SOTERIA: Privacy-Preserving Machine Learning for Apache Spark

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

Abstract—The adoption of third-party secure machine learning (ML) cloud services is highly dependent on the security guarantees provided, and on the performance penalty they incur on workloads for model training and inference. In this paper, we explore 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). In turn, this allows us to considerably improve performance of privacy-preserving solutions, without exposing the protocol to pervasive ML attacks.
We propose SOTERIA, a system for distributed privacy-preserving ML that leverages Trusted Execution Environments (*e.g.* Intel SGX). Unlike in previous work, where all ML-related computation is performed at trusted enclaves, we follow this insight to have a hybrid deployment, combining computation done inside and outside these enclaves. The conducted experimental evaluation validates that our approach improves the runtime performance of distinct ML algorithms by up to 1.7x, when compared to previous related work. Our protocol is accompanied by a security proof, as well as a discussion regarding resilience against a wide spectrum of ML attacks.

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

1 INTRODUCTION

The ubiquitous environment provided by cloud computing providers offers a scalable, reliable, and performant solution to deploy machine learning (ML) workloads, which typically requires a considerable amount of computational capabilities to provide timely results. However, many of these workloads operate over users' sensitive information (*e.g.*, medical records). Therefore, outsourcing ML data storage and computation to third-party cloud services leaves users vulnerable to attacks that may compromise the integrity and confidentiality of their data. This exposes the users of this paradigm to critical ethical concerns, and is not compliant with regulations such as HIPAA and GDPR [1].

The common machine learning workflow includes two main phases: data training and data inference. During such phases, users' data is susceptible to several attacks, such as *adversarial attacks*, *model extraction* and *inversion*, and *reconstruction attacks* [2], [3], [4]. Many of these attacks can be prevented by using cryptographic schemes that enable privacy-preserving computation over sensitive data, such as homomorphic encryption or secure multi-party computation [5], [6]. However, these impose a significant performance toll that restricts their applicability to practical scenarios [7].

Trusted Execution Environments (TEEs) (*e.g.*, Intel SGX (SGX) [8], AMD-SEV [9], ARM TrustZone [10]) present an hardware-based alternative solution to deliver a secure processing environment where data can be handled in its original form (*i.e.*, plaintext) at untrusted servers. State-of-the-art solutions follow a straightforward design where ML workloads are fully deployed inside TEE enclaves [11], [12], [13]. However, as the amount of computational and I/O operations done at the enclaves increases, the performance of ML training and inference is affected negatively, thus limiting, once again, the applicability of these solutions to practical scenarios [14].

This paper builds upon the insight that such ML performance could be improved if one can reduce the amount of operations done at enclaves. However, choosing the correct set of operations to run outside of TEEs is a very complex task. Ideally, these operations: *i)* should significantly reduce the enclaves' computational and I/O load for different ML workloads; and *ii)* must not leak critical information during the execution of the aforementioned ML attacks. We argue that statistical operations (*i.e.*, the calculation of confidence results, table summaries, ROC/AUC curves, and probability distributions for classes) are good candidates to be offloaded from enclaves. We support this claim by analyzing and detailing its security and performance implications for different ML workloads and attacks.

To demonstrate the feasibility of our approach, we propose SOTERIA, a system for distributed privacy-preserving machine learning, that leverages the scalability and reliability provided by Apache Spark and its ML library (MLlib). Unlike previous solutions [15], [16], SOTERIA supports a wide variety of ML algorithms without changing how users build and run these within Apache Spark. It is also the first TEE-based design to facilitate the exploration of this security/performance interplay, by enabling fine-grained configurations regarding which operations and information are to be processed within protected enclaves.

SOTERIA ensures that critical operations, which enable existing attacks to reveal sensitive information from ML datasets and models, are only performed on secure enclaves.

- *C. Brito, R. Oliveira and J. Paulo are with INESC TEC & University of Minho. E-mail: claudia.v.brito@inesctec.pt, joao.t.paulo@inesctec.pt, rcmo@inesctec.pt*
- *P. Ferreira and B. Portela are with INESC TEC & Faculty of Sciences, University of Porto.E-mail: pferreira@ipatimup.pt, bernardo.portela@fc.up.pt.*

This means that the sensitive information only exists in plaintext form when inside the enclave, while being encrypted in the remainder data flow (*e.g.*, network, storage). This design enables robust security guarantees, protecting the confidentiality of ML training and inference workloads while covering a larger spectrum of attacks than in current related work [11], [12], [15].

In order to showcase the performance benefits of differentiating the ML operations done at secure enclaves, SOTERIA also supports an alternative design, similar to the ones proposed in previous work, where the full set of ML operations are deployed at enclaves. Furthermore, we compare our solution with a non-secure deployment of Apache Spark and the state-of-the-art SGX-Spark solution [15]. Our experiments, resorting to the HiBench benchmark [17] and including seven different ML algorithms, show that SOTERIA outperforms SGX-Spark by up to 1.7x, while offering stronger security guarantees.

In summary, we present the following contributions:

- SOTERIA, a provably secure privacy-preserving system based on Apache Spark that leverages TEEs for the secure execution of ML workloads.
- An open-source prototype[1] that resorts to the Graphene-SGX Library OS [18] to ease the integration of Intel SGX within Apache Spark's workflow.
- An extensive evaluation comparing the performance of SOTERIA with a baseline (non-secure) deployment of Apache Spark and state-of-the-art SGX-based solutions.

2 BACKGROUND

2.1 Apache Spark and MLlib

Apache Spark is a distributed cluster computing framework, which uses in-memory data processing engines that support Extraction, Treatment, and Loading (ETL), analytical, machine learning, and graph processing over large volumes of data. Spark can be deployed on a cluster of servers that may access several data sources (*e.g.*, HBase, HDFS) for reading the data to be processed and store the corresponding output and logs [19]. Spark is commonly compared to Hadoop's two-stage disk-based MapReduce computation engine. However, it is able to perform most of the computation in-memory, thus promoting better performance for data-intensive applications such as machine learning ones [19]. Spark follows a distributed architecture composed of a Master and several Worker nodes, typically deployed across distinct cluster servers.

The MLlib library [20] enables Spark users to build end-to-end machine learning workflows. This is depicted in Figure 1, which divides the two main phases of ML into 5 stages. The first stage goes from the collection of data to its treatment. In the second stage, while transforming data into train and test datasets, it is chosen or developed a new or pre-defined (*e.g.*, Logistic Regression, Random Forest) ML algorithm. The third stage is the training stage, where data is iterated to deliver an optimized trained model, in the fourth stage. In the fifth stage, the trained model can then be saved (persisted) and loaded (accessed) for inference purposes.

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

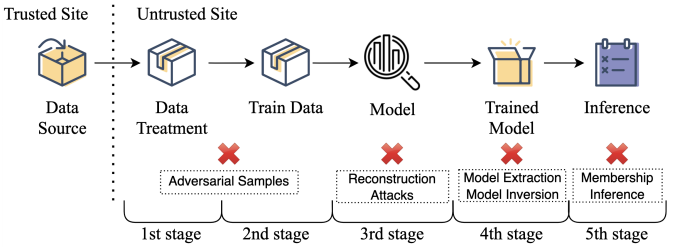


Fig. 1: Machine learning pipeline and attacks defined within our security and threat model.

2.2 Intel Software Guard Extensions

Intel SGX provides a set of new instructions, available on Intel processors, that applications can use to create protected and trusted memory regions. These regions (enclaves) are isolated from any other code at the host system, thus not allowing other processes, even the ones running at higher privilege levels, to access their content [8], [21]. To ensure that the desired computation was correctly executed in a secure enclave, the integrity of computational results, as well as the identity of the enclave, can be verified remotely via the remote attestation mechanism provided by SGX [22].

Since SGX protects code and data from privileged access (i.e, host OS, hypervisor, BIOS), sensitive plaintext information can be processed at the enclave without compromising its privacy. Therefore, TEEs enable better performance than traditional cryptographic computational techniques (*e.g.*, searchable encryption, homomorphic encryption) [23]. Nonetheless, the enclave has limited memory capacity (128MB per CPU) before requiring memory swapping, which is a costly mechanism in terms of performance [14]. Thus, solutions resorting to SGX must balance the number of I/O operations and the amount of data handled by enclaves as well as the Trusted Computing Base (TCB) in order to optimize performance.

In this paper, we chose Intel SGX over other TEE's (*e.g.*, ARM TrustZone [10]) due to its broad use in academia [13], [24], [25] 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. Similarly, novel research and optimizations for SGX, which are orthogonal to the work discussed in this paper and that solve issues such as *Denial of Service (DoS)*, *side-channel* attacks, or *memory access patterns*, can be applied to SOTERIA [27], [28].

3 APPLICATIONS AND SECURITY MODEL

SOTERIA is designed to enable the secure outsourcing of ML training and inference workloads. These are scenarios where the data owner holds sensitive information (*e.g.*, a private dataset, model), and wants to perform some ML workload on it, using an external cloud provider.

This is a very common setting in health-related contexts, such as rare disease identification [29]. Hospitals and clinics are unwilling to (or outright unable to) offload patient data to a potentially malicious cloud service, even if such service provides computational resources to feasibly process large quantities of data. Alternatively, insurance companies also collect large quantities of data and leverage it for the

forecasting of traffic accidents [30]. The company would also significantly benefit from using the resources of third-party cloud infrastructures, but risking the leakage of said predictive model entails losing valuable market advantage.

Therefore, this paper addresses the scenario where the data owner (client) must be protected against a malicious adversary at the cloud provider (server), which is targeting its data, the generated model, and the queries performed over it. Crucially, we assume that the data owner will not actively compromise the protocol, either by inputting false data to the model training, or performing malicious queries to the model on behalf of the adversary. Essentially, we assume that the data owner is responsible to enforce standard mechanisms for data access and control over its own machine. These kind of access control mechanisms are orthogonal to this work, as they are widely used, and are adequately documented by previous research [31].

3.1 Security

Many applications have demonstrated how Intel SGX can be used to design provably secure solutions for secure outsourced computation. Our work builds upon the work of Bahmani et. al. [32], where Intel SGX behaves as a trusted anchor in untrusted environments for general secure computation. Fundamentally, their guarantees rely on building protocols for secure computation following a very specific construction: a bootstrapping stage, where the client establishes a secure channel with the remote enclave, and an online stage, where the client provides inputs and receives outputs via this secure channel. Here, the enclave is simply executing a pre-determined, arbitrary function. In this work, we extend this notion by considering the access of enclave workers to external untrusted storage. This is achieved via standard mechanisms for authenticated encryption, using a key provided by the client via secure channels. We demonstrate that this entails the same level of security.

SOTERIA is designed to behave as a service for privacy-preserving machine learning. As such, we assume the following deployment model. The client (data owner) will be trusted. It will provide the input for processing, and submitting queries for machine learning tasks. Then, we will have a Master node, and N Worker nodes. These 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 has stored its input data securely within this backend, which is also considered to be untrusted throughout the protocol execution.

The main security goal is to ensure that clients can provide input data for training and inference in a way that is not vulnerable to breaches in confidentiality, for both Master and Worker nodes. Concretely, we want our system to behave as a black-box for executing ML scripts according to Apache Spark specifications.

Our model considers semi-honest adversaries, which means that security is discussed considering an adversary that attempts to break the confidentiality of data and model, but that will not actively deviate from the protocol specification. This is a common setting for cloud-based systems, where vulnerability data breaches allow for malicious entities to temporarily read internal processing data. Nevertheless, we provide additional counter-measures against adversarial queries and the injection of data samples into the storage backend or the model being trained.

However, we assume that the number and duration of computation steps, and the size of output data, are explicit leakage. This is because, despite using secure channels, all of these parameters can be inferred by an adversary observing network communication between Spark Nodes and their storage accesses. Ensuring the privacy of such data would entail additional steps to enforce constant-round execution and fixed-size outputs, which would significantly reduce the performance of ML workloads [33]. We consider this relevant future work but outside the scope of this paper.

3.2 Machine Learning Workflow Attacks

SOTERIA is aimed at scenarios where machine learning pipelines are outsourced to untrusted environments. By default, ML datasets and models are stored and processed in plaintext, which leads to the leakage of sensitive information to adversaries at untrusted premises. However, even if this information is encrypted, there are other attacks that may compromise the security of ML at different stages, namely Adversarial Attacks, Model Extraction, Model Inversion and Membership Inference and, Reconstruction Attacks, as depicted in Figure 1.

Adversarial Attacks. These attacks can be decomposed into five categories, where the main goal is to inject adversarial samples. Data poisoning relies on the injection of adversarial samples directly on the training dataset, to intentionally control the model and make it incorrectly label samples. This requires direct access to the training dataset [34]. Gradient-based attacks require an adversary to have detailed information of the models, including input data and model gradients. Score-based attacks rely on adversarial knowledge of the predicted results and class probabilities (*i.e*, inferring the probabilities of each class label). Transfer-based attacks require the attacker to know the full training dataset, which is then used to train a similar model able to replace the original one. Finally, decision-based attacks rely on the adversary observing the final output obtained by the model, e.g., the top-1 class label [35].

Model Extraction. In this attack, the adversary learns a close approximation of the objective function of the trained model (f), finding $f'(x) = f(x)$. This close approximation is based on the exact confidence values and response labels obtained by inference [2]. Briefly, an adversary queries the system in a black-box manner, and must achieve its goal with as few queries as possible.

This class of attacks can be decomposed in four types. Equation-solving attacks are based on class probabilities and confidence values and require to know the dimension of the original training dataset[2], and are performed by running $d + 1$ queries on the d-dimensional input vector x. Equation-solving is not applicable to decision trees. Path-finding attacks are focused on decision trees, and are performed by querying the tree to recreate the path of each leaf, requiring the adversary to have knowledge on the tree

2. State-of-the-art attacks rely on publicly available datasets to which adversaries have full access to.

and data features, and confidence values. Class-only attacks target binary models, and require an adversary with concrete samples from the training dataset [2]. Recently, data-free knowledge distillation (DFKD) has been proposed for performing model extraction attacks. This attack has shown to be effective with only knowledge of publicly available information from training data, namely statistics information (*e.g.,* gender and/or demographic statistics) [36].

Model Inversion and Membership Inference. While model extraction aims to retrieve an approximation of the model, model inversion and membership inference target the recovery of values from the training dataset. Both attacks consider an adversary that queries the ML system in a black-box fashion. Model inversion attacks are currently based on ML services, which define publicly their trained models (e.g., BigML explains its trained models on their website[3]) and the confidence values. Alongside the use of ML services, to accomplish the attack proposed by [3], the attacker must have partial knowledge of the training dataset's features to infer and query the model with specific queries. Unfortunately, these features tend to be considered as non-sensitive (*e.g., age, gender*) and may be public, which facilitates their access to a potential adversary [2].
Membership inference aims to test if a specific data point d was used as training data. These attacks are also based on publicly accessible ML services and require the adversary to know a subset of samples used for training the model (that does not contain d). The adversary then compares the predictions between these data points and others that the model might have not encountered during training, to infer if d was in the original training dataset [37].

Reconstruction Attacks. In this attack, the adversary intends to reconstruct raw data used for training the model. Most works claim that the main difference from model inversion is that the adversary must have white-box access to the model since it requires access to model-specific information such as feature vectors. The reconstruction attack is most common for ML algorithms that use feature vectors, such as Support Vector Machines or K-Nearest Neighbor [38]. Nevertheless, [39] has recently proposed a black-box approach to reconstruction attacks. In this setting, the attacker has 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). Also, even though the attacker has black-box access to the model, it needs additional information about the model architecture and its hyperparameters (*e.g.,* learning rate, loss function) which can be obtained by resorting to attacks similar to [4].

Unlike previous works [11], [12], [15], [16], which typically consider a small subset of these attacks, our proposal aims at providing mechanisms that cover the full range of mentioned exploits. Table 1 presents relevant state-of-the-art solutions, the security attacks covered by these, and the attacks addressed by SOTERIA. Our system implements authentication mechanisms to prevent the execution of non-client queries. As such, we assume that the adversary is not given public (black-box) access to the systems, which would make these security mechanisms redundant. Indeed, the other solutions presented at the table also rely on such assumption to deter black-box attacks. Furthermore, attacks that rely on knowing network and storage I/O patterns, or based on the number of computation steps or output sizes, are not covered by SOTERIA. These issues are considered to be orthogonal, as they are common to all systems performing secure computation in untrusted servers. It is possible to remove access pattern leakage from SOTERIA by overlaying mechanisms such as Oblivious RAM [40], however these techniques impose a performance toll that is usually prohibitive for practical deployment.

3. https://bigml.com/api/models

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

Attacks		Systems [11]	[12]	[16]	[15]*	SOTERIA
Adversarial	Gradient-based	✗	✗	✓	✗	✓
	Score-based	✗	✗	✓	✗	✓
	Transfer-based	✗	✗	✓	✗	✓
	Decision-based	✗	✗	✓	✗	✓
Model Extraction	Equation-solving	✓	✓	✗	✓	✓
	Path-finding	✓	✓	✗	✗	✓
	Class-only	✓	✓	✗	✗	✓
	DFKD	✓	✓	✗	✓	✓
Model Inversion		✓	✓	?	✓	✓
Reconstruction Attacks		✓	✓	✓	✓	✓
Membership Inference		✗	✗	✗	✗	✓

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

4 SOTERIA

SOTERIA is a system for distributed privacy-preserving machine learning that avoids changing the architecture and processing flow of Apache Spark and MLlib, keeping their usability, scalability and fault tolerance properties.

4.1 Apache Spark - Architecture and Flow

As depicted in Figure 2 by the gray boxes, a Spark cluster is composed of a Master and several Worker nodes. Before submitting ML tasks (*e.g.,* machine learning training and inference operations) to the Spark cluster, the user must load its local datasets and models to a distributed storage backend supported by Apache Spark. The user can then submit ML processing tasks to the Spark client that is responsible for forwarding these tasks (scripts) to the Master node. Namely, tasks are submitted to the Spark Driver that generates a Spark Context which enables accessing the resource manager and then distributing the tasks to a set of Worker nodes according to their computational resources. A task can be divided into smaller processing steps, each done by an independent Worker in parallel. Also, each Worker launches one or more executors (JVM processes) that do local processing concurrently (*e.g.,* training the ML model and collecting train statistics).

When the Worker nodes require accessing data at the storage backend (*e.g.,* for reading and training an ML dataset, for loading a stored model and data for inference) they usually use an abstraction called *Resilient Distributed Datasets (RDD)* [41]. This representation eases the partitioning of data into shards that, ideally, are collocated with the Worker nodes requesting them, thus improving storage I/O latency. To improve storage performance, data is also

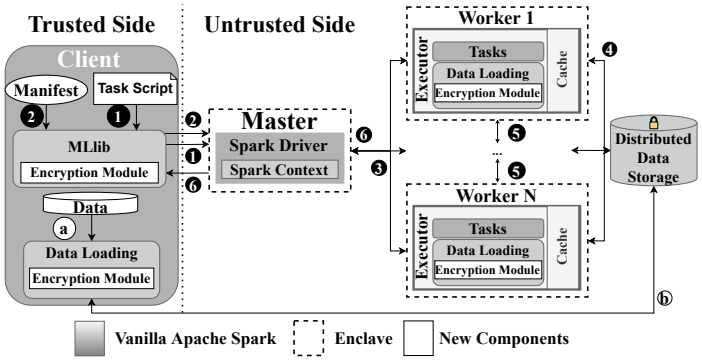


Fig. 2: SOTERIA architecture and operations flow.

kept at an in-memory cache at each executor process. As Workers may be executing different steps of a given task, these need to be able to transfer information (*e.g.*, model parameters), through the network, among each other. After finishing the desired computational steps, the workers send back their outputs to the Master node, which is responsible for merging the outputs and replying to the client.

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

4.2 SOTERIA Client

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

4.2.1 Data Loading

For the first operation, the user must specify the data to be loaded to the storage backend. However, this data has to be encrypted before leaving the trusted user premises. This step is done by extending Spark's data loading module, which is responsible for this operation, with a transparent encryption module (Figure 2-ⓐ). This module is responsible for encrypting the data being loaded into the distributed storage backend with a symmetric-key encryption scheme (Figure 2-ⓑ). Note that the data to be loaded may be the train and validation dataset, for training purposes, or the model and data to be inferred, for inference purposes.

4.2.2 Tasks submission

The task submission phase includes two main files, namely, the ML task script and the *Manifest file*. For this phase, the transparent encryption module is also integrated with MLlib. This module is used to encrypt the ML task script (Figure 2-❶), which may contain sensitive arguments (*i.e,* model parameters) and processing logic, and to decrypt the outputs (*e.g.*, trained model or inference result) returned by Spark's Master node after completing the corresponding tasks. The *Manifest file*, which requires user input, contains the location (*e.g.*, directory) at the storage backend where the corresponding training or inference data is kept (Figure 2-❷). Also, and as further discussed below, the encryption module is responsible for exchanging the user's symmetric encryption key, task's scripts, and *Manifest file* with the Master node's SGX enclave (Figure 2-❶❷). This is done once, at the task's bootstrapping phase, and requires establishing a secure channel between the client and Master's enclave. This secure channel guarantees the secure exchange of the user's encryption key and the *Manifest file* whilst the encrypted tasks scripts can be safely sent via a regular channel.

With the previous design, we ensure that sensitive data is only accessed in its plaintext format at the trusted user premises or inside trusted enclaves. This includes users' encryption keys, the information contained in ML task scripts, and the output of executing these tasks.

4.3 SOTERIA Cluster

As aforementioned, the 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 several Worker nodes. Therefore, 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 needs to access the distributed storage backend to retrieve the stored ML model. The user's encryption key is again necessary so that the encrypted model is only decrypted and processed at the secure enclave. Moreover, the *Manifest file*, also sent by the trusted client through the established secure channel, ensures that the Master node only has access to the storage locations specified at the file (Figure 2-❷). This is important to prevent malicious attackers from accessing stored data (*e.g*, datasets, models) that these should not have access to.

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[4] along with the user's encryption key and *Manifest file* (Figure 2-❸). The user's encryption key is needed at the Worker nodes so that these can read encrypted data (*e.g.*, train dataset or data to be inferred) from the storage backend while decrypting and processing it in a secure enclave environment (Figure 2-❹). The *Manifest file* is used to prevent unwanted accesses to stored data. Furthermore, the enclaves at the Worker nodes also establish secure channels across each other for transferring sensitive metadata information such as model training parameters (Figure 2-❺).

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

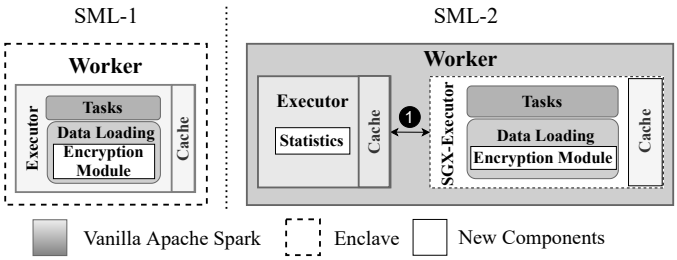


Fig. 3: Comparison between SML-1 and SML-2 designs.

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

4.4 SOTERIA Twofold Design

SOTERIA is fundamentally designed considering two main designs. The first (SML-1) is a straightforward approach, where all ML operations are done within secure enclaves. This is a useful general-purpose solution, and presents a relevant baseline to compare with the novel design in terms of security and performance. The second (SML-2) enables the aforementioned fine-grained configuration of which ML operations are processed within protected enclaves. We stress that SML-1 and SML-2 only differ on the amount of operations done at the enclaves of Spark Worker nodes, while for enclaves deployed at other Spark components (*i.e.*, Master node), the operations are identical.

Secure ML Design 1 (SML-1). In the first design, all computation done by Spark Workers is included in a trusted environment. The executor processes launched by each Worker node are deployed inside an enclave, as depicted in Figure 3. Outside the enclave, data is always encrypted in an authenticated fashion, which allows the Worker to decrypt and validate data integrity within the enclave.

Secure ML Design 2 (SML-2). Our novel design 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 datasets 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 done. SML-2 decouples statistical processing, used for assessing the performance of inference and training tasks, from the actual computation of the ML algorithms done over plaintext information. Concretely, SML-2 reveals the calculation of confidence results (accuracy, precision, recall and F1-scores), table summaries and ROC/AUC curves, and probability distributions for classes. This decoupling builds directly upon MLlib and refactors the needed classes without requiring any changes to the way users submit ML tasks. As depicted in Figure 3, statistical processing is done by executor processes at the untrusted environment, while the remaining processing endeavors are done by another set of executors inside a trusted enclave (Figure 3-❶).

4.5 Security

Formally, our security goal is defined using the real-versus-ideal world paradigm, similarly to the Universal Composability [42] framework. Succinctly, we prove that SOTERIA is indistinguishable from an idealized service for running ML scripts to an arbitrary external environment that can collude with a malicious insider adversary. We then use that abstraction to demonstrate how it prevents real-world attacks. This idealized service is specified as a black-box functionality parametrized with the input data, which simply executes the tasks described in the ML task script, and returns the output to the client via a secure channel. We first discuss the main security argument, and then demonstrate how this idealized black-box ensures effective mitigation of the attacks specified in Section 3.2.

In the real-world, we run SOTERIA as specified in Section 4, and the adversary can observe all messages exchanged between participants, as well as requests to external storage done by Workers. In the ideal world, we instead run the idealized service, and the adversary is presented with a simulated view of the world. The security reasoning is that, if the views provided to the adversary in both worlds are indistinguishable, then SOTERIA reveals nothing more than what would be revealed if we were to use such an idealized service. This entails revealing network and storage I/O patterns, number of computation steps, and output sizes.

Our security proof is dependent on an intermediate result that follows naturally from [32], which states that if a functionality can be securely computed via SGX, then the same functionality using inputs from external storage can also be securely computed. The full security proof of SOTERIA can be found in A. We now outline the intuition for the proof.

The role played by SOTERIA 's Master node can be seen as an extension of the client, establishing secure channels, providing storage encryption keys, and receiving outputs. Given that this follows the methodology described in [32], the Master node can be replaced by a reactive functionality performing the same tasks. Following the same reasoning for the ML processing stage, 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 respective role within the ML task script. At this stage, we have a client interacting with a functionality that forwards encryption keys to other functionalities performing ML processing.

Given that all processing is done in ideal functionalities, and that all access to external storage is fixed by the ML Task Script and the Manifest, we can have the functionalities processing over hard-coded client data, and replace the secure data storage with dummy encryptions. Finally, we can collapse the Master node functionality as part of the Client, since it is simply forwarding client requests. We have now reached the ideal world, where all ML computation is performed in an ideal black-box functionality, and all other

protocol interactions are simulated given the Task Script and Manifest files.

For simplicity, our analysis refers to SML-1. The reasoning for SML-2 is identical, with the caveat that non-sensitive statistical data is explicitly revealed as leakage by the functionality to the simulator in the ideal world. Thus, the proof for SML-2 is analogous, but instead considers an ideal functionality that reveals the confidence results and class probabilities associated with the ML task, which are then used by the simulator to emulate the real-world view.

4.5.1 Relation to ML Attacks and Applications

The previous discussion allows us to reason over ML workflow attacks in a very concrete way. The security of SOTERIA ensures that the adversary's interaction with our system is analogous to observing the idealized service interact with the client via secure channels, with the only additional information received being confidence values and class probabilities. Considering the pervasive pre-requisite for many attacks of having black-box access to the models, a crucial security question to answer is: *how do confidence values and class probabilities relate to black-box model access, i.e. are these equivalent in any way?* Extracting model access from only confidence values is an on-going area of research. However, current attacks suggest one is unable to do this in any efficient way [43], which supports our assumption that revealing these statistical values is insufficient to enable attacks relying on black-box model access.

We now overview the four types of attacks referred in Section 3.2 in a case-by-case basis. An in-depth security analysis considering alternative adversary assumptions in SML-1 and SML-2 can be found in B. The adversary considered in the security proof is semi-honest, which does not allow input forgery. However, this possibility must be considered if SOTERIA is to be resistant against the cloud provider itself. This is achieved by relying on authenticated data encryption. This means that the input dataset is authenticated by the data owner and explicitly defined in the *Manifest file*, allowing enclave Worker nodes to check the authenticity of all input data. Thus, no forged data is accepted for processing, a necessary pre-requisite for performing any type of *adversarial attack*. By relying on secure channels between the TEE at the Master node and the client, an external adversary is prevented from observing legitimate query input/outputs, or submitting forged 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 some instances of *reconstruction attacks* that require black-box access to the model.

Finally, the most common *reconstruction attacks* require white-box access to the model. The previous security result demonstrates that our system does not reveal the trained model, which includes the important feature vectors required for this attack to occur. Alternatively, *reconstruction attacks* requiring black-box access to the model are strictly stronger. However, since only confidence values and class probabilities are revealed by SOTERIA, these are insufficient to emulate black-box access to the model, which excludes the prerequisites for such attacks.

4.6 Implementation

SOTERIA's prototype is implemented using Java and Scala, and is based on Apache Spark 2.3.0. The current implementation requires modifying Spark's data loading and MLlib, while not changing any core modules of the framework.

Spark's data loading library was extended to include transparent encryption and decryption. This library is used by the Client when loading data into the distributed storage backend, submitting ML task scripts to the Master node, and receiving the corresponding inference and training results. Data encryption and decryption are done by resorting to the AES-GCM-128 authenticated encryption cipher mode, which is standardized by NIST and provides both privacy and integrity guarantees [44].

We recall that our current prototype assumes a bootstrapping phase for establishing a secure channel between the trusted Client module and the Master's node SGX enclave, which is used to securely exchange messages between these nodes. The establishment of this secure channel is not currently supported by our prototype, as the main goal of this paper is to analyze the extra performance overhead of doing server-side ML computations with the aid of Intel SGX enclaves. Nevertheless, this channel could be implemented with one of many key exchange protocols for SGX, with minimal performance overhead [32], [45].

However, it is important to note that the remaining secure channels used by the Master node enclave to communicate with the Workers' enclaves, and for Workers' enclaves to exchange information among each other, are fully supported by our prototype and their overhead is considered in our experimental evaluation. Indeed, these secure channels are provided by Graphene-SGX [18].

Graphene-SGX. SOTERIA uses version 1.0 of Graphene-SGX, which is an open-source library OS that facilitates the portability of native applications and libraries to run inside SGX enclaves. Briefly, the Graphene-SGX library works similarly to a paravirtualization environment, enabling native applications and libraries to run unmodified on an isolated enclave space. I/O and other system calls from the application are replaced by Graphene-SGX to ensure their security. For instance, I/O operations to the storage backend are encrypted transparently by Graphene-SGX, while also enabling the creation of secure encrypted channels to communicate with other enclaves. One of the main components of Graphene-SGX is its *Manifest file*. It ensures that applications, running within Graphene-SGX, can attest the integrity of libraries and data being used/read by them and, moreover, cannot access other libraries or data that are not specified in this file.

Applying Graphene-SGX to SOTERIA's prototype. The *Manifest file*, discussed at Section 4.2, is directly mapped to a Graphene-SGX file in our prototype. For SML-1's implementation, besides the path to the data and ML task scripts inputted by the users, this file also has the necessary MLlib and core Spark libraries that must run on a secure SGX enclave at the untrusted cluster deployment. Briefly, these libraries are the ones that are used by the Spark Master and Worker nodes to perform ML tasks, including statistical computation. Note that these are already included in the file and the user does not need to add them manually.

Furthermore, by using Graphene-SGX, SOTERIA does not change any of the implementation code at the Spark libraries. The only exception is on our modified data loading package, used at the Master and Worker nodes. Here, we will transparently decrypt data read from the distributed storage backend, and encrypt inference and training results before sending them back to the client.

For SML-2, a distinct *Manifest file* is used which does not include the MLlib statistical libraries. To accomplish this, we needed to be intrusive and change the original MLlib's implementation, decoupling it into two sub-libraries, one that contains the statistical logic, and another with the remaining ML computational logic.

Both designs leverage Graphene-SGX to ensure the establishment of secure channels between the enclaves at the Master and Worker nodes. This is attained by resorting to the TLS-PSK with AES-GCM cryptographic protocols [46].

5 METHODOLOGY

Our evaluation aims at answering three main questions:

1) How does SOTERIA impact the execution time of ML workloads?
2) How does SOTERIA's novel optimization (SML-2) compares, in terms of performance, with standard state-of-the-art approaches (*i.e.*, SML-1 and SGX-Spark)?
3) Can SOTERIA handle different algorithms and dataset sizes efficiently?

Environment. The experiments use a Cloudera 6.3 cluster with eight Dell OptiPlex 3070 Small-Form Desktops, 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 and driver 1.8. The client and Spark Master run in one server while Spark Workers are deployed in the remaining seven servers. Also, SGX memory is configured to use 4GB.

Workloads. We resort to the HiBench benchmark [17], which allows evaluating different ML algorithms broadly used and natively implemented on top of MLlib. Our evaluation considers seven algorithms, which are detailed in Table 2. We do not consider the use deep neural networks since MLlib does not natively support them. For each algorithm, the benchmark suite offers different workload sizes ranging from *Tiny* to *Gigantic* configurations.

Setups and metrics. To validate SOTERIA's performance, and the benefits of fine-grained differentiation of secure ML operations, we compare both our architectures, namely SML-1 and SML-2, with a baseline deployment of Apache Spark that does not provide any privacy guarantees (Vanilla). Moreover, we compare our solution with SGX-Spark [15], a state-of-the-art SGX-based secure Spark solution that shares similar goals with SOTERIA.

SGX-Spark aims at protecting both analytical and ML computation done at Apache Spark. Implemented with SGX-LKL[5], this solution is designed to process sensitive information inside SGX enclaves. Therefore, this design can be considered as the most similar to the one proposed in this paper. However, SGX-Spark can only ensure that *User Defined Functions (UDFs)* [54] processing is done at a secure enclave. This decision leaves a large codebase of Spark outside the protected memory region and, consequently, limits the users to only be able to execute privacy-preserving machine learning algorithms that leverage the UDF mechanism. Also, similarly to our threat model, SGX-Spark does not consider traffic analysis attacks [55], side-channel attacks [27] or speculative attacks [28].

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

5. https://github.com/lsds/sgx-lkl

6 EVALUATION

We split our evaluation into two different stages to present our results more clearly. Section 6.1 summarizes the main evaluation observations, while Section 6.2 analyzes these observations and provides key insights.

6.1 Performance Overview

Figure 4 shows the execution time of all the setups for the 7 algorithms when using a huge-sized workload configuration. Moreover, Figures 5a, 5b, 5c and 5d present the performance evaluation for *PCA*, *GBT*, *ALS* and *Linear* 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 4, whilst *Observations 9* to *12* refer to Figure 5.

Observation 1. Vanilla Spark's execution times for ALS, LDA, Kmeans, PCA, Bayes, Linear, and GBT algorithms, are, respectively, 55, 401, 155, 655, 33, 657, and 189 seconds.

Observation 2. The execution time for the ALS algorithm increases by 3.62x and 4.35x for SML-2 and SML-1, respectively. SGX-Spark incurs an execution overhead of 4x. Thus, the three setups have similar results while requiring approximately more 150 seconds of processing time than the vanilla deployment. Nevertheless, SML-2 performs slightly better than the other two approaches.

Observation 3. The LDA algorithm exhibits higher execution overhead of 17.40x, 8.89x, and 15.08x for SML-1, SML-2, and SGX-Spark setups, respectively. SML-2 outperforms SGX-Spark by a difference of 41.5 minutes.

Observation 4. When compared with the vanilla deployment, SML-1 increases execution time by 9.37x and SML-2 by 6.68x. SGX-Spark has an overhead of 9.7x, which, in comparison with SML-2, requires more 468 seconds (7.8 minutes) to execute.

Observation 5. For PCA, SML-1 and SML-2 have an execution overhead of 3.67x and 2.85x, respectively, over the vanilla results, while SGX-Spark increases the computational time by 3.95x. Thus, both SML-1 and SML-2 surpass SGX-Spark. Moreover, when comparing SML-2 with SGX-Spark, we observe a decrease of 768 seconds, nearly 12 minutes, in execution time.

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

Algorithms	Tasks	Time Complexity	Workloads			
			Tiny	Large	Huge	Gigantic
Alternating Least Squares (ALS)	RS	$O((m+n)k^3 + mnk^2)$ [47]	193KB	345MB	2GB	4GB
Principal Compt. Analysis (PCA)	DR	$O(nm * min(n,m) + m^3)$ [48]	256KB	92MB	550MB	688MB
Gradient Boosted Trees (GBT)	P	$O(n * y * n_{trees})$ [49]	36KB	46MB	92MB	183MB
Linear Regression (LR)	C + P	$O(m * n^2 + n^3)$ [50]	11GB	134GB	335GB	894GB
Sparse Naive Bayes (Naive Bayes)	MC	$O(nm)$ [51]	-	-	5GB	-
Latent Dirichlet Allocation (LDA)	DR	$O(mnt + t^3)$ [52]	-	-	2GB	-
K-means clustering (K-means)	Cl	$O(n^2)$ [53]	-	-	56GB	-

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

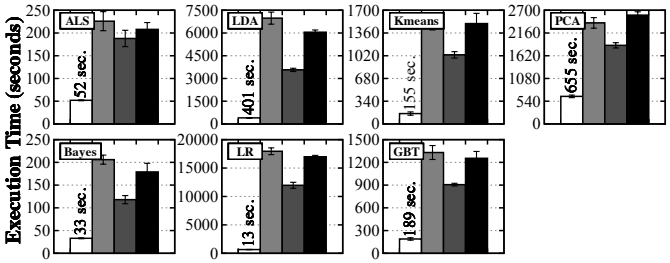


Fig. 4: Execution time for each algorithm with *Huge* workload. The legend is as follows: □ Vanilla Spark; ■ SML-1; ■ SML-2; ■ SGX-Spark.

Observation 6. With *Huge* workload and Naive Bayes, SOTERIA exhibits an overhead of 6.24x for SML-1, which is higher than the 5.33x observed for SGX-Spark. Also, SML-2 continues to present lower overhead (3.58x) when compared with SGX-Spark. The absolute difference of execution time between SML-2 and SML-1 is 88 seconds, whilst with SGX-Spark, SML-2 decreases execution time by 61 seconds.

Observation 7. For the Linear Regression algorithm, SML-1 shows an average overhead of 27.31x and SML-2 lowers this execution time to 18.2x, whilst SGX-Spark shows an overhead similar with SML-1. These results portray the greatest decrease in execution time when comparing SML-2 and the SGX-Spark state-of-the-art solution, corresponding to 1.4 hours of execution time.

Observation 8. With the GBT algorithm, SML-1 shows similar execution times when compared with SGX-Spark, with a 7.04x and 6.64x increase, respectively. SML-2 outperforms both approaches, presenting an overhead of 4.79x, about 248 seconds less than SGX-Spark.

Observation 9. For *Tiny* and *Large* workloads with the PCA algorithm, SOTERIA performs similarly for our two designs, while outperforming SGX-Spark. With larger workload sizes, the overhead imposed by our solutions increases, however, it continues to show better performance than SGX-Spark. SML-1 has an overhead of 1.96x to 5.15x, for *Tiny* and *Gigantic* workloads, whilst SML-2 incurs an overhead of 1.72x to 3.79x. When compared with SGX-Spark, the results show an absolute difference of 4 and 436 seconds, for SML-1, and 7 seconds and 33 minutes, for SML-2.

Observation 10. Regarding the GBT algorithm, SOTERIA shows significant variance in terms of execution time when dealing with different workload sizes. For the *Tiny* workload, the overhead of SML-1, SML-2, and SGX-Spark are similar. However, when increasing the workload size, the difference between the three approaches is more visible. SML-2 (*Tiny-2.13x and Gigantic-5.88x*) outperforms both approaches, while SML-1 (*Tiny-2.18x, Gigantic-9.35x*) and SGX-Spark (*Tiny-2.3x, Gigantic-10.34x*) have similar overheads.

Observation 11. With the ALS algorithm, SOTERIA maintains a more constant increase of the execution time between the four workload size configurations. SML-2 shows an execution time for the *Tiny* and *Gigantic* workloads of 2.04x and 3.28x when compared with vanilla Spark. SML-2 achieves lower overhead than SML-1 and SGX-Spark for all the workloads, with the execution time decreasing 8 seconds for the *Tiny* and 191 seconds for the *Gigantic* workloads.

Observation 12. For the linear regression algorithm, SOTERIA exhibits more overhead for increasing data sizes. With the *Tiny* workload, SML-1 has an overhead of 14.39x and SML-2 shows an overhead of 12.95x. As for the *Gigantic* workload, SML-1 incurs an overhead of 30.04x and SML-2 of 23.89x. If one compares with SGX-Spark, our second design decreases the execution time in 43 seconds for the *Tiny* workload and 4.31 hours for the *Gigantic* workload.

Observation 13. Overall, the resource consumption (CPU and memory) and network traffic for both SOTERIA designs are similar to the vanilla Spark baseline. In more detail, the SML-1 design with Linear Regression presents the upper-bound limit for both memory and CPU, showing an increase of 9% in both when compared with vanilla Spark (20%). Whilst the network shows an upper-bound increase of 15% (vanilla Spark shows an upper-bound network of 135MB) in SML-1 with LDA due to the extra encrypted data paddings being sent between Worker nodes.

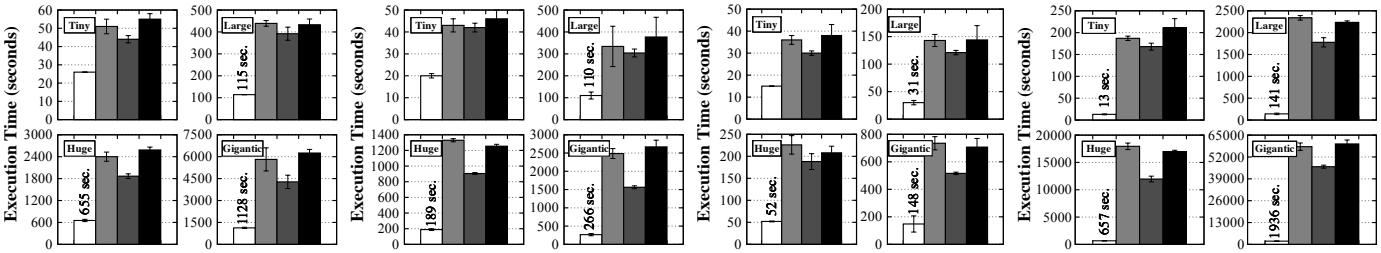
Observation 14. SOTERIA does not impact the accuracy of ML workloads. For all experiments, we measured the corresponding accuracy metrics (*e.g.*, accuracy, root mean square error, ROC). The results corroborate that both SML-1 and SML-2 show accuracy values similar to the vanilla Spark version.

6.2 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).

6.2.1 Dataset size

Figure 5 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, SML-1 and SML-2 perform



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

Fig. 5: Runtime execution for PCA, GBT, ALS and Linear Regression for *Tiny*, *Large*, *Huge* and *Gigantic* workloads. The legend is as follows: □ Vanilla Spark; ■ SML-1; ■ SML-2; ■ SGX-Spark.

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 [14], [56]. For example, when compared to the vanilla setup, the PCA algorithm overhead for SML-1 varies between 1.96x, for *Tiny* workload, and 5.15x, for *Gigantic* workload. While for SML-2, the execution time increases 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). When compared with SGX-Spark, SML-2 deals better with the increase of data volume. Indeed, as seen in *Observations 9-12*, we are able to reduce the execution time from a few seconds to more than 4 hours when compared to SGX-Spark.

6.2.2 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. Whilst, 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 3* emphasize the performance of these two algorithms for a similar workload size. Similar to LDA, *Observations 5 and 9* show that PCA complexity and performance overhead also increase with the processed data volume. Commonly classified as regression and classification algorithms, Bayes and GBT have similar performance, as seen in *Observation 6* and *Observation 8*. 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 one time 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 7*).

6.2.3 Size of TCB

The results discussed at Section 6.1 show that SGX-Spark outperforms SML-1 for some of the evaluated algorithms (*Observation 2, 3, 6-8*). As SGX-Spark only protects UDFs, the performance overhead imposed by the larger trusted computing base of our solution is naturally higher. Nevertheless, when compared to SGX-Spark, SML-1 covers a wider range of machine learning attacks, while keeping performance overhead below 1.59x. Indeed, for algorithms such as PCA, and Kmeans, SML-1 has a similar or slightly inferior execution time (*Observations 4 and 5*). This happens because, for these algorithms, both SGX-Spark and SML-1 perform similar computations at the secure enclaves, while the UDF mechanism is not the most optimized choice for running some of these workloads.

Finally, SML-2 always outperforms SGX-Spark and SML-1 (*Observations 2-8*). This is due to the TCB reduction present in our second design. The results show that this solution can outperform SGX-Spark by up to 41%, namely for the LDA algorithm with the *Huge* workload (*Observation 3*).

6.3 Discussion

The experimental results show that SML-2 outperforms state-of-the-art approaches, namely SGX-Spark, for all the considered ML algorithms. Also, SML-2 achieves better performance than the more standard SML-1 design, while offering similar security guarantees when considering the most prevalent ML attacks (Section 4.5). This is made possible by filtering key operations to be done outside enclaves.

In more detail, when compared to SML-1, our new design (SML-2) reduces ML workloads' execution time by up to 1.6x. When compared with SGX-Spark, the execution time is reduced by up to 1.7x. Interestingly, for the *Linear Regression* algorithm using a *Gigantic* workload (*894GB*), SML-2 decreases computation time by 3.8 hours and 3.3 hours, when compared with SGX-Spark and SML-1, respectively.

Finally, the performance overhead of SML-2 for the seven different algorithms ranges from 1.7x to 23.8x when compared to Vanilla Spark.

7 Related Work

Secure ML solutions can be classified into four main groups based on the privacy-preserving techniques being used: *i)* encryption-based [7], [57], [58], *ii)* secure multi-party computation [59], [60], *iii)* differential privacy [61], [62] and, *iv)* trusted execution environments (TEEs) [11], [12], [21], [22]. This paper is included in group *iv)*.

Privacy-preserving machine learning with TEEs. Chiron [11] enables training ML models on a cloud service without

revealing information about the training dataset. Also, once the model is trained, only the data owners can query it. *Myelin* [12] offers a similar solution to *Chiron* while adding differential privacy and data oblivious protocols to the algorithms to mitigate the exploits from *side-channels* and the information leaked by the model parameters. SOTERIA differs from these works as it is able to cover both training and inference phases while providing additional protection against *adversarial samples*, *reconstruction*, and *membership inference* attacks (Table 1). In [21], five machine learning algorithms are re-implemented with data oblivious protocols. These protocols are combined with TEEs to ensure strong privacy guarantees while preventing the exploitation of *side-channel* attacks that observe memory, disk, and network access patterns to infer private information. Unlike this solution, SOTERIA aims at transparently supporting all machine learning algorithms built with the MLlib Spark's API. Also, we underline that *side-channel* attacks are currently considered orthogonal to our work. Nevertheless, solutions have been proposed as countermeasures to such attacks [63].

Privacy-preserving analytics with TEEs. TEEs have also been used to ensure privacy-preserving computation for general-purpose analytical frameworks [13], [16]. In comparison to SGX-Spark [15], detailed in Section 5, SOTERIA supports a broader set of algorithms (*i.e.*, any algorithm that users can build with the MLlib API) while protecting users from a more complete set of attacks to the machine learning pipeline, as shown in Table 1. Opaque [24] and Uranus [25] also resort to SGX to provide secure analytics but only support a very restricted set of ML algorithms. The first solution combines SGX with oblivious protocols while requiring the re-implementation of default Apache Spark UDF operators. The second solution aims at simplifying the combination of Big Data applications with SGX enclaves. Namely, it addresses an Apache Spark use-case where it shows that UDF processing can be ported to secure enclaves. However, the proposed use-case only includes a single machine learning workload. SOTERIA aims at supporting a broader spectrum of ML algorithms (*i.e.*, it is not limited to algorithms built on top of Spark UDFs) while avoiding changing internal Spark operators to achieve privacy.

Privacy-preserving deep learning with TEEs. TEEs have also been applied to the training and inference of deep neural networks [22], [64]. 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 on ML algorithms.

8 Conclusion

This paper presents SOTERIA, a system for distributed privacy-preserving machine learning. 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 design (SML-2) that allows fine-grained differentiation of which ML operations can be deployed outside of trusted enclaves, maintaining the security guarantees of the protocol. 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 attacks than in previous related work. Furthermore, this decision enables SOTERIA to perform better than existing solutions, such as SGX-Spark, while reducing ML workloads execution time by up to 41%.

Acknowledgment

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

References

[1] Lun Wang, Joseph P Near, Neel Somani, Peng Gao, Andrew Low, David Dao, and Dawn Song. Data capsule: A new paradigm for automatic compliance with data privacy regulations. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 2019.

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

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

[4] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018.

[5] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*

[6] Claudio Orlandi. Is multiparty computation any good in practice? In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2011.

[7] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 2017.

[8] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *Hasp@ isca*, 2013.

[9] AMD. Amd secure encrypted virtualization (sev). https://developer.amd.com/sev/. (Accessed on 24/02/2021).

[10] Tiago Alves. Trustzone: Integrated hardware and software security. *White paper*, 2004.

[11] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018.

[12] Nick Hynes, Raymond Cheng, and Dawn Song. Efficient deep learning on multi-source private data. *arXiv preprint arXiv:1807.06689*, 2018.

[13] Do Le Quoc, Franz Gregor, Jatinder Singh, and Christof Fetzer. Sgx-pyspark: Secure distributed data analytics. In *The World Wide Web Conference*, 2019.

[14] Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, Valerio Schiavoni, Pascal Felber, and Daniel Hagimont. Everything you should know about intel sgx performance on virtualized systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2019.

[15] Large-Scale Data & Systems (LSDS) Group. Sgx-spark. https://github.com/lsds/sgx-spark. (Accessed on 15/02/2021).

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

[17] Intel. Hibench is a big data benchmark suite. https://github.com/Intel-bigdata/HiBench. (Accessed on 21/02/2021).

[18] Chia-Che Tsai, Donald E Porter, and Mona Vij. Graphene-sgx: A practical library os for unmodified applications on sgx. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017.

[19] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 2016.

[20] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 2016.

[21] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[22] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

[23] Joseph I Choi and Kevin RB Butler. Secure multiparty computation and trusted hardware: Examining adoption challenges and opportunities. *Security and Communication Networks*, 2019, 2019.

[24] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017.

[25] XC Jianyu Jiang, CW Tzs, On Li, T Shen, and S Zhao. Uranus: Simple, efficient sgx programming and its applications. In *Proceedings of the 15th ACM ASIA Conference on Computer and Communications Security (ASIACCS '20)*, 2020.

[26] Microsoft Azure. Azure confidential computing. https://azure.microsoft.com/en-us/solutions/confidential-compute/. (Accessed on 05/01/2021).

[27] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. Varys: Protecting sgx enclaves from practical side-channel attacks. In *USENIX Annual Technical Conference*, 2018.

[28] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *27th $USENIX$ Security Symposium ($USENIX$ Security 18)*, 2018.

[29] Jinmeng Jia, Ruiyuan Wang, Zhongxin An, Yongli Guo, Xi Ni, and Tieliu Shi. Rdad: a machine learning system to support phenotype-based rare disease diagnosis. *Frontiers in genetics*, 2018.

[30] Seema Rawat, Aakankshu Rawat, Deepak Kumar, and A Sai Sabitha. Application of machine learning and data visualization techniques for decision support in the insurance sector. *International Journal of Information Management Data Insights*, 2021.

[31] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 1994.

[32] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure multiparty computation from sgx. In *International Conference on Financial Cryptography and Data Security*. Springer, 2017.

[33] Jonas Bushart and Christian Rossow. Padding ain't enough: Assessing the privacy guarantees of encrypted dns. In *10th USENIX Workshop on Free and Open Communications on the Internet*, 2020.

[34] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

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

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

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

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

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

[40] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of ACM SIGSAC conference on Computer & communications security*, 2013.

[41] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, San Jose, CA, 2012. USENIX.

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

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

[44] Morris J Dworkin. *Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac*. National Institute of Standards & Technology, 2007.

[45] Takanori Machida, Dai Yamamoto, Ikuya Morikawa, Hirotaka Kokubo, and Hisashi Kojima. Poster: A novel framework for user-key provisioning to secure enclaves on intel sgx. 2018.

[46] Graphene-SGX. Performance tuning and analysis — graphene documentation. https://graphene.readthedocs.io/en/latest/devel/performance.htmls. (Accessed on 07/02/2021).

[47] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2016.

[48] Tarek Elgamal and Mohamed Hefeeda. Analysis of pca algorithms in distributed environments. *arXiv preprint arXiv:1503.05214*, 2015.

[49] Si Si, Huan Zhang, Sathiya Keerthi, Druv Mahajan, Inderjit Dhillon, and Cho-Jui Hsieh. Gradient boosted decision trees for high dimensional sparse output. In *International conference on machine learning*, 2017.

[50] Jim 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).

[51] Charles Elkan. Boosting and naive bayesian learning. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1997.

[52] Deng Cai, Xiaofei He, and Jiawei Han. Training linear discriminant analysis in linear time. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 2008.

[53] Malay K Pakhira. A linear time-complexity k-means algorithm using cluster shifting. In *2014 International Conference on Computational Intelligence and Communication Networks*. IEEE, 2014.

[54] Scalar User Defined Functions (UDFs). Apache spark. https://spark.apache.org/docs/latest/sql-ref-functions-udf-scalar.html. (Accessed on 03/02/2021).

[55] Shay Gueron. A memory encryption engine suitable for general purpose processors. *IACR Cryptol. ePrint Arch.*, 2016, 2016.

[56] ChongChong Zhao, Daniyaer Saifuding, Hongliang Tian, Yong Zhang, and ChunXiao Xing. On the performance of intel sgx. In *13th web information systems and applications conference*. IEEE, 2016.

[57] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[58] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying

neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, 2016.
[59] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *Symposium on Security and Privacy*. IEEE, 2017.
[60] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.
[61] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015.
[62] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016.
[63] Tânia Esteves, Ricardo Macedo, Alberto Faria, Bernardo Portela, João Paulo, José Pereira, and Danny Harnik. Trustfs: An sgx-enabled stackable file system framework. In *38th International Symposium on Reliable Distributed Systems Workshops*. IEEE, 2019.
[64] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. Tensorscone: a secure tensorflow framework using intel sgx. *arXiv:1902.04413*, 2019.
[65] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure multiparty computation from sgx. Springer International Publishing.

APPENDIX A
SOTERIA PROOF

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

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

Formally, this security goal is defined using the real-versus-ideal world paradigm, similarly to the Universal Composability [42] framework.

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

A.1 Formal Security Model

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

Algorithm $\mathsf{Setup}(i, m)$<G>:
$\mathsf{k} \leftarrow\$\ \Theta.\mathsf{Gen}()$
$\mathsf{c} \leftarrow\$\ \Theta.\mathsf{Enc}(\mathsf{k}, i)$
$G[m] \leftarrow \mathsf{c}$
Return (m, k)

Fig. 6: Secure external storage setup.

functionality is parametrised by this external storage $\mathcal{F}$¡G¿, and will reveal the access patterns via leakage function $\mathcal{L}$.[6]

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.[7] The Client will then execute protocol Π, sending messages to M, W_1 and W_2. When the script is concluded, output is provided to C, finally being returned to $\mathcal{Z}$. $\mathcal{A}$ can observe all communication between C, M, W_1, W_2 and G.

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

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

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

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

A.2 Intermediate Result

For convenience, SOTERIA does not require the Client to provide input data at the time of the ML processing, and instead the Workers are given access to an external storage from which to retrieve this data. When discussing the security in the context of secure outsourced computation for SGX, this is functionally equivalent to classical scenarios where the Client provides these inputs via secure channel (Theorem 3 in [65]). The reasoning is simply that, if a protocol securely realises a functionality with a given input provided via secure channel, then the same functionality can be securely realised with the same input fixed in an external storage, securely accessed by the enclave.

Consider a protocol Π_1 that securely realises some functionality $\mathcal{F}$ with simulator $\mathcal{S}_1$ according to Theorem 3 of [42]. We construct protocol Π_2 built on top of this secure protocol Π_1, where input data is pre-established and

6. Reasoning for the security of SML-2 instead would only require this function to also reveal statistical data to the simulator, which we consider to be non-sensitive.

7. 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 $\mathsf{AC}()$
$\mathsf{k} \leftarrow\$\ \Theta.\mathsf{Gen}()$
Return $\mathcal{S}_1.\mathsf{AC}()$

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

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

Fig. 7: Simulator for Π_2.

provided to the enclave via an initial Setup stage where inputs are stored in encrypted fashion (Figure 6 describes a simplified version of the process for a single entry). Inputs to Π_2 are exactly the same as those for Π_1, but instead of being transmitted via the secure channel established with Attested Computation, they are retrieved from storage using a key sent via the same channel. The Client-server communication increases by a constant (the key length), which can be trivially simulated, and the rest of the input can be simulated in a similar way using the IND-CPA properties of Θ. This protocol behavior will be key for all SOTERIA Workers. Our theorem is as follows.

Theorem 1. Let Π_1 be a protocol that securely realises functionality $\mathcal{F}$ according to Theorem 3 in [65]. Then Π_2, constructed as discussed above, securely realises $\mathcal{F}$ according to Definition 1.

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

We begin by deconstructing $\mathcal{S}_1$ in two parts: $\mathcal{S}_1.\mathsf{AC}()$ will produce the view for the establishment of secure channel, while $\mathcal{S}_1.\mathsf{Send}(l)$ will produce a simulated view of Client inputs, given their length. In turn, our simulator will share the same functions, but also include a third $\mathcal{S}_2.\mathsf{Get}(l)$ to simulate information being retrieved from G, given its length. Our simulator is depicted in Figure 7.

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

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

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

Let

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

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

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

and Theorem 1 follows.

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

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

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

Fig. 8: SOTERIA Components. Client C (left), Master node M (middle) and Worker node 1 W (right).

A.3 SOTERIA Client, Master and Workers

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

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

The SOTERIA components can be analysed in Figure 8 and are as follows. The Client C (left of Figure 8) 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 8) 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.[8] Worker W_1 (right of Figure 8) receives the parameters from M and starts processing the script: retrieves encrypted data from G, decrypts, processes and exchanges intermediate results with the other Worker. When the script is concluded, it returns its output to M. The behavior of W_2 is the same, but connection is established instead with W_1.

8. 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.

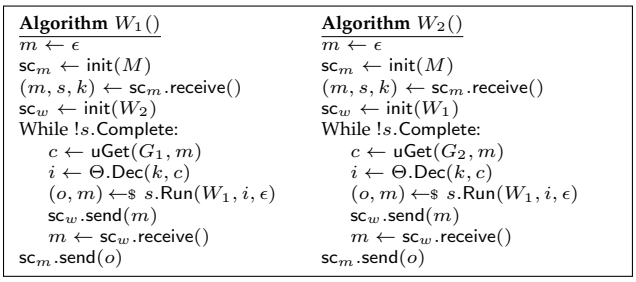


Fig. 9: SOTERIA Workers with split storage.

A.4 Full Proof

Given the description of SOTERIA components in Figure 8, the SOTERIA protocol Π_{xyz} is straightforward to describe. Considering a pre-encrypted storage G, the Client C, Master M and Workers W_1, W_2 execute following their respective specifications. Our theorem for the security of SOTERIA is as follows.

Theorem 2. Π_{xyz}, assuming the setup of Figure 6 and constructed as discussed above, securely realises $\mathcal{F}$ according to Definition 1.

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

The first is a simplification step, where, instead of using a single storage G, we slice the storage to consider G_1 and G_2. Figure 9 represents this change. This enables us to split the execution environment of W_1 and W_2 seamlessly, and can be done trivially since manifest file m by construction will never require different Workers to access the same parts of G. Since these two games are functionally equal, the adversarial advantage is exactly 0.

The second step is a hybrid argument, where we sequentially replace both Workers by ideal functionalities performing partial steps of the ML script. Concretely, we argue as follows. Replace W_1 with a functionality for its part of the ML script $\mathcal{F}_{W1}$, according to Definition 1. From Theorem 1 we can establish that this adaptation entails negligible advantage to $\mathcal{A}$ provided that the protocol without external access realises the same functionality. However this is necessarily the case, as it follows the exact structure as the constructions in [65]. We can repeat this process for W_2.[9] As such, using the intermediate result, we can thus upper bound the advantage adversary to distinguish these two scenarios by applying twice the result of Theorem 1.

The third step replaces the Master by an ideal functionality $\mathcal{F}_M$ that simply forwards requests to the Worker functionalities. This one follows the same logic as the previous one, without requiring the external storage, as the protocol also follows the exact structure as the constructions in [65].

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

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

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

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

Let

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

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

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

and Theorem 2 follows.

APPENDIX B
ML WORKFLOW ATTACKS

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

B.1 Attacker against SOTERIA

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

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

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

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

Fig. 10: Adversary interacting with SOTERIA.

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

Fig. 11: Model for dataset manipulation attack.

to the model is not possible if SOTERIA assumes external adversaries.

Let Π_{xyz} denote the full training protocol of SOTERIA. It receives external storage G and task script s as inputs, and produces a model m, which can then be queried. Based on the security result of Appendix A, the interaction of an adversary with our system can be described in Figure 10. The adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ can first try and manipulate the input dataset G to G'. This is then used for Π_{xyz}, which will produce the model m and the additional leakage l (SML-1 has no additional leakage, so $l = \epsilon$). Finally, the adversary can interact black-box with the model until a conclusion r is produced. This will be provided to a Success predicate, which will state if the attack was successful. This predicate is specific to the attack, and allows us to generally describe attacks such as *adversarial samples*, where the goal is to make the resulting model deviate, as well as *membership inference*, where the goal is to retrieve information from the original dataset.

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

B.2 Dataset manipulation

Dataset manipulation attacks are defined by an adversary with the capability of inserting, removing or manipulating dataset information. These align with the setting considered for attacks via *adversarial samples*. Figure 11 is an experiment that describes what constitutes a successful attack for dataset manipulation. The adversary $\mathcal{A}$ is given full knowledge of G[10], and must produce an alternative input dataset G'. We then train the model (protocol Π) over that data to produce model m, and the adversary is successful if said model satisfies some attack success criteria $\mathsf{T}/\mathsf{F} \leftarrow \mathsf{Success}$.

We now argue that the integrity guarantees of the authenticated encryption used by our external storage G ensure that these attacks do not occur for Π_{xyz}. We do this by showing that any adversary that performs an *adversarial samples* attack on Π_{xyz} can be used to construct a successful attack on the the security of the authenticated encryption scheme. First, observe that no attack can be successful if the adversary makes no changes on the input dataset, so if $G = G'$ then $\mathsf{F} \leftarrow \mathsf{Success}$. Furthermore, if Π_{xyz} aborts, then no model is produced, so it naturally follows that the attack is unsuccessful $\mathsf{F} \leftarrow \mathsf{Success}$.[11]

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

B.3 Black-box Attacks

All the remaining attacks with the exception of some *reconstruction attacks* follow a similar setting, where the adversary leverages a black-box access to the trained model, depicted in the experiment of Figure 12. We begin by running Π 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 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 [43]. We now go case-by-case, assuming an adversary can play the role of a genuine client to our system.

Our arguments for Π_{xyz} depend on being able to rely on a successful adversary $\mathcal{A}$ of Figure 12 to perform the same attack in Figure 10. 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 12 will also be successful in Figure 10, meaning that both for SML-1 and SML-2 are vulnerable. Preventing these attacks requires restricting access to the model to untrusted participants.

10. 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.

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

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

Fig. 12: Model for black-box attacks.

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

Fig. 13: Model for white-box attacks.

- *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 SML-1 or SML-2 if there is also an efficient attack for *model inversion* in SML-1 or SML-2, 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 SML-1 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 SML-2 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 SML-1 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 SML-2 is vulnerable to such attacks under these assumptions.

We can generalize the security of our system to these types of attacks as follows. If no additional information z is required, then Π_{xyz} is vulnerable to an adversary that can play the role of an honest client. If z can be extracted from black-box access to the model, then we can still rely on said adversary to attack Π_{xyz}. Otherwise, SML-1 is secure, as no additional information is leaked. Furthermore, the security of SML-2 will depend 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 $z \leftarrow\!\!\$\ \mathsf{F}^m(l)$, then Π_{xyz} for leakage l is secure against attacks requiring additional data z.

B.4 White-box Attacks

White-box attacks capture a scenario where an adversary requires white-box access to the model. These align with the setting of *reconstruction attacks* that explicitly requires white-box access to the model. Figure 13 is an experiment that describes what constitutes a successful *reconstruction attack* in this context. We begin by training the model (protocol Π) over the original dataset to produce the model. We then provide the trained model directly to the adversary, which will reconstruct raw data r. Finally, the success of the attack is validated with respect to the original dataset.

We now argue that these attacks do not occur for Π_{xyz}, as long as it is not possible to extract the model from the confidence values and from black-box access to the model. This is because the attacker of Figure 13 receives explicitly the model m, whereas the adversary $\mathcal{A}_2$ in Figure 10 receives the confidence values in l, and black-box access to the model. To rely on such an attacker, $\mathcal{A}_2$ must therefore be able to produce input m from its own view of the system. As such, relying on such an adversary implies there is an efficient way $m \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{bb}}^m(l)$ to retrieve the model m from confidence values l and black-box access to the model m, which is exactly the setting of *model extraction* attacks in the previous section. This adversary $\mathcal{A}_{\mathsf{bb}}$ can then be called by $\mathcal{A}_2$ to produce input m, which is then forwarded to the adversary of Figure 13 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 12 for the *model extraction* attack.

B.5 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 the different contexts.

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

White-box based attacks explicitly require additional information, such as feature vectors, over black-box access to the model [39]. 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 [43] 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

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

TABLE 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.

{3b} is false. [2] suggests {4a} is true, but {4b} is not. {5} will fundamentally depend on the application [2], [43].