freshness.

6.3.2 Initialization & Users Communication

MP registers the ML model $\mathcal{M}$ via sending the model through a secure channel (encryption in place). Once the model is received and verified in AISolace's enclave, it is sealed and stored outside the enclave until it is requested to have a fairness analysis on it by FA. Then FA, through a secure channel, sends the test dataset $(\mathcal{X}, \mathcal{Y})$, grouping criteria $\mathcal{G}^{\mathcal{C}}$ and fairness metrics $\mathcal{F}$. After the analysis is completed, the result is sent back to the FA. Note that in cases that we cannot have enough buffer available inside the enclave, the buffer is securely kept sealed (authenticated encryption), at reasonable granularity. For instance, at row level for the test data.

6.3.3 Secure Inference

The first stage in the fairness operation, is having the model predictions $\mathcal{Y}' = \mathcal{M}(\mathcal{X})$ gathered for each record in the test dataset by running an inference task on them. Once these predictions are computed, AISolace no longer needs the model to have resources allocated to it in enclave.

Inference Data Obliviousness

Most computational aspects (prominent layers such convolution) of inference in modern DL is data oblivious out of the box because they rely on matrix multiplication, which maintains the same memory access footprint given the dimensions of the inputs regardless of the contents within the matrices. Fully-connected, and convolutional (Goodfellow et al., 2016c) layers are prominent examples of such cases. Nevertheless, there are certain classes of layers such as **ReLU**, which has a data dependent memory access footprint. For these classes, AISolace relies on *in-register* **CMOV** assembly instructions to hide the memory access patterns for conditional assignments (Listing 1).

6.3.4 Secure Grouping

Algorithm 1 Oblivious Grouping

```
1: function GroupBy(X,g,s)  ▷ Sorts the data based on grouping and segment columns and assigns a unique id for each element in a group
2:     id ← 0                                                    ▷ The running id
3:     ID ← zeros(X.nrows())                                     ▷ ID column that will be appended
4:     X ← X.bitonic_sort(g, s)▷ Sorts the array obliviously for row_id ← 2 · ·X.nrows() do
5:
        end
        cond ← X[row_id, (g, s)] == X[row_id − 1, (g, s)]
6:     id ← cond_select(cond, id + 1, 0)
7:     ID[row_id] ← id
8:
9:     return X.col_concat(ID)
10: end function
```

FA is capable of devising intricate (sub)groupings so that the fairness metrics are computed over each group indecently and can be compared against each other. The goal is to decide the group each entry in the dataset belongs to, given the grouping scheme (criteria). And the end result will transform the $(\mathcal{X}, \mathcal{Y}, \mathcal{Y}')$ to $(\mathcal{Y}, \mathcal{Y}', [\mathcal{G}^C])$. Note that after the correct group is associated with the row, we no longer need the initial test data provided by

Algorithm 2 Oblivious Clustering

1: **function** CLUSTER(X, cat, num, K, N) ▷ Clustering based on categorical and numerical columns
2: $X \leftarrow X$.concat_col(rand$(0, K,$ "C")) ▷ Assigns to a random cluster
3: $C \leftarrow$ CLUSTERINIT(X, K) **while** $N > 0$ **do**
end
$row\_id \leftarrow 1 \cdot \cdot X$.nrows() ▷ Assign the row to the minimum distance cluster
4: $(min\_dist, min\_cluster) = (\infty, \perp)$ **for** c ***in*** C **do**
5:
end
$num\_dist \leftarrow$ NUMERICALDISTANCE$(X[row_id, num], c[num])$
6: $cat\_dist \leftarrow$ CATEGORICALDISTANCE$(X[row_id, cat], c[cat])$
7: $dist \leftarrow num\_dis + cat\_dis$
8: cond_assign$(dist < min_dist, min_dist, dist)$ ▷ Updates the minimum distance if condition is met
9: cond_assign$(dist < min_dist, min_cluster, c$.id()$)$
10:
11: $X[row\_id,$ "C"$] = min\_cluster$ ▷ Assigning to the minumum distance cluster to Column "C"
12:
13: $X \leftarrow$ GROUPBY(X, "C", $\perp$) ▷ Rows belonging to the same cluster are in sequential order
14: UPDATENUMERICALCOLUMNSMEAN(C, num) ▷ For numerical columns compute the arithmetic mean via SUM and COUNT
15: UPDATECATEGORICALCOLUMNSMODE(C, cat) ▷ We assume each column can have maximum L categories to avoid leaking cluster size
16: $N \leftarrow N - 1$
17:
18: **end function**
19: **function** CATEGORICALDISTANCE(cat_cols, $centroid\_cat$) ▷ Distance for categorical columns from a cluster centroid
20: $dist \leftarrow 0$ **for** $col \leftarrow 1 \cdot \cdot centroid\_cat.ncols()$ **do**
21:
end
$dist \leftarrow$ cond_select$(centroid_cat[col] == cat_cols[col], dist + 0, dist + 1)$
22:
23: **return** $dist$
24: **end function**
25: **function** NUMERICALDISTANCE(num_cols, $centroid\_num$) ▷ Oblivious, because it should be a mathematical funtion
26: $\cdots$
27: **return** $dist$
28: **end function**

```
GROUP BY ethnicity, age
    SEGMENT OVER gender
        PROTECTED  VALUES ('Female','NA')
        PRIVILEGED VALUES ('Male')
    HAVING ethnicity IN ('WHITE','BLACK')
```

Listing 4: Example of a grouping based on ethnicity, and age segmented over gender for fairness analytics.

the FA, and it is safe to release the resources allocated for the sake of performance. Moreover, because it is very likely for a FA to provide multiple grouping schemes, we use a list notation and AISolace associates a column per group scheme that encapsulates the grouping assignment based on grouping criteria.

Predicate-based Grouping

This form of grouping enables FA to perform grouping semantics aligned with what is customary in the literature as of now. For instance, **IBM's AIF360** (Bellamy et al., 2018) or **Microsoft Fairlearn** (Bird et al., 2020). In this scheme of grouping, FA must provide a list of grouping columns. Additionally, FA must provide the column that signifies the segmentation column over which the *protected*, and *privileged* assignments is based on. FA is allowed to filter out groupings based on the predicates. However, these predicates should only be based on the grouping columns. In example Listing 4, the FA is interested in to conduct fairness analytics for the people only with black, and white ethnicity, covering all the age groups, and segmenting the resultant rows over the gender as protected and privileged.

Grouping Data Obliviousness

AISolace uses a sorting approach over hashing to group the rows. However, regular sorting algorithms such as merge sort is not inherently data oblivious. Thus, *bitonic* sort (Batcher, 1968) is used as the main sorting algorithm. It has the same sorting order of comparison

and the memory access pattern can only be affected by the size of the data, not the content of the data. Nonetheless, in-enclave comparison and swapping needs also be data oblivious. To mitigate the swapping access pattern leakage, AISolace relies on CMOV instruction.

Initially, AISolace runs a filtering selection to only consider the rows passing the HAVING clause. Note that, it is a little different from ANSI-SQL in that it more acts as a WHERE clause and the filtering happens prior to the grouping. If FA provides filtering criteria, these criteria are matched against each row and rows that should be included will have a new column with the value '1', while non-matching rows will have a value of '0'. The rows that pass the filtering criteria are obliviously selected. In particular, A selection column is associated to the dataset and for each entry, a value of 0 (indicating not matching), or a value of 1 (indicating pass) is assigned (oblivious conditional assignment). Then the dataset is sorted obliviously in descending order of the new column. Afterwards, the dataset scanned and the rows with value 1 are selected. Finally, the selection column is dropped. Note that the size of the matching and non-matching rows is leaked. For instance, in case of the example in Listing 4 the size of the rows having the ethnicity as 'WHITE' or 'BLACK' is leaked, but the adversary cannot match the rows to the initial dataset submitted by the user.

Once the predicates in the HAVING clause are processed (in case there are any) and the passing rows are selected, a traversal is run to obliviously assign a new column (here we call it '*segment*'). This column is appended to the dataset, which signifies whether the row belongs to the protected or the privileged class. AISolace assigns 0 for the protected class, and 1 for the privileged class. Next, the dataset is obliviously sorted based on the grouping columns (in the order indicated by the user), and the segment column calculated in the previous step. Here, the segment column will be the last column that ordering is based on. After sorting is finished, the dataset is in the proper ordering to calculate the necessary aggregates. In example Listing 4 in the end, we will have the rows sorted based

```
USING 3 CLUSTERS
    BY ethnicity, age, zipcode, salary
    SEGMENT OVER gender
        PROTECTED  VALUES ('Female','NA')
        PRIVILEGED VALUES ('Male')
    HAVING ethnicity IN ('WHITE','BLACK')
```

Listing 5: Example of a clustering based on ethnicity, age, zip code, and salary segmented over gender for fairness analytics.

on the ethnicity, age, and the segment column. A generic case of group by is illustrated in Algorithm 1.

Clustering-based Grouping

We are the first to propose another approach for FAs to conduct fairness analysis based on *K-Prototypes* clustering (Huang, 1997), especially when the columns of the test dataset are mixed, i.e., numerical and categorical values. For instance, in Listing 5, salary, and age (here assume not bucketized) are numerical feature columns while the rest are categorical. FA can prototype and test different values of K, and clustering columns to further delve into the clusters where the model needs improvements and does show a degree of bias based for the protected attributes. In addition, FA can provide arbitrary distance metrics for the numerical columns, as well, and further shed the light on different ramifications of the model predictions based on different distance metrics. For instance, FA may want to add extra weight for the distance based on the age, which they can do by supplying the distance function definition. Besides the restriction on the distance function, which can only access the numerical columns, the distance function cannot also utilize the segment column because it is going to be used to calculate the fairness metrics within a cluster.

Similar to the previous case (Section 6.3.4), the ultimate goal is to consider the cluster association of each row as a singular grouping column (call it 'cluster') criteria, and perform

the same approach as in the Section 6.3.4 by using the cluster, and segment column values for each row to calculate aggregations and fairness metrics.

Clustering Data Obliviousness

K-Prototypes (Huang, 1997) combines K-Modes (Chaturvedi et al., 2001) and K-Means (Hartigan and Wong, 1979) to enable clustering analysis for datasets that have both numerical and categorical features. In general, the flow of a clustering algorithm is as follows: 1) For each row in the dataset, calculate the distance to every cluster centroid, And assign that entry to the cluster with the minimum distance to the centroid. 2) Update each cluster's centroid based on the rows belonging to the cluster. A naive approach (non-oblivious) leaks row-wise cluster assignment, and cluster sizes, which is not desirable when FA owns sensitive data.

AISolace proposes a simpler but data-oblivious variation of K-Prototype. A generic flow of the clustering is illustrated in Algorithm 2.

Initialization Each point is randomly assigned to a cluster, and the cluster assignment of each row is stored in a new column. It is worth mentioning that at the end of the algorithm, this column will be referenced as the cluster group column when performing the grouping. This step has no data-dependent memory access, hence, it is data-oblivious.

Centroid Recalculation At this step, each centroid should be assigned with an updated value per column (involved in clustering 'BY' statement). For numerical columns, the mean should be calculated, and for categorical columns the mode. AISolace utilizes the data-oblivious sorting primitive to sort the dataset based on the cluster column. After the sorting step is done, points belonging to a cluster are in sequential order, and AISolace can proceed with calculating the aggregates (mode for categorical, and mean for numerical) in one scan.

Since the number of clusters are usually not significant, when processing each row to calculate the partial aggregates, AISolace holds all the clusters meta in an array and accesses every entry of the array, and manipulates the irrelevant clusters with dummy (or inconsequential) updates. For numerical columns, we need to keep tack of the counts, and sum, while for categorical columns we should keep track of the frequencies. To obliviously handle the frequencies of each variant, AISolace assumes each categorical column i can have a maximum of M_i variants. Once the sequential scan is over, for numerical columns the mean is calculated. However, for the Mode, it needs to obliviously traverse the sorted rows, and for each column of every centroid keeps a counter. Then for every categorical column, the centroid picks the maximum frequency variant, to be used for calculating distance in the next step.

Cluster Reassignment At this stage, for every entry in the dataset, AISolace assigns the entry to the closest cluster. Every row's distance to all the centroids is calculated. The distance is a weighted sum of the distances of the numerical columns and categorical columns. For numerical columns, because it is a mathematical function with no conditional logic, it is data-oblivious out of the box. However, for categorical columns, it needs to perform the calculation in a data-oblivious manner. The overall distance for categorical columns depends on conditional evaluation and comparison to the centroids (statistical mode). If they match, the distance remains the same, otherwise it is increased by 1. Note that users can set reasonable weights, to balance off the distance computed for categorical and numerical values.

The centroid with the minimum distance will be selected as the cluster candidate. This step can be done via oblivious selection. By the end of this step, the cluster column will have the new cluster assignments for each row, and it is ready for the centroid recalculation stage.

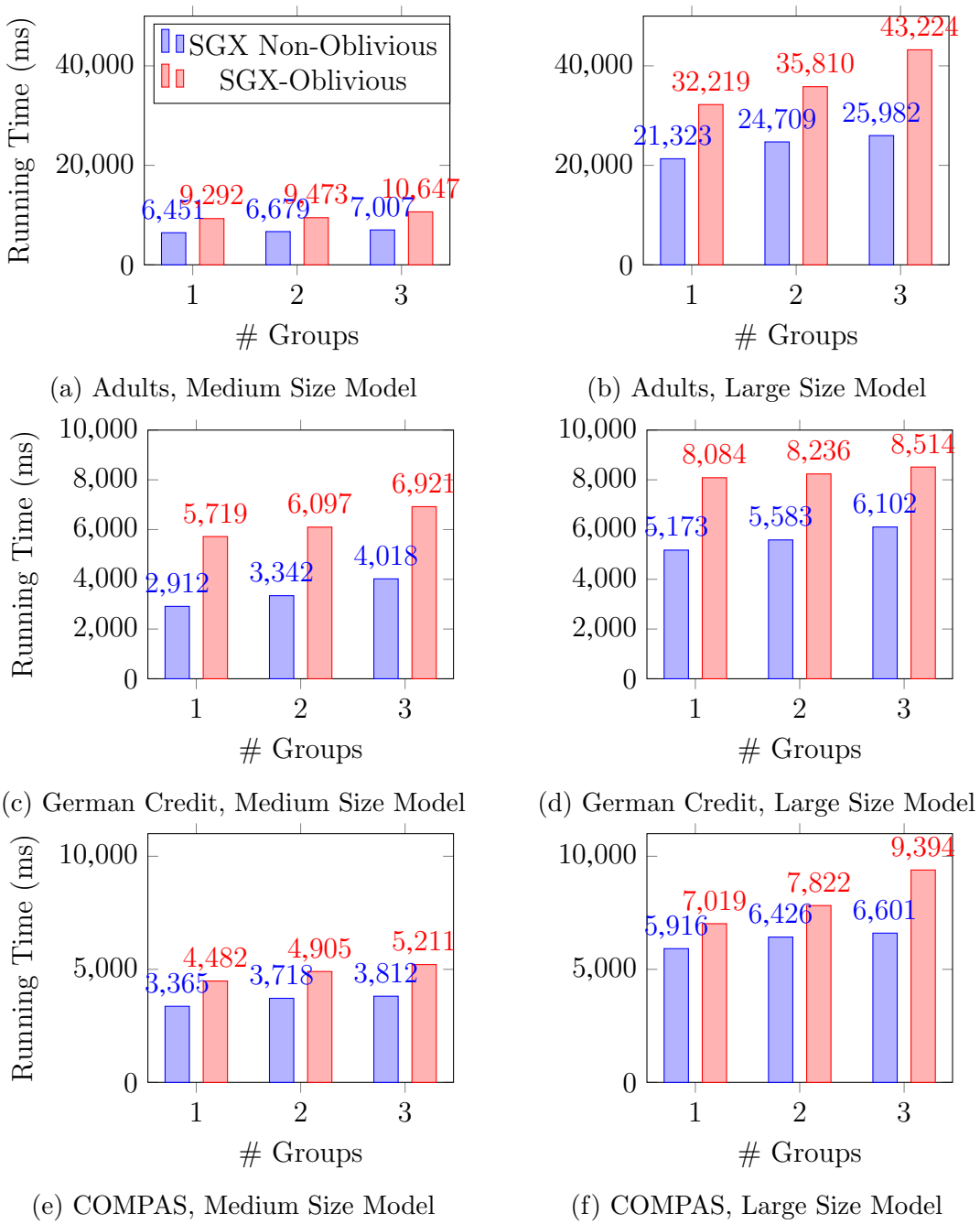
6.3.5 Secure Fairness Analytics

Once the test set has the grouping assignment (either via predicate-grouping or clustering), it is viable to run numerous fairness metrics on each group separately, and compare and contrast them. We want to stress how our framework fits in to the current popular frameworks such as AIF360 (Bellamy et al., 2018), and Fairlearn (Bird et al., 2020) which do not provide privacy guarantees as of now. First, we have the notion of segment column in our framework, which is the equivalent of the protected attribute in AIF360 in which you can provide specific values for column to designate whether they belong to a protected group or not. Another concept that we borrow is from Fairlearn in which they have the concept of *control* columns. Control column allows separating the dataset into groups so that the fairness metrics are computed for each group separately (inter-group), and they can be compared with each other across groups (intra-group). Our oblivious grouping mechanism are for the exact same purpose and is equivalent to control column(s) feature of Fairlearn. In short, AISolace tries to take the best of the both worlds to enable FAs conduct varous fairness analytics.

6.4 Experiments

The experiments conducted on a computer with Linux operating system that had 64GB of RAM, and Intel(R) Xeon(R) W-10855M @ 2.80GHz CPU. Further, AISolace prototype is developed in Rust language, and relies on Fortanix EDP (Fortanix, 2021). The experiments are divided into two sections. First, we investigate the performance of AISolace for regular grouping criteria. Next, we investigate the performance for clustering-based grouping.

We use three datasets that are popular in fairness literature. First dataset is Adult(Dua and Graff, 2017) (Census Income) dataset which 9000 records from the test set is used. The classifier determines whether the individual makes more than $50K a year. The second dataset is COMPAS(ProPublica, 2016) which we use 3000 of its records as the test set. Also,



(a) Adults, Medium Size Model

(b) Adults, Large Size Model

(c) German Credit, Medium Size Model

(d) German Credit, Large Size Model

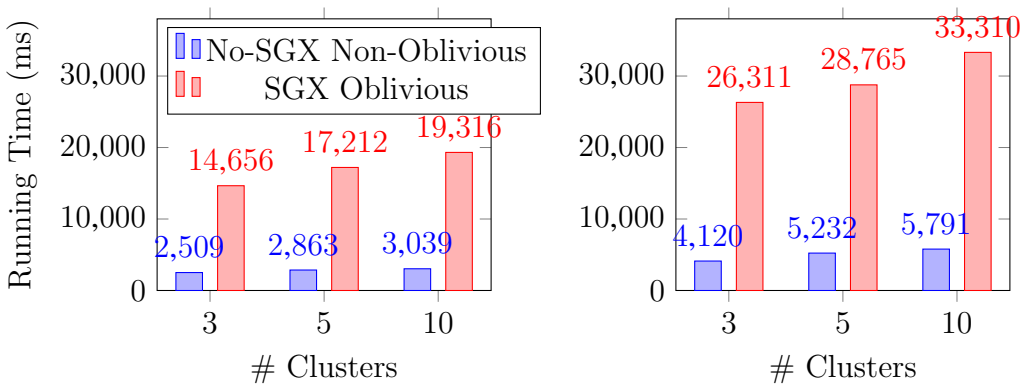
(e) COMPAS, Medium Size Model

(f) COMPAS, Large Size Model

Figure 6.2: Running time of regular groupings

selected 21 of the columns[1]. Finally, for the German Credit dataset (Dua and Graff, 2017),

[1]columns such as first name, last name, etc. are dropped



(a) COMPAS (b) Adult

Figure 6.3: Clustering Running Time without including the inference time

the objective of model built on this dataset is to determine whether the individual is a good or bad credit risk. We use all the 1000 records and all the 18 columns present in the dataset.

For the ML models, we use deep neural networks due to their recent popularity in ML applications. For DNN models, we devised two configurations. The first, which we refer to as the "medium" model, has four fully connected layers with the output sizes (16, 32, 16, 1). The "large" configuration has nine fully connected layers with the output sizes (32, 32, 32, 64, 64, 128, 64, 32, 1). We only modify the input size of the first fully connected layer to match the input dimension per given dataset. Finally, for all the cases, we computed four general statistics. 1) True positives, which is the count of instances that model predicts positive, and they match the provided label. 2) False positives, the counts of the instances where model predicts positive, however the label from ground truth is negative. 3) True negatives, that counts the number of instances where the model predicts negative, and it matches the ground truth. 4) False negatives, that counts the number of instances that model predicts negative, however the ground truth label is positive. We would like to stress that all the grouping based fairness criteria reported in practical tools such as IBM AI fairness 360 (Bellamy et al., 2018) can be computed using these four general statistics per given group.

6.4.1 Grouping Performance

For each dataset, we evaluate two cases: 1) the computation is using data-oblivious mode, 2) the computation leaks memory access pattern. In addition, we test for three different grouping size configuration. The first configuration is across the segment column, while the other two has additional column(s) that can be used to compute fairness metrics across subgroups.

Figure 6.2 illustrates the running time of AISolace for the configurations mentioned thus far. Note that these results include the time taken for the model inference.

For Adult dataset (figure 6.2 a—b), first, we observe the larger model results in a higher analysis runtime. Second, in terms of the number of the grouping columns, in both model configurations, the overall runtime increases as it is expected. The case for non-oblivious does not involve any sorting, and we use a hash table to store the statistics for each group variation. On the other hand, for oblivious grouping, the operation involves bitonic-sort and oblivious aggregation. For larger model cases, since we need to get a larger enclave to accommodate the inference, the overall fairness analysis is slower (for the 3 columns case, 66% overhead vs 52% overhead between for oblivious mode vs non-oblivious mode).

For German Credit dataset (figure 6.2 c—d), for medium model size the overall run-time in oblivious mode incurs 72% to 96% overhead, while in the large model case it is between 39% to 47%. For COMPASS dataset (figure 6.2 e—f), the overhead of oblivious mode for medium model is between 31% to 36%, while for the large model the overhead is between 18% to 42%. Finally, it is worth mentioning that we also run the experiments in a setting without the SGX, for Adult dataset, and it was 40**X** faster. The majority of the difference is due to inference time and the fact that the non-SGX package outsource the matrix multiplication calls to an efficient library where it is not available to the SGX.

6.4.2 Clustering Performance

To conduct experiments on clustering and evaluate the running time, we restrict our experiments to medium size models only. Furthermore, we only consider Adults and COMPASS datasets. Additionally, we exclude the inference time, and the results are reported for the grouping and fairness metric computation only. Also, for each dataset, we used three categorical columns, and two numeric columns to compute the *mode* and *euclidean* distances, respectively with 3,5, and 10 clusters. Finally, we only evaluate in two settings, between in-SGX oblivious and non-SGX non-oblivious for 10 iterations of clustering. For the COPASS dataset (figure. 6.3a) the oblivious mode incurs an overhead of 5.8**X** to 6.4**X**, and as the number of clusters increases the growth rate of the overhead slightly shrinks. For the Adult dataset (figure. 6.3b), we observe the overhead is between 5.5**X** to 6.6**X**.

6.5 Conclusion

In this chapter, we presented AISolace; a data-oblivious privacy-preserving framework for fairness analysis of models built based on deep learning. We assumed model provider and fairness analyst do not trust each other, so they relied on AISolace to provide confidentiality with respect to the model (parameters), and the fairness test data. Moreover, AISolace allows for complex groupings, and clustering, to run the fairness analysis across subgroups over predefined set of protected group definitions. Finally, we want to stress that the data-obliviousness is not supported by TEEs by default, however, lack of such a feature can have huge repercussions. AISolace runs all the operation in a data-oblivious manner so as long as the data size and the fairness analysis is fixed, the memory access pattern in not data-dependent.

CHAPTER 7

INVESTIGATION OF A DIFFERENTIAL CRYPTANALYSIS INSPIRED APPROACH FOR TROJAN AI DETECTION[1]

7.1 Introduction

Deep Learning(DL) is playing a major role in our lives and its high-pace improvements will continue to broaden its footprint in our daily routines. However, as the systems that are using more complex DL techniques, architectures, etc. increases, the research community must keep up with ways in which these systems can act unexpectedly and it should not remain undetected. In this work, we focus on trojan backdoor attacks (Liu et al., 2017; Gu et al., 2017b) which are not easy to detect or fully mitigate, however, these attacks can be materialized effortlessly through simple data-poisoning during training or through transfer learning as the base model can carry the faulty hidden behavior into the freshly trained model. We investigate the behavior of models w/w.o. backdoor on a dataset of random unrelated images to the application domain these models are trained on. We try to answer how similar the behavior of the clean models are? Or how dissimilar their behavior are between clean/poisoned (contains a backdoor) models, given the unrelated inputs?

The main findings of this work are as follows:

- We built a meta-classifier to detect poisoned/clean models with an average accuracy of 77%.

- We discovered that random samples drawn from a normal or uniform distribution is not an effective measure to derive meaningful signals to build a meta-classifier, due to

[1]©2021 SPIE Reprinted. with permission, from A. Asvadishirehjini, M. Kantarcioglu and Y. Zhou, "Investigation of a differential cryptanalysis inspired approach for Trojan AI detection ", In Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III, vol. 11746, p. 117460X. International Society for Optics and Photonics, 2021.

its inefficiency in approximating a true random image distribution. Hence, we believe huge datasets from unrelated domains can significantly improve the performance of the meta-classifier.

7.2 Methodology

In this section, we formalize our methodology for analyzing the behavior of models on random inputs[2]. Given what we discussed so far, we wanted to see if there are efficient ways to build a meta-classifier that can efficiently determine whether a model contains a backdoor (i.e. poisoned) or not w.o. the need for costly gradient optimizations on the input domain (Wang et al., 2019).

7.2.1 Cryptographic View On Backdoors

For a classification model $\mathcal{F}^\theta$ with parameters θ, that gets an input $\mathcal{X}$ in the domain $\mathbb{D}^x$, and returns the output $\mathcal{Y}$ in the domain $\mathbb{D}^y$:

$$\mathcal{F}^\theta : \mathbb{D}^x \rightarrow \mathbb{D}^y$$
$$\mathcal{Y} = \mathcal{F}^\theta(\mathcal{X})$$

In practice, the prediction result is a probability distribution over the classification categories ($\sum_{i \in C} \mathcal{Y}_i = 1$) that highlights the model's belief for each class. Typically, the final decision class is the class c highest value.

In the presence of a backdoor (like an unknown encryption key) $\mathcal{K}^{c_t}$ in a poisoned model $\mathcal{P}^\theta$, for any malicious input $\mathcal{X}'$ prediction is c_t with a high probability regardless of the true

[2]either from unrelated inputs or sampled from a random distribution

classification category.

$$\mathcal{K}^{c_t} : \mathcal{D}^x \times \mathcal{D}^\theta \rightarrow \mathcal{D}^y$$
$$\mathcal{P}^\theta : \mathcal{D}^x \rightarrow \mathcal{D}^y$$
$$\mathcal{X}' = \mathcal{K}^{c_t}(\mathcal{X}, \mathcal{P}^\theta)$$
$$\mathcal{Y}' = \mathcal{P}^\theta(\mathcal{X}')$$

7.2.2 Problem Definition

Given the above construction, we aim to find a mechanism to activate trojan behavior that distinguish the clean models from the poisoned ones.

Hypothesis 1. *A poisoned Model $\mathcal{P}^\theta$ with an unknown backdoor function $\mathcal{K}^{c_t}$, in comparison to a clean model $\mathcal{F}^\theta$, has a distinguishable prediction distribution over random samples drawn from an unrelated domain.*

Definition 1. *We define the distance Δ between a model's prediction class distribution C from a uniform prior $\mathcal{U}$ as:*

$$\begin{aligned} \Delta^i(C, \mathcal{U}) &= |C_i - \mathcal{U}_i| \\ \Delta &= \tfrac{1}{2} \sum_{i \in C} \Delta^i \end{aligned} \tag{7.1}$$

7.2.3 Prediction Class Probability Computation

To compute the distance for a model in either of $\mathcal{P}^\theta$ or $\mathcal{F}^\theta$ (we refer to both as $\mathcal{M}^\theta$), an unrelated dataset $\mathcal{S}$ (e.g. random noise, or an unseen dataset), is used. The entries of $\mathcal{S}$ are fed to the model for prediction, and a prediction class probability vector C is computed with two different approaches.

Hard Label

In this approach, we only count the final classification label which is the class with the highest probability. Simply every item i in C is just the number of samples that are labeled

as i divided by the total number of the samples in $\mathcal{S}$.

$$C_i = \frac{\sum_{s \in \mathcal{S}} \mathbf{1}[argmax(\mathcal{M}^{\theta}(s)) = i]}{|\mathcal{S}|} \tag{7.2}$$

Soft Label

The hard-label approach loses information about other classes. In contrast, the soft-label approach sums all the assigned classification probabilities per class across the $\mathcal{S}$ and then normalizes the outcome into a probability distribution. This way, we train more information at the expense of some memory cost which is not very significant.

$$C_i = \frac{\sum_{s \in \mathcal{S}} \mathcal{M}^{\theta}(s)[i]}{\sum_{s \in \mathcal{S}} \sum_{j \in C} \mathcal{M}^{\theta}(s)[j]} \tag{7.3}$$

7.3 Experiments

In this section, we discuss our empirical analysis. First, we explain the configuration for the environment and experiment setup. Next, we go over the empirical results we gained and their ramifications.

7.3.1 Experiment Setup

These experiments are performed on the CIFAR10 (Krizhevsky et al., 2014) dataset. It consists of 60K image samples in 10 classification categories. We trained 200 models with an adapted ResNet(20) (He et al., 2016) architecture for CIFAR10 dimensions. 100 of the trained models are "clean", while 100 of the models are "poisoned"; meaning there exists a backdoor pattern that triggers the malicious behavior. The attacks performed on poison models can differ on the backdoor pattern characteristics, and also the attack target labels as described in Table 7.1. All these models have been trained for 250 epochs. Poisoned models

Table 7.1: Poisoned Models Attack Configuration Parameters

Configuration	Values
Attack target class	[5,8]
Pattern location	[top left, bottom left, top right, bottom right]
Pattern size	[small(2x2), medium(5x5), big(8x8)]
Pattern color	[red, blue, green, pink]

usually have a small accuracy performance degradation of around 2%. Each dataset analysis consists of the two approaches, i.e hard-label and soft-label for prediction class probability distance (Δ) computation. For each approach there are 10 (CIFAR10 classes) probability features that is used to build a Random Forest (Breiman, 2001) model classifier to distinguish clean and poisoned models.

7.3.2 Analysis On ImageNet

Table 7.2: Hard and Soft Label Per Class Distance Statistics for ImageNet

Approach	Type	Statistic	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	RF acc
soft	clean	mean	.017	.058	.057	.076	.029	.029	.009	.04	.02	.022	
		std	.004	.002	.005	.007	.003	.005	.003	.003	.004	.004	mean=.77
		median	.016	.059	.057	.076	.029	.028	.009	.04	.029	.021	
	poisoned	mean	.018	.058	.053	.071	.028	.031	.009	.038	.025	.02	
		std	.004	.003	.011	.014	.011	.006	.005	.008	.008	.010	std=.019
		median	.018	.059	.055	.075	.032	.031	.008	.04	.027	.02	
hard	clean	mean	.017	.06	.058	.079	.031	.027	.008	.043	.029	.024	
		std	.004	.002	.005	.008	.003	.005	.003	.003	.004	.005	mean=.76
		median	.016	.006	.059	.079	.031	.027	.008	.043	.03	.014	
	poisoned	mean	.019	.061	.055	.076	.029	.027	.008	.042	.026	.024	
		std	.004	.004	.011	.013	.001	.008	.005	.007	.009	.01	std=.05
		median	.019	.061	.057	.078	.034	.027	.008	.043	.028	.025	

Our analysis starts with the ImageNet (Deng et al., 2009) dataset. We used the training set which has more than 1.2M images in 1000 categories, however, we ignored the labels. Then we built a random forest classifier to better understand the potentially complex relationship between the distances we derived from a uniform prior. Denote that initially, we tried to find a simple threshold (based on every single metric, and one metric that summed

over the distances like the total variation distance). Nonetheless, it did not work well, even when we separated the experiments based on the attack target class (Table 7.1). In Table 7.2, we show the statistical properties related to each prediction class distance across the models. Also, for each prediction class probability calculation we build a separate classifier. We repeated a 5-fold cross-validation for 10 rounds. The statistical mean and the standard deviation of the 50 models' performance on the hold out set (20%) have been reported. We can see that it is able to correctly classify the poisoned models from the clean ones with 77% accuracy while having a low standard deviation. Finally, we observe that the poisoned model consistently tends to have a higher standard deviation as opposed to the clean models.

7.3.3 Analysis On Tiny-ImageNet

Tiny-ImageNet (Wu et al., 2017) dataset, is an enhanced subset of the imagenet dataset with 200 classes and 100K training samples. We performed the same steps as in the previous case, and the best result had a mean accuracy of 70% (the hard-label case was 67%) with a standard deviation of 0.07 (not a percentage scale). Clearly, we can see that having a bigger dataset that contains a wider array of variations can elicit better signals for our methodology.

7.3.4 Analysis On Random Uniform Input

Finally, we tested our methodology on an artificial dataset of 200K images all drawn from a normal distribution $\mathcal{N}(\mu = 0, \sigma = 1)$. This approach had the least success with a mean accuracy of 61% with a standard deviation of 0.07. We believe a mere random sampling cannot model the complexities of a true image, however, we plan to incorporate Generative Adversarial Networks (GAN) (Goodfellow et al., 2014) to randomly generate high-quality images.

7.4 Conclusion

In this chapter, we investigated the performance of models with backdoors and clean models using mainly two categories of datasets. 1) Random unrelated real images and 2) Random samples were drawn from a normal distribution. We tried to elicit signals from the prediction outcomes and use them to tell apart the clean and poisoned models. ImageNet (Deng et al., 2009) proved to have the highest performance with about 77% accuracy. We believe the more images that are available to extract the metrics we discussed, the higher the accuracy will be. For the future, we plan to incorporate GANs (Goodfellow et al., 2014) to generate synthetic large datasets, that can resemble natural image distributions to a high degree.

CHAPTER 8

CONCLUSION

In this dissertation, we presented solutions to ensure and increase the trust in machine learning by using TEEs (e.g., Intel SGX). In particular, we emphasized on finding the silver lining between performance and security. When it was possible to tolerate a level of performance overhead, we provided a stronger guarantee on the security, wherein we considered protections against memory access pattern leakage. Nonetheless, for applications of training a deep neural network, we followed a pursuit of realizing a practical solution for real-world scenarios in cloud environments, thus we focused on delivering integrity guarantees during the training without a significant performance overhead.

As a starting point, in Chapter 4, we presented a practical solution to accommodate training deep neural networks by combining GPU with TEE and enforcing a set of narrow gradient clipping on the update step, which thwarts an attacker from maliciously training a model to ensure that an attacker will get caught with a high probability.

Next, in Chapter 5, we proposed an analytics framework for deep learning inference combined with genomic GWAS analytics that takes into account the protection against memory access pattern side-channel leaks.

Later in Chapter 6, we delivered a data-oblivious and privacy-preserving solution for fairness tests for machine learning wherein the data owner and model owner do not trust each other and both the ML model and the test data needs to be kept private.

Finally, in Chapter 7, we investigated the feasibility of detecting models that have hidden trojan through building a machine learning metamodel that distinguishes between poisoned (with trojan) and clean (without trojan).

In summary, this dissertation demonstrated the applications of trusted execution environments in enabling solutions for integrity, privacy, and fairness for machine learning. For

future work, enabling an end-to-end operational pipeline for machine learning from data-collection, training, inference, and fairness in a distributed setting (blockchains for provenance of the inputs, and outputs) while maintaining privacy and integrity properties, should be an interesting direction to pursue.

REFERENCES

(2017). Idash privacy & security workshop 2017. `http://www.humangenomeprivacy.org/2017/about.html`.

(2019). Idash privacy & security workshop 2019. `http://www.humangenomeprivacy.org/2019/competition-tasks.html`.

A., K. Intro to optimization in deep learning: Momentum, rmsprop and adam. `https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/`.

Akgün, M., A. O. Bayrak, B. Ozer, and M. Ş. Sağıroğlu (2015). Privacy preserving processing of genomic data: A survey. *Journal of biomedical informatics 56*, 103–111.

Arasu, A. and R. Kaushik (2013). Oblivious query processing. *arXiv preprint arXiv:1312.4012*.

Asvadishirehjini, A., M. Kantarcioglu, and B. Malin (2020). Goat: Gpu outsourcing of deep learning training with asynchronous probabilistic integrity verification inside trusted execution environment. *arXiv preprint arXiv:2010.08855*.

Batcher, K. E. (1968). Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pp. 307–314.

Bellamy, R. K. E., K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilovic, S. Nagar, K. N. Ramamurthy, J. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang (2018, October). AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias.

Bird, S., M. Dudík, R. Edgar, B. Horn, R. Lutz, V. Milan, M. Sameki, H. Wallach, and K. Walker (2020, May). Fairlearn: A toolkit for assessing and improving fairness in AI. Technical Report MSR-TR-2020-32, Microsoft.

Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152.

Brasser, F., U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi (2017, August). Software grand exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC. USENIX Association.

Breiman, L. (2001). Random forests. *Machine learning 45*(1), 5–32.

Bron, C. (1971). Proof of a merge sort algorithm.

Bulck, J. V., M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx (2018, August). Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, pp. 991–1008. USENIX Association.

Carpov, S. and T. Tortech (2018). Secure top most significant genome variants search: idash 2017 competition. Cryptology ePrint Archive, Report 2018/314. `https://eprint.iacr.org/2018/314`.

Chaturvedi, A., P. E. Green, and J. D. Caroll (2001). K-modes clustering. *Journal of classification 18*(1), 35–55.

Chen, F., M. Dow, S. Ding, Y. Lu, X. Jiang, H. Tang, and S. Wang (2016). Premix: Privacy-preserving estimation of individual admixture. In *AMIA Annual Symposium Proceedings*, Volume 2016, pp. 1747. American Medical Informatics Association.

Chen, F., C. Wang, W. Dai, X. Jiang, N. Mohammed, M. M. Al Aziz, M. N. Sadat, C. Sahinalp, K. Lauter, and S. Wang (2017). Presage: Privacy-preserving genetic testing via software guard extension. *BMC medical genomics 10*(2), 48.

Chen, F., S. Wang, X. Jiang, S. Ding, Y. Lu, J. Kim, S. C. Sahinalp, C. Shimizu, J. C. Burns, V. J. Wright, et al. (2016). Princess: Privacy-protecting rare disease international network collaboration via encryption through software guard extensions. *Bioinformatics 33*(6), 871–878.

Chen, G., S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai (2019). Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 142–157. IEEE.

Chen, X., C. Liu, B. Li, K. Lu, and D. Song (2017). Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*.

Chen, X., Z. S. Wu, and M. Hong (2020). Understanding gradient clipping in private sgd: A geometric perspective.

Chouldechova, A., D. Benavides-Prado, O. Fialko, and R. Vaithianathan (2018). A case study of algorithm-assisted decision making in child maltreatment hotline screening decisions. In *Conference on Fairness, Accountability and Transparency*, pp. 134–148. PMLR.

Chouldechova, A. and A. Roth (2018). The frontiers of fairness in machine learning. *arXiv preprint arXiv:1810.08810*.

Constable, S. D., Y. Tang, S. Wang, X. Jiang, and S. Chapin (2015, Dec). Privacy-preserving gwas analysis on federated genomic datasets. *BMC Medical Informatics and Decision Making 15*(5), S2.

Corbett-Davies, S., E. Pierson, A. Feller, S. Goel, and A. Huq (2017). Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd acm sigkdd international conference on knowledge discovery and data mining*, pp. 797–806.

Costan, V. and S. Devadas (2016). Intel sgx explained. *IACR Cryptology ePrint Archive 2016*(086), 1–118.

Court, D. S. (2018). Forensic genealogy: Some serious concerns. *Forensic Science International: Genetics 36*, 203 – 204.

Csongor, R. Tesla raises the bar for self-driving carmakers.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee.

Devadas, S., M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs (2016). Onion oram: A constant bandwidth blowup oblivious ram. In *Theory of Cryptography Conference*, pp. 145–174. Springer.

Dua, D. and C. Graff (2017). UCI machine learning repository.

Dwork, C., M. Hardt, T. Pitassi, O. Reingold, and R. Zemel (2012). Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pp. 214–226.

Edge, M. D., B. F. B. Algee-Hewitt, T. J. Pemberton, J. Z. Li, and N. A. Rosenberg (2017). Linkage disequilibrium matches forensic genetic records to disjoint genomic marker sets. *Proceedings of the National Academy of Sciences*.

Evarist Fomenko, Vadim Pirogov, R. D. (2020). onednn. `https://github.com/oneapi-src/oneDNN.git`.

Fortanix (2021). Enclave development platform.

Freivalds, R. (1977). Probabilistic machines can use less running time. In *IFIP congress*, Volume 839, pp. 842.

G., K. and M. H. (2018). The golden state killer is tracked through a thicket of dna, and experts shudder. `https://www.nytimes.com/2018/04/27/health/dna-privacy-golden-state-killer-genealogy.html`.

Gao, Y., C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal (2019). Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 113–125.

Gilad-Bachrach, R., N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing (2016). Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pp. 201–210. PMLR.

Goldreich, O. (2009). *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

Goldreich, O. and R. Ostrovsky (1996). Software protection and simulation on oblivious rams. *Journal of the ACM (JACM) 43*(3), 431–473.

Goodfellow, I., Y. Bengio, and A. Courville (2016a). 10.11 optimization for long-term dependencies. *Deep Learning*, 408–411.

Goodfellow, I., Y. Bengio, and A. Courville (2016b). 6.5 back-propagation and other differentiation algorithms. *Deep Learning*, 200–220.

Goodfellow, I., Y. Bengio, and A. Courville (2016c). *Deep learning*. MIT press.

Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.

Götzfried, J., M. Eckert, S. Schinzel, and T. Müller (2017). Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, pp. 1–6.

Gras, B., K. Razavi, H. Bos, and C. Giuffrida (2018). Translation leak-aside buffer: Defeating cache side-channel protections with {TLB} attacks. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 955–972.

Gu, T., B. Dolan-Gavitt, and S. Garg (2017a). Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*.

Gu, T., B. Dolan-Gavitt, and S. Garg (2017b). Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*.

Guttmacher, A. E. and F. S. Collins (2003). Welcome to the genomic era.

Hanzlik, L., Y. Zhang, K. Grosse, A. Salem, M. Augustin, M. Backes, and M. Fritz (2021). Mlcapsule: Guarded offline deployment of machine learning as a service. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3300–3309.

Hardt, M., E. Price, and N. Srebro (2016). Equality of opportunity in supervised learning. *Advances in neural information processing systems 29*, 3315–3323.

Harrison, G. (2015). Database survey. In *Next Generation Databases: NoSQL, NewSQL, and Big Data*, pp. 217–228. Berkeley, CA: Apress.

Hartigan, J. A. and M. A. Wong (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics) 28*(1), 100–108.

Hashemi, H., Y. Wang, and M. Annavaram (2021). Privacy and integrity preserving training using trusted hardware. *arXiv preprint arXiv:2105.00334*.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hoare, C. A. R. (1961, July). Algorithm 64: Quicksort. *Commun. ACM 4*(7), 321.

Huang, Z. (1997). Clustering large data sets with mixed numeric and categorical values. In *Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(PAKDD)*, pp. 21–34. Citeseer.

Humbert, M., E. Ayday, J.-P. Hubaux, and A. Telenti (2013). Addressing the concerns of the lacks family: Quantification of kin genomic privacy. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, New York, NY, USA, pp. 1141–1152. ACM.

Hunt, T., C. Song, R. Shokri, V. Shmatikov, and E. Witchel (2018). Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*.

Hynes, N., R. Cheng, and D. Song (2018). Efficient deep learning on multi-source private data. *arXiv preprint arXiv:1807.06689*.

Intel. Intel sgx developer reference.

Intel. (2015). Intel software guards extensions. `https://web.stanford.edu/class/ee380/Abstracts/150415-slides.pdf`.

Intel, I. (64). and ia-32 architectures software developer's manual. *Volume 3A: System Programming Guide, Part 1*(64), 64.

Islam, M. S., M. Kuzu, and M. Kantarcioglu (2012). Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Ndss*, Volume 20, pp. 12.

Johnson, S., V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen (2016). Intel software guard extensions: Epid provisioning and attestation services. *White Paper 1*(1-10), 119.

Juvekar, C., V. Vaikuntanathan, and A. Chandrakasan (2018). {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1651–1669.

Kim, H.-Y. (2017). Statistical notes for clinical researchers: Chi-squared test and fisher's exact test. *Restorative dentistry & endodontics 42*(2), 152–155.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kiourti, P., K. Wardega, S. Jha, and W. Li (2019). Trojdrl: Trojan attacks on deep reinforcement learning agents. *CoRR abs/1903.06638*.

Knott, B., S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten (2021). Crypten: Secure multi-party computation meets machine learning. *arXiv preprint arXiv:2109.00984*.

Kocher, P., D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom (2018). Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*.

Krizhevsky, A., V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

Krizhevsky, A., V. Nair, and G. Hinton (2014). The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html 55*, 5.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems 25*, 1097–1105.

Kunkel, R., D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer (2019). Tensorscone: A secure tensorflow framework using intel sgx. *arXiv preprint arXiv:1902.04413*.

Lambrecht, A. and C. Tucker (2019). Algorithmic bias? an empirical study of apparent gender-based discrimination in the display of stem career ads. *Management science 65*(7), 2966–2981.

LeCun, Y. and C. Cortes (2010). MNIST handwritten digit database.

Lipp, M., M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg (2018). Meltdown. *arXiv preprint arXiv:1801.01207*.

Liu, C., X. S. Wang, K. Nayak, Y. Huang, and E. Shi (2015). Oblivm: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy*, pp. 359–376. IEEE.

Liu, Y., W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang (2019). Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1265–1282.

Liu, Y., S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang (2017). Trojaning attack on neural networks.

Maas, M., E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song (2013). Phantom: Practical oblivious computation in a secure processor. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 311–324.

Mandal, A., J. C. Mitchell, H. Montgomery, and A. Roy (2018). Data oblivious genome variants search on intel sgx. Cryptology ePrint Archive, Report 2018/732. `https://eprint.iacr.org/2018/732`.

McGrew, D. A. and J. Viega (2004). The security and performance of the galois/counter mode (gcm) of operation. In *International Conference on Cryptology in India*, pp. 343–355. Springer.

Mehrabi, N., F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR) 54*(6), 1–35.

Microsoft (2021). Azure confidential computing. `https://azure.microsoft.com/en-us/solutions/confidential-compute`.

Mittos, A., B. Malin, and E. D. Cristofaro (2017). Systematizing genomic privacy research - A critical analysis. *CoRR abs/1712.02193*.

Mohassel, P. and P. Rindal (2018). Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 35–52.

Mohassel, P. and Y. Zhang (2017). Secureml: A system for scalable privacy-preserving machine learning. *IACR Cryptology ePrint Archive 2017*, 396.

Murdock, K., D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens (2020). Plundervolt: Software-based fault injection attacks against intel sgx. In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1466–1482. IEEE.

Naveed, M., E. Ayday, E. W. Clayton, J. Fellay, C. A. Gunter, J.-P. Hubaux, B. A. Malin, and X. Wang (2015). Privacy in the genomic era. *ACM Computing Surveys (CSUR) 48*(1), 6.

Ng, L. K., S. S. Chow, A. P. Woo, D. P. Wong, and Y. Zhao (2021). Goten: Gpu-outsourcing trusted execution of neural network training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 35, pp. 14876–14883.

Ohrimenko, O., F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa (2016). Oblivious multi-party machine learning on trusted processors. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 619–636.

Ozga, W., C. Fetzer, et al. (2021). Perun: Confidential multi-stakeholder machine learning framework with hardware acceleration support. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pp. 189–208. Springer.

Panwar, H., P. Gupta, M. K. Siddiqui, R. Morales-Menendez, and V. Singh (2020). Application of deep learning for fast detection of covid-19 in x-rays using ncovnet. *Chaos, Solitons & Fractals 138*, 109944.

Phong, L. T., Y. Aono, T. Hayashi, L. Wang, and S. Moriai (2018). Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security 13*(5), 1333–1345.

Priebe, C., K. Vaswani, and M. Costa (2018). Enclavedb: A secure database using sgx. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE.

ProPublica (2016). Machine bias. `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing`.

Qui, P., D. Wang, Y. Lyu, and G. Qu (2020). Voltjockey: Abusing the processor voltage to break arm trustzone. *GetMobile: Mobile Computing and Communications 24*(2), 30–33.

Raji, I. D. and J. Buolamwini (2019). Actionable auditing: Investigating the impact of publicly naming biased performance results of commercial ai products. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 429–435.

Ramakrishnan, R. and J. Gehrke (2000). *Database management systems*. McGraw Hill.

Rane, A., C. Lin, and M. Tiwari (2015, August). Raccoon: Closing digital side-channels through obfuscated execution. In *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C., pp. 431–446. USENIX Association.

Redmon, J. (2013–2016). Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`.

Robbins, H. and S. Monro (1951). A stochastic approximation method. *The annals of mathematical statistics*.

Russinovich, M. (2016). Introducing azure confidential computing.

SAMTools (2021). The variant call format specification. `https://samtools.github.io/hts-specs/VCFv4.3.pdf`.

Sasy, S., S. Gorbunov, and C. W. Fletcher (2017). Zerotrace: Oblivious memory primitives from intel sgx. *Cryptology ePrint Archive*.

Schwarz, M., S. Weiser, D. Gruss, C. Maurice, and S. Mangard (2017). Malware guard extension: Using sgx to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 3–24. Springer.

Shafahi, A., W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein (2018). Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems 31*.

Shaon, F., M. Kantarcioglu, Z. Lin, and L. Khan (2017). Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1211–1228.

Simonyan, K. and A. Zisserman (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*.

Stallkamp, J., M. Schlipsing, J. Salmen, and C. Igel (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*.

statgen.org (2012). A derivation for armitage's trend test for the 2 by 3 genotype table. http://statgen.org/wp-content/uploads/2012/08/armitage.pdf.

Stefanov, E., M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas (2013). Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 299–310.

Sun, L. (2020). Natural backdoor attack on text data.

Szegedy, C., S. Ioffe, V. Vanhoucke, and A. A. Alemi (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Tian, D. J., J. I. Choi, G. Hernandez, P. Traynor, and K. R. B. Butler (2019). A practical intel sgx setting for linux containers in the cloud. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, CODASPY '19, New York, NY, USA, pp. 255–266. Association for Computing Machinery.

Tkachenko, O., C. Weinert, T. Schneider, and K. Hamacher (2018). Large-scale privacy-preserving statistical computations for distributed genome-wide association studies. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, New York, NY, USA, pp. 221–235. ACM.

Tramer, F. and D. Boneh (2018). Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*.

Tsai, C.-C., D. E. Porter, and M. Vij (2017). Graphene-sgx: A practical library {OS} for unmodified applications on {SGX}. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pp. 645–658.

Van Bulck, J., N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx (2017). Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association.

van Schaik, S., M. Minkin, A. Kwong, D. Genkin, and Y. Yarom (2021). Cacheout: Leaking data on intel cpus via cache evictions. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 339–354. IEEE.

Wang, B., Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao (2019). Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 707–723. IEEE.

Wang, H., P. Wang, Y. Ding, M. Sun, Y. Jing, R. Duan, L. Li, Y. Zhang, T. Wei, and Z. Lin (2019). Towards memory safe enclave programming with rust-sgx. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2333–2350.

Wang, S., S. Nepal, C. Rudolph, M. Grobler, S. Chen, and T. Chen (2020). Backdoor attacks against transfer learning with pre-trained deep learning models. *arXiv preprint arXiv:2001.03274*.

Wang, W., G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter (2017). Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, New York, NY, USA, pp. 2421–2434. Association for Computing Machinery.

Wang, X., H. Chan, and E. Shi (2015). Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 850–861.

Wang, X. S., K. Nayak, C. Liu, T. H. Chan, E. Shi, E. Stefanov, and Y. Huang (2014). Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 215–226.

Wu, J., Q. Zhang, and G. Xu (2017). Tiny imagenet challenge.

Xu, J. (2014). *Next-generation Sequencing*. Caister Academic Press.

Yue, T. and H. Wang (2018). Deep learning for genomics: A concise overview. *arXiv preprint arXiv:1802.00810*.

Zahur, S. and D. Evans (2015). Obliv-c: A language for extensible data-oblivious computation. *IACR Cryptol. ePrint Arch. 2015*, 1153.

Zhang, J., T. He, S. Sra, and A. Jadbabaie (2019). Why gradient clipping accelerates training: A theoretical justification for adaptivity.

Zheng, W., A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica (2017). Opaque: An oblivious and encrypted distributed analytics platform. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pp. 283–298.

BIOGRAPHICAL SKETCH

Aref Asvadishirehjini received his bachelor's in Computer Engineering from Amirkabir University of Technology in 2013. Then, he received his master's in Information Technology in 2015. Afterwards, in 2015, he joined The University of Texas at Dallas's Computer Science doctoral program. His main research interests are privacy-preserving deep learning, and increasing social trust in AI/ML systems.

CURRICULUM VITAE

Aref Asvadishirehjini

November 4, 2021

Educational History:

BS, Computer Engineering, Amirkabir University of Technology, 2013
MS, Information Technology, Arkansas Tech University, 2015

Employment History:

Research Scientist, Facebook Inc, January 2021 – present
Research Assistant, The University of Texas at Dallas, January 2016 – December 2020
Teaching Assistant, The University of Texas at Dallas, August 2015 – December 2015
Graduate Assistant, Arkansas Tech University, August 2013 – May 2015