Soteria: Privacy-Preserving Machine Learning for Apache Spark

Cláudia Brito, Pedro Ferreira*, Bernardo Portela†, Rui Oliveira, João Paulo
INESC TEC
*INESC TEC & Faculty of Sciences, University of Porto
†NOVA LINCS & University of Porto

***Abstract*—Privacy and security are prime obstacles to the wider adoption of machine learning services offered by cloud computing providers. Namely, trusting users' sensitive data to a third-party infrastructure, vulnerable to both external and internal malicious attackers, restricts many companies from leveraging the scalability and flexibility offered by cloud services.**

We propose SOTERIA, a system for distributed privacy-preserving machine learning that combines the Apache Spark system, and its machine learning library (MLlib), with the confidentiality features provided by Trusted Execution Environments (*e.g.*, Intel SGX). SOTERIA supports two main designs, each offering specific guarantees in terms of security and performance. The first encapsulates most of the computation done by Apache Spark on a secure enclave, thus offering stronger security. The second fine-tunes the Spark operations that must be done at the secure enclave to reduce the needed trusted computing base, and consequently the performance overhead, at the cost of an increased attack surface.

An extensive evaluation of SOTERIA, with classification, regression, dimensionality reduction, and clustering algorithms, shows that our system outperforms state-of-the-art solutions, reducing their performance overhead by up to 41%. Moreover, we show that privacy-preserving machine learning is achievable while providing strong security guarantees.

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

I. INTRODUCTION

The ubiquitous environment provided by the cloud computing paradigm offers a scalable, reliable, and performant solution to deploy data analytics applications (*e.g.*, Apache Spark [79], Hadoop [32]). As the amount of data and complexity of analytical processing grows, these third-party services become increasingly desirable while avoiding the extra costs and concerns of owning a private infrastructure.

However, data and computation outsourcing to cloud computing services leaves users vulnerable to attacks that may compromise the integrity and confidentiality of their sensitive information [80], [45], [81]. Also, with the increasing number of regulations such as HIPAA and GDPR, sensitive data from users cannot be processed or sent to untrusted third-party infrastructures without users' consent and the adoption of strong security policies [59], [88].

Machine learning (ML) provides a strong use-case for the aforementioned challenges since it deals with the analysis of sensitive data. Also, machine learning requires significant computational power as data is iterated over and over until models are fully optimized. The common machine learning workflow presents three main phases: data preparation, training, and inference. During such phases, users' data is susceptible to several attacks that must be prevented such as adversarial attacks, model extraction, and inversion, reconstruction attacks, among others [49], [90], [26], [3].

The adoption of software-based cryptographic techniques (*e.g.*, homomorphic encryption, secure multi-party computation [2], [28], [60], [64]) intends to overcome these setbacks while offering privacy-preserving machine learning solutions [66], [58], [36]. Despite that, such techniques limit the operations that can be conducted over encrypted data and/or impose a significant performance toll that restricts their applicability to practical scenarios [6]. Trusted Execution Environments (TEEs) (*e.g.*, Intel SGX [55], AMD-SEV [5], ARM TrustZone [4]) are increasingly considered as an alternative solution to deliver a secure processing environment where data can be handled in its original form (*i.e.*, plaintext) at an untrusted storage server [72], [14]. Though, choosing which data and computations must be done at the secure enclave is not a trivial task. Ideally, all operations should be done in a secure space in order to have strong security guarantees. However, as the amount of operations and data increases at the enclave so does the performance overhead of the solution [18], [96]. Indeed, there is a balance in terms of security guarantees and performance that needs to be further explored. Such balance is highly dependent on the true definition of which data is sensitive and which computations do not leak sensitive information.

Previous solutions based on TEEs focused their efforts on building new secure frameworks from the ground-up [75], [62] or on providing privacy-preserving systems that are only applicable to a small subset of ML operations [97], [31] or other specific fields such as neural networks [48].

Contrarily, this paper is the first to cover a larger spectrum of machine learning attacks while providing two TEE-based designs that balance different trade-offs in terms of security and performance. Also, it supports a wide variety of machine learning algorithms while not changing how users build and run their algorithms within the Apache Spark framework. Namely, we present SOTERIA[1] , a system for distributed privacy-preserving machine learning. It leverages the scalabil-

[1] SOTERIA was known as the greek goddess of safety and salvation, deliverance, and preservation from harm.

ity and reliability provided by Apache Spark and its machine learning library (MLlib) while combining these with TEEs to promote a privacy-preserving solution.

SOTERIA introduces two new designs that showcase different levels of confidentiality, integrity, and performance one can expect from a full-fledged machine learning pipeline (i.e., including both training and inference phases) supported by Apache Spark. The first ensures that all operations done at the untrusted Spark backend are performed on secure enclaves. Thus, sensitive data is only processed in plaintext inside the enclave, while being encrypted in the remainder data flow (*e.g.*, network, storage). This design enables robust security guarantees while covering a larger spectrum of attacks than in current related work [40], [42], [31].

The second design aims at reducing the trusted computing base (TCB) that must be executed in a secure space, thus lowering the respective number of processing operations and data being transferred to the enclave. When compared to the first approach, this decision enables better performance at the cost of relaxing the offered security guarantees. Nevertheless, we argue that such is possible while still ensuring that all sensitive computation is executed inside trusted enclaves, thus never disclosing users' plaintext data at the untrusted servers. Indeed, our experimental results demonstrate that performance overhead can be significantly reduced by just offloading non-sensitive statistical operations to the untrusted Spark backend. We conduct an extensive evaluation over seven different machine learning algorithms with HiBench [44] to demonstrate the feasibility and usability of our system. The results show that SOTERIA outperforms the state-of-the-art SGX-Spark solution [31], [73]. In general, SOTERIA's first design offers better security guarantees than SGX-Spark, while exhibiting a similar performance behavior. In contrast, the second design is able to surpass SGX-Spark by up to 41% in execution time, however, it relaxes the security guarantees when compared with the first design. Also, when compared to a non-secure baseline deployment of Apache Spark, SOTERIA exhibits a performance overhead ranging from 1.7x to 30.8x.

Our contributions are summarized as follows:

- We propose SOTERIA, a novel twofold privacy-preserving machine learning system based on Apache Spark that leverages TEEs for secure execution of machine learning pipelines.
- SOTERIA is provided as an open-source prototype that resorts to the Graphene-SGX Library OS [85] to ease the integration of Intel SGX within Apache Spark's workflow. Our implementation benefits from the increased security isolation and host platform compatibility provided by Graphene-SGX. The source code can be found here https://github.com/claudiavmbrito/Soteria.
- We conduct an extensive evaluation resorting to seven machine learning algorithms within HiBench. The results show that SOTERIA's prototype outperforms state-of-the-art solutions by up to 41%, while providing stronger security guarantees.

The paper is structured as follows: Section II presents relevant background. Section III defines our threat model and security properties. Section IV discusses SOTERIA design principles and details implementation decisions. Section V establishes the chosen evaluation methodology and Section VI presents the experimental results. Section VIII concludes the paper.

II. BACKGROUND

This section overviews Apache Spark, MLlib, and Intel SGX as these are used by SOTERIA to provide a distributed and secure machine learning solution.

A. Apache Spark and MLlib

Apache Spark is an open-source 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 large cluster of servers that may access several data sources (*e.g.*, HBase [34], Hive [38], Cassandra [25], HDFS [32]) for reading the data to be processed and store the corresponding output and logs [95], [79]. 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 [94]. In general, Spark follows a distributed architecture composed of a Master and several Worker nodes, typically deployed across distinct cluster servers. Further details about this architecture are discussed in Section IV-A.

The MLlib library provides machine learning capabilities to the latter framework allowing the development of end-to-end machine learning pipelines [57], [56]. The data workflow of MLlib is similar to the one found in other machine learning solutions, with the addition of the initial data collection stage. It also provides a set of tools and utilities for feature extraction, model persistence, among others. Figure 1 shows the typical machine learning workflow for data engineers, whereas the first step goes from the collection of data to its treatment. Alongside the creation of a new machine learning algorithm or choosing a pre-defined one (*e.g.*, Logistic Regression, Random Forest), in the second stage, data is transformed into two datasets, a training dataset and a test dataset. The third stage represents the training stage, where data is iterated to, in the end, deliver an optimized trained model, depicted in the fourth stage. In a fifth stage, this trained model can then be saved (persisted) and loaded (accessed) for further inference.

B. Intel Software Guard Extensions (SGX)

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 [55], [62]. 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 [83].

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. Also, these TEEs enable better performance than traditional cryptographic computational techniques (*e.g.*, searchable encryption, homomorphic encryption) [17].

The enclave has a limited memory capacity (128MB per CPU) before requiring memory swapping, which is a costly mechanism in terms of performance [14]. Solutions must balance the number of I/O operations and the amount of data handled by SGX enclaves in order to optimize their performance.

In this paper, we chose Intel SGX over other TEE's (*e.g.*, ARM TrustZone [4]) due to its broad use in academia [72], [47], [97], [41] and industry [8], [43] as well as its security guarantees and computing reliability. However, our solution is generic, in the sense that it 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 [92], [33], [63], [87].

III. Security Properties and Threat Model

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. [9], 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, and the enclave is simply executing an 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 his 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 will be trusted. It will provide the input for processing, and submit queries for machine learning tasks. Then, we will have a Master node, and N Worker nodes. These will be deployed in untrusted environments, equipped with Intel SGX technology. Externally, we also consider a distributed data storage backend. The protocol assumes a previous setup where the client has stored its input data securely within this backend, which is also considered to be untrusted throughout the protocol execution.

In particular, we want our protocol 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 presented considering an adversary that attempts to break the confidentiality of data and model, but that will not actively deviate from the protocol specification. This a common adversary for cloud-based systems, where vulnerability data breaches allow for malicious entities to read internal processing data. Additionally, we provide countermeasures against adversarial queries and the injection of data samples into the storage backend or the model being trained (*i.e.*, breaches in data integrity).

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 our adversary, by observing network communication between Spark Nodes and their storage accesses. Ensuring the privacy of such data would require additional steps to ensure constant-round execution and fixed-size outputs, which we consider to be relevant future work but outside the scope of this paper.

Formally, our security goal is defined using the real-versus-ideal world paradigm, similarly to the Universal Composability [16] framework. 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. Concretely, we specify this idealized service 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 via a secure channel.

In the real-world, we run the SOTERIA as specified in Section IV, and the adversary can observe all messages exchanged between participants, as well as requests to external storage by the Worker nodes. In the ideal world, we instead run the idealized service, and the adversary is instead 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 no information other than network and storage I/O patterns, number of computation steps, and output sizes.

A. Machine Learning Workflow Attacks

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

Adversarial Samples This attack, also known as Data Poisoning, relies on the injection of adversarial samples into the machine learning model. These examples are intentionally built to control the model by introducing noisy samples and

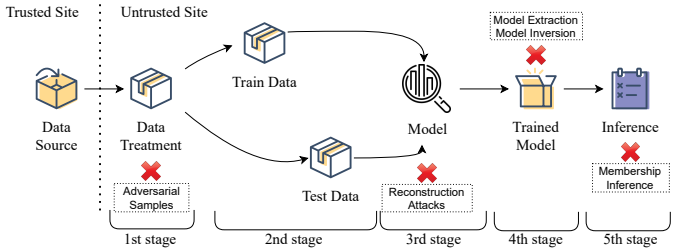


Fig. 1: Machine learning pipeline attacks compromising data and model. The figure represents the attacks defined within our security and threat model.

making the model incorrectly label the samples [49], [39], [13]. These attacks occur when the client loses control of its initial dataset and allows adversarial samples to be introduced inside its own data or at any stage of the model training.

Model Extraction The intellectual property of a model relies on its architecture and its ability to learn. In a Machine Learning-as-a-Service setting, the importance of keeping the model unknown to its users leverages its potential and market value. In a model extraction attack, an adversarial client learns a close approximation of the objective function of the trained model (f) using as few queries as possible, finding $f'(x) = f(x)$. This close approximation is based on the exact confidence values and response labels obtained by inference [84]. On the same note, this attack has also been proposed for disclosing the hyperparameters of ML models [90], [61].

Model Inversion These attacks rely on the capacity to invert the objective of the trained model; the output is used as input to understand the raw train data. This attack is based on the confidence results in comparison to the model extraction attack. Following the basis of this attack, researchers have been able to perform the reconstruction of facial images from the original dataset [26], [37].

Reconstruction Attacks In this attack, the adversary intends to reconstruct raw data used for building the model. The reconstruction attack is most common in ML algorithms that use feature vectors, such as Support Vector Machines or K-Nearest Neighbor. This exploit is seen as a white-box access attack since the adversary must have access to the feature vectors. The goal of this attack is similar to model inversion ones, i.e., reconstruction of the original data [3].

Membership Inference This attack implies querying the trained model on adversarial points and record the answer and with this, detect if the sample was used as training data [7], [77]. Thus, it is intended to infer whether a sample was in the training set based on the model output.

Unlike previous works [40], [42], [75], [31], which typically consider a small subset of these attacks, our proposal aims at providing mechanisms that cover the full range of mentioned exploits. Table I presents the relevant state-of-the-art solutions and the type of security attacks covered by these. Further, it shows the attacks addressed by the two SOTERIA's designs (SML-1 and SML-2). It is important to note, however, that attacks analogous to model extraction or model inversion that rely on knowing network and storage I/O patterns and, with these, inferring the number of computation steps or output sizes, are not covered by SOTERIA. As stated previously, we leave such attacks to future research work, which can be built on top of the proposed solution.

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

	Systems					
Attacks	*Chiron [40]*	*Myelin [42]*	*SGX-BigMatrix [75]*	*SGX-Spark [31]*	*SOTERIA SML-1*	*SOTERIA SML-2*
Adversarial Samples	✗	✗	✗	✗[2]	✓	✓
Model Extraction	✓	✓	✗	✓*	✓*	✗
Model Inversion	✓	✓	✓	✓*	✓*	✗
Reconstruction Attacks	✗	✓	✗	✓	✓	✓
Membership Inference	✗	✗	✗	✗	✓	✓

*Vulnerable to model extraction and inversion attacks if such exploits are based on access patterns, training time or output sizes.

IV. SOTERIA

SOTERIA is a system for distributed privacy-preserving machine learning, which leverages Apache Spark's design and its MLlib APIs.

A. Apache Spark - Architecture and Flow

Our solution was designed to avoid changing the architecture and processing flow of Apache Spark, keeping its scalability and fault tolerance properties. 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.

After the data loading step, the user can then submit ML processing tasks to Spark's client that is responsible for forwarding these tasks (scripts) to the Master node. Namely, tasks are submitted to the Spark Driver component which generates a Spark Context allowing to access the resource manager and then distribute the tasks to a set of Worker nodes accordingly to its needs. Therefore, the Spark Driver must have direct access to the computations, processing logic, or ML task scripts, before delegating the tasks to the Workers to optimizing resources' usage.

A task can be divided into smaller processing steps, each done by an independent Worker in parallel. Furthermore, each Worker launches one or more executors (JVM processes) that can further increase parallelism by doing local processing concurrently (*e.g.*, locally training the ML model, and collecting train statistics in parallel).

[2]Data encryption is not a module on the open-source version.

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)* [93]. 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 kept at an in-memory cache at each executor process.

As Workers may be executing concurrently different steps of a given task, these need to be able to transfer information, through the network, among each other. For example, in a distributed ML training task, this information can contain model parameters that must be exchanged frequently across Workers to increase training's accuracy.

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.

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.

Similar to the regular flow of Apache Spark, also SOTERIA can be divided into two main components or sides, the SOTERIA Client or trusted side and the SOTERIA Cluster or untrusted side. With this, the SOTERIA Client represents the user and its site and the SOTERIA Cluster represents the cloud environment.

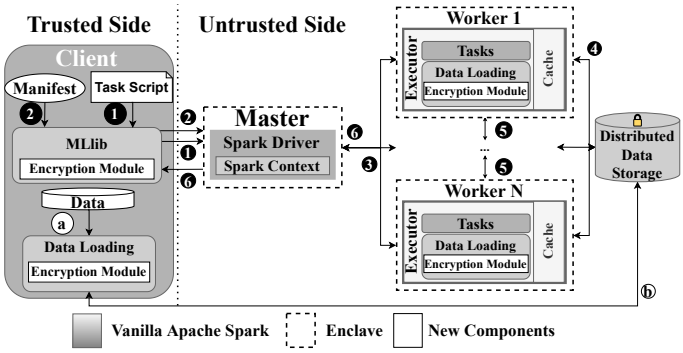


Fig. 2: SOTERIA architecture and operations flow. Main components of Apache Spark vanilla are depicted in gray boxes, whilst dashed boxes represent the components inside enclaves and white boxes depict the new components implemented in SOTERIA.

B. SOTERIA *Client*

Although SOTERIA provides an extended version of the Spark client module and MLlib, these modifications are transparent to users, thus not changing the way they usually load data into the distributed storage backend, build new ML algorithms, and execute ML tasks (*e.g.,* training and inference) at the Spark cluster. The only exception is that users must specify additional information in a *Manifest configuration file* as further described next.

SOTERIA's modified client module is used by the 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.

a) Data Loading: For the first operation, similarly to the regular usage of Apache Spark, 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.

b) Tasks submission: This stage has two main steps and two main files, namely, the ML task script and the *Manifest file*. For these operations, the transparent encryption module is integrated with MLlib. It is then 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 only 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, including users' encryption keys, and data, the information contained in ML task scripts, and the output of executing these tasks, is only accessed in its plaintext format at the trusted user premises or inside trusted enclaves.

C. SOTERIA *Cluster*

As mentioned previously, 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 accessed without breaching security. Note that the cryptograms 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 directly between the trusted client module and the secure enclave.

For inference operations, the Master node needs to access the distributed storage backend to retrieve the stored ML model. As so, 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 Worker nodes to send the necessary computational instructions[3] along with the user's encryption key and *Manifest file* (Figure 2-❸). Secure channels are established between secure enclaves to avoid external attackers from gaining access to sensitive information passing through the channels.

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

a) **SOTERIA *Workers*:** Secure SGX enclaves are applied in two distinct ways at the Worker nodes. The main rationale for this is to enable two designs that offer a different balance in terms of security guarantees and performance.

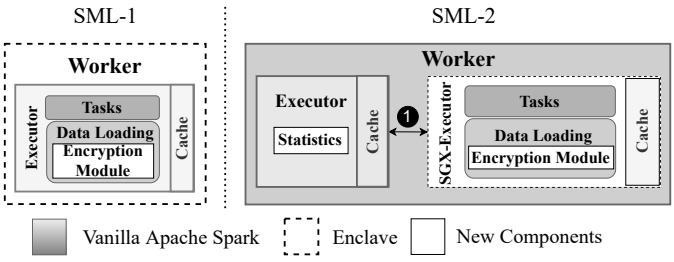


Fig. 3: Comparison between SML-1 and SML-2 designs.

b) Secure ML Design 1 (SML-1): The first design aims at ensuring that all computation done by Spark Workers is done in a trusted environment. In more detail, executor processes launched by each Worker node are deployed inside an enclave, as depicted in Figure 3. Since data read from the storage backend is always encrypted and is only decrypted at the enclave, this design ensures that all computation over plaintext information is done in a trusted enclave environment. Therefore, this design provides the best security guarantees. However, as shown in Section VI-A, this extra security has a toll on performance due to the extra amount of computation and I/O calls done at the enclave.

c) Secure ML Design 2 (SML-2): The second design is based on the observation that ML inference and training operations are composed of different computational steps. Ones that must operate directly over plaintext information (*e.g.,* train and inference datasets and model), and by others that do not require access to this data and are just calculating and collecting general statistics about the operations being done.

Following this observation, SML-2 decouples statistical processing, used for assessing the performance of inference and training tasks (*e.g.,* the calculation of confidence results, loss functions, table summaries, and probability distributions), from the actual computation of the ML algorithms done over plaintext information. As depicted in Figure 3, statistical processing is therefore 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).

With this approach, one is relaxing security guarantees, as further discussed next in Section IV-D, while improving execution time of ML tasks, as discussed in Section VI-B. It is important to note that both SML-1 and SML-2 designs can be used transparently by users, thus not affecting the way ML tasks are submitted and the API to do so.

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

D. Security Analysis

Our security proof is dependent on an intermediate result that follows naturally from [9], 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. Next, we provide the main arguments for SOTERIA's proof while the full description is in Appendix A.

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 [9], the Master node can be replaced by a reactive functionality performing these tasks. Following the same reasoning for the ML processing stage, each SOTERIA Worker behaves simultaneously as a processing node, and as a client providing inputs to the computation of other Workers (*e.g.*, model training parameters). This enables us to do a hybrid argument, where SOTERIA 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 that are 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, we can have the functionalities processing over hardcoded 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 only forwarding

[3]The same metadata sent by a vanilla Spark deployment so that Workers know what computational operations they must perform.

the client requests, and we reach 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. We note that, for simplicity, our analysis refers to SML-1. The reasoning for SML-2 is exactly the same, with the caveat that the non-sensitive statistical data is explicitly revealed as leakage by the functionality to the simulator in the ideal world.

We now focus on the ML attacks mentioned in Section III-A and on discussing how SOTERIA aims at mitigating these.

a) Adversarial Samples and Membership Inference: Authenticated data encryption and the *Manifest file* aid SML-1 and SML-2 to preserve the confidentiality and integrity of the input dataset. These mechanisms ensure that a malicious attacker cannot introduce new or modified samples to the input dataset when the corresponding model is being trained. Namely, the input dataset is encrypted by the user and explicitly defined in the *Manifest file* at the user's trusted premises. The Worker enclaves are bound to only access the data specified at the *Manifest file* during the training phase. Since the model is fully trained inside secure enclaves, a malicious attacker, at the untrusted servers, does not have access to query the model being trained.

b) Model Extraction and Model Inversions: SML-1 prevents both of these attacks by protecting the confidence values for the model being trained. Namely, these values are, again, computed inside the trusted enclave and never revealed in plaintext to external attackers at the untrusted servers.

However, SML-2 is not able to prevent these two attacks. This is a consequence of outsourcing statistical computation, specifically, the confidence intervals to an untrusted environment that can be accessed by external attackers.

As mentioned in Section III, analogous attacks to the previous two that explore instead I/O access patterns or computing output sizes and time, are not protected by SOTERIA and should be considered as future research work.

c) Reconstruction Attack: This attack requires access to the algorithm's feature vectors. In both SML-1 and SML-2, the feature vectors are only disclosed inside the trusted enclaves, thus preventing attackers from accessing them.

E. Implementation

SOTERIA's open-source prototype is implemented in Java and Scala, the preferred languages of Apache Spark, and on top of Apache Spark 2.3.0. Both SML-1 and SML-2 designs are implemented without requiring any modifications to the core Apache Spark implementation. Namely, our implementation only requires modifying Spark's data loading and MLlib libraries. Thus, our prototype does not change the distributed and fault-tolerant nature of the original Spark 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 [21].

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 [54], [9].

Nevertheless, 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 resorting to Graphene-SGX [85].

Graphene-SGX. SOTERIA's prototype uses version 1.0 of Graphene-SGX [85], 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 store data at a storage backend can be encrypted transparently by Graphene-SGX, while also enabling the creation of secure encrypted channels to communicate with other enclaves.

Graphene-SGX is based on Drawbridge's *picoprocess* [68], which provides an isolated address space for generic applications. 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 in SOTERIA's prototype. It must be noted that the *Manifest file*, discussed at Section IV-B, 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 that we use our modified data loading package, which is also used at the Master and Worker nodes to transparently decrypt data read from the distributed storage backend and encrypt inference and training results before sending them back to the client.

For the SML-2 implementation, 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 capabilities 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 [30].
We highlight that our prototype fully supports both these designs and the user can choose which one to use by just selecting the corresponding *Manifest file*.

V. Methodology

We define our evaluation to answer the following questions:

1) How do the two SOTERIA's designs impact the execution time of ML workloads?
2) How does SOTERIA compares, in terms of performance, with state-of-the-art solutions?
3) Can SOTERIA handle different algorithms and dataset sizes efficiently?

A. Environment

For the evaluation of our solution, we 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 [74]. For this environment we increase the memory of SGX to 4GB, providing a memory swapping space of 4GB. In our use case, we chose one server as client and Master and the remaining seven servers as Spark Workers.

B. Workloads

We resort to the HiBench benchmark [44], which allows evaluating different machine learning algorithms broadly used and natively implemented on top of MLlib. Namely, our evaluation considers seven algorithms, which are detailed in Table II. For each algorithm, the benchmark suite offers different workload sizes ranging from *Tiny* to *Gigantic* configurations, as seen in Table II.

C. Setups and metrics

To validate SOTERIA's performance, we compare both our architectures, namely SML-1 and SML-2, with a vanilla deployment of Apache Spark that does not provide any privacy guarantees (Vanilla). Moreover, we compare our solution with SGX-Spark [31], [73], 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 [69], this solution is designed to process sensitive information inside SGX enclaves. Therefore, this is the design that can be considered as the most similar to the one proposed in this paper. However, SGX-Spark can only ensure that UDFs 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.
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* [67] monitoring tool was used to collect the CPU, RAM, and network consumption at each cluster node.

VI. Evaluation

We split our evaluation into two different stages to present our results more clearly. Section VI-A summarizes the main evaluation observations, while Section VI-B further analyzes these observations and provides key insights.

A. 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. The vanilla Spark deployment'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 is increased 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. When executing with the LDA algorithm the results show a higher execution overhead of 17.40x, 8.89x, and 15.08x for the SML-1, SML-2, and SGX-Spark setups, respectively. Moreover, SML-2 outperforms SGX-Spark by a difference of 41.5 minutes.
Observation 4. Both SOTERIA's designs surpass the state-of-the-art solution when evaluating Kmeans. Namely, 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. When evaluated with PCA, SML-1 and SML-2 have an execution time of 3.67x and 2.85x, respectively,

TABLE II: Representation of the corresponding tasks and time complexity of each algorithm and the data sizes for different workload configurations.

Algorithms	Tasks	Time Complexity	Workloads			
			Tiny	Large	Huge	Gigantic
Alternating Least Squares (ALS) [71]	Recommendation Systems	$O((m+n)k^3 + mnk^2)$ [35]	193KB	345MB	2GB	4GB
Principal Compt. Analysis (PCA) [91]	Dimensionality Reduction	$O(nm*min(n,m)+m^3)$ [22]	256KB	92MB	550MB	688MB
Gradient Boosted Trees (GBT) [27]	Prediction	$O(n*y*n_{trees})$ [78]	36KB	46MB	92MB	183MB
Linear Regression (LR) [89]	Classification & Prediction	$O(m*n^2+n^3)$ [19]	11GB	134GB	335GB	894GB
Sparse Naive Bayes (Naive Bayes) [98]	Multi-class classification	$O(nm)$ [23]	-	-	5GB	-
Latent Dirichlet Allocation (LDA) [11]	Dimensionality Reduction	$O(mnt+t^3)$ [15]	-	-	2GB	-
K-means clustering (K-means) [20]	Clustering	$O(n^2)$ [65]	-	-	56GB	-

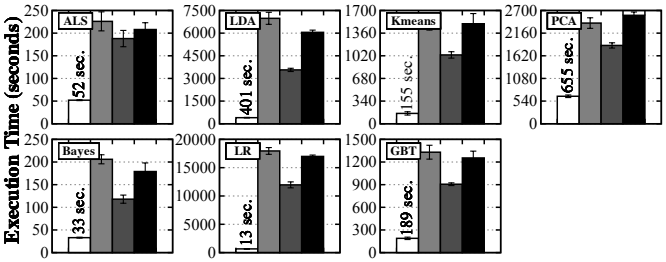


Fig. 4: Execution time for each algorithm with *Huge* workload. The legend is as follows: □ Vanilla Spark; ■ SML-1; ■ SML-2; ■ SGX-Spark.

over the vanilla results, while SGX-Spark increases the computational time by 3.95x. Similar to Kmeans, 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.

Observation 6. With the *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 the 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. Following the previous results, 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 approaches, while outperforming SGX-Spark. When dealing 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, these results portray 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. Namely, 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*) maintain a similar overhead.

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 the vanilla Spark. Also, SML-2 continues to present a lower overhead than SML-1 and SGX-Spark for all the workloads, with the execution time decreasing 8 seconds for the *Tiny* workload and 191 seconds for the *Gigantic* workload.

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 the vanilla Spark. Whilst the network shows an upper-bound limit of 15% in SML-1 with LDA due to the extra encrypted data paddings being sent between Worker nodes.

Observation 14. As a set point, we do not intend to trade-off accuracy for security. The metrics used for evaluation (*e.g.*, accuracy, root mean square error, ROC) are the ones

commonly used for each algorithm. The accuracy results obtained corroborate such claims were both SML-1 and SML-2 show accuracy results similar to the Vanilla version.

B. Analysis

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

a) ***Dataset size****:* Figure 5 shows the performance degradation expected for the PCA, GBT, ALS, and Linear Regression algorithms with increasing dataset sizes. Namely, the results show that, for PCA, GBT, and ALS workloads with smaller datasets, SML-1 and SML-2 perform similarly. On the other side, as the size of the datasets increases, the 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 [18], [96]. 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. The most expensive algorithm in terms of performance is Linear Regression as it is the one that processes more data for the distinct workload sizes (Table II).

When compared with SGX-Spark, our second design 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 the state-of-the-art solution.

b) ***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 the ALS algorithm, 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.

Similarly 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 a 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*).

c) ***Size of TCB****:* By analyzing the results from Section VI-A, we can observe that SGX-Spark outperforms SML-1 for some of the evaluated algorithms (*Observation 2, 3, 6-8*). As SGX-Spark only protects *User Defined Functions (UDFs)*[4], the performance overhead imposed by the larger trusted computing base of our solution is naturally higher.

Nevertheless, when compared to SGX-Spark, SML-1 never surpasses a performance overhead superior to 1.15x, while covering a wider range of machine learning attacks. 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. For instance, in PCA, the eigenvalue decomposition needs to be performed inside the enclave.

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

C. Discussion

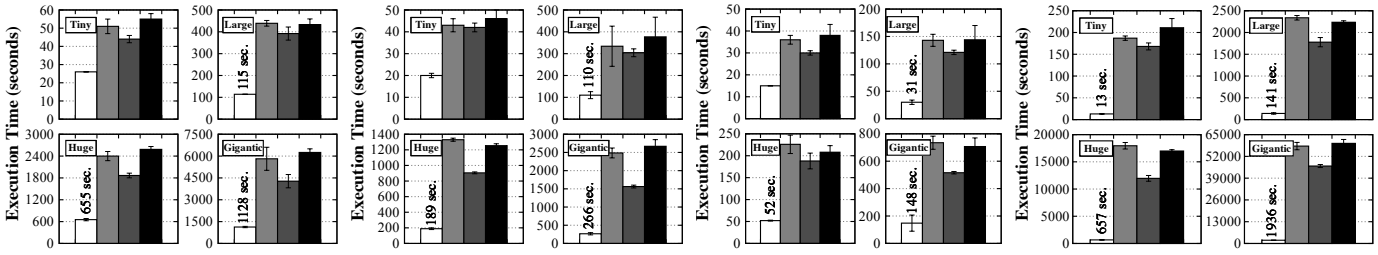
The results discussed in this section show that SOTERIA can surpass a state-of-the-art solution, namely SGX-Spark, regarding performance and coverage of machine learning attacks (Table I).

In more detail, SOTERIA's SML-1 design provides a more robust solution in terms of security, when compared with the state-of-the-art approaches, by being the first approach to consider the adversarial samples, model extraction, model inversion, reconstruction, and membership inference attacks discussed at Section III-A. Notably, this design is able to do so while exhibiting similar performance to SGX-Spark.

The SML-2 design relaxes security guarantees in order to offer significantly better execution times for all tested algorithms. This is possible because the TCB for this solution is reduced, thus mitigating the performance bottlenecks imposed by the SGX technology. Interestingly, this solution is still able to cope with adversarial samples, reconstruction, and membership inference attacks.

With this, choosing which design to use relies on the user's needs regarding security and performance. The first design should be chosen if the main goal is to guarantee the confidentiality and integrity of both models' intellectual property and sensitive data. On the other hand, the second design, while relaxing the security guarantees, discards the need to protect the model. So, if a user wants to train a common model without novel and state-of-the-art parameters, the second design still guarantees the protection of sensitive data while ensuring that a malicious attacker is unable to introduce adversarial samples to the training set.

[4]*User Defined Functions* are a feature offered by Apache Spark where the user can define a new Spark function by defining a Scala function, these functions, contrarily to Spark predefined functions, are not optimized [86].



(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.

VII. Related Work

Secure machine learning solutions can be classified into four main groups based on the privacy-preserving techniques being used: *i)* encryption-based [12], [6], [29], *ii)* secure multi-party computation [58], [70], [53], *iii)* differential privacy [76], [1] and, *iv)* trusted execution environments (TEEs) [62], [83], [40], [82], [42]. This paper is included in group *iv)*.

Privacy-preserving machine learning with TEEs. Chiron [40] enables training machine learning 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* [42] 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 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 I).

In [62], 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 within the MLlib Spark's API. Also, we underline that side-channel attacks are currently not considered by, and are therefore orthogonal, to our work. Nevertheless, solutions such as LibSodium [52] would work as countermeasures to these attacks [24].

Privacy-preserving analytics with TEEs. TEEs have also been applied to ensure privacy-preserving computation and storage for general-purpose analytic frameworks [14], [50], [75].

In comparison to SGX-Spark [31], detailed in Section V-C, 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 common attacks to the machine learning pipeline, as shown in Table I.

Opaque [97] and Uranos [46] also resort to SGX to provide secure analytics but only support a very restricted set of ML algorithms. Briefly, 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 is shown that UDF processing can be ported to secure enclaves. However, the proposed use-case only contemplates a single machine learning workload. SOTERIA differs from these works since it supports a broader spectrum of machine learning 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. It is also worth mentioning that TEEs have also been applied to ensure privacy for the training and inference of deep neural networks[83], [82], [48], [51]. However, there is a substantial difference between the internals of ML and DL frameworks and algorithms. Therefore, the privacy-preserving considerations and designs must be adapted to each scenario.

To the best of our knowledge, SOTERIA is the first framework to cover a large spectrum of machine learning exploits (Table I) and to provide two TEE-based designs that balance different trade-offs in terms of security and performance. Also, it supports a wide variety of machine learning algorithms, as shown in our evaluation, while not changing how users build and run their algorithms with Spark MLlib.

VIII. 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 machine learning training and inference phases.

In more detail, we propose two different designs with different balances in terms of performance and security. The first (SML-1) builds on the idea that to propose robust security guarantees and cover a larger spectrum of attacks than related work, SGX must be used for a majority of Apache Spark processing steps. The second design (SML-2), trades off security guarantees to reduce the Apache's trusted computing base that must be executed on a secure enclave. This entails a smaller performance overhead at the cost of not being able to protect users from model extraction and inversion attacks.

We perform an extensive evaluation of SOTERIA and compare it with the SGX-Spark state-of-the-art solution. Results show that SOTERIA's SML-1 is able to maintain similar performance to SGX-Spark while covering a wider range of known machine learning attacks. However, by relaxing the security offered by SOTERIA's first design, while still providing stronger guarantees than SGX-Spark, SML-2 surpasses the state-of-the-art results by up to 41%. Also, for both designs, we show that the performance overhead for seven different algorithms ranges from 1.7x to 30.8x when compared to a non-secure baseline deployment of Apache Spark.

We strongly believe that SOTERIA's twofold approach is a fundamental asset to push forward a wider usage of secure machine learning in third-party cloud computing services while enabling users to choose the best compromises in terms of performance and security for their data and workloads.

ACKOWLEDGEMENT

We thank Ricardo Macedo, Tânia Esteves, Rogério Pontes, Mariana Miranda and Vítor Enes for their comments and suggestions to improve the paper. 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.

REFERENCES

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[2] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 79:1–79:35, Jul. 2018.

[3] M. Al-Rubaie and J. M. Chang, "Reconstruction attacks against mobile-based continuous authentication systems in the cloud," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 12, pp. 2648–2663, 2016.

[4] T. Alves, "Trustzone: Integrated hardware and software security," *White paper*, 2004.

[5] AMD, "Amd secure encrypted virtualization (sev)," https://developer.amd.com/sev/, (Last accessed on 24/02/2021).

[6] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[7] G. Ateniese, G. Felici, L. V. Mancini, A. Spognardi, A. Villani, and D. Vitali, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *arXiv preprint arXiv:1306.4447*, 2013.

[8] M. Azure, "Azure confidential computing," https://azure.microsoft.com/en-us/solutions/confidential-compute/, (Last accessed on 05/01/2021).

[9] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from sgx," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 477–497.

[10] ——, "Secure multiparty computation from sgx," in *Financial Cryptography and Data Security*, A. Kiayias, Ed. Cham: Springer International Publishing, 2017, pp. 477–497.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[12] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data." in *NDSS*, vol. 4324, 2015, p. 4325.

[13] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," in *International Conference on Learning Representations*, 2018.

[14] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, "Securekeeper: confidential zookeeper using intel sgx," in *Proceedings of the 17th International Middleware Conference*, 2016, pp. 1–13.

[15] D. Cai, X. He, and J. Han, "Training linear discriminant analysis in linear time," in *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 2008, pp. 209–217.

[16] R. Canetti, "Universally composable security: a new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, 2001, pp. 136–145.

[17] J. I. Choi and K. R. Butler, "Secure multiparty computation and trusted hardware: Examining adoption challenges and opportunities," *Security and Communication Networks*, vol. 2019, 2019.

[18] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything you should know about intel sgx performance on virtualized systems," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, pp. 1–21, 2019.

[19] J. Dowling, "Id2223 lecture 2: Distributed ml and linear regression," https://www.kth.se/social/files/5a040fe156be5be5f93667e9/ID2223-02-ml-pipelines-linear-regression.pdf, november 2017, (Last accessed on 01/12/2020).

[20] R. Dubes and A. K. Jain, "Clustering techniques: the user's dilemma," *Pattern Recognition*, vol. 8, no. 4, pp. 247–260, 1976.

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

[22] T. Elgamal and M. Hefeeda, "Analysis of pca algorithms in distributed environments," *arXiv preprint arXiv:1503.05214*, 2015.

[23] C. Elkan, "Boosting and naive bayesian learning," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1997.

[24] T. Esteves, R. Macedo, A. Faria, B. Portela, J. Paulo, J. Pereira, and D. Harnik, "Trustfs: An sgx-enabled stackable file system framework," in *2019 38th International Symposium on Reliable Distributed Systems Workshops (SRDSW)*. IEEE, 2019, pp. 25–30.

[25] T. A. S. Foundation, "Apache cassandra," https://cassandra.apache.org/, (Last accessed on 24/02/2021).

[26] M. Fredrikson, S. Jha, and T. 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, pp. 1322–1333.

[27] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[28] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford University Stanford, 2009, vol. 20, no. 09.

[29] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*. PMLR, 2016, pp. 201–210.

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

[31] L.-S. D. . S. L. Group, "Sgx-spark," https://github.com/lsds/sgx-spark, (Last accessed on 15/02/2021).

[32] A. Hadoop.

[33] M. Hahnel, W. Cui, and M. Peinado, "High-resolution side channels for untrusted operating systems," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 299–312.

[34] A. HBase, https://hbase.apache.org/, (Last accessed on 24/02/2021).

[35] X. He, H. Zhang, M.-Y. Kan, and T.-S. 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, pp. 549–558.

[36] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 123–142, 2018.

[37] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 603–618.

[38] A. Hive.

[39] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011, pp. 43–58.

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

[41] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," *ACM Transactions on Computer Systems (TOCS)*, vol. 35, no. 4, pp. 1–32, 2018.

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

[43] IBM, "Data shield - ibm cloud data shield," https://www.ibm.com/cloud/data-shield, (Last accessed on 05/01/2021).

[44] Intel, "Intel-bigdata/hibench: Hibench is a big data benchmark suite." https://github.com/Intel-bigdata/HiBench, (Last accessed on 21/02/2021).

[45] S. Iqbal, M. L. M. Kiah, B. Dhaghighi, M. Hussain, S. Khan, M. K. Khan, and K.-K. R. Choo, "On cloud security attacks: A taxonomy and intrusion detection and prevention as a service," *Journal of Network and Computer Applications*, vol. 74, pp. 98–120, 2016.

[46] X. J. Jiang, C. Tzs, O. 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.

[47] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han, "A first step towards leveraging commodity trusted execution environments for network applications," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, 2015, pp. 1–7.

[48] R. Kunkel, D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "Tensorscone: a secure tensorflow framework using intel sgx," *arXiv preprint arXiv:1902.04413*, 2019.

[49] A. Kurakin, I. J. Goodfellow, and S. 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.

[50] D. Le Quoc, F. Gregor, J. Singh, and C. Fetzer, "Sgx-pyspark: Secure distributed data analytics," in *The World Wide Web Conference*, 2019, pp. 3564–3563.

[51] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using sgx," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–17.

[52] LibSodium, "Introduction - libsodium documentation," https://libsodium.gitbook.io/doc/, (Last accessed on 03/02/2021).

[53] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 619–631.

[54] T. Machida, D. Yamamoto, I. Morikawa, H. Kokubo, and H. Kojima, "Poster: A novel framework for user-key provisioning to secure enclaves on intel sgx."

[55] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." *Hasp@ isca*, vol. 10, no. 1, 2013.

[56] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[57] MLlib, https://spark.apache.org/mllib/, (Last accessed on 07/02/2021).

[58] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[59] U. D. of Heatlh and H. Services, "Hipaa for professionals — hhs.gov," https://www.hhs.gov/hipaa/for-professionals/index.html, last accessed on 21/02/2021.

[60] M. Ogburn, C. Turner, and P. Dahal, "Homomorphic encryption," *Procedia Computer Science*, vol. 20, pp. 502–509, 2013.

[61] S. J. Oh, M. Augustin, M. Fritz, and B. Schiele, "Towards reverse-engineering black-box neural networks," in *International Conference on Learning Representations*, 2018.

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

[63] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer, "Varys: Protecting sgx enclaves from practical side-channel attacks," in *2018 $Usenix$ Annual Technical Conference ($USENIX$ ATC 18)*, 2018, pp. 227–240.

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

[65] M. K. Pakhira, "A linear time-complexity k-means algorithm using cluster shifting," in *2014 International Conference on Computational Intelligence and Communication Networks*. IEEE, 2014, pp. 1047–1051.

[66] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 399–414.

[67] A. Pollock, "dstat(1) - linux man page," Dec. 2020, (Last accessed on 03/02/2021). [Online]. Available: https://linux.die.net/man/1/dstat

[68] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, "Rethinking the library os from the top down," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, 2011, pp. 291–304.

[69] C. Priebe, D. Muthukumaran, J. Lind, H. Zhu, S. Cui, V. A. Sartakov, and P. Pietzuch, "Sgx-lkl: Securing the host os interface for trusted execution," *arXiv preprint arXiv:1908.11143*, 2019.

[70] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 707–721.

[71] R. Sands and F. W. Young, "Component models for three-way data: An alternating least squares algorithm with optimal scaling features," *Psychometrika*, vol. 45, no. 1, pp. 39–67, 1980.

[72] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 38–54.

[73] C. Segarra, R. Delgado-Gonzalo, M. Lemay, P.-L. Aublin, P. Pietzuch, and V. Schiavoni, "Using trusted execution environments for secure stream processing of medical data," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2019, pp. 91–107.

[74] I. SGX, "Intel(r) software guard extensions for linux* os," https://github.com/intel/linux-sgx, (Last accessed on 21/12/2020).

[75] F. Shaon, M. Kantarcioglu, Z. Lin, and L. 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, pp. 1211–1228.

[76] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[77] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 3–18.

[78] S. Si, H. Zhang, S. Keerthi, D. Mahajan, I. Dhillon, and C.-J. Hsieh, "Gradient boosted decision trees for high dimensional sparse output," in *International conference on machine learning*, 2017.

[79] A. Spark, "Overview - spark 2.4.0 documentation," https://spark.apache.org/docs/2.4.0/, last accessed on 20/02/2021.

[80] S. J. Stolfo, M. B. Salem, and A. D. Keromytis, "Fog computing: Mitigating insider data theft attacks in the cloud," in *2012 IEEE Symposium on Security and Privacy Workshops*, 2012, pp. 125–128.

[81] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 24–31, 2010.

[82] S. Tople, K. Grover, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and secure dnn inference," *arXiv preprint arXiv:1810.00602*, 2018.

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

[84] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 601–618.

[85] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: A practical library os for unmodified applications on sgx," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.

[86] S. U. D. F. (UDFs), "Spark 3.0.1 documentation," https://spark.apache.org/docs/latest/sql-ref-functions-udf-scalar.html, (Last accessed on 03/02/2021).
[87] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. 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, pp. 991–1008.
[88] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.
[89] H. M. Wagner, "Linear programming techniques for regression analysis," *Journal of the American Statistical Association*, vol. 54, no. 285, pp. 206–212, 1959.
[90] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 36–52.
[91] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.
[92] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 640–656.
[93] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. 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: USENIX, 2012, pp. 15–28.
[94] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," *Proceedings of the USENIX conference on Hot topics in cloud computing*, p. 10, 2010.
[95] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.
[96] C. Zhao, D. Saifuding, H. Tian, Y. Zhang, and C. Xing, "On the performance of intel sgx," in *2016 13Th web information systems and applications conference (WISA)*. IEEE, 2016, pp. 184–187.
[97] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 283–298.
[98] Z. Zheng, Y. Cai, Y. Yang, and Y. Li, "Sparse weighted naive bayes classifier for efficient classification of categorical data," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 691–696.

APPENDIX

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 schemes.

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, 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 [16] framework.

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

A. Formal Security Model

Our model considers external environment $\mathcal{Z}$ and internal adversary $\mathcal{A}$. Π denotes the protocol running in the real world, and $\mathcal{S}$ and $\mathcal{F}$ denote the simulator and functionality, respectively, running in the ideal-world. The real-world considers a Client C, a Master node M and 2 Worker nodes W_1 and W_2. This is for simplicity, as the definition and proof can be easily generalized to consider any number of Worker nodes. We also consider a global storage G, which is initialized by $\mathcal{Z}$ before starting the protocol. The Ideal functionality is parametrised by this external storage $\mathcal{F}$¡G¿, and will reveal the access patterns via leakage function $\mathcal{L}$.[5]

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.[6] 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 realizes $\mathcal{F}$ for storage G if, for all environments $\mathcal{Z}$ and all adversaries $\mathcal{A}$,

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

B. Intermediate Result

For convenience, SOTERIA does not require the Client to provide input data at the time of the ML processing, and instead the Workers are given access to 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 [10]). The reasoning is simply that, if a protocol securely

[5]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.

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

Algorithm Setup(i, m)¡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.

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

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

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

Fig. 7: Simulator for Π_2.

realises a functionality with a given input provided via secure channel, then the same functionality can be securely realized with the same input fixed in an external storage, securely accessed by the enclave.

Consider a protocol Π_1 that securely realizes some functionality $\mathcal{F}$ with simulator $\mathcal{S}_1$ according to Theorem 3 of [16]. We construct protocol Π_2 built on top of this secure protocol Π_1, where input data is pre-established and provided to the enclave via an initial Setup stage where inputs are stored in encrypted fashion (Figure 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 [10]. Then Π_2, constructed as discussed above, securely realizes $\mathcal{F}$ 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.

C. SOTERIA *Client, Master and Workers*

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

Secure channels are established with enclaves. We define $\mathsf{init}(P)$ 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 collaborative 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.[7] 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.

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

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

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

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

Fig. 9: SOTERIA Workers with split storage.

D. Full Proof

Given the description of SOTERIA components in Figure 8, the SOTERIA protocol Π_{sot} 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_{sot}, assuming the setup of Figure 6 and constructed as discussed above, securely realizes $\mathcal{F}$ 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 [10]. We can repeat this process for W_2.[8] 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 [10].

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

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

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.

[8] 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.