SOTERIA: Preserving Privacy in Distributed Machine Learning

Cláudia Brito, Pedro Ferreira, Bernardo Portela, Rui Oliveira, and João Paulo

Abstract—We propose SOTERIA, a system for distributed privacy-preserving Machine Learning (ML) that leverages Trusted Execution Environments (*e.g.* Intel SGX) to run code in isolated containers (enclaves). Unlike previous work, where all ML-related computation is performed at trusted enclaves, we introduce a hybrid scheme, combining computation done inside and outside these enclaves. The conducted experimental evaluation validates that our approach reduces the runtime of ML algorithms by up to 41%, when compared to previous related work. Our protocol is accompanied by a security proof, as well as a discussion regarding resilience against a wide spectrum of ML attacks.

Index Terms—Privacy-preserving, Machine Learning, Apache Spark, SGX, Outsourcing

1 INTRODUCTION

Outsourcing Machine Learning (ML) data storage and computation to third-party services (*e.g.*, cloud computing) leaves users vulnerable to attacks that may compromise the integrity and confidentiality of their data. Indeed, the ML pipeline encompasses several stages, both for model training and inference, in which users' data is known to be susceptible to different attacks such as *adversarial attacks*, *model extraction*, and *inversion*, and *reconstruction attacks* [1], [2].

Recent works have addressed these attacks with solutions based on homomorphic encryption or secure multi-party computation schemes. However, these cryptographic schemes impose a significant performance toll that restricts their applicability to practical scenarios [3]. To circumvent this performance penalty, another line of research is that of exploring hardware technologies enabling Trusted Execution Environments (TEEs), such as Intel SGX [4]. These technologies allow the execution of code within isolated processing environments (*i.e.,* enclaves) where data can be securely handled in its original form (*i.e.*, plaintext) at untrusted servers.

The latter approach typically deploys full ML workloads inside TEEs [5], [6]. However, as the amount of computational and I/O operations performed at the enclaves increases, the performance of ML training and inference is noticeably affected by hardware limitations, limiting the design's applicability in practice [7].

This paper builds upon the idea that ML runtime performance could be improved by reducing the number of operations done at enclaves. In fact, this insight is backed up by previous work [8], [9] exploring the partitioning of computation across trusted and untrusted environments, but in contexts (*e.g.*, SQL processing, MapReduce, distributed coordination) with different security requirements and processing logic than the ones found for ML workloads.

Therefore, the key challenge addressed by this paper is to understand and define the set of ML operations to run inside/outside TEEs. Ideally, these operations should significantly reduce the enclaves' overall computational and I/O load for different ML workloads; and doing so should not leak critical sensitive information during the execution of ML workloads.

Our reasoning is twofold: *i.)* the majority of current attacks on the ML pipeline is only successful if the attacker has some knowledge about the datasets and/or models being used [2], [10]; and *ii.)* studies show that such knowledge cannot be inferred from the information leaked by statistical operations, such as the calculation of confidence results, table summaries, ROC/AUC curves, and probability distributions for classes [11]. As a result, these operations are ideal candidates to be offloaded from enclaves. We support these claims by analyzing the security and performance implications of different ML workloads and attacks.

Thus, we propose SOTERIA, an open-source system for distributed privacy-preserving ML [1] that leverages the scalability and reliability provided by Apache Spark and its ML library (MLlib). Unlike previous solutions [12], [13], SOTERIA supports a wide variety of ML algorithms without changing how users build and run these within Spark. It ensures that critical operations, which enable existing attacks to reveal sensitive information from ML datasets and models, are exclusively performed in secure enclaves. This means that the sensitive information being processed only exists in plaintext when inside the enclave, being encrypted in the remainder data flow (*e.g.,* network, storage). This solution enables robust security guarantees, ensuring data privacy during ML training and inference.

SOTERIA introduces a new computation partitioning scheme for Apache Spark's MLlib, SOTERIA-P, that offloads non-critical statistical operations from the trusted enclaves

This work has been accepted on The 38th ACM/SIGAPP Symposium On Applied Computing (SAC'23). Copyright may be transferred without notice, after which this version may no longer be accessible. For any questions, please address to claudia.v.brito@inesctec.pt.

1. https://github.com/claudiavmbrito/soteria

to untrusted environments. SOTERIA-P is accompanied by a formal security proof for how data remains private during ML workloads and an analysis of how this guarantee ensures resilience against various ML attacks. Furthermore, SOTERIA offers a baseline scheme, SOTERIA-B, where all ML operations are done inside trusted enclaves without a fine-grained differentiation between critical and non-critical operations. SOTERIA-B provides a performance and security baseline for comparison against our new partitioned scheme.

We compare experimentally both approaches with a non-secure deployment of Apache Spark and a state-of-the-art solution, namely SGX-Spark [12]. Our experiments, resorting to the HiBench benchmark [14] and including four different ML algorithms, show that SOTERIA-P, while considering a larger subset of ML attacks, reduces training time by up to 41% for Gradient Boosted Trees workloads and up to 4.3 hours for Linear Regression workloads, when compared to SGX-Spark. Also, when compared to SOTERIA-B, SOTERIA-P reduces execution time by up to 37% for the Gradient Boosted Trees workloads and up to 3.3 hours for the Linear Regression workloads.

2 Background

2.1 Apache Spark and MLlib

Apache Spark is a distributed cluster computing framework that supports ETL, analytical, ML, and graph processing over large volumes of data. Spark follows a Master/Workers distributed architecture and can be deployed on a cluster of servers in the cloud that may access several data sources (*e.g.*, HBase, HDFS) for reading the data to be processed and storing the corresponding output and logs [15]. Spark is able to perform most of the computation in-memory, thus promoting better performance for data-intensive applications when compared to Hadoop's MapReduce.

The MLlib library [16] enables Spark users to build end-to-end ML workflows. These workflows are divided into 5 stages (Figure 1). The first stage goes from the collection of data to its treatment. In the second stage, data is split into train and test datasets, and a given ML algorithm is chosen. The third stage is the training stage, where data is iterated to deliver an optimized trained model at the fourth stage. In the fifth stage, the trained model can then be saved (persisted) and loaded (accessed) for inference purposes.

2.2 Intel Software Guard Extensions

Intel SGX provides a set of new instructions, available on Intel processors, that applications can use to create trusted memory regions. These regions (enclaves) are isolated from any other code on the host system, preventing other processes, including those with higher privilege levels (such as the host OS, hypervisor, and BIOS), from accessing their content [4], [17].

Since SGX protects code and data from privileged access, sensitive plaintext data can be processed at the enclave without compromising its privacy. Thus, TEEs outperform typical traditional cryptographic computational techniques (*e.g.*, searchable encryption, homomorphic encryption) [17]. Even though the second generation of SGX has improved

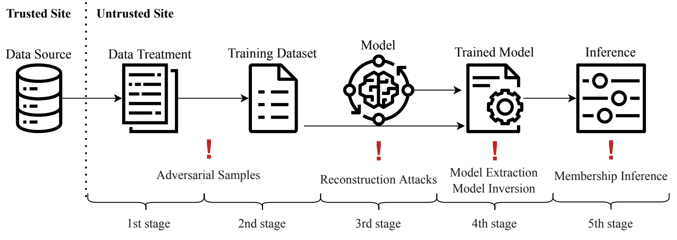


Fig. 1: ML pipeline and known attack vectors.

the size of the protected memory region, it still defines the Enclave Page Cache (EPC) to 128MB per CPU [18]. When such limitation is met, memory swapping occurs, which is a performance-costing mechanism [7]. Thus, SGX-based solutions must balance the number of I/O operations and the amount of data handled by enclaves as well as the Trusted Computing Base (TCB) to optimize performance.

We chose SGX over other TEEs in this paper because of its broad availability and use in academia [8], [9] and industry [19].

3 Applications and Security Model

3.1 SOTERIA Threat Model

SOTERIA enables the secure outsourcing of ML training and inference workloads. These are scenarios where the data owner holds sensitive information (a private dataset and/or model) and wants to perform some ML workload on it using an external cloud provider.

Our deployment model is depicted in Figure 2 and is as follows. The client (data owner) will be trusted and will provide input for ML tasks Then, a Spark Master node and N Worker nodes will be deployed in an untrusted environment (cloud provider), equipped with Intel SGX technology. Externally, we also consider a distributed data storage backend. The protocol assumes an implicit setup where the client stores its input data securely within this backend, which is also considered untrusted throughout the protocol execution.

We consider semi-honest adversaries, which means that security is defined according to a threat that attempts to break the confidentiality of data and model, but that will not actively deviate from the protocol specification. This is a good fit for cloud-based systems, where data breaches are common and malicious entities can read internal processing data temporarily [20]. In brief, our security goal is to allow clients to provide input data for training and inference in a way that is not vulnerable to breaches in confidentiality.

3.2 ML Workflow Attacks

Throughout the paper, we follow the black-box setting of [21]. Essentially, when we state that an adversary has black-box access to a model, it can query any input x and receive the predicted class probabilities $P(y|x)$ for all classes y. This allows the adversary to interact with the trained model without retrieving additional information, e.g. computing the gradients. Ensuring security against attacks on this pipeline entails including countermeasures against a wide array of attack vectors, as depicted in Figure 1.

TABLE 1: Comparison between state-of-the-art solutions and SOTERIA regarding the safety against ML attacks.

Attacks		Systems				
		[5]	[6]	[13]	[12]*	SOTERIA
Adversarial	Gradient-based	✗	✗	✓	✗	✓
	Score-based	✗	✗	✓	✗	✓
	Transfer-based	✗	✗	✓	✗	✓
	Decision-based	✗	✗	✓	✗	✓
Model Extraction	Equation-solving	✓	✓	✗	✓	✓
	Path-finding	✓	✓	✗	✗	✓
	Class-only	✓	✓	✗	✗	✓
	DFKD	✓	✓	✗	✓	✓
Model Inversion		✓	✓	?	✓	✓
Reconstruction Attacks		✓	✓	✓	✓	✓
Membership Inference		✗	✗	✗	✗	✓

*Data encryption is not provided on the open-source version.
✓ - Protected; ✗ - Non-protected; ? - Not disclosed.

Adversarial attacks. These attacks are characterized by the injection of malicious data samples, to manipulate the model and to disclose information about the original data being used for training or inference purposes. Successful attacks in the literature require the attacker to have direct access to the training dataset (data poisoning, transfer-based, and gradient-based attacks), the model and gradients (gradient-based attacks), or the full results and class probabilities (score-based attacks) [10], [22].

Model Extraction. These attacks aim at learning a close approximation to an objective function of the trained model. This approximation is based on the exact confidence values and response labels obtained by inference. To attain the desired output, the attacker must know the dimension of the original training dataset (equation-solving attacks), the dimension of the decision trees, data features and the final confidence values (path-finding attacks), or hold real samples from the training dataset (class-only attacks and data-free knowledge distillation (DFKD)) [1], [23].

Model Inversion and Membership Inference. These attacks target the recovery of values from the training dataset. Both consider an adversary that queries the ML system in a black-box fashion and both are currently based on ML services, which define publicly their trained models and the confidence values. In model inversion, the adversary must have partial knowledge of the training dataset's features to infer and query the model with specific queries [2]. Membership inference aims to test if a specific data point d was used as training data and requires the adversary to know a subset of samples used for training the model (that does not contain d) [24].

Reconstruction attacks. The goal of this attack is similar to that of membership inference, but instead of testing for the existence of a specific data point, the adversary intends to reconstruct raw data used for training the model. To be successful, some attacks require the adversary to have model-specific information, namely feature vectors (*e.g.*, Support Vector Machines or K-Nearest Neighbor) [25], others only require black-box access to the model [26].

Summary. Unlike previous works [5], [6], [12], [13], which typically consider a small subset of ML attacks, our proposal aims at providing mechanisms that cover the full range of the above-mentioned exploits. Table 1 presents relevant state-of-the-art solutions, the security attacks covered by these, and the attacks addressed by SOTERIA. Intuitively, the resilience of our system is the result of combining several mechanisms, which are only partially ensured by other systems: i.) authenticity verification of inputs excludes injections necessary for *adversarial attacks*; ii.) isolation guarantees of our protocol ensure that malicious workers gather no additional information other than statistical data, an essential aspect for preventing most attacks, and iii.) query input via secure channel prevents the adversary from performing arbitrary queries to our system, which is also a central requirement for *model inversion* or *reconstruction attacks*. This is analysed in detail in Section 4.5.

TEE-related security issues such as *side-channel* and *memory access pattern* attacks are considered orthogonal and complementary to our design goals. Indeed, mechanisms such as ObliviousRAM [27] can be layered over Soteria to address these, at the cost of additional performance overhead.

4 SOTERIA

SOTERIA is a privacy-preserving ML solution that avoids changing Apache Spark's main architecture and processing flow while retaining its usability, scalability, and fault tolerance properties.

4.1 Apache Spark: Architecture and Flow

As depicted in Figure 2, Apache Spark's operational flow is as follows. Before submitting ML tasks (*e.g.*, model training, and/or inference operations) to the Spark cluster, users must load their local datasets and models to a distributed storage backend. Users can then submit ML processing tasks, specified as ML task scripts, to the Spark client, which is responsible for forwarding these scripts to the Master node. At the Master node, tasks are forwarded to the Spark Driver, which generates a Spark Context that then distributes the tasks to a set of Worker nodes.

As Workers may be executing different steps of a given task, they need to be able to transfer information (*e.g.*, model parameters) among each other through the network. After finishing the desired computational steps, Workers send back their outputs to the Master node, which merges the outputs and replies back to the client.

Similar to the regular flow of Apache Spark, SOTERIA can be divided into two main environments or sides: the SOTERIA Client, trusted side, and the SOTERIA Cluster, untrusted side, (*e.g.*, cloud environment). Next, we describe the main modifications required by SOTERIA to the original Apache Spark's design, depicted in Figure 2 by the white dashed and solid line boxes.

4.2 SOTERIA Client

SOTERIA's client module is used by users for three main operations: i) loading data into the distributed storage backend, ii) sending ML training tasks to the Spark cluster, and iii) sending ML inference tasks to the Spark cluster. SOTERIA does not change the way users typically specify and perform the previous operations. The only exception is that users need to provide additional information in a *Manifest* configuration file, as described next.

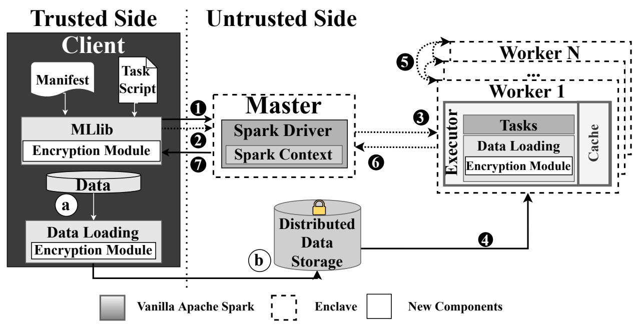


Fig. 2: SOTERIA architecture and operations flow.

Data Loading. For the first operation, the user must specify the data to be loaded to the storage backend. However, such data has to be encrypted before leaving the trusted user premises. This step is done by extending Spark's data loading component with a transparent encryption module (Figure 2-ⓐ), This module encrypts the data being loaded into the distributed storage backend with a symmetric-key encryption scheme (Figure 2-ⓑ).

Tasks submission. ML training and inference operations include two main files: the ML task script and the *Manifest* file. The transparent encryption module, also integrated within MLlib, is used to encrypt the ML task script (Figure 2-❶), which contains sensitive arguments (*i.e,* model parameters) and the ML's workload processing logic, and to decrypt the outputs (*e.g.*, trained model or inference result) returned by Spark's Master node to the client.

The *Manifest* file contains the libraries to be used by the ML task script, as well as the path at the storage backend where the training or inference data, for that specific task, is kept (Figure 2-❷). Briefly, and as explained in the next sections, this file ensures that different Spark components can attest the integrity of libraries and data being used/read by them and, moreover, cannot access other libraries or data that these are not supposed to.

The encryption module is in charge of securely exchanging the *Manifest file*, and the user's symmetric encryption key with the SGX enclave on the Master node (Figure 2-❶❷). This is done once, at the ML task's bootstrapping phase, and requires establishing a secure channel between the client and Master's enclave. This channel guarantees the security and integrity of the user's encryption key and the *Manifest file*, while the encrypted ML task scripts can be safely sent via an unprotected channel.

With the previous design, sensitive data is only accessed in its plaintext format at trusted user premises or inside trusted enclaves. This includes users' encryption keys, the information in the *Manifest file* and ML task scripts, as well as the final output.

4.3 SOTERIA Cluster

Training and inference ML task scripts are sent encrypted to Spark's Master node to avoid revealing sensitive information. However, the node requires access to the plaintext information contained in these cryptograms to distribute the required computational load across Workers. So, the Spark Driver and Context modules must be deployed in a secure SGX enclave where the cryptograms can be decrypted and the plaintext information can be securely accessed. The cryptograms, however, can only be decrypted if the secure enclave has access to the user's encryption key, thus explaining why the key must be sent through a secure channel established between the client module and the enclave.

For inference operations, the Master node also needs to access the distributed storage backend to retrieve the stored ML model. The user's encryption key is necessary so that the encrypted model is only decrypted and processed at the secure enclave. The *Manifest* file ensures that only the storage locations specified in the file are accessible to the Master Node (Figure 2-❷).

After processing the ML task scripts, the Master's enclave establishes secure channels with the enclaves of a set of Workers to send the necessary computational instructions[2] along with the user's encryption key and *Manifest file* (Figure 2-❸). The user's encryption key is needed at the Worker nodes so that these can read encrypted data (*e.g.*, train dataset or data to be inferred) from the storage backend while decrypting and processing it in a secure enclave environment (Figure 2-❹). The *Manifest file* is used, once again, to prevent unwanted access to stored data. Furthermore, the enclaves at the worker nodes establish secure channels between themselves to transfer sensitive metadata information such as model training parameters (Figure 2-❺).

Finally, after completing the desired computational tasks, the Workers send the corresponding inference or training outputs to the Master node, through the established secure channel (Figure 2-❻). The Master node then merges the partial outputs into the final result and sends it encrypted, with the user's encryption key, to the trusted client module (Figure 2-❼). At the latter, the result (*i.e.,* trained model or inference output) is decrypted by the transparent encryption module and returned to the user in plaintext.

4.4 SOTERIA Design

SOTERIA proposes a novel partitioning scheme, SOTERIA-P, that does fine-grained partitioning of which operations execute inside and outside secure enclaves. Note that this partitioning is only done for ML operations executed at Spark Worker nodes. The remaining operations done at other Spark components (*i.e.*, Master) are always executed inside trusted enclaves.

To better understand the novelty of our partitioning scheme, we first introduce a common state-of-the-art approach, SOTERIA-B, which is also supported by our system and is used in this paper as a security and performance baseline.

SOTERIA Baseline (SOTERIA-B). In SOTERIA-B, all computation done by Spark Workers is included in a trusted environment. The executor processes launched by each Worker node are deployed inside an enclave, as depicted in Figure 3. Outside the enclave, data is always encrypted in an authenticated fashion, which allows the Worker to decrypt and validate data integrity within the enclave.

SOTERIA Partitioning Scheme (SOTERIA-P). Our novel scheme is based on the observation that ML workloads

2. The same metadata sent by a vanilla Spark deployment so that Workers know the computational operations to perform.

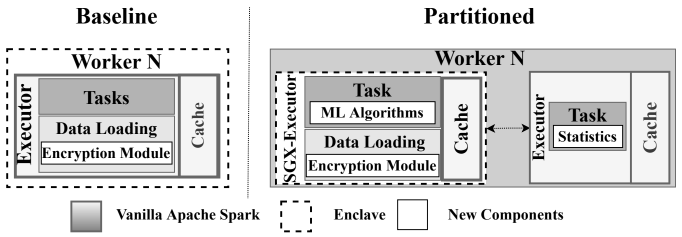


Fig. 3: Comparison between SOTERIA-B and SOTERIA-P schemes.

are composed of different computational steps. Some must operate directly over sensitive plaintext information (*e.g.*, train and inference data and model), while others do not require access to this type of data and are just calculating and collecting general statistics about the operations being made. For instance, in a multiclass ML task, where the user may want to predict multiple classes, the evaluation of such an algorithm would need to measure the precision and the probability of each individual class. These measurements can be performed independently of other operations over sensitive information.

Therefore, SOTERIA-P decouples statistical processing, used for assessing the performance of inference and training tasks, from the actual computation of the ML algorithms done over sensitive plaintext information. This decoupling builds directly upon MLlib and refactors its implementation without requiring any changes to the way users submit ML tasks. As depicted in Figure 3, statistical processing is done by executor processes in the untrusted environment, while the remaining processing endeavors are done by another set of executors inside a trusted enclave.

This decoupled scheme leads SOTERIA-P to reveal the following statistical information during the execution of ML workloads: the calculation of confidence results (accuracy, precision, recall and F1-scores), table summaries and ROC/AUC curves, and probability distributions for classes.

4.5 Security

Formally, our security goal is defined using the real-versus-ideal world paradigm, similarly to the Universal Composability [28] framework. Succinctly, we prove that SOTERIA is indistinguishable from an idealized service for running ML scripts in an arbitrary external environment that can collude with a malicious insider adversary. We then use that abstraction to demonstrate how SOTERIA is resilient to real-world ML attacks. This idealized service is specified as a functionality parametrized with the input data, which simply executes the tasks described in the ML task script, and returns the output to the client via a secure channel.

The full proof of SOTERIA can be found in Appendix A. The outline is as follows. The role played by SOTERIA 's Master node can be seen as an extension of the client, establishing secure channels, providing storage encryption keys, and receiving outputs. We follow the reasoning of [29] and replace the Master node with a reactive functionality performing the same tasks. Similarly, each SOTERIA Worker behaves simultaneously as a processing node and as a client node, providing inputs to the computation of other Workers (e.g., model training parameters). This enables us to do a hybrid argument, where Worker nodes are sequentially replaced by idealized reactive functionalities executing their roles in the task script.

Finally, all processing is done in ideal functionalities, and all access to external storage is fixed by the ML task script and the Manifest file, so we can refactor the functionalities to process over hard-coded client data, and replace the secure data storage with dummy encryptions. We have now reached the ideal world, where all ML computation is done in an isolated service, and all other protocol interactions are simulated given the ML task script and Manifest files. Our analysis refers to SOTERIA-B, and thus establishes the baseline security result when no computation is done outside the enclave (no leakage). The reasoning for SOTERIA-P is identical, with the caveat that statistical data is explicitly revealed as leakage in the ideal world.

4.5.1 Security implications of statistical leakage

To show that our system is resilient against ML attacks, we must consider a common prerequisite for such attacks to be successful: the adversary must have black-box access to the model (Section 3.2). Our result implies that adversaries cannot infer internal data from the workers, and the secure channel between client and Master prevents adversaries from injecting queries into the system. This would intuitively suggest that our adversary is unable to perform queries in a black-box fashion to the model, however, SOTERIA-P has the aforementioned additional leakage of statistical information.

As such, a crucial security question to answer is: *how does statistical information relate to black-box model access, i.e. does the first imply the second in any way?* Extracting model access from statistical data is an ongoing area of research. However, current attacks suggest one is unable to do this in any successful way [11]. This supports our thesis that statistical values *are not sensitive information*, in the sense that their leakage does not expose our system to these types of attacks. It follows that SOTERIA-P *scheme is resilient to any attack that requires black-box access to the model to succeed*.

4.5.2 Relation to ML Attacks

We now overview the four types of attacks referred to in Section 3.2 on a case-by-case basis. Appendix B contains a more in-depth analysis of these attacks.

Resistance against input forgery is achieved by SOTERIA through authenticated data encryption. This means that the input dataset is authenticated by the data owner and explicitly defined in the *Manifest file*, allowing enclave Worker nodes to check the authenticity of all input data. Thus, no forged data is accepted for processing, which is necessary for performing any type of *adversarial attack*.

The secure channels between the TEE at the Master node and the client ensure that an external adversary cannot observe legitimate query input/outputs, and cannot submit arbitrary queries to SOTERIA. This query privacy feature is crucial to block illegitimate model access, which allows us to protect against *model extraction*, *model inversion*, *membership inference* as well as instances of *reconstruction attacks* that require black-box access to the model.

Finally, *reconstruction attacks* require additional knowledge about internal ML model data. Our security result shows that SOTERIA is indistinguishable from an idealized ML service, which does not reveal the trained model. This includes the important feature vectors required for this attack to occur, which cannot be inferred from confidence values and class probabilities alone. Alternatively, *reconstruction attacks* requiring black-box access to the model are strictly stronger, but this, as we have argued, is not possible only with knowledge of confidence values, class probabilities, ROC/AUC curves, and table summaries (the explicit leakage of SOTERIA-P) (refer to Appendix A and B).

4.6 Implementation

SOTERIA's prototype is built on top of Apache Spark 2.3.0 and implemented using both Java and Scala. Spark's data loading library was extended to include SOTERIA's transparent encryption module. The latter uses the AES-GCM-128 authenticated encryption cypher mode, which provides both data privacy and integrity guarantees.

Both SOTERIA-B and SOTERIA-P schemes are supported by our prototype. For SOTERIA-P's implementation, Spark's MLlib implementation was decoupled into two sub-libraries, one with the statistical processing (to be executed outside SGX), and another with the remaining ML computational logic (to be executed inside SGX).

Graphene-SGX 1.0 was used for the overall management of Intel SGX enclaves' life cycle, for specifying the computation (*i.e.*, internal Spark and MLlib libraries) to run at each enclave, and for establishing secure channels (*i.e.*, with the TLS-PSK protocol) between the enclaves at the Master and Worker nodes [30]. SOTERIA's *Manifest* file was also provided by Graphene.

5 METHODOLOGY

Environment. The experiments use a cluster with eight servers, with a 6-core 3.00 GHz Intel Core i5-9500 CPU, 16 GB RAM, and a 256GB NVMe. The host OS is Ubuntu 18.04.4 LTS, with Linux kernel 4.15.0. Each machine uses a 10Gbps Ethernet card connected to a dedicated local network. We use Apache Spark 2.3.0 and version 2.6 of the Intel SGX Linux SDK (driver 1.8). The client and Spark Master run in one server while Spark Workers are deployed in the remaining seven servers. SGX memory is configured to use 4GB.

Workloads. We resort to the HiBench benchmark [14] for evaluating four ML algorithms (Table 3), that are broadly used and natively implemented on top of MLlib, namely: Alternating Least Squares (ALS), Principal Component Analysis (PCA), Gradient Boosted Trees (GBT) and Linear Regression (LR). For each algorithm, the benchmark suite offers different workload sizes ranging from *Tiny* to *Gigantic* configurations.

Setups and metrics. To validate SOTERIA's performance, and the benefits of fine-grained differentiation of secure ML operations, we compare the implementations of our system with the SOTERIA-B and SOTERIA-P schemes. These setups are compared with a deployment of Apache Spark that does not offer privacy guarantees (Vanilla).

Moreover, we test SGX-Spark [12], a state-of-the-art SGX-based solution that protects both analytical and ML computation done with Apache Spark. It is designed to process sensitive information inside SGX enclaves, so it can be considered the most similar to SOTERIA. However, SGX-Spark can only guarantee that *User Defined Functions (UDFs)* are processed in secure enclaves. This decision leaves a large codebase of Spark outside the protected memory region and, consequently, limits the users to only being able to execute privacy-preserving ML algorithms based on UDFs.

TABLE 2: Representation of the tasks of each ML algorithm and the data sizes for different workloads.

Algorithms	Tasks	Workloads			
		Tiny	Large	Huge	Gigantic
ALS	RS	193KB	345MB	2GB	4GB
PCA	DR	256KB	92MB	550MB	688MB
GBT	P	36KB	46MB	92MB	183MB
LR	C + P	11GB	134GB	335GB	894GB

RS: Recommendation Systems; DR: Dimensionality Reduction; P: Prediction; C: Classification.

For each experiment discussed in the next section, we include the average algorithm execution time and standard deviation for 3 independent runs. The *dstat* monitoring tool was used to collect the CPU, RAM, and network consumption at each cluster node.

6 EVALUATION

Our evaluation answers three main questions: *i) How does* SOTERIA *impacts the execution time of ML workloads? ii) How does the* SOTERIA-P *scheme compares, in terms of performance, with state-of-the-art approaches* (i.e., SOTERIA-B *and SGX-Spark)? iii) Can* SOTERIA *efficiently handle different algorithms and dataset sizes?*

6.1 Methodology

Environment. The experiments use a cluster with eight servers, with a 6-core 3.00 GHz Intel Core i5-9500 CPU, 16 GB RAM, and a 256GB NVMe. The host OS is Ubuntu 18.04.4 LTS, with Linux kernel 4.15.0. Each machine uses a 10Gbps Ethernet card connected to a dedicated local network. We use Apache Spark 2.3.0 and version 2.6 of the Intel SGX Linux SDK (driver 1.8). The client and Spark Master run in one server while Spark Workers are deployed in the remaining seven servers. SGX memory is configured to use 4GB.

Workloads. We resort to the HiBench benchmark [14] for evaluating four ML algorithms (Table 3), that are broadly used and natively implemented on top of MLlib, namely: Alternating Least Squares (ALS), Principal Component Analysis (PCA), Gradient Boosted Trees (GBT) and Linear Regression (LR). For each algorithm, the benchmark suite offers different workload sizes ranging from *Tiny* to *Gigantic* configurations.

Setups and metrics. To validate SOTERIA's performance, and the benefits of fine-grained differentiation of secure ML operations, we compare the implementations of our system with the SOTERIA-B and SOTERIA-P schemes. These setups are compared with a deployment of Apache Spark that does not offer privacy guarantees (Vanilla).

Moreover, we test SGX-Spark [12], a state-of-the-art SGX-based solution that protects both analytical and ML computation done with Apache Spark. It is designed to process

TABLE 3: Representation of the tasks of each ML algorithm and the data sizes for different workloads.

Algorithms	Tasks	Workloads			
		Tiny	Large	Huge	Gigantic
ALS	RS	193KB	345MB	2GB	4GB
PCA	DR	256KB	92MB	550MB	688MB
GBT	P	36KB	46MB	92MB	183MB
LR	C + P	11GB	134GB	335GB	894GB

RS: Recommendation Systems; DR: Dimensionality Reduction; P: Prediction; C: Classification.

sensitive information inside SGX enclaves, so it can be considered the most similar to SOTERIA. However, SGX-Spark can only guarantee that *User Defined Functions (UDFs)* are processed in secure enclaves. This decision leaves a large codebase of Spark outside the protected memory region and, consequently, limits the users to only being able to execute privacy-preserving ML algorithms based on UDFs.

For each experiment discussed in the next section, we include the average algorithm execution time and standard deviation for 3 independent runs. The *dstat* monitoring tool was used to collect the CPU, RAM, and network consumption at each cluster node.

6.2 Performance Overview

Figures 4a, 4b, 4c and 4d present the performance evaluation for *PCA*, *GBT*, *ALS* and *LR* algorithms for different workload sizes. Next, we list our main observations to aid in the characterization of these results. Unless stated otherwise, the performance overhead values discussed in this section correspond to the number of times that the algorithm's execution time increases for a given setup when compared to the Vanilla Spark deployment results. *Obs. 1* to *5* correspond to the *Huge* workload for the defined algorithms, whilst *Obs. 6* to *9* refer to the overall results in Figure 4.

Observation 1. Vanilla Spark's execution times for ALS, PCA, LR, and GBT algorithms are, respectively, 55, 655, 657, and 189 seconds.

Observation 2. The execution time for ALS increases by 3.62x and 4.35x for SOTERIA-P and SOTERIA-B, respectively. SGX-Spark incurs an execution overhead of 4x. Thus, the three setups have similar results while requiring approximately 150 seconds more processing time than the vanilla deployment. Nevertheless, SOTERIA-P performs slightly better than the other two approaches.

Observation 3. For PCA, SOTERIA-B and SOTERIA-P have an execution overhead of 3.67x and 2.85x, while SGX-Spark increases the computational time by 3.95x. When compared to SGX-Spark, SOTERIA-P decreases the execution time by 12 minutes (27.8%).

Observation 4. For LR, SOTERIA-B and SGX-Spark exhibit an overhead of 27.31x, while SOTERIA-P reduces this value to 18.2x. This reduction of 29.6% allows SOTERIA-P to complete this workload 1.4 hours earlier.

Observation 5. With the GBT algorithm, SOTERIA-B shows similar execution times when compared to SGX-Spark, with a 7.04x and 6.64x increase, respectively. SOTERIA-P outperforms both approaches, with an overhead of 4.79x, 27.8% less than SGX-Spark.

Observation 6. For *Tiny* and *Large* workloads with the PCA algorithm, SOTERIA performs similarly for our two schemes, while outperforming SGX-Spark. With larger workload sizes, the overhead imposed by our solutions increases, however, it continues to show better performance than SGX-Spark. SOTERIA-B has an overhead of 1.96x to 5.15x for *Tiny* and *Gigantic* workloads, whilst SOTERIA-P incurs an overhead of 1.72x to 3.79x. When compared with SGX-Spark, the results show an absolute difference of 4 seconds and 7 minutes (7%), for SOTERIA-B, and 7 seconds and 33 minutes (19% and 31%) respectively, for SOTERIA-P.

Observation 7. Regarding the GBT algorithm, and the *Tiny* workload, the overhead of SOTERIA-B, SOTERIA-P, and SGX-Spark are similar. However, the difference between the three approaches is more visible when increasing the workload size. SOTERIA-P (*Tiny-2.13x and Gigantic-5.88x*) outperforms both approaches, while SOTERIA-B (*Tiny-2.18x, Gigantic-9.35x*) and SGX-Spark (*Tiny-2.3x, Gigantic-10.34x*) have similar results. SOTERIA-P is able to surpass SGX-Spark's execution time in the *Gigantic* workload by up to 41%.

Observation 8. With ALS, SOTERIA-P shows an execution time overhead of 2.04x and 3.28x, for the *Tiny* and *Gigantic* workloads, respectively. SOTERIA-P achieves lower overhead than SOTERIA-B and SGX-Spark for all dataset sizes, with the execution time decreasing by 8 seconds (9%) for the *Tiny* and 191 seconds (27%) for the *Gigantic* workloads.

Observation 9. For LR, with the *Tiny* workload, SOTERIA-B and SOTERIA-P increase execution time by 14.39x and 12.95x, respectively. As for the *Gigantic* workload, SOTERIA-B incurs an overhead of 30.04x and SOTERIA-P of 23.89x. Compared to SGX-Spark, our SOTERIA-P decreases the execution time by 43 seconds for the *Tiny* workload and by 4.31 hours for the *Gigantic* workload (22.6%).

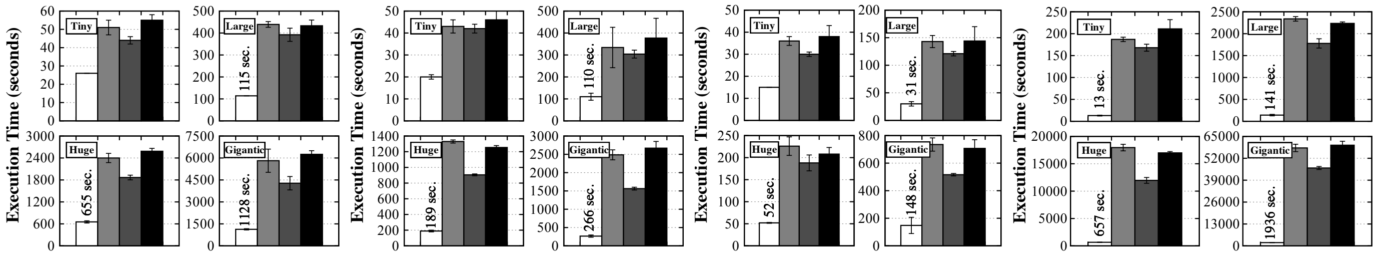
Observation 10. Overall, the CPU, RAM, and network usage for both SOTERIA schemes is similar to the vanilla Spark baseline. In more detail, SOTERIA-B with LR presents the upper-bound limit for both memory and CPU, showing an increase of 9% in both when compared with vanilla Spark (20%). Whilst the network shows an upper-bound increase of 10% (vanilla Spark shows an upper-bound network of 135MB) in SOTERIA-B with PCA due to extra encrypted data paddings being sent between Spark Workers.

Observation 11. SOTERIA does not impact the accuracy of ML workloads. For all experiments, we measured the corresponding accuracy metrics (*e.g.*, accuracy, root mean square error, or ROC). The results corroborate that both SOTERIA-B and SOTERIA-P show accuracy values similar to the vanilla Spark version.

6.3 Analysis

We analyze the results based on *i)* dataset size; and *ii)* size of trusted computing base (TCB).

Dataset size. For PCA, GBT, and ALS with smaller datasets, SOTERIA-B and SOTERIA-P perform similarly (Figure 4). However, as the size of the datasets increases, more operations and data must be transferred to the SGX enclave, thus taking a more noticeable toll on the overall performance. The page swapping mechanism of SGX, which occurs due to its memory limitations, incurs a significant performance penalty [7], [18]. For example, when compared to the vanilla



(a) PCA (b) GBT (c) ALS (d) LR

Fig. 4: Runtime execution for PCA, GBT, ALS and Linear Regression for *Tiny*, *Large*, *Huge* and *Gigantic* workloads. The legend is as follows: □ Vanilla Spark; ▒ Soteria-B; ▓ Soteria-P; ■ SGX-Spark.

setup, the PCA algorithm overhead for Soteria-B varies between 1.96x for *Tiny* workload and 5.15x for *Gigantic* workload. While for Soteria-P, the execution time increases 1.78x in the *Tiny* workload and 3.79x in the *Gigantic* workload.

Soteria-P is the setup that scales better as the amount of data to be processed grows. Indeed, as seen in *Obs. 6-9*, it is able to reduce execution time from 9% up to 31% when compared to SGX-Spark.

Size of TCB. SGX-Spark outperforms Soteria-B for some of the evaluated algorithms (*Obs. 2, 4, and 5*). As SGX-Spark only protects UDFs, the performance overhead imposed by the larger TCB of Soteria-B is higher. Nevertheless, when compared to SGX-Spark, Soteria-B covers a wider range of ML attacks, while keeping performance overhead below 1.59x. Indeed, for algorithms such as PCA, Soteria-B has similar or slightly inferior execution times (*Obs. 3*) which is due to both setups performing similar computations at the enclaves while the UDF mechanism is not fully optimized.

On the other hand, Soteria-P always outperforms SGX-Spark and Soteria-B (*Obs. 2-5*). This is due to the TCB reduction present in our novel partitioning scheme. The results show that this solution can reduce the training time by up to 30%, namely for the LR algorithm with the *Huge* workload (*Obs. 4*).

Discussion. The results show that Soteria-P outperforms other state-of-the-art approaches, namely SGX-Spark, for all the considered ML algorithms. Also, Soteria-P achieves better performance than the Soteria-B setup, while offering similar security guarantees when considering distinct ML attacks (Section 4.5). This is made possible by filtering key operations to be done outside enclaves.

In detail, when compared to Soteria-B, Soteria-P reduces ML workloads' execution time by up to 37%. When compared with SGX-Spark, the execution time is reduced by up to 41%. Interestingly, for the LR algorithm using a *Gigantic* workload (*894GB*), Soteria-P decreases computation time by 4.3 hours and 3.3 hours, when compared with SGX-Spark and Soteria-B, respectively. The performance overhead of Soteria-P for the four different algorithms ranges from 1.7x to 23.8x when compared to Vanilla Spark.

7 Related Work

Privacy-preserving ML with TEEs. Chiron [5] enables training ML models on a cloud service without revealing information about the training dataset. *Myelin* [6] offers a similar solution to *Chiron* while adding differential privacy and data oblivious protocols to the algorithms to mitigate the exploits from *side-channels* and the information leaked by the model parameters. Soteria differs from these works as it is able to cover both the training and inference phases while providing additional protection against *adversarial samples*, *reconstruction*, and *membership inference* attacks (Table 1).
In [17], five ML algorithms are re-implemented with data oblivious protocols. These protocols combined with TEEs ensure strong privacy guarantees while preventing the exploitation of *side-channel* attacks that observe memory, disk, and network access patterns to infer private information. Unlike this solution, Soteria aims at transparently supporting all ML algorithms built with MLlib.

Privacy-preserving analytics with TEEs. TEEs have also been used to ensure privacy-preserving computation for general-purpose analytical frameworks [13]. In comparison to SGX-Spark [12], detailed in Section 6.1, Soteria supports a broader set of algorithms (*i.e.*, any algorithm that can be built with the MLlib API), while protecting users from a more complete set of ML attacks (Table 1).
Opaque [8] and Uranus [9] resort to SGX to provide secure general-purpose analytical operations, while only supporting a restricted set of ML algorithms. Opaque combines SGX with oblivious protocols and requires the re-implementation of default Apache Spark UDF operators. Uranus is also based on porting UDF processing to SGX enclaves but includes a single ML workload. Differently, Soteria is targeted at ML workloads and is not limited by UDF-based algorithms that, when compared with MLlib-based ones, exhibit lower performance for some ML workloads [31]. Therefore, the design, implementation, and security requirements to be considered are distinct when comparing with Soteria.

8 Conclusion

We propose Soteria, a system for distributed privacy-preserving ML. Our solution builds upon the combination of Apache Spark and TEEs to protect sensitive information being processed at third-party infrastructures during the ML training and inference phases.

The innovation of Soteria stems from a novel partitioning scheme (Soteria-P) that allows specific ML operations to be deployed outside trusted enclaves. Namely, we show that it is possible to offload non-sensitive operations (*i.e.*, statistical calculations) from enclaves, while still covering a larger spectrum of black-box ML attacks than in previous related work. Also, this decision enables Soteria to perform better than existing solutions, such as SGX-Spark, while reducing ML workloads execution time by up to 41%.

Acknowledgment

This work was supported by the Portuguese Foundation for Science and Technology through a PhD Fellowship (SFRH/-BD/146528/2019 – Cláudia Brito) and the project AIDA - Adaptive, Intelligent and Distributed Assurance Platform (reference POCI-01-0247-FEDER-045907 - João Paulo), co-financed by the ERDF - European Regional Development Fund through the Operacional Program for Competitiveness and Internationalisation - COMPETE 2020 and by the Portuguese Foundation for Science and Technology - FCT under CMU Portugal. Also, it was also funded by the European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project nº 047264; Funding Reference: POCI-01-0247-FEDER-047264 - Bernardo Portela].

References

[1] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and other. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium*, 2016.

[2] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[3] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. 2017.

[4] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, et al. Innovative instructions and software model for isolated execution. 2013.

[5] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, et al. Chiron: Privacy-preserving machine learning as a service. 2018.

[6] Nick Hynes, Raymond Cheng, and Dawn Song. Efficient deep learning on multi-source private data. 2018.

[7] Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, et al. Everything you should know about intel sgx performance on virtualized systems. 2019.

[8] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, et al. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017.

[9] Jianyu Jiang, Xusheng Chen, TszOn Li, Cheng Wang, et al. Uranus: Simple, efficient sgx programming and its applications. In *15th ACM Asia Conference on Computer and Communications Security*, 2020.

[10] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *6th International Conference on Learning Representations,*, 2018.

[11] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, et al. Exploring connections between active learning and model extraction. In *29th USENIX Security Symposium*, 2020.

[12] Large-Scale Data & Systems (LSDS) Group. Sgx-spark. https://github.com/lsds/sgx-spark. (Accessed on 22/10/2022).

[13] Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, and Latifur Khan. Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors. In *ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[14] Intel. Hibench is a big data benchmark suite. https://github.com/Intel-bigdata/HiBench. (Accessed on 22/10/2022).

[15] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, et al. Apache spark: a unified engine for big data processing. 2016.

[16] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, et al. Mllib: Machine learning in apache spark. 2016.

[17] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, et al. Oblivious multi-party machine learning on trusted processors. In *25th USENIX Security Symposium*, 2016.

[18] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, et al. Benchmarking the second generation of intel sgx hardware. In *Data Management on New Hardware*. 2022.

[19] Microsoft Azure. Azure confidential computing. https://azure.microsoft.com/en-us/solutions/confidential-compute/. (Accessed on 22/10/2022).

[20] Salman Iqbal, Miss Laiha Mat Kiah, Babak Dhaghighi, Muzammil Hussain, Suleman Khan, Muhammad Khurram Khan, and Kim-Kwang Raymond Choo. On cloud security attacks: A taxonomy and intrusion detection and prevention as a service. 2016.

[21] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *10th ACM workshop on artificial intelligence and security*, 2017.

[22] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *5th International Conference on Learning Representations*, 2017.

[23] Jean-Baptiste Truong, Pratyush Maini, Robert J Walls, and Nicolas Papernot. Data-free model extraction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[24] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Symposium on Security and Privacy (SP)*, 2017.

[25] Mohammad Al-Rubaie and J Morris Chang. Privacy-preserving machine learning: Threats and solutions. IEEE, 2019.

[26] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and other. Updates-leak: Data set inference and reconstruction attacks in online learning. In *29th USENIX Security Symposium*, 2020.

[27] Emil Stefanov, Marten Van Dijk, Elaine Shi, T-H Hubert Chan, et al. Path oram: an extremely simple oblivious ram protocol. 2018.

[28] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE Symposium on Foundations of Computer Science*, 2001.

[29] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, et al. Secure multiparty computation from sgx. In *International Conference on Financial Cryptography and Data Security*. Springer, 2017.

[30] Chia-Che Tsai, Donald E Porter, and Mona Vij. Graphene-sgx: A practical library os for unmodified applications on sgx. In *USENIX Annual Technical Conference*, 2017.

[31] Databricks. Optimizing apache spark udfs. https://www.databricks.com/session_eu20/optimizing-apache-spark-udfs. (Accessed on 27/10/2022).

[32] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure multiparty computation from sgx. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security*, pages 477–497, Cham, 2017. Springer International Publishing.

Appendix A
Soteria Proof

We now discuss the privacy-preserving security of the Soteria protocol. The goal is to reduce the security of our system to the security of the underlying security mechanisms, namely the isolation guarantees of Intel SGX and the bootstrapped secure channels, and the indistinguishability properties of encryption.

The security goal consists in demonstrating that Soteria ensures privacy-preserving machine learning. Concretely, this means that the behavior displayed by Soteria in the real-world is indistinguishable from the one displayed by an idealized functionality in the ideal-world, which simply computes over the task script and provides an output via a secure channel. The only information revealed during this process is the length of I/O, the number of computation steps, and the access patterns to the external storage where data is kept

Formally, this security goal is defined using the real-versus-ideal world paradigm, similarly to the Universal Composability [28] framework.

We begin with a more formal description of our security model. Then, we present an intermediate result for ensuring the security of enclaves relying on external storage. We can finally specify the behavior of the Client, Master and Workers, and present the full proof.

A.1 Formal Security Model

Our model considers external environment $\mathcal{Z}$ and internal adversary $\mathcal{A}$. Π denotes the protocol running in the real world, and $\mathcal{S}$ and $\mathcal{F}$ denote the simulator and functionality, respectively, running in the ideal-world. The real-world considers a Client C, a Master node M and 2 Worker nodes W_1 and W_2. This is for simplicity, as the definition and proof can be easily generalized to consider any number of Worker nodes. We also consider a global storage G, which is initialized by $\mathcal{Z}$ before starting the protocol. The Ideal functionality is parametrised by this external storage $\mathcal{F}$¡G¿, and will reveal the access patterns via leakage function $\mathcal{L}$.[3]

In the real-world, $\mathcal{Z}$ begins by providing public inputs to C in the form of (s, m), where s is the task script, and m is the manifest detailing data in G to be retrieved.[4] The Client will then execute protocol Π, sending messages to M, W_1 and W_2. When the script is concluded, output is provided to C, finally being returned to $\mathcal{Z}$. $\mathcal{A}$ can observe all communication between C, M, W_1, W_2 and G.

In the ideal world, (s, m) are provided to dummy Client C, which in turn forwards them to $\mathcal{F}$¡G¿. The functionality will simply run the protocol and forward the output to C, which in turn is returned to $\mathcal{Z}$. All the communication observed by $\mathcal{A}$ must be emulated by simulator $\mathcal{S}$, which receives (s, m), leakage $\mathcal{L}$ produced from the functionality interaction with storage G, and the size of the output.

Security is predicated on ensuring that $\mathcal{S}$ does not require any sensitive information (contained in G) to emulate the communication to $\mathcal{A}$. Given that we consider a semi-honest adversary, we can simplify the interaction with the system and instead discuss equality of views, as $\mathcal{Z}$ and $\mathcal{A}$ are unable to deviate the system from its expected execution. This is captured by the following definition.

Definition 1. Let Real denote the view of $\mathcal{Z}$ in the real-world, and let Ideal denote the view of $\mathcal{Z}$ in the ideal-world. Protocol Π securely realises $\mathcal{F}$ for storage G if, for all environments $\mathcal{Z}$ and all adversaries $\mathcal{A}$,

$$\mathsf{Real}_{\mathcal{Z},\mathcal{A},\Pi}(G) \approx \mathsf{Ideal}_{\mathcal{Z},\mathcal{A},\mathcal{S},\mathcal{F}}(G)$$

A.2 Intermediate Result

For convenience, SOTERIA does not require the Client to provide input data at the time of the ML processing, and instead the Workers are given access to an external storage from which to retrieve this data. When discussing the security in the context of secure outsourced computation for SGX, this is functionally equivalent to classical scenarios where the Client provides these inputs via secure channel (Theorem 3 in [32]). The reasoning is simply that, if a protocol securely realises a functionality with a given input provided via secure channel, then the same functionality can be securely realised with the same input fixed in an external storage, securely accessed by the enclave.

```
Algorithm Setup($i, m$)<$G$>:
k ←$ $\Theta$.Gen()
c ←$ $\Theta$.Enc(k, $i$)
$G[m]$ ← c
Return ($m$, k)
```

Fig. 5: Secure external storage setup.

```
Algorithm AC()
k ←$ $\Theta$.Gen()
Return $\mathcal{S}_1$.AC()

Algorithm Send($l$)
Return $\mathcal{S}_1$.Send($l$)

Algorithm Get($l$)
$i$ ← $\{0\}^l$
c ←$ $\Theta$.Enc(k, $i$)
Return c
```

Fig. 6: Simulator for Π_2.

Consider a protocol Π_1 that securely realises some functionality $\mathcal{F}$ with simulator $\mathcal{S}_1$ according to Theorem 3 of [28]. We construct protocol Π_2 built on top of this secure protocol Π_1, where input data is pre-established and provided to the enclave via an initial Setup stage where inputs are stored in encrypted fashion (Figure 5 describes a simplified version of the process for a single entry). Inputs to Π_2 are exactly the same as those for Π_1, but instead of being transmitted via the secure channel established with Attested Computation, they are retrieved from storage using a key sent via the same channel. The Client-server communication increases by a constant (the key length), which can be trivially simulated, and the rest of the input can be simulated in a similar way using the IND-CPA properties of Θ. This protocol behavior will be key for all SOTERIA Workers. Our theorem is as follows.

Theorem 1. Let Π_1 be a protocol that securely realises functionality $\mathcal{F}$ according to Theorem 3 in [32]. Then Π_2, constructed as discussed above, securely realises $\mathcal{F}$ according to Definition 1.

PROOF. To demonstrate this result, we construct simulator $\mathcal{S}_2$ using $\mathcal{S}_1$, then argue that, given that $\mathcal{S}_1$ is a valid simulator for the view of Π_1, then the simulated view must be indistinguishable from the one of the real world of Π_2.

We begin by deconstructing $\mathcal{S}_1$ in two parts: $\mathcal{S}_1.\mathsf{AC}()$ will produce the view for the establishment of secure channel, while $\mathcal{S}_1.\mathsf{Send}(l)$ will produce a simulated view of Client inputs, given their length. In turn, our simulator will share the same functions, but also include a third $\mathcal{S}_2.\mathsf{Get}(l)$ to simulate information being retrieved from G, given its length. Our simulator is depicted in Figure 6.

The view presented to $\mathcal{A}$ is composed of three different types of messages:

- Messages exchanged during the secure channel establishment are exactly the same as in Π_1. Thus they remain indistinguishable from Π_2.

3. Reasoning for the security of SOTERIA-P instead would only require this function to also reveal statistical data to the simulator, which we consider to be non-sensitive.

4. SOTERIA Clients are trusted. As such, we assume (s, m) to both be *valid*, in the sense that they are correct ML scripts and data sets in G, and thus can be interpreted by ideal functionality $\mathcal{F}$.

- Outputs received via the secure channel follow the exact same simulation strategy than Π_1, and thus are indistinguishable from Π_2.
- Messages produced from G in Π_2 are encryption of data in $G[m]$, while the values presented by $\mathcal{S}_2$ are dummy encryptions with the same length. We can thus reduce the advantage of $\mathcal{A}$ to distinguish these views to the advantage of the same adversary to attack the IND-CPA guarantees of encryption scheme Θ, which is negligible.

As such, if $\mathcal{S}_1$ is a valid simulator for Π_1 to $\mathcal{A}$, then the view presented by $\mathcal{S}_2$ must also be indistinguishable for Π_2 to $\mathcal{A}$.

Let

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Dist}}{}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) = \\ |\Pr[\mathsf{Real}^{G}_{\mathcal{Z},\mathcal{A},\Pi_2} \Rightarrow \mathsf{T}] - \Pr[\mathsf{Ideal}^{G}_{\mathcal{Z},\mathcal{A},\mathcal{S}_2,\mathcal{F}} \Rightarrow \mathsf{T}]|\end{aligned}$$

To conclude, we have that, for negligible function μ,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_2,\mathcal{S}_2,\mathcal{F}}(G) &= \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_1,\mathcal{S}_1,\mathcal{F}}() + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\Theta,\mathcal{A}}() \\ &\leq \mu()\end{aligned}$$

and Theorem 1 follows.

A.3 Soteria Client, Master and Workers

The Soteria components follow standard methodologies for ensuring secure outsourced computation using SGX. As such, and given the complexity of ML tasks described in the script, we consider the following set of functions.

Secure channels are established with enclaves. We define $\mathsf{init}(P)$ as the bootstrapping process, establishing a channel with participant P. This produces an object that can be used to send and receive data via send and receive. Untrusted storage is not protected with secure channels, and can be accessed using the call $\mathsf{uGet}(G, m)$, which retrieves data from G considering manifest file m. Concretely, this is achieved using the open-source library Graphene-SGX, which we assume to correctly implement this mechanism. Finally, the script s defines the actual computation that must be performed by the system, and will be executed collaboratively with both Workers. As such, we define s as a stateful object with the main method $\mathsf{Run}(\mathsf{id}, i_1, i_2)$, where input id is the identifier of the Worker, i_1 is input from storage and i_2 is intermediate input (e.g. model parameters), returning (o_1, o_2), where o_1 is the (possibly) final output, and o_2 is the (optional) intermediate output for dissemination. For simplicity, we also define method Complete that returns T if the task is complete, or F otherwise.

The Soteria components can be analysed in Figure 7 and are as follows. The Client C (left of Figure 7) simply establishes the channel with M, sends the parameters (manifest file, task script and storage key), and awaits computation output. Observe that we assume that the key k has been previously initialized, and that the actual data has been previously encrypted in G using it. The Master M (middle of Figure 7) will receive the parameters from C and establish channels with W_1 and W_2, forwarding them the same parameters and awaiting computation output. When it arrives, it is forwarded to the Client.[5] Worker W_1

5. In the actual protocol, the Master has additional steps to process the output. We describe it like this for simplicity, as it does not change the proof.

```
Algorithm $C(m, s, k)$
$\mathsf{sc} \leftarrow \mathsf{init}(M)$
$\mathsf{sc.send}(m, s, k)$
$o \leftarrow \mathsf{sc.receive}()$
Return $o$

Algorithm $M()$
$\mathsf{sc}_c \leftarrow \mathsf{init}(C)$
$(m, s, k) \leftarrow \mathsf{sc}_c.\mathsf{receive}()$
$\mathsf{sc}_1 \leftarrow \mathsf{init}(W_1)$
$\mathsf{sc}_1.\mathsf{send}(m, s, k)$
$\mathsf{sc}_2 \leftarrow \mathsf{init}(W_2)$
$\mathsf{sc}_2.\mathsf{send}(m, s, k)$
$o_1 \leftarrow \mathsf{sc}_1.\mathsf{receive}()$
$o_2 \leftarrow \mathsf{sc}_2.\mathsf{receive}()$
$\mathsf{sc}_c.\mathsf{send}((o_1, o_2))$

Algorithm $W_1()$
$m \leftarrow \epsilon$
$\mathsf{sc}_m \leftarrow \mathsf{init}(M)$
$(m, s, k) \leftarrow \mathsf{sc}_m.\mathsf{receive}()$
$\mathsf{sc}_w \leftarrow \mathsf{init}(W_2)$
While $!s.$Complete:
  $c \leftarrow \mathsf{uGet}(G, m)$
  $i \leftarrow \Theta.\mathsf{Dec}(k, c)$
  $(o, m) \leftarrow\$ \; s.\mathsf{Run}(W_1, i, \epsilon)$
  $\mathsf{sc}_w.\mathsf{send}(m)$
  $m \leftarrow \mathsf{sc}_w.\mathsf{receive}()$
$\mathsf{sc}_m.\mathsf{send}(o)$
```

Fig. 7: Soteria Components. Client C (left), Master node M (middle) and Worker node 1 W (right).

```
Algorithm $W_1()$
$m \leftarrow \epsilon$
$\mathsf{sc}_m \leftarrow \mathsf{init}(M)$
$(m, s, k) \leftarrow \mathsf{sc}_m.\mathsf{receive}()$
$\mathsf{sc}_w \leftarrow \mathsf{init}(W_2)$
While $!s.$Complete:
    $c \leftarrow \mathsf{uGet}(G_1, m)$
    $i \leftarrow \Theta.\mathsf{Dec}(k, c)$
    $(o, m) \leftarrow\$ \; s.\mathsf{Run}(W_1, i, \epsilon)$
    $\mathsf{sc}_w.\mathsf{send}(m)$
    $m \leftarrow \mathsf{sc}_w.\mathsf{receive}()$
$\mathsf{sc}_m.\mathsf{send}(o)$

Algorithm $W_2()$
$m \leftarrow \epsilon$
$\mathsf{sc}_m \leftarrow \mathsf{init}(M)$
$(m, s, k) \leftarrow \mathsf{sc}_m.\mathsf{receive}()$
$\mathsf{sc}_w \leftarrow \mathsf{init}(W_1)$
While $!s.$Complete:
    $c \leftarrow \mathsf{uGet}(G_2, m)$
    $i \leftarrow \Theta.\mathsf{Dec}(k, c)$
    $(o, m) \leftarrow\$ \; s.\mathsf{Run}(W_1, i, \epsilon)$
    $\mathsf{sc}_w.\mathsf{send}(m)$
    $m \leftarrow \mathsf{sc}_w.\mathsf{receive}()$
$\mathsf{sc}_m.\mathsf{send}(o)$
```

Fig. 8: Soteria Workers with split storage.

(right of Figure 7) receives the parameters from M and starts processing the script: retrieves encrypted data from G, decrypts, processes and exchanges intermediate results with the other Worker. When the script is concluded, it returns its output to M. The behavior of W_2 is the same, but connection is established instead with W_1.

A.4 Full Proof

Given the description of Soteria components in Figure 7, the Soteria protocol Π_{xyz} is straightforward to describe. Considering a pre-encrypted storage G, the Client C, Master M and Workers W_1, W_2 execute following their respective specifications. Our theorem for the security of Soteria is as follows.

Theorem 2. Π_{xyz}, assuming the setup of Figure 5 and constructed as discussed above, securely realises $\mathcal{F}$ according to Definition 1.

The proof is presented as a sequence of four games. We begin in the real-world, and sequentially adapt our setting until we arrive in the ideal world. We then argue that all steps up to that point are of negligible advantage to $\mathcal{A}$, and thus the views must be indistinguishable to $\mathcal{Z}$.

The first is a simplification step, where, instead of using a single storage G, we slice the storage to consider G_1 and G_2. Figure 8 represents this change. This enables us to split the execution environment of W_1 and W_2 seamlessly, and can be done trivially since manifest file m by construction will never require different Workers to access the same parts of G. Since these two games are functionally equal, the adversarial advantage is exactly 0.

The second step is a hybrid argument, where we sequentially replace both Workers by ideal functionalities performing partial steps of the ML script. Concretely, we argue as follows. Replace W_1 with a functionality for its part of the ML script $\mathcal{F}_{W1}$, according to Definition 1. From Theorem 1 we can establish that this adaptation entails negligible advantage to $\mathcal{A}$ provided that the protocol without external access realises the same functionality. However this is necessarily the case, as it follows the exact structure as the constructions in [32]. We can repeat this process for W_2.[6] As such, using the intermediate result, we can thus upper bound the advantage adversary to distinguish these two scenarios by applying twice the result of Theorem 1.

The third step replaces the Master by an ideal functionality $\mathcal{F}_M$ that simply forwards requests to the Worker functionalities. This one follows the same logic as the previous one, without requiring the external storage, as the protocol also follows the exact structure as the constructions in [32].

In the final step, we have 3 functionalities $(\mathcal{F}_M, \mathcal{F}_{W1}, \mathcal{F}_{W2})$ playing the roles of (M, W_1, W_2), respectively. We finalize by combining them to a single functionality $\mathcal{F}$ for ML script processing. This can be done by constructing a big simulator S that builds upon the simulators for the individual components (S_M, S_{W1}, S_{W2}). The simulator S behaves as follows:

- Run S_M to construct the communication trace that emulates the first part of $\mathcal{F}$.
- Run the initial step of S_{W1} and S_{W2} to construct the communication trace for establishing secure channels between Workers and Master.
- Call leakage function $\mathcal{L}$ to retrieve the access patterns to G. Use the result to infer which part of the storage is being accessed, and run S_{W1} or S_{W2} to emulate the computation stage.

Given that the view produced by S is exactly the same as the one provided by the combination of S_M, S_{W1} and S_{W2}, the adversarial advantage is exactly 0.

We are now exactly in the ideal world specified for Definition 1.

Let

$$\begin{aligned}&\mathsf{Adv^{Dist}}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) = \\ &\qquad |\Pr[\mathsf{Real}^{G}_{\mathcal{Z},\mathcal{A},\Pi_{\mathsf{xyz}}} \Rightarrow \mathsf{T}] - \Pr[\mathsf{Ideal}^{G}_{\mathcal{Z},\mathcal{A},\mathcal{S},\mathcal{F}} \Rightarrow \mathsf{T}]|\end{aligned}$$

To conclude, we have that, for the negligible function μ,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) &= 2 \cdot \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_{W1},\mathcal{S}_{W1},\mathcal{F}_{W1}}(G) \\ &\quad + \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_M,\mathcal{S}_M,\mathcal{F}_M}() \\ &\leq \mu()\end{aligned}$$

and Theorem 2 follows.

Appendix B
ML Workflow attacks

This section presents the attacks in Section 3.2 in further detail and argues in which circumstances SOTERIA is secure against each attack. First, we will describe a general adversarial model against SOTERIA that follows the security restrictions justified in Appendix A. Then, we will present an experiment that captures what constitutes a valid attack under each definition, as described in Section 3.1. For each attack, we consider our protocol to be secure if we can demonstrate that one cannot rely on a valid adversary against the experiment of said attack under the constraints of SOTERIA. In some instances, this will depend on known attack limitations, which we detail case-by-case.

6. Again, this technique extends for an arbitrary number of Workers. N number of Workers would just require us to adapt the multiplication factor in the final formula, which would still be negligible.

```
Game AdvXYZ$_{\mathcal{A},\Pi_{\mathsf{xyz}}}(G, s)$:
$(G') \leftarrow\$ \mathcal{A}_1(G, s)$
$(m, l) \leftarrow \Pi_{\mathsf{xyz}}(G', s)$
$r \leftarrow\$ \mathcal{A}_2^{m}(l)$
Return Success$(G, s, m, r)$
```

Fig. 9: Adversary interacting with SOTERIA.
ff

B.1 Attacker against SOTERIA

Our goal is to present a model that details the conditions in which these attacks are possible. As such, it must be both generic, to capture the multiple success conditions of attacks, as well as expressive, so that it can be easy to relate to each specific attack.

In this definition, we will also consider an adversary that can play the role of an honest client, and thus will have black-box access to the produced model. We stress that, in practice, this will not be the case in many circumstances. In those scenarios, since queries to the model are made via secure channel, an external adversary is unable to arbitrarily request queries to the model without causing it to abort. This means that any attack that requires black-box access to the model is not possible if SOTERIA assumes external adversaries.

Let Π_{xyz} denote the full training protocol of SOTERIA. It receives external storage G and task script s as inputs, and produces a model m, which can then be queried. Based on the security result of Appendix A, the interaction of an adversary with our system can be described in Figure 9. The adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ can first try and manipulate the input dataset G to G'. This is then used for Π_{xyz}, which will produce the model m and the additional leakage l (SOTERIA-B has no additional leakage, so $l = \epsilon$). Finally, the adversary can interact black-box with the model until a conclusion r is produced. This will be provided to a Success predicate, which will state if the attack was successful. This predicate is specific to the attack, and allows us to generally describe attacks such as *adversarial samples*, where the goal is to make the resulting model deviate, as well as *membership inference*, where the goal is to retrieve information from the original dataset.

Remark Observe that $\mathcal{A}_1$ and $\mathcal{A}_2$ do not share state. This is because they play different roles within this experiment: the first influences the system by attempting to manipulate the training dataset G, while the second interacts with the model m and leakage l to try and extract information. Indeed, our first step will be to show that $\mathcal{A}_1$ is unable to rely on G' to meaningfully convey any additional knowledge gained by observing (G, s).

B.2 Dataset manipulation

Dataset manipulation attacks are defined by an adversary with the capability of inserting, removing or manipulating

```
Game $\mathsf{DSetMan}_{\mathcal{A},\Pi}(G, s)$:
    $G' \leftarrow\!\!\$\; \mathcal{A}(G, s)$
    $m \leftarrow \Pi(G', s)$
    Return $\mathsf{Success}(G, s, m)$
```

Fig. 10: Model for dataset manipulation attack.

dataset information. These align with the setting considered for attacks via *adversarial samples*. Figure 10 is an experiment that describes what constitutes a successful attack for dataset manipulation. The adversary $\mathcal{A}$ is given full knowledge of G[7], and must produce an alternative input dataset G'. We then train the model (protocol Π) over that data to produce model m, and the adversary is successful if said model satisfies some attack success criteria $\mathsf{T}/\mathsf{F} \leftarrow \mathsf{Success}$.

We now argue that the integrity guarantees of the authenticated encryption used by our external storage G ensure that these attacks do not occur for Π_{xyz}. We do this by showing that any adversary that performs an *adversarial samples* attack on Π_{xyz} can be used to construct a successful attack on the the security of the authenticated encryption scheme. First, observe that no attack can be successful if the adversary makes no changes on the input dataset, so if $G = G'$ then $\mathsf{F} \leftarrow \mathsf{Success}$. Furthermore, if Π_{xyz} aborts, then no model is produced, so it naturally follows that the attack is unsuccessful $\mathsf{F} \leftarrow \mathsf{Success}$.[8]

As such, the only cases in which $\mathsf{T} \leftarrow \mathsf{Success}$ are those in which $G' \neq G$ and Π_{xyz} does not abort. But this means that the adversary was able to forge an input that correctly decrypts, breaking the integrity of the underlying encryption scheme. Since the security guarantees of authenticated encryption ensure that the probability of existing such an adversary is negligible, the probability of such an attacker in SOTERIA will also been negligible.

B.3 Black-box Attacks

All the remaining attacks with the exception of some *reconstruction attacks* follow a similar setting, where the adversary leverages a black-box access to the trained model, depicted in the experiment of Figure 11. We begin by running Π to produce our model and leakage, and then run an additional procedure $\mathsf{Extract}$ to obtain additional information from the original dataset, which cannot be retrieved by simply querying the model. This procedure captures whatever knowledge regarding the underlying ML training might be necessary for the attack to be successful (e.g. information about data features). We then provide this additional information to the adversary, and give it black-box access to the model. The success criteria depends on the specific attack, and is validated with respect to the original dataset, model, and the task script being run. E.g. for *model extraction* attack, the goal might be to present a model m' that is similar to m, evaluated by the $\mathsf{Success}$ predicate.

For simplicity, we first exclude all attacks for an external adversary, which does not have black-box access to the model of SOTERIA. This is true if we can show that one cannot emulate black-box access to the model using confidence values and class probabilities. Albeit an interesting research topic, current attacks are still unable to do this in an efficient way [11]. We now go case-by-case, assuming an adversary can play the role of a genuine client to our system.

Our arguments for Π_{xyz} depend on being able to rely on a successful adversary $\mathcal{A}$ of Figure 11 to perform the same attack in Figure 9. As such, the security of our system will depend on the amount of additional information z, on how it can be extracted from the view of the adversary of SOTERIA.

- For *membership inference*, *reconstruction attacks* based on black-box access to the model, *model inversion*, and *model extraction* via *data-free knowledge distillation*, no additional information z is required. This means that any successful adversary in Figure 11 will also be successful in Figure 9, meaning that both for SOTERIA-B and SOTERIA-P are vulnerable. Preventing these attacks requires restricting access to the model to untrusted participants.
- *Class-only* attacks for *model extraction* require additional knowledge from the dataset. Specifically, z must contain concrete training dataset samples. This means that, to leverage such an adversary $\mathcal{A}$, one must first be able to use $\mathcal{A}_2^m(l)$ to extract such a z. This exactly matches the setting of *model inversion* attacks. This means that SOTERIA is vulnerable to *class-only* attacks for *model extraction* in SOTERIA-B or SOTERIA-P if there is also an efficient attack for *model inversion* in SOTERIA-B or SOTERIA-P, respectively.
- *Equation-solving model extraction* requires knowledge of the dimension of the training dataset G. This is additional information z that is not revealed by querying the model, which means no adversary $\mathcal{A}_2^m(\epsilon)$ can retrieve z, and thus SOTERIA-B is secure against said attacks. However, combining public data with confidence values might allow for $\mathcal{A}_2^m(l)$ to extract a sufficient z to perform the attack, which makes SOTERIA-P vulnerable to *equation-solving model extraction*.
- *Path-finding model extraction* attacks require information regarding leaf count, tree depth and leaf ID. As such, all this must be encapsulated in z. [1] suggests that such information is not retrievable from only black-box access to the model [1], which means no adversary $\mathcal{A}_2^m(\epsilon)$ can produce z and thus SOTERIA-B is secure. However, this is information that can be extracted from confidence values, which suggests that an efficient adversary $z \leftarrow\!\!\$\; \mathcal{A}_2^m(l)$ is likely to exist, and thus SOTERIA-P is vulnerable to such attacks under these assumptions.

We can generalize the security of our system to these types of attacks as follows. If no additional information z is required, then Π_{xyz} is vulnerable to an adversary that can play the role of an honest client. If z can be extracted from black-box access to the model, then we can still rely on said adversary to attack Π_{xyz}. Otherwise, SOTERIA-B is secure, as no additional information is leaked. Furthermore, the security of SOTERIA-P will depend whether one can

7. Realistically, an attacker would have less information, but for our purposes we can go for the worst case and give him all the information regarding the computation and its input.

8. The only circumstance in which this could be considered a successful attack was if the goal was to perform a denial-of-service attack, which we consider to fall outside the scope of an adversarial sample attack.

```
**Game** $\mathsf{BlackBox}_{\mathcal{A},\Pi_{\mathsf{xyz}}}(G, s)$:
$m \leftarrow \Pi(G, s)$
$z \leftarrow\!\!\$\ \mathsf{Extract}(G, s)$
$r \leftarrow\!\!\$\ \mathcal{A}^{m}(z)$
Return $\mathsf{Success}(G, s, m, r)$
```

Fig. 11: Model for black-box attacks.

```
**Game** $\mathsf{WhiteBox}_{\mathcal{A},\Pi_{\mathsf{xyz}}}(G, s)$:
$m \leftarrow \Pi(G, s)$
$r \leftarrow\!\!\$\ \mathcal{A}(m)$
Return $\mathsf{Success}(G, s, m, r)$
```

Fig. 12: Model for white-box attacks.

infer z from l and from the black-box access to the model. Concretely, if we can show that no (efficient) function F exists, such that $z \leftarrow\!\!\$\ \mathsf{F}^{m}(l)$, then Π_{xyz} for leakage l is secure against attacks requiring additional data z.

B.4 White-box Attacks

White-box attacks capture a scenario where an adversary requires white-box access to the model. These align with the setting of *reconstruction attacks* that explicitly requires white-box access to the model. Figure 12 is an experiment that describes what constitutes a successful *reconstruction attack* in this context. We begin by training the model (protocol Π) over the original dataset to produce the model. We then provide the trained model directly to the adversary, which will reconstruct raw data r. Finally, the success of the attack is validated with respect to the original dataset.

We now argue that these attacks do not occur for Π_{xyz}, as long as it is not possible to extract the model from the confidence values and from black-box access to the model. This is because the attacker of Figure 12 receives explicitly the model m, whereas the adversary $\mathcal{A}_2$ in Figure 9 receives the confidence values in l, and black-box access to the model. To rely on such an attacker, $\mathcal{A}_2$ must therefore be able to produce input m from its own view of the system. As such, relying on such an adversary implies there is an efficient way $m \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{bb}}^{m}(l)$ to retrieve the model m from confidence values l and black-box access to the model m, which is exactly the setting of *model extraction* attacks in the previous section. This adversary $\mathcal{A}_{\mathsf{bb}}$ can then be called by $\mathcal{A}_2$ to produce input m, which is then forwarded to the adversary of Figure 12 to produce a successful attack r. As such, SOTERIA is vulnerable to white-box *reconstruction attacks* if there exists an efficient adversary $\mathcal{A}'$ that successfully wins the experiment of Figure 11 for the *model extraction* attack.

B.5 Summary

Table 4 summarizes the attacks discussed. These are divided between all the identified classes of attacks, as well as whether the adversary is external or if it can query the model as a client. In many instances, the security of our system hinges on another argument over specific restrictions assumed for the adversary.

We now list the arguments that propose the security of our system in the different contexts.

{1}: an adversary is unable to retrieve the (white-box) model from confidence values {1a}, black-box access to the model {1b}, or both {1c}.
{2}: an adversary is unable to emulate black-box access to the model from confidence values.
{3}: an adversary is unable to retrieve the dimension of the dataset from black-box access to the model {3a} and confidence values {3b}.
{4}: an adversary is unable to retrieve information of leaf count, depth and ID from black-box access to the model {4a} and confidence values {4b}.
{5}: no *model inversion* attack exists for retrieving dataset samples for SOTERIA-B {5a} and SOTERIA-P {5b}.

White-box based attacks explicitly require additional information, such as feature vectors, over black-box access to the model [26]. This is something that supports {1b} directly, and since confidence values are not computed from feature vectors, so would be {1a} and {1c}. Extracting model access from only confidence values is an active area of research, but current attacks [11] are still unable to do this in an efficient way {2}. Typically, one cannot infer dimension from simply querying the model, which suggests {3a} is true, but this is unclear for confidence values, and thus one might consider {3b} is false. [1] suggests {4a} is true, but {4b} is not. {5} will fundamentally depend on the application [1], [11].

		External		Client	
		SOTERIA-B	SOTERIA-P	SOTERIA-B	SOTERIA-P
Adversarial samples		✓	✓	✓	✓
Reconstruction	WB	✓	{1a}	{1b}	{1c}
	BB	✓	{2}	✗	✗
Membership inference		✓	{2}	✗	✗
Model inversion		✓	{2}	✗	✗
Model extr	Equation	✓	{2}	{3a}	{3b}
	Path	✓	{2}	{4a}	{4b}
	Class	✓	{2}	{5a}	{5b}
	DF KD	✓	{2}	✗	✗

TABLE 4: Summary of attacks against SOTERIA. ✓ means SOTERIA is resilient to the attacks, ✗ means SOTERIA is vulnerable to the attacks, and {X} means SOTERIA is secure if argument {X} is also true.