

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

# Privacy-Preserving Machine Learning on Apache Spark

CLÁUDIA V. BRITO<sup>1,2</sup>, PEDRO G. FERREIRA<sup>1,3</sup>, BERNARDO L. PORTELA<sup>1,3</sup>, RUI C. OLIVEIRA<sup>1,2</sup>, and JOÃO T. PAULO<sup>1,2</sup>

<sup>1</sup>INESC TEC

<sup>2</sup>University of Minho (e-mail: {claudia.v.brito, rcmo, jtpaulo}@inesctec.pt)

<sup>3</sup>Faculty of Sciences, University of Porto (e-mail: {pgferreira, bernardo.portela}@fc.up.pt)

Corresponding author: Cláudia V. Brito (e-mail: claudia.v.brito@inesctec.pt).

This work is an extended version of the work published in [1]. This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020 and through Ph.D. Fellowship SFRH/BD/146528/2019 - Cláudia Brito. This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

**ABSTRACT** The adoption of third-party machine learning (ML) cloud services is highly dependent on the security guarantees and the performance penalty they incur on workloads for model training and inference. This paper explores security/performance trade-offs for the distributed Apache Spark framework and its ML library. Concretely, we build upon a key insight: in specific deployment settings, one can reveal carefully chosen non-sensitive operations (e.g. statistical calculations). This allows us to considerably improve the performance of privacy-preserving solutions without exposing the protocol to pervasive ML attacks. In more detail, we propose Soteria, a system for distributed privacy-preserving ML that leverages Trusted Execution Environments (e.g. Intel SGX) to run computations over sensitive information 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 experimental evaluation validates that our approach reduces the runtime of ML algorithms by up to 41% compared to previous related work. Our protocol is accompanied by a security proof and a discussion regarding resilience against a wide spectrum of ML attacks.

**INDEX TERMS** Privacy-preserving, Machine Learning, Distributed Systems, Apache Spark, Trusted Execution Environments, Intel SGX.

## I. INTRODUCTION

The ubiquitous environment provided by cloud computing providers offers a scalable, reliable, and performant environment to deploy compute-intensive Machine Learning (ML) workloads. However, outsourcing ML data storage and computation to third-party services 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 [2]–[4].

Recent works have addressed these attacks with solutions based on homomorphic encryption or secure multi-party computation schemes [5], [6]. However, these cryptographic schemes impose a significant performance toll that restricts their applicability to practical scenarios [7]. To circumvent

this performance penalty, another line of research is that of exploring hardware technologies enabling Trusted Execution Environments (TEEs), such as Intel SGX [8]. 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 [9]–[11]. 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 [12].

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 [13]–[15] 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 [3], [16]; 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 [17]. 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.

### A. CONTRIBUTIONS

Thus, we propose Soteria, an open-source system for distributed privacy-preserving ML that leverages the scalability and reliability provided by Apache Spark and its ML library (MLlib). Unlike previous solutions [18], [19], Soteria supports a wide variety of ML algorithms without changing how users build and run these within Spark. Further, 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 data being processed only exists in plaintext inside the enclave and is encrypted in the remainder dataflow (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 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 [18]. Our experiments, resorting to the HiBench benchmark [20] and including four different ML algorithms, show that Soteria-P, while considering a more significant 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, 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.

### B. OUTLINE

The remainder of this paper is organized as follows. Section II gives brief background information on Apache Spark, MLlib, and Intel SGX. Then, Section III presents the threat model of Soteria and a description of current ML attacks. Next, we concretely define the design goals of Soteria and the architecture and flow of Apache Spark. This section also encompasses the architecture of Soteria, its building blocks, the novel partitioning scheme, and the security reasoning. Section V depicts the testbed performed, the main observations, and its analysis and discussion. Next, Section VI presents the state-of-the-art approaches over the past years, emphasizing the main differences with Soteria. Finally, Section VII concludes and highlights the main contributions proposed in this paper.

## II. BACKGROUND

### A. 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. Spark performs most of the computation in-memory, thus promoting better performance for data-intensive applications when compared to Hadoop's MapReduce [21].

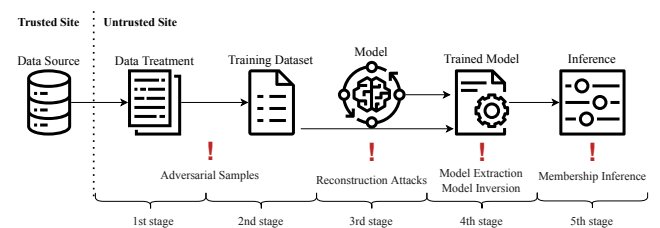


FIGURE 1. ML pipeline and known attack vectors.

The MLlib library [22] enables Spark users to build end-to-end ML workflows. These workflows are divided into five 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 be saved (persisted) and loaded (accessed) for inference.

## B. 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 [8], [23].

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) [23]. Nonetheless, even though the second generation of SGX has improved the size of the protected memory region, it still defines the Enclave Page Cache (EPC) to 128MB per CPU [24]. Memory swapping occurs when such a limitation is met, which is a performance-costing mechanism [12]. Thus, SGX-based solutions must balance the number of I/O operations, the amount of data handled by enclaves, and the Trusted Computing Base (TCB) to optimize performance.

In this paper, we chose Intel SGX over other TEE's (e.g., ARM TrustZone [25]) due to its general use in academia [11], [13], [14] and industry [26], availability, as well as its security guarantees and computing reliability. However, our solution is generic and can be applied to other TEE technology that follows similar design principles to SGX.

## III. THREAT MODEL AND ATTACKS

### A. 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 securely stores its input data within this backend, which is also considered untrusted throughout the protocol execution.

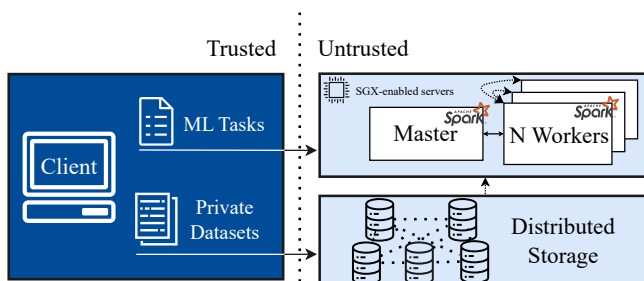


FIGURE 2. SOTERIA deployment model.

We consider semi-honest adversaries, meaning that security is defined according to a threat that attempts to break the confidentiality of data and model but 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 [27]. 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.

### B. ML WORKFLOW ATTACKS

Throughout the paper, we consider the black-box setting proposed by [28], which is as follows. When an adversary is given *black-box* access to a model, it means that 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 under this threat model entails including countermeasures against a wide array of attack vectors. Given this context, we now further detail pervasive attacks on the ML pipeline and establish their adversarial assumptions. A scheme summarizing these attacks is depicted in Figure 1.

#### Adversarial attacks

These attacks are characterized by injecting malicious data samples to manipulate the model and 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 (*i.e.*, the output of inference) and class probabilities (score-based attacks) [16], [29].

#### 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 obtain the desired result, the attacker must know the dimension of the original training dataset (equation-solving attacks), the dimension of the decision trees, the 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)) [2], [17], [30].

#### 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 publicly disclose their trained models and confidence values. In model inversion attacks, the adversary must have partial knowledge of the training dataset's features to infer and query the model with specific queries [2], [3]. Membership inference aims to test whether

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

Attacks		Systems				
		[9]	[10]	[19]	[18]*	SOTERIA
Adversarial	Gradient-based	X	X	✓	X	✓
	Score-based	X	X	✓	X	✓
	Transfer-based	X	X	✓	X	✓
	Decision-based	X	X	✓	X	✓
Model Extraction	Equation-solving	✓	✓	X	✓	✓
	Path-finding	✓	✓	X	X	✓
	Class-only	✓	✓	X	X	✓
	DFKD	✓	✓	X	✓	✓
Model Inversion		✓	✓	?	✓	✓
Reconstruction Attacks		✓	✓	✓	✓	✓
Membership Inference		X	X	X	X	✓

\*Data encryption is not provided on the open-source version.

✓ - Protected; X - Non-protected; ? - Not disclosed.

a specific data point  $d$  was used as training data and requires the adversary to know a subset of samples used to train the model (that does not contain  $d$ ) [31].

#### 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) [32], others only require black-box access to the model [33]. Nonetheless, in this setting, the attacker needs to have access to another dataset with the same distribution as the original training dataset (i.e., local dataset and training dataset are subsets from a larger dataset).

#### Summary

Unlike previous works [9], [10], [18], [19], 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 a 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 analyzed in detail in Section IV-F.

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 [34] can be layered over Soteria to address these at the cost of additional performance overhead [35].

## IV. SOTERIA

### A. DESIGN GOALS

SOTERIA is a distributed privacy-preserving machine learning system that avoids changing the architecture and processing flow of Apache Spark and MLlib, retaining its usability, scalability, and fault tolerance properties. It is built under the assumption that ML runtime performance can be improved if one can diminish the number of operations done inside secure enclaves. Thus, Soteria proposes a partitioning scheme to split the computation to be performed inside and outside these.

Soteria builds upon four core principles:

**General Applicability for ML workloads.** Soteria aims to offer an encompassing solution for several ML algorithms by relying on Apache Spark’s MLlib.

**Privacy-by-design.** In Soteria, sensitive data is only on plaintext inside the enclaves, being encrypted in the remaining workflow. This is achieved by resorting to trusted execution environments and encryption mechanisms that safeguard data privacy.

**Balanced overhead.** Soteria offers a partitioning scheme that balances the imposed performance overhead of the privacy measures and the leakage of such a solution.

**Low intrusiveness.** Both the processing flow of Apache Spark and the user’s interaction with the system remain unchanged or require minor changes.

### B. ARCHITECTURE AND FLOW

As depicted in Figure 3, 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 3 by the white dashed and solid line boxes.

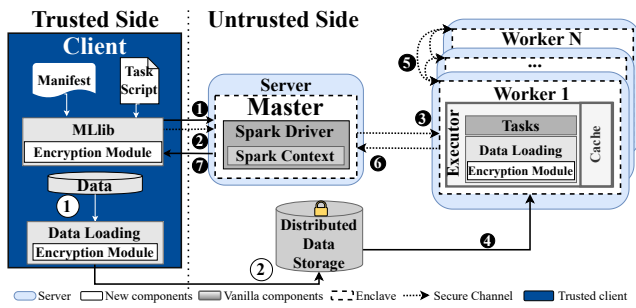


FIGURE 3. Soteria architecture and flow of operations.

### C. CLIENT

Users use Soteria's client module 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 how 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.

**Data Loading.** First, the user must specify the data to be loaded to the storage backend for the first operation. 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 3-①). This module encrypts the data being loaded into the distributed storage backend with a symmetric-key encryption scheme (Figure 3-②).

**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 3-③), 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 3-④). Briefly, and as explained in the next sections, this file ensures that different Spark components can attest to 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 3-①②). 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.

### D. 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 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 3-②).

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<sup>1</sup> along with the user's encryption key and Manifest file (Figure 3-③). 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 3-④). Once again, the Manifest file prevents 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 3-⑤).

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 3-⑥). The master node then merges the partial outputs into the final result, which is done inside a trusted enclave, and sends it encrypted, with the user's encryption key, to the trusted client module (Figure 3-⑦). 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.

### E. PARTITIONED DESIGN

Soteria proposes a novel partitioning scheme, Soteria-P, that does fine-grained partitioning, of which operations execute

<sup>1</sup>The same metadata sent by a vanilla Spark deployment so that workers know the computational operations to perform.

inside and outside secure enclaves. Note that this partitioning is only done for ML operations executed at Spark worker nodes. The remaining operations performed 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.

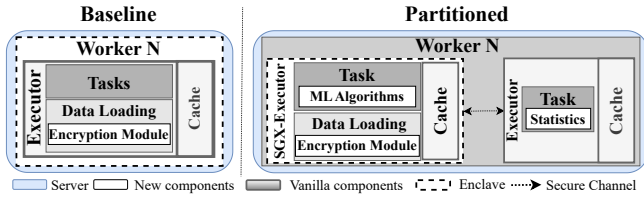


FIGURE 4. Comparison between SOTERIA-B and SOTERIA-P schemes.

**SOTERIA Baseline (SOTERIA-B).** In Soteria-B, all computation done by Spark workers is included in a trusted environment. Namely, the executor processes launched by each worker node are deployed inside an enclave, as depicted in Figure 4. Outside the enclave, data is always encrypted in an authenticated fashion, allowing 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 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 changes to how users submit ML tasks. As depicted in Figure 4, 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 (*SGX-executors*).

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

To provide a more concrete example, Algorithm 1 depicts the pseudo-code of a Linear Regression algorithm under the Soteria-P scheme and behaves as follows. The goal is to minimize the loss function, in this particular example, the *Root Mean Squared Error*. First, in the SGX-executor, an instance of Spark loads the dataset, creating a dataframe  $(X, y)$ . This dataframe is further split into train and test data,  $(X_{train}, y_{train}, X_{test}, y_{test})$ . After this first pre-processing, an instance of a Linear Regression algorithm ( $lrM$ ) is trained with the training data, and with the testing data, the first predicted values are inferred ( $P$ ).

With these values, the Root Mean Squared Error (RMSE) is calculated at the non-secure executor ( $rmse$ ). This computation is depicted in Algorithm 2, which intends to find the minimum error value. If no initial error is available, the algorithm returns the calculated RMSE. Otherwise, it returns the newly calculated RMSE ( $newRMSE$ ). This is the only part of the computation within Soteria-P that is done outside trusted hardware, and thus, it is highlighted in orange.

After receiving the result, the SGX-executor continues the model training according to the number of maximum iterations ( $maxIter$ ) defined by the user. In each iteration, it trains a new model ( $lrM'$ ) and predicts new values ( $P'$ ), which are iteratively used to calculate a new RMSE. If the new RMSE is lower than the previous, the worker keeps the newly trained model. The master node then collects the results from the workers, maps and reduces the model parameters, and returns the encrypted final model to the client.

---

#### Algorithm 1 LinearRegression(DS, prms, $lrM$ ):

---

```

1:  $(X, y) = \text{Spark.load}(\text{DS})$ 
2:  $(X_{\text{train}}, y_{\text{train}}), (X_{\text{test}}, y_{\text{test}}) = (X, y).\text{split}()$ 
3:  $lrM.\text{fit}(X_{\text{train}}, y_{\text{train}}, \text{prms})$ 
4:  $P \leftarrow lrM.\text{transform}(X_{\text{test}}, y_{\text{test}})$ 
5:  $rmse \leftarrow \text{Untrusted}(y_{\text{test}}, P, \epsilon)$ 
6: for  $i = 1$  to  $maxIter$  do
7:    $lrM' \leftarrow lrM.\text{fit}(X_{\text{train}}, y_{\text{train}}, \text{prms})$ 
8:    $P' \leftarrow lrM'.\text{transform}(X_{\text{test}}, y_{\text{test}})$ 
9:    $rmse' \leftarrow \text{Untrusted}(y_{\text{test}}, P', rmse)$ 
10:  if  $rmse' < rmse$  then
11:     $lrM \leftarrow lrM'$ 
12:     $rmse \leftarrow rmse'$ 
13:  end if
14: end for
15: return  $lrM, rmse$ 

```

---

## F. SECURITY

Our security goal is formally defined using the real-versus-ideal world paradigm, similar to the Universal Composability framework [36]. 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

**Algorithm 2** Untrusted( $y_{\text{test}}, y_{\text{new}}, \text{rmse}$ ):

---

```

1:  $n \leftarrow \text{len}(y_{\text{test}})$ 
2:  $\text{newRMSE} \leftarrow \sqrt{\frac{1}{n} \sum_{i=1}^n (y_{\text{test}_i} - y_{\text{new}_i})^2}$ 
3: if  $\text{rmse} = \epsilon$  then
4:   return  $\text{newRMSE}$ 
5: else
6:   return  $\min(\text{rmse}, \text{newRMSE})$ 
7: end if

```

---

attacks. This idealized service is specified as a functionality parameterized 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-P and Soteria-B schemes can be found in Appendix A. The outline is as follows. The role played by the 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 [37] 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.

## 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 (as per definition on Section III-B). Our result implies that adversaries cannot infer internal data from the workers, and the secure channel between the 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? Specifically, our argument is by reduction: if an attack based on black-box access to the model occurs in Soteria-P, then the adversary must have been able to extract such black-box access from the statistical information revealed. Given how statistical data depends on the underlying ML script, consider the concrete example provided in Algorithm 1. Here, the leakage can be defined as the sum of all data revealed to the untrusted execution, namely the set of predictive labels  $y_{\text{test}}$ , and the results of  $\text{maxIter}$  number of predictions after  $\text{lrM.fit}(X_{\text{train}}, y_{\text{train}}, \text{prms})$ . Concretely, the leakage  $l$  of an execution of Algorithm 1 can be defined as:

$$l = (y_{\text{test}}, \sum^{maxIter} \text{lrM.transform}(X_{\text{test}}, y_{\text{test}}))$$

This quantifies the amount of information explicitly revealed to the adversary of Soteria. As such, attacks requiring black-box access to the model can only occur if there exists an efficient algorithm that can take  $l$  and produce a sufficient approximation to  $\text{lrM}$  for black-box attacks to be conducted.

For the general case, 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 [17]. 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. From this, it follows that Soteria-P scheme is resilient to any attack requiring black-box access to the model to succeed.

## 2) Relation to ML Attacks

We now overview the four types of attacks referred to in Section III-B 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 the SGX-executors 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, 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, as defined in Appendix A).

### G. 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 cipher mode, providing 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 [38]. Soteria's Manifest file was also provided by Graphene.

## V. 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?

### A. 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 [20] for evaluating seven ML algorithms (Table 2) 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), Sparse Naive Bayes (Naive Bayes), Latent Dirichlet Allocation (LDA) and K-means clustering (K-means).

The benchmark suite offers different workload sizes for each algorithm ranging from *Tiny* to *Gigantic* configurations.

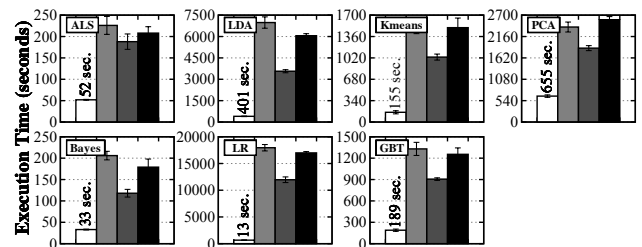
**Setups and metrics.** To validate Soteria's performance and the benefits of fine-grained differentiation of secure ML operations, we compare the implementations of our system with the Soteria-B and Soteria-P schemes. These setups are compared with a deployment of Apache Spark that does not offer privacy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

Moreover, we test SGX-Spark [18], 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 system 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 three independent runs. The dstat monitoring tool was used to collect the CPU, RAM, and network consumption at each cluster node.

### B. PERFORMANCE OVERVIEW

Figure 5 shows the execution time of all the setups for the seven algorithms when using a huge-sized workload configuration. Moreover, Figures 6a, 6b, 6c and 6d 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. Observations 1 to 8 correspond to Figure 5, whilst Observations 9 to 12 refer to Figure 6.



**FIGURE 5.** Execution time for each algorithm with *Huge* workload. The legend is as follows: □ Vanilla Spark; ■ SOTERIA-B; ■ SOTERIA-P; ■ SGX-Spark.

**Observation 1.** Vanilla Spark's execution times for ALS, PCA, LR, and GBT algorithms are 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.



**TABLE 2.** Representation of each ML algorithm's tasks, time complexity, and data sizes for different workloads.

Algorithms	Tasks	Time Complexity	Workloads			
			Tiny	Large	Huge	Gigantic
ALS	RS	$O((m+n)k^3 + mnk^2)$ [39]	193KB	345MB	2GB	4GB
PCA	DR	$O(nm * \min(n, m) + m^3)$ [40]	256KB	92MB	550MB	688MB
GBT	P	$O(n * y * n_{trees})$ [41]	36KB	46MB	92MB	183MB
LR	C + P	$O(m * n^2 + n^3)$ [42]	11GB	134GB	335GB	894GB
Naive Bayes	MC	$O(nm)$ [43]	-	-	5GB	-
LDA	DR	$O(mnt + t^3)$ [44]	-	-	2GB	-
Kmeans	Cl	$O(n^2)$ [45]	-	-	56GB	-

RS: Recommendation Systems; DR: Dimensionality Reduction; P: Prediction; C: Classification; MC: Multi-class Classification Cl: Clustering.

SGX-Spark incurs an execution overhead of 4x. Thus, the three setups have similar results, requiring approximately 150 seconds more of 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 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.** The LDA algorithm exhibits higher execution overhead of 17.40x, 8.89x, and 15.08x for Soteria-B, Soteria-P, and SGX-Spark setups, respectively. Soteria-P outperforms SGX-Spark by a difference of 41.5 minutes (i.e., reduces execution time by 41%).

**Observation 7.** When compared with the vanilla deployment, Soteria-B increases execution time of KMeans by 9.37x and Soteria-P by 6.68x. SGX-Spark has an overhead of 9.7x, which means that, in comparison with Soteria-P, it requires additional 468 seconds (7.8 minutes) to execute, i.e., Soteria-P is 31% faster.

**Observation 8.** With Huge workload and Naive Bayes, Soteria exhibits an overhead of 6.24x for Soteria-B, which is higher than the 5.33x observed for SGX-Spark. Also, Soteria-P continues to present a lower overhead (3.58x) compared to SGX-Spark. The absolute difference in execution time between Soteria-P and Soteria-B is 88 seconds, whilst, with SGX-Spark, Soteria-P decreases execution time by 61 seconds (34%).

**Observation 9.** 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, Soteria 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.

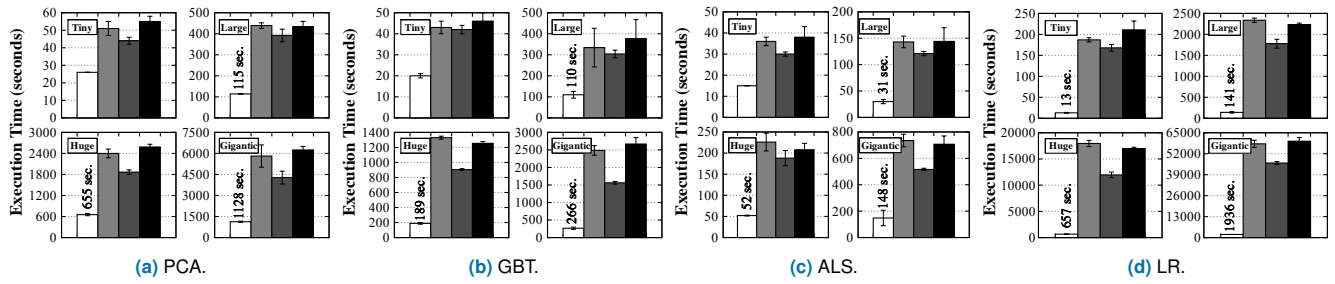
**Observation 10.** Regarding the GBT algorithm, and the Tiny workload, the overhead of Soteria-B, Soteria-P, and SGX-Spark are similar. However, the difference between the three approaches is more visible when increasing the workload size. Soteria-P (Tiny-2.13x and Gigantic-5.88x) outperforms both approaches, while Soteria-B (Tiny-2.18x, Gigantic-9.35x) and SGX-Spark (Tiny-2.3x, Gigantic-10.34x) have similar results. Soteria-P surpasses SGX-Spark's execution time in the Gigantic workload by up to 41%.

**Observation 11.** 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 12.** 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, 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 13.** Overall, the resource consumption (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% CPU usage). 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 14.** 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,



**FIGURE 6.** 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.

or ROC). The results corroborate that both Soteria-B and Soteria-P show accuracy values similar to the vanilla Spark version.

### C. ANALYSIS

We now further analyze the experimental observations according to three topics, i) dataset size; ii) algorithm complexity; iii) size of trusted computing base (TCB).

#### 1) Dataset size

Figure 6 shows the performance degradation for the PCA, GBT, ALS, and Linear Regression algorithms with increasing dataset sizes. Results show that, for PCA, GBT, and ALS workloads with smaller datasets, Soteria-B and Soteria-P perform similarly. However, as the size of the datasets increases, more operations and data must be transferred to the SGX enclave, thus having a more noticeable toll on the overall performance. Indeed, the page swapping mechanism of SGX, which occurs due to its memory limitations, incurs a significant performance penalty [12], [46]. For example, when compared to the vanilla 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 by 1.78x in the *Tiny* workload and 3.79x in the *Gigantic* workload. Linear Regression is the most expensive algorithm in terms of performance as it processes more data for the distinct workload sizes (Table 2). Compared with SGX-Spark, Soteria-P deals better with the data volume increase. Indeed, as seen in Observations 9-12, we reduce the execution time from 9% up to 41% when compared to SGX-Spark. Also, compared with our baseline, Soteria-P achieves up to 33% less execution time.

#### 2) Algorithm Complexity

The execution times of ALS and LDA algorithms are very different even though their dataset size is similar. These results are explained by the computational complexity of each algorithm. For ALS, the synthetic workload data generated by the benchmark has a low hidden  $k$  dimension with a low ranking of 10, simplifying the required computation and decreasing execution time. For the LDA algorithm, the computational complexity, and consequently the execution time, are

increased due to the higher number of dependencies between values at the generated synthetic workload data. Observations 2 and 6 emphasize the performance of these two algorithms for a similar workload size. Like LDA, Observations 3 and 9 show that PCA complexity and performance overhead increase with the processed data volume. Commonly classified as regression and classification algorithms, Bayes and GBT have similar performance, as seen in Observation 8 and Observation 5. The data sizes of these two algorithms are completely different, where GBT uses 91.7MB, and Bayes has 5.2GB. However, the Bayes algorithm iterates only once over the data, while GBT iterates over several decision trees to find its best model. Kmeans' performance is highly dependent on the chosen dataset size. This is also true for the Linear Regression algorithm (Observations 4 and 12).

#### 3) Size of TCB

The results discussed in Section V-B show that SGX-Spark outperforms Soteria-B for some of the evaluated algorithms (Observation 2, 4-6, 8). As SGX-Spark only protects UDFs, the performance overhead imposed by our solution's larger trusted computing base is naturally higher. Nevertheless, when compared to SGX-Spark, Soteria-B covers a wider range of machine learning attacks while keeping performance overhead below 1.59x. Indeed, for algorithms such as PCA, and Kmeans, Soteria-B has a similar or slightly inferior execution time (Observations 3 and 7). This happens because, for these algorithms, both SGX-Spark and Soteria-B perform similar computations at the secure enclaves, while the UDF mechanism is not the most optimized choice for running some of these workloads.

Finally, due to the TCB reduction by our second scheme, Soteria-P always outperforms SGX-Spark and Soteria-B (Observations 2-12). The results show that this solution can reduce the training time by up to 41%, namely for the LDA algorithm with the *Huge* workload (Observation 6).

#### 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 IV-F). 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.

## VI. RELATED WORK

Secure ML solutions can be classified into four main groups based on the privacy-preserving techniques being used: i) encryption-based [5]–[7], ii) secure multi-party computation [47], [48], iii) differential privacy [49], [50] and, iv) trusted execution environments (TEEs) [9], [10], [23], [51]. This paper is included in group iv).

### A. PRIVACY-PRESERVING ML WITH TEEs

Chiron [9] enables training ML models on a cloud service without revealing information about the training dataset. Myelin [10] 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 can cover both the training and inference phases while providing additional protection against adversarial samples, reconstruction, and membership inference attacks (Table 1).

In [23], five ML algorithms are re-implemented with data oblivious protocols. Combined with TEEs, these protocols ensure strong privacy guarantees while preventing exploiting side-channel attacks that observe memory, disk, and network access patterns to infer private information. Unlike this solution, Soteria aims to support all ML algorithms built with MLlib transparently.

### B. PRIVACY-PRESERVING ANALYTICS WITH TEEs

TEEs have also been used to ensure privacy-preserving computation for general-purpose analytical frameworks [19]. In comparison to SGX-Spark [18], detailed in Section V-A, 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 [13], and Uranus [14] 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 the 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, compared with MLlib-based ones, exhibit lower performance for some ML workloads [52].

Therefore, the design, implementation, and security requirements are distinct compared to Soteria.

### C. PRIVACY-PRESERVING DEEP LEARNING WITH TEEs

TEEs have also been applied to the training and inference of deep neural networks [51], [53]. However, there is a substantial difference between the internals of ML and DL frameworks and algorithms, thus, requiring significantly different privacy-preserving designs for each scenario. Since MLlib does not natively support DL workloads, the focus of Soteria is solely on ML algorithms.

### D. COMPUTATION PARTITIONING WITH TEEs

Recent solutions have been proposed to address the challenge of partitioning and selecting specific code from applications that should run at trusted execution environments.

Glamdring [54] proposes a static analysis tool that infers a partition between trusted and untrusted code in an application. It tries to achieve a balanced distribution of partitions to minimize the number of edges crossing between components. Another approach, Civet [55], focuses solely on partitioning Java applications. It provides an annotation-based approach for partitioning Java applications and ensures inter-object communication and consistent garbage collection across the partitioned components. Finally, Uranus [14] and Montsalvat [56] propose two automatic partitioning tools. Uranus [14] addresses the challenges of automatic partitioning between trusted and untrusted code in Intel SGX enclaves. However, unlike Glamdring, Uranus tries to enforce the trusted and untrusted code partition at runtime. Conversely, Montsalvat [56] provides an automatic partitioning tool for GraalVM images that automatically annotates trusted and untrusted code to be computed inside trusted execution environments.

Soteria is different from these solutions as it does not try to propose a partitioning tool. Namely, Soteria proposes a specific partitioning scheme for Apache Spark's MLlib.

**Summary.** To the best of our knowledge, Soteria is the first privacy-preserving ML framework that proposes an alternative TEE-based scheme (Soteria-P), which can improve the performance of training and inference workloads by reducing the number of operations done at secure enclaves. Also, it is the first solution that covers a large spectrum of ML exploits (Table 1) and supports various ML algorithms while not changing how users build and run their algorithms within Spark MLlib.

## VII. 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 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%.

## REFERENCES

- [1] C. Brito, P. Ferreira, B. Portela, R. Oliveira, and J. a. Paulo, "Soteria: Preserving privacy in distributed machine learning," ser. SAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 135–142. [Online]. Available: <https://doi.org/10.1145/3555776.3578591>
- [2] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and other, "Stealing machine learning models via prediction apis," in 25th USENIX Security Symposium, 2016.
- [3] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015.
- [4] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018.
- [5] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in NDSS, vol. 4324, 2015, p. 4325.
- [6] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in International Conference on Machine Learning. PMLR, 2016, pp. 201–210.
- [7] Y. Aono, T. Hayashi, L. Wang, S. Moriai et al., "Privacy-preserving deep learning via additively homomorphic encryption," 2017.
- [8] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas et al., "Innovative instructions and software model for isolated execution," 2013.
- [9] T. Hunt, C. Song, R. Shokri, V. Shmatikov et al., "Chiron: Privacy-preserving machine learning as a service," 2018.
- [10] N. Hynes, R. Cheng, and D. Song, "Efficient deep learning on multi-source private data," 2018.
- [11] D. Le Quoc, F. Gregor, J. Singh, and C. Fetzer, "Sgx-pyspark: Secure distributed data analytics," in The World Wide Web Conference, 2019.
- [12] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana et al., "Everything you should know about intel sgx performance on virtualized systems," 2019.
- [13] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa et al., "Opaque: An oblivious and encrypted distributed analytics platform," in 14th USENIX Symposium on Networked Systems Design and Implementation, 2017.
- [14] J. Jiang, X. Chen, T. Li, C. Wang et al., "Uranus: Simple, efficient sgx programming and its applications," in 15th ACM Asia Conference on Computer and Communications Security, 2020.
- [15] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, "Securekeeper: confidential zookeeper using intel sgx," in Proceedings of the 17th International Middleware Conference, 2016, pp. 1–13.
- [16] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," in 6th International Conference on Learning Representations., 2018.
- [17] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha et al., "Exploring connections between active learning and model extraction," in 29th USENIX Security Symposium, 2020.
- [18] L.-S. D. . S. L. Group, "Sgx-spark," <https://github.com/lstds/sgx-spark>, (Accessed on 22/10/2022).
- [19] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan, "Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors," in ACM SIGSAC Conference on Computer and Communications Security, 2017.
- [20] Intel, "Hibench is a big data benchmark suite." <https://github.com/Intel-bigdata/HiBench>, (Accessed on 22/10/2022).
- [21] M. Zaharia, R. S. Xin, P. Wendell, T. Das et al., "Apache spark: a unified engine for big data processing," 2016.
- [22] X. Meng, J. Bradley, B. Yavuz, E. Sparks et al., "Mllib: Machine learning in apache spark," 2016.
- [23] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta et al., "Oblivious multi-party machine learning on trusted processors," in 25th USENIX Security Symposium, 2016.
- [24] M. El-Hindi, T. Ziegler, M. Heinrich, A. Lutsch et al., "Benchmarking the second generation of intel sgx hardware," in Data Management on New Hardware, 2022.
- [25] T. Alves, "Trustzone: Integrated hardware and software security," White paper, 2004.
- [26] M. Azure, "Azure confidential computing," <https://azure.microsoft.com/en-us/solutions/confidential-compute/>, (Accessed on 22/10/2022).
- [27] S. Iqbal, M. L. M. Kiah, B. Dhaghghi, M. Hussain, S. Khan, M. K. Khan, and K.-K. R. Choo, "On cloud security attacks: A taxonomy and intrusion detection and prevention as a service," 2016.
- [28] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. 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.
- [29] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in 5th International Conference on Learning Representations, 2017.
- [30] J.-B. Truong, P. Maini, R. J. Walls, and N. Papernot, "Data-free model extraction," in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021.
- [31] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in Symposium on Security and Privacy (SP), 2017.
- [32] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions." IEEE, 2019.
- [33] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and other, "Updates-leak: Data set inference and reconstruction attacks in online learning," in 29th USENIX Security Symposium, 2020.
- [34] E. Stefanov, M. V. Dijk, E. Shi, T.-H. H. Chan et al., "Path oram: an extremely simple oblivious ram protocol," 2018.
- [35] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer, "Varys: Protecting sgx enclaves from practical side-channel attacks," in USENIX Annual Technical Conference, 2018.
- [36] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in 42nd IEEE Symposium on Foundations of Computer Science, 2001.
- [37] R. Bahmani, M. Barbosa, F. Brasser, B. Portela et al., "Secure multiparty computation from sgx," in International Conference on Financial Cryptography and Data Security. Springer, 2017.
- [38] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: A practical library os for unmodified applications on sgx," in USENIX Annual Technical Conference, 2017.
- [39] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in 39th International ACM SIGIR conference on Research and Development in Information Retrieval, 2016.
- [40] T. Elgamal and M. Hefeeda, "Analysis of pca algorithms in distributed environments," 2015.
- [41] S. Si, H. Zhang, S. Keerthi, D. Mahajan, I. Dhillon, and C.-J. Hsieh, "Gradient boosted decision trees for high dimensional sparse output," in International conference on machine learning, 2017.
- [42] J. Dowling, "Distributed ml and linear regression," <https://www.kth.se/social/files/5a040fe156be5be5f93667e9/ID2223-02-ml-pipelines-linear-regression.pdf>, november 2017, (Accessed on 01/12/2020).
- [43] C. Elkan, "Boosting and naive bayesian learning," in International Conference on Knowledge Discovery and Data Mining, 1997.
- [44] D. Cai, X. He, and J. Han, "Training linear discriminant analysis in linear time," in 2008 IEEE 24th International Conference on Data Engineering. IEEE, 2008.
- [45] M. K. Pakhira, "A linear time-complexity k-means algorithm using cluster shifting," in 2014 International Conference on Computational Intelligence and Communication Networks. IEEE, 2014.
- [46] C. Zhao, D. Saifuding, H. Tian, Y. Zhang, and other, "On the performance of intel sgx," in 13th web information systems and applications conference, 2016.
- [47] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017, pp. 19–38.
- [48] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 619–631.

- [49] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 2015, pp. 1310–1321.
- [50] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 308–318.
- [51] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," arXiv preprint arXiv:1806.03287, 2018.
- [52] Databricks, "Optimizing apache spark udfs," [https://www.databricks.com/session\\_eu20/optimizing-apache-spark-udfs](https://www.databricks.com/session_eu20/optimizing-apache-spark-udfs), (Accessed on 27/10/2022).
- [53] R. Kunkel, D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "Tensorscone: a secure tensorflow framework using intel sgx," arXiv preprint arXiv:1902.04413, 2019.
- [54] J. Lind, C. Priebe, D. Muthukumar, D. O’Keeffe, P. Aublin, F. Kelbert, T. Reiher, D. Goltzsche, D. Eyers, R. Kapitza et al., "Glamdring: Automatic application partitioning for intel sgx." USENIX, 2017.
- [55] C.-C. Tsai, J. Son, B. Jain, J. McAvey, R. A. Popa, and D. E. Porter, "Civet: An efficient java partitioning framework for hardware enclaves," in 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 505–522.
- [56] P. Yuhala, J. Ménétrey, P. Felber, V. Schiavoni, A. Tchana, G. Thomas, H. Guiroux, and J.-P. Lozi, "Montsalvat: Intel sgx shielding for graalvm native images," in Proceedings of the 22nd International Middleware Conference, 2021, pp. 352–364.
- [57] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from sgx," in Financial Cryptography and Data Security, A. Kiayias, Ed. Cham: Springer International Publishing, 2017, pp. 477–497.

## APPENDIX A SOTERIA SECURITY 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 real-world behavior displayed by Soteria 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, similar to the Universal Composability [36] 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. FORMAL SECURITY MODEL

Our model considers external environment  $\mathcal{Z}$  and internal adversary  $\mathcal{A}$ .  $\Pi$  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 global storage  $G$ , which is initialized by  $\mathcal{Z}$  before starting the protocol. The Ideal functionality is parametrised by this external storage  $\mathcal{F}\langle G \rangle$ , and will reveal the access patterns via leakage function  $\mathcal{L}$ .<sup>2</sup>

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.<sup>3</sup> The Client will then execute protocol  $\Pi$ , sending messages to  $M$ ,  $W_1$  and  $W_2$ . When the script is concluded, the 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  $\mathcal{F}\langle G \rangle$ . 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 output size.

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

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

### B. 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 external storage from which they retrieve the 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 a secure channel (Theorem 3 in [57]). The reasoning is simply that if a protocol securely realizes a functionality with a given input provided via a secure channel, then the same functionality can be securely realized with the same input fixed in an external storage, securely accessed by the enclave.

Consider a protocol  $\Pi_1$  that securely realises some functionality  $\mathcal{F}$  with simulator  $\mathcal{S}_1$  according to Theorem 3 of [36]. We construct protocol  $\Pi_2$  built on top of this secure protocol  $\Pi_1$ , where input data is pre-established and provided to the enclave via an initial Setup stage where inputs are stored in

<sup>2</sup>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.

<sup>3</sup>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  $\mathcal{F}$ .

```

Algorithm Setup( $i, m$ ) $\langle G \rangle$ :
 $k \leftarrow \Theta.Gen()$ 
 $c \leftarrow \Theta.Enc(k, i)$ 
 $G[m] \leftarrow c$ 
Return ( $m, k$ )
    
```

FIGURE 7. Secure external storage setup.

an encrypted fashion (Figure 7 describes a simplified version of the process for a single entry). Inputs to  $\Pi_2$  are exactly the same as those for  $\Pi_1$ , 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  $\Theta$ . This protocol behavior will be key for all Soteria Workers. Our theorem is as follows.

*Theorem 1:* Let  $\Pi_1$  be a protocol that securely realises functionality  $\mathcal{F}$  according to Theorem 3 in [57]. Then  $\Pi_2$ , 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  $\Pi_1$ , then the simulated view must be indistinguishable from the one of the real world of  $\Pi_2$ .

We begin by deconstructing  $\mathcal{S}_1$  in two parts:  $\mathcal{S}_1.AC()$  will produce the view for establishing a secure channel, while  $\mathcal{S}_1.Send(l)$  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  $\mathcal{S}_2.Get(l)$  to simulate information being retrieved from  $G$ , given its length. Our simulator is depicted in Figure 8.

<b>Algorithm</b> AC()	<b>Algorithm</b> Send( $l$ )	<b>Algorithm</b> Get( $l$ )
$k \leftarrow \Theta.Gen()$	Return $\mathcal{S}_1.Send(l)$	$i \leftarrow \{0\}^l$
Return $\mathcal{S}_1.AC()$		$c \leftarrow \Theta.Enc(k, i)$
		Return $c$

FIGURE 8. Simulator for  $\Pi_2$ .

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  $\Pi_1$ . Thus they remain indistinguishable from  $\Pi_2$ .
- Outputs received via the secure channel follow the exact same simulation strategy than  $\Pi_1$ , and thus are indistinguishable from  $\Pi_2$ .
- Messages produced from  $G$  in  $\Pi_2$  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  $\Theta$ , which is negligible.

As such, if  $\mathcal{S}_1$  is a valid simulator for  $\Pi_1$  to  $\mathcal{A}$ , then the view presented by  $\mathcal{S}_2$  must also be indistinguishable for  $\Pi_2$  to  $\mathcal{A}$ .

Let

$$Adv^{Dist}_{\mathcal{Z}, \mathcal{A}, \Pi, \mathcal{S}, \mathcal{F}}(G) = |\Pr[Real_{\mathcal{Z}, \mathcal{A}, \Pi_2}^G \Rightarrow T] - \Pr[Ideal_{\mathcal{Z}, \mathcal{A}, \mathcal{S}_2, \mathcal{F}}^G \Rightarrow T]| \quad (1)$$

$$\leq \mu() \quad (2)$$

To conclude, we have that, for negligible function  $\mu$ ,

$$Adv^{Dist}_{\mathcal{Z}, \mathcal{A}, \Pi_2, \mathcal{S}_2, \mathcal{F}}(G) = Adv^{Dist}_{\mathcal{Z}, \mathcal{A}, \Pi_1, \mathcal{S}_1, \mathcal{F}}() + Adv^{IND-CPA}_{\Theta, \mathcal{A}}() \quad (3)$$

$$\leq \mu() \quad (4)$$

and Theorem 1 follows.

### C. 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  $init(P)$  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  $uGet(G, m)$ , which retrieves data from  $G$  considering manifest file  $m$ . Concretely, this is achieved using the open-source library Graphene-SGX, which we assume to implement this mechanism correctly. 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  $Run(id, i_1, i_2)$ , 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 analyzed in Figure 9 and are as follows. The Client  $C$  (left of Figure 9) 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 9) 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.<sup>4</sup> Worker  $W_1$  (right of Figure 9) 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 the connection is established instead with  $W_1$ .

<sup>4</sup>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.

<b>Algorithm</b> $C(m, s, k)$	<b>Algorithm</b> $M()$	<b>Algorithm</b> $W_1()$
$sc \leftarrow \text{init}(M)$	$sc_c \leftarrow \text{init}(C)$	$m \leftarrow \epsilon$
$sc.\text{send}(m, s, k)$	$(m, s, k) \leftarrow$	$sc_m \leftarrow \text{init}(M)$
$o \leftarrow sc.\text{receive}()$	$sc_c.\text{receive}()$	$(m, s, k) \leftarrow sc_m.\text{receive}()$
Return $o$	$sc_1 \leftarrow \text{init}(W_1)$	$sc_w \leftarrow \text{init}(W_2)$
	$sc_1.\text{send}(m, s, k)$	While $!s.\text{Complete}$ :
	$sc_2 \leftarrow \text{init}(W_2)$	$c \leftarrow \text{uGet}(G, m)$
	$sc_2.\text{send}(m, s, k)$	$i \leftarrow \Theta.\text{Dec}(k, c)$
	$o_1 \leftarrow sc_1.\text{receive}()$	$(o, m) \leftarrow s.\text{Run}(W_1, i, \epsilon)$
	$o_2 \leftarrow sc_2.\text{receive}()$	$sc_w.\text{send}(m)$
	$sc_c.\text{send}((o_1, o_2))$	$m \leftarrow sc_w.\text{receive}()$
		$sc_m.\text{send}(o)$

FIGURE 9. SOTERIA Components. Client  $C$  (left), Master node  $M$  (middle), and Worker node 1  $W$  (right).

## D. FULL PROOF

Given the description of Soteria components in Figure 9, the Soteria protocol  $\Pi_{xyz}$  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:**  $\Pi_{xyz}$ , assuming the setup of Figure 7 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  $\mathcal{Z}$ .

<b>Algorithm</b> $W_1()$	<b>Algorithm</b> $W_2()$
$m \leftarrow \epsilon$	$m \leftarrow \epsilon$
$sc_m \leftarrow \text{init}(M)$	$sc_m \leftarrow \text{init}(M)$
$(m, s, k) \leftarrow sc_m.\text{receive}()$	$(m, s, k) \leftarrow sc_m.\text{receive}()$
$sc_w \leftarrow \text{init}(W_2)$	$sc_w \leftarrow \text{init}(W_1)$
While $!s.\text{Complete}$ :	While $!s.\text{Complete}$ :
$c \leftarrow \text{uGet}(G_1, m)$	$c \leftarrow \text{uGet}(G_2, m)$
$i \leftarrow \Theta.\text{Dec}(k, c)$	$i \leftarrow \Theta.\text{Dec}(k, c)$
$(o, m) \leftarrow s.\text{Run}(W_1, i, \epsilon)$	$(o, m) \leftarrow s.\text{Run}(W_1, i, \epsilon)$
$sc_w.\text{send}(m)$	$sc_w.\text{send}(m)$
$m \leftarrow sc_w.\text{receive}()$	$m \leftarrow sc_w.\text{receive}()$
$sc_m.\text{send}(o)$	$sc_m.\text{send}(o)$

FIGURE 10. SOTERIA Workers with split storage.

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 10 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  $\mathcal{F}_{W_1}$ , 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 realizes the same functionality. However, this is necessarily the case, as it follows the exact structure as the

constructions in [57]. We can repeat this process for  $W_2$ .<sup>5</sup> 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 with 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 external storage, as the protocol also follows the exact structure as the constructions in [57].

In the final step, we have 3 functionalities  $(\mathcal{F}_M, \mathcal{F}_{W_1}, \mathcal{F}_{W_2})$  playing the roles of  $(M, W_1, W_2)$ , respectively. We finalize by combining them into 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  $(S_M, S_{W_1}, S_{W_2})$ . 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  $S_{W_1}$  and  $S_{W_2}$  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  $S_{W_1}$  or  $S_{W_2}$  to emulate the computation stage.

Given that the view produced by  $S$  is exactly the same as the one provided by the combination of  $S_M, S_{W_1}$ , and  $S_{W_2}$ , the adversarial advantage is exactly 0.

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

Let

$$\text{Adv}_{\mathcal{Z}, \mathcal{A}, \Pi, S, \mathcal{F}}^{\text{Dist}}(G) = |\Pr[\text{Real}_{\mathcal{Z}, \mathcal{A}, \Pi, S, \mathcal{F}}^G \Rightarrow \top] - \Pr[\text{Ideal}_{\mathcal{Z}, \mathcal{A}, S, \mathcal{F}}^G \Rightarrow \top]| \quad (5)$$

To conclude, we have that, for the negligible function  $\mu$ ,

$$\text{Adv}_{\mathcal{Z}, \mathcal{A}, \Pi, S, \mathcal{F}}^{\text{Dist}}(G) = 2 \cdot \text{Adv}_{\mathcal{Z}, \mathcal{A}, \Pi, W_1, S_{W_1}, \mathcal{F}_{W_1}}^{\text{Dist}}(G) \quad (7)$$

$$+ \text{Adv}_{\mathcal{Z}, \mathcal{A}, \Pi, M, S_M, \mathcal{F}_M}^{\text{Dist}}() \quad (8)$$

$$\leq \mu() \quad (9)$$

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

<sup>5</sup>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.

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.

### A. 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 a 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.

<p><b>Game</b> AdvXYZ<sub>A,Π<sub>xyz</sub></sub>(<math>G, s</math>):</p> <p><math>(G') \leftarrow \\$ \mathcal{A}_1(G, s)</math></p> <p><math>(m, l) \leftarrow \Pi_{xyz}(G', s)</math></p> <p><math>r \leftarrow \\$ \mathcal{A}_2^m(l)</math></p> <p>Return Success(<math>G, s, m, r</math>)</p>
---

FIGURE 11. Adversary interacting with SOTERIA.

ff

Let  $\Pi_{xyz}$  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 11. 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  $\Pi_{xyz}$ , 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 influence 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. DATASET MANIPULATION

Dataset manipulation attacks are defined by an adversary with the capability of inserting, removing or manipulating dataset information. These align with the setting considered for attacks via adversarial samples. Figure 12 is an experiment that describes what constitutes a successful attack for dataset manipulation. The adversary  $\mathcal{A}$  is given full knowledge of  $G^6$ , and must produce an alternative input dataset  $G'$ . We then train the model (protocol  $\Pi$ ) over that data to produce model  $m$ , and the adversary is successful if said model satisfies some attack success criteria  $T/F \leftarrow \text{Success}$ .

<p><b>Game</b> DSetMan<sub>A,Π</sub>(<math>G, s</math>):</p> <p><math>G' \leftarrow \\$ \mathcal{A}(G, s)</math></p> <p><math>m \leftarrow \Pi(G', s)</math></p> <p>Return Success(<math>G, s, m</math>)</p>
--

FIGURE 12. Model for dataset manipulation attack.

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  $\Pi_{xyz}$ . We do this by showing that any adversary that performs an adversarial samples attack on  $\Pi_{xyz}$  can be used to construct a successful attack on 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  $F \leftarrow \text{Success}$ . Furthermore, if  $\Pi_{xyz}$  aborts, then no model is produced, so it naturally follows that the attack is unsuccessful  $F \leftarrow \text{Success}$ .<sup>7</sup>

As such, the only cases in which  $T \leftarrow \text{Success}$  are those in which  $G' \neq G$  and  $\Pi_{xyz}$  do 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 be negligible.

### C. 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 13. We begin by running  $\Pi$  to produce our model and leakage and then run an additional procedure 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

<sup>6</sup>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.

<sup>7</sup>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.



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 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 [17]. We now go case-by-case, assuming an adversary can play the role of a genuine client in our system.

Our arguments for  $\Pi_{xyz}$  depend on being able to rely on a successful adversary  $\mathcal{A}$  of Figure 13 to perform the same attack in Figure 11. 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 13 will also be successful in Figure 11, 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  $z$ . [2] suggests that such information is not retrievable from only black-box access to the model [2], 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  $\Pi_{xyz}$  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  $\Pi_{xyz}$ . Otherwise, Soteria-B is secure, as no additional information is leaked. Furthermore, the security of Soteria-P will depend on whether one can 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  $z \leftarrow F^m(l)$ , then  $\Pi_{xyz}$  for leakage  $l$  is secure against attacks requiring additional data  $z$ .

**Game**  $\text{BlackBox}_{\mathcal{A}, \Pi_{xyz}}(G, s)$ :

$m \leftarrow \Pi(G, s)$

$z \leftarrow \text{Extract}(G, s)$

$r \leftarrow \mathcal{A}^m(z)$

Return  $\text{Success}(G, s, m, r)$

FIGURE 13. Model for black-box attacks.

#### D. 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 require white-box access to the model. Figure 14 is an experiment that describes what constitutes a successful reconstruction attack in this context. We begin by training the model (protocol  $\Pi$ ) 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.

**Game**  $\text{WhiteBox}_{\mathcal{A}, \Pi_{xyz}}(G, s)$ :

$m \leftarrow \Pi(G, s)$

$r \leftarrow \mathcal{A}(m)$

Return  $\text{Success}(G, s, m, r)$

FIGURE 14. Model for white-box attacks.

We now argue that these attacks do not occur for  $\Pi_{xyz}$ , 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 14 receives explicitly the model  $m$ , whereas the adversary  $\mathcal{A}_2$  in Figure 11 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  $m \leftarrow \mathcal{A}_{bb}^m(l)$  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  $\mathcal{A}_{bb}$  can then be called by  $\mathcal{A}_2$  to produce input  $m$ , which is then forwarded to the adversary of Figure 14 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 13 for the model extraction attack.

**E. SUMMARY**

Table 3 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 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 [33]. 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 [17] 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. [2] suggests {4a} is true, but {4b} is not. {5} will fundamentally depend on the application [2], [17].



CLÁUDIA BRITO is currently a Ph.D. student in the Doctoral Program in Informatics at the University of Minho and a Researcher at HASLab, one of the research units of INESC TEC and the University of Minho. Her current research interests rely on the intersection of machine learning and privacy-preserving techniques, as well as the application of such solutions in the biomedical field. For more information, please visit:

<https://www.inesctec.pt/en/people/claudia-vanessa-brito>



PEDRO G. FERREIRA received a Ph.D. in Artificial Intelligence from the University of Minho in 2007. He was a Postdoctoral Fellow at the Center for Genomic Regulation, Barcelona (2008-2012) and at the University of Geneva (2012-2014). Currently, he is an Assistant Professor at the Department of Computer Science, Faculty of Sciences of the University of Porto and a researcher at INESC TEC-LIADD and i3s/Ipatimup. His main research focus is in genomic data science. In particular, he is interested in unraveling the role of genomics on human health and disease. For more information, please visit:

<https://www.inesctec.pt/en/people/pedro-gabriel-ferreira>



BERNARDO L. PORTELA received a Ph.D. under the MAP-i Doctoral Programme from the Universities of Minho, Portugal, Aveiro, and Porto, in 2018. He is an Assistant Professor at the Department of Computer Science, Faculty of Sciences of the University of Porto and a researcher at HASLab, one of the research units of INESC TEC. His research interests rely on cryptography and information security, more specifically regarding secure multiparty computation protocols relying on trusted hardware. For more information, please visit:

<https://www.inesctec.pt/en/people/bernardo-luis-portela>



RUI C. OLIVEIRA received the Ph.D. degree in 2000 from the École Polytechnique Fédérale de Lausanne. He is an Associate Professor with Habilitation at the Informatics Department of the University of Minho, a member of the Board of Directors of INESC TEC, director of the Minho Advanced Computing Centre, and co-director of the UT Austin Portugal program. His research contributions have been in the field of fault-tolerant distributed agreement and epidemic communication algorithms and in the conception, development, and assessment of dependable database systems. For more information, please visit:

<https://www.inesctec.pt/en/people/rui-carlos-oliveira>



JOÃO T. PAULO received the Ph.D. degree, in the context of the MAP-i Doctoral Programme from the Universities of Minho, Portugal, Aveiro, and Porto, in 2015. He is an Assistant Professor at the University of Minho, Portugal, and a senior researcher at HASLab, one of the research units of INESC TEC and the University of Minho in Portugal. His research is focused on large-scale distributed systems with an emphasis on storage and database systems' scalability, performance, and dependability. For more information, please visit:

<https://www.inesctec.pt/en/people/joao-tiago-paulo>

...

		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 extraction	Equation	✓	{2}	{3a}	{3b}
	Path	✓	{2}	{4a}	{4b}
	Class	✓	{2}	{5a}	{5b}
	DF KD	✓	{2}	✗	✗

**TABLE 3.** 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.