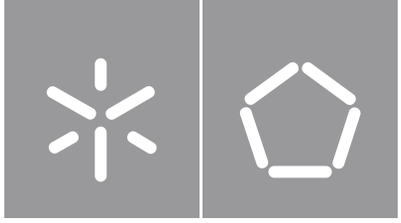
Universidade do Minho
Escola de Engenharia

Cláudia Vanessa Martins de Brito

Towards a Privacy-Preserving Distributed Machine Learning Framework

março de 2024

Towards a Privacy-Preserving Distributed Machine Learning Framework
Cláudia Brito
UMinho | 2024



Universidade do Minho
Escola de Engenharia

Cláudia Vanessa Martins de Brito

Towards a Privacy-Preserving Distributed Machine Learning Framework

Tese de Doutoramento
Doutoramento em Informática

Trabalho efetuado sob a orientação de
João Tiago Medeiros Paulo
Pedro Gabriel Dias Ferreira

março de 2024

Acknowledgements

Este documento marca o fim de um ciclo e a todos os que de alguma forma fizeram parte desta caminhada, o meu obrigada.

Começo por agradecer aos meus orientadores pelas inúmeras discussões durante estes últimos 5 anos e por me terem orientado neste percurso que por vezes pareceu interminável. O meu obrigada ao Prof. Dr. João Paulo e ao Prof. Dr. Pedro Ferreira. Não posso, ainda, deixar de salientar e agradecer ao João Paulo pois durante 5 anos acreditou mais em mim do que o que eu alguma vez consegui fazer. Obrigada João por me teres dado abertura para crescer academicamente e por me teres ajudado a definir o meu caminho na investigação. Espero que esta parceria continue por mais algum tempo.

Quero também agradecer a todos os meus coautores por todo o tempo despendido nos documentos que conseguimos produzir em conjunto para o sucesso deste percurso. Adicionalmente, o meu obrigada ao Bernardo Portela por todas as discussões, iterações e modificações durante os 4 anos de submissões do sistema base desta tese (além das provas de segurança).

Estendo o meu agradecimento a todos os membros e ex-membros do HASLab que acompanharam parte ou toda esta jornada: Ana Alonso, Beatriz Cepa, Bernardo Portela, Catarina Fernandes, Carlos Baquero, Daniel Cruz, Diogo Ribeiro, Fábia Pereira, Fábio Coelho, Francisco Cruz, Francisco Maia, Francisco Neves, João Dias, João Marco, José Orlando Pereira, Luís Meruje, Mariana Miranda, Miguel Peixoto, Nuno Machado, Paula Rodrigues, Pedro Moreira, Ricardo Macedo, Ricardo Vilaça, Rogério Pontes, Rui Monteiro, Rui Ribeiro, Rui Carlos Oliveira, Susana Marques, Tânia Esteves e Vítor Enes, bem como todo o corpo docente e todos aqueles com quem tive o privilégio de trabalhar desde 2017. Obrigada também a todos os alunos de mestrado a quem tive o privilégio de co-orientar e que me ajudaram a crescer academicamente.

Deixo aqui também o meu agradecimento e reconhecimento ao Grupo de *Storage*, João, Ricardo e Tânia pela integração de uma incógnita no vosso grupo, apesar de continuarem sem saber onde me colocar. Contudo, espero um dia deixar de ser apenas *a pessoa do ML* ou *a pessoa de biocenas* e que realmente encaixe por lá. Tânia, foi um prazer partilhar estes últimos 5 anos contigo assim como as dores do crescimento académico. Ricardo, obrigado por teres sido o primeiro a ver além do ML e por me teres introduzido a um mundo de *storage* que eu não tinha tido oportunidade de conhecer. Obrigada aos três.

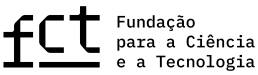
Um agradecimento especial ao Prof. Dr. António Sousa pela oportunidade dada há 7 anos, pela amizade e por me ter dado sempre um porto seguro onde ele não existia.

Ao meu *Apostulado*, juntamente com Chica, Patrícia e Reizinho, que me acompanham há cerca de 11 anos, obrigada por me ajudarem a manter a sanidade mental (ou a falta dela). À Inês, obrigada por me teres ensinado a diferença entre adenosina, adrenalina e atropina. Assim como pelos últimos 15 anos de amizade, apoio, partilha de traumas e mais recentemente as viagens aos mercados de Natal.

Finalmente, o meu obrigada à minha família. Os últimos anos foram, no mínimo, desafiantes e, em especial, um obrigada à minha mãe que sempre me apoiou e ensinou desde cedo a perseverança e resiliência mesmo quando o mundo parece estar contra nós. Agradeço também ao meu irmão, cunhada, madrinha, afilhado e avô que sempre me apoiaram e incentivaram neste percurso. Não posso deixar de dedicar este documento a três daqueles que partiram antes do fim. À minha avó, uma segunda mãe, que me impulsionou a seguir este sonho. Ao meu avô que sempre me ensinou a lutar por aquilo em que acredito. Ao meu pai que fora todos os valores incutidos, também me ensinou a ouvir boa música. A todos eles, o meu obrigada.

Para todos os que possam entender: agradeço mais do que tudo ao meu Tico e Teco, foram eles os percursores de tudo.

Adicionalmente, os meus agradecimentos às instituições que apoiaram o trabalho aqui apresentado. A Fundação para a Ciência e Tecnologia (FCT) apoiou este trabalho através da bolsa de doutoramento SFRH/BD/146528/2019. O Departamento de Informática da Universidade do Minho, o HASLab, e INESC TEC ofereceram-me as condições necessárias para o desenvolvimento deste trabalho.



STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

Braga, March 4th, 2024

(Cláudia Vanessa Martins de Brito)

Resumo

Rumo a uma Plataforma Para Aprendizagem Máquina Privada e Distribuída

A Aprendizagem Máquina (AM) tornou–se uma técnica essencial para vários sectores (*p.ex.*, saúde, finanças) que pretendem extrair novas informações dos seus dados. No entanto, estes tendem a conter informações sensíveis, levantando preocupações sobre a privacidade e segurança e levando ao desenvolvimento de soluções de Aprendizagem Máquina com Preservação da Privacidade (AMPP). Isto é particularmente relevante quando esses dados, assim como a computação feita sobre eles, precisam de ser transferidos para infraestruturas de terceiros (*i.e.,* computação em nuvem) ou diretamente processados no dispositivo móvel do utilizador. Nesta tese, mostramos que as soluções atuais apresentam várias limitações, sendo apenas aplicáveis a casos de uso específicos, exigem que os utilizadores reimplementem os seus algoritmos de AM ou comprometem significativamente o desempenho das cargas de trabalho.

Para responder a estes desafios e melhorar a adoção prática de soluções AMPP, propomos três contribuições. Primeiro, introduzimos o *Soteria*, um novo sistema que aproveita a escalabilidade e a fiabilidade do *Apache Spark* e da sua biblioteca ML (MLlib). Este garante que as operações críticas são realizadas exclusivamente em enclaves seguros fornecidos por Ambientes de Computação Confiáveis (ACC). Isto significa que os dados sensíveis a ser processados só existem em claro dentro do enclave, estando cifrados no resto do fluxo de dados (*i.e.*, rede, armazenamento). Esta solução assegura a privacidade dos dados durante o treino e inferência. Embora o *Soteria* se revele uma solução prática de AMPP para AM genérica, este não suporta outros tipos de dados, como é o caso dos dados genómicos. Assim, propomos o *Gyosa*, uma nova solução de computação distribuída para estudos de associação do genoma (*GWAS*) com preservação da privacidade. Diferente de outras soluções, o *Gyosa* oferece uma diferenciação fina entre informação sensível e não sensível processada por *GWAS* num ambiente distribuído. Finalmente, o *TAPUS*, centra–se no compromisso entre precisão e privacidade para ambientes de AM distribuída sem acesso a hardware especializado. Para tal, combinamos a Aprendizagem Federada e a Privacidade Diferencial (PD) e avaliamos o impacto de diferentes algoritmos baseados em PD sobre o desafio de compreender as preferências dos utilizadores em termos de modalidades de transporte.

Com estas contribuições, melhoramos o atual estado da arte dos sistemas de aprendizagem automática distribuídos e com preservação da privacidade.

Palavras-chave: Ambientes de Computação Confiáveis, Aprendizagem Máquina, Computação Distribuída, Privacidade, Privacidade Diferencial

Abstract

Towards a Privacy-Preserving Distributed Machine Learning Framework

Machine Learning (ML) has become an essential technique for several sectors (*e.g.*, Healthcare, Finances) that wish to extract novel insights from their data. Nonetheless, such data tends to contain sensitive information, which raises concerns about privacy and security and leads to the development of privacy–preserving machine learning (PPML) solutions. This is particularly relevant when such data, along with the computation done over it, need to be outsourced to third–party infrastructures (*i.e.,* cloud computing, HPC) or directly performed in the user's mobile device. In this thesis, we show that the current state–of–the–art solutions still pose several limitations as these are only applicable to specific use cases, require users to reimplement their ML algorithms, or significantly compromise the performance of these workloads.

To answer these challenges and improve the practical adoption of PPML solutions, we propose three main contributions. First, we introduce *Soteria*, a novel system that leverages the scalability and reliability of Apache Spark and its ML library (MLlib). It ensures that critical operations are exclusively performed in secure enclaves provided by Trusted Execution Environments (TEEs). This means the sensitive data being processed only exists in plaintext inside the enclave and is encrypted in the remainder of the dataflow (*i.e.*, network, storage). This solution enables robust security guarantees, ensuring data privacy during ML training and inference. While *Soteria* proves to be a practical PPML solution for generic ML, it does not support other types of data or workloads that may benefit from privacy–preserving guarantees, which is the case of genomic data. Therefore, we propose *Gyosa*, a novel distributed computing solution for privacy–preserving genome–wide association studies. Different from other solutions, Gyosa offers a fine–grained differentiation between sensitive and nonsensitive information processed by GWAS in a distributed environment. Finally, *TAPUS*, focuses on the trade–offs between accuracy and privacy for distributed ML setups that do not have access to specialized hardware. This is done by combining federated learning and differential privacy and evaluating the impact of different DP–based algorithms over the challenge of understanding users' transportation modality preferences.

With these contributions, we improve the current state–of–the–art of privacy–preserving and distributed machine learning systems.

Keywords: Differential Privacy, Distributed Computing, Machine Learning, Privacy–preserving, Secure Enclaves

Contents

Appendices

List of Figures

List of Tables

List of Algorithms

Acronyms

2PC two-party computation 29, 30
3PC three-party computation 29
4PC four-party computation 30

ACC Ambientes de Computação Confiáveis vi
AI Artificial Intelligence 10, 89, 91, 99, 112, 114
AM Aprendizagem Máquina vi
AMPP Aprendizagem Máquina com Preservação da Privacidade vi
API Application Programming Interface 46, 71
AWS Amazon Web Services 1

CPU Central Processing Unit 10

DL Deep Learning 7, 10, 13, 19, 21, 22, 34, 37, 38, 69, 71, 100, 104, 105, 109
DML Distributed Machine Learning 1, 2, 3, 9, 14, 21, 113
DNN Deep Neural Network 18, 35, 37, 69
DP Differential Privacy vii, 2, 3, 4, 5, 6, 7, 25, 30, 31, 32, 33, 34, 41, 86, 90, 91, 92, 94, 99, 100, 102, 103, 105, 106, 110, 113, 114
DT Decision Trees 30, 33, 37

EPC Enclave Page Cache 47
ERM Empirical Risk Minimization 32
ETL Extract, Transform and Load 14

FHE Fully Homomorphic Encryption 27, 28

FL Federated Learning 1, 3, 4, 5, 7, 15, 19, 21, 32, 33, 34, 35, 88, 90, 91, 92, 93, 96, 97, 98, 99, 100, 102, 103, 104, 105, 109, 110, 113, 114

GC Garbled Circuits 25, 26, 29, 40
GCP Google Cloud Platform 1, 103
GHG Greenhouse Gas 89, 96, 97, 99, 101, 103, 109, 110
GPU Graphics Processing Unit 1, 10, 22, 114
GWAS Genome-Wide Association Studies vi, 4, 5, 6, 7, 9, 37, 71, 74, 75, 76, 79, 80, 81, 82, 88, 112

HE Homomorphic Encryption 2, 3, 25, 27, 29, 30, 34, 35, 40, 41, 86
HPC High-Performance Computing vii, 1, 112

IOT Internet of Things 3, 41, 88, 91, 113

LCA Life Cycle Assessment 96, 97, 101, 109
LibOS Library Operating System 36
LLM Large Language Model 114
LR Linear Regression 4, 30

MCC MedCloudCare 8
ML Machine Learning vii, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 20, 21, 22, 23, 25, 29, 31, 33, 34, 40, 41, 43, 44, 45, 46, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 62, 66, 67, 68, 69, 70, 71, 78, 88, 90, 92, 96, 97, 98, 100, 104, 109, 110, 112, 113, 114, 147, 149, 150, 151, 154
MPI Message Passing Interface 18

NFS Network File System 103
NN Neural Network 29, 30, 37, 105

OS Operating System 46, 47
OT Oblivious Transfer 25, 26, 37, 38

PATE Private Aggregation of Teacher Ensembles 33
PD Privacidade Diferencial vi
PHE Partially Homomorphic Encryption 27, 28
PPDL Privacy-Preserving Deep Learning 29

PPDML Privacy-Preserving Distributed Machine Learning 2, 3, 4, 5, 41, 42, 43, 44, 45, 70, 71, 74, 112, 113, 114

PPFL Privacy-Preserving Federated Learning 4, 5, 7, 9, 90, 91, 109, 110, 111

PPML Privacy-Preserving Machine Learning vii, 2, 3, 5, 9, 10, 32, 37, 38, 39, 68, 112, 113

PS Parameter Server 15

SGD Stochastic Gradient Descent 13

SGX Software Guard Extensions 3, 35, 37, 38, 44, 46, 88

SIMD Single Instruction Multiple Data 28

SMPC Secure Multi-party Computation 2, 3, 25, 29, 30, 34, 35, 37, 86, 87, 110

SS Secret Sharing 25, 26, 29

SVM Support Vector Machine 30, 32, 37

SWHE Somewhat Homomorphic Encryption 27, 28

TCB Trusted Computing Base 35, 36, 40, 41, 44, 74

TEE Trusted Execution Environments vii, 2, 3, 4, 6, 25, 35, 37, 38, 40, 41, 42, 44, 47, 68, 74, 86, 87, 91, 111, 112, 113

UDF User-Defined Function 38

VANET Vehicular Ad hoc Networks 110

VCF Variant Call Format 4, 82

VM Virtual Machine 103

WtW Well-to-Wheel 101, 109

ZKP Zero-Knowledge Proof 30

1 Introduction

In the era of digitalization and ubiquitous connectivity, data has emerged as the cornerstone of modern technological progress. Combined with Machine Learning (ML), large-scale data analysis has revolutionized industries ranging from healthcare to finance, enabling insights and predictions that were previously unimaginable [218].

Nonetheless, the exponential data increase has led to a paradigm shift in how this data is processed. The traditional processing approach in a single machine is no longer feasible. Instead, data is distributed across different nodes, and the computation is parallelized to decrease the execution time. This distribution of data and computation has led to the emergence of Distributed Machine Learning (DML), which aims to provide a scalable solution to process large volumes of data while resorting to ML techniques.

Briefly, DML enables training models across different nodes in a collaborative and distributed manner. However, the task of processing large volumes of data still poses several challenges. Although it distributes the load to improve the runtime performance, DML still requires efficient computation nodes (*e.g.,* with Graphics Processing Units (GPUs), high-speed network, and storage) to process the data. This often leads to the necessity of outsourcing the computation to third-party infrastructures, such as cloud providers (*e.g.,* Google Cloud Platform (GCP), Amazon Web Services (AWS), Azure [1, 21, 60]) or High-Performance Computing (HPC) facilities, or the strict collaboration between different entities (*e.g.,* hospitals, financial institutions). For example, several entities may collaborate to train a centralized model that will profit from several data sources. The computation is done on each entity's premises, and only the resulting values from the trained model (*i.e.,* gradients) are shared with a centralized server that aggregates this information to update the centralized model. This is a common setting in Federated Learning (FL) [27], which is particularly relevant in the healthcare domain. In this area *i)* the amount of data is increasing exponentially, and *ii)* patterns or insights on rare diseases, *i.e.*, patterns within small subsets of data from different healthcare facilities, need to be found. If the first problem can be solved by outsourcing the computation to third-party infrastructures, the second can be solved by collaborating with other entities to train a model that would profit from distinct data sources [67].

However, this shift towards distributed data-driven decision-making in untrusted third-party infrastructures has brought to the forefront a critical concern: preserving privacy in the face of ever-increasing data

sharing and the outsourcing of data processing. The risk of inadvertent data exposure or unauthorized access increases as data is shared between multiple entities or computed in third-party infrastructures [131, 279].

Indeed, the ML pipeline can be complex as it encompasses several stages, which go from data collection and preprocessing to model training and inference, each with its distinct computational load (*e.g.,* regarding data and computation workflow). In detail, the data preprocessing stage depends on the data format and the transformations (*e.g.,* data cleaning or pruning, image rotation, oversampling) performed on it as these often require human judgment. On the other hand, the model training and inference depend on the model's complexity and the amount of data used to train it. This extensive pipeline when outsourced becomes susceptible to a broad spectrum of attacks, such as *adversarial attacks*, *model extraction* and *inversion* and *reconstruction attacks* [89, 286, 296]. By aiming to extract sensitive information, these attacks can compromise the integrity and confidentiality of the data (*e.g.,* model inversion) and the model (*e.g.,* model extraction). As such, these attacks hinder the outsourcing ML tasks to untrusted third-party infrastructures, such as cloud providers [131]. Furthermore, data privacy regulations are tightening globally, and public awareness of personal data's value is surging [118, 295].

Addressing these challenges is of paramount importance to enable the adoption of DML solutions in highly sensitive domains, such as healthcare, finances or urban mobility. Although acknowledging the importance of these privacy concerns, traditional distributed computation frameworks like Apache Spark [314] lack security guarantees to protect the data or the models against these attacks.

To overcome these challenges, an extensive compendium of privacy-preserving solutions has been proposed in the literature, which has led to the emergence of Privacy-Preserving Machine Learning (PPML). This new subfield of ML aims at providing privacy guarantees to ML systems. Most of these privacy-preserving solutions can be decomposed into two main categories: *PPML* and *Privacy-Preserving Distributed Machine Learning (PPDML)*. The former refers to applying privacy-preserving techniques to a single machine learning model being trained by a single server residing at an untrusted infrastructure. The latter refers to using privacy-preserving techniques in a distributed architecture. Moreover, these distributed and privacy-preserving approaches can be further decomposed into *collaborative learning*, where the goal is to train a model using data from different sources and within each entity's premises. In *distributed computation* the goal is to outsource the training or inference of a model to third-party premises which will distribute the computation, by parallelizing and consequently, decreasing the execution time of these tasks.

Solutions to PPDML range from the usage of Homomorphic Encryption (HE) or Secure Multi-party Computation (SMPC) [32, 97] to the exploration of Trusted Execution Environments (TEE) (*i.e.,* to allow data computation to be securely handled and processed in a secure environment) [126, 129, 153] and even the application of Differential Privacy (DP) [3, 266]. The broad spectrum of solutions and the different cryptographic primitives used to protect data and computations have led to a wide range of trade-offs between privacy, performance, and utility.

1.1 Problem Statement and Objectives

Despite the growing interest in PPML, current solutions are either too restrictive or performance-wise inefficient. Also, they can require changes in how users interact with the system. HE and SMPC schemes impose a significant performance toll that restricts their applicability to practical scenarios [231].

When shifting the focus to TEEs, such as Intel Software Guard Extensions (SGX), these environments provide a more efficient alternative to protect sensitive data and computations. Nonetheless, the need to resort to specific hardware and SGX's memory limitations that impose high performance overhead still hinder its broad usage [120]. In detail, confidentiality is guaranteed by the hardware when resorting to a TEE (*i.e.,* through trusted regions known as enclaves). Based on this, current SGX-based solutions typically deploy full ML workloads inside these enclaves [126, 129, 153]. As the amount of computational and I/O operations performed at the enclaves increases, the performance of ML training and inference is noticeably affected by hardware limitations, limiting the design's applicability in practice [73]. This shows that the trade-off of TEEs is made between privacy and performance. Computation partitioning (*i.e.,* splitting the computation between trusted and untrusted environments) has been studied as an alternative approach to performing this processing more efficiently but applied to different types of computation such as SQL processing, distributed coordination, or MapReduce [36, 135, 321]. ML workloads have different security requirements and processing logic than the ones found for these workloads. As such, the trade-offs between privacy and performance may differ from each use case.

The need to use specific hardware limits the applicability of TEE-based solutions to environments where such hardware is not available (*e.g.,* mobile, Internet of Things (IOT)). DP has been widely explored and applied in the context of DML, namely in Federated Learning (FL). However, this solution lingers on the trade-off between data privacy and utility since introducing noise or perturbation to the data or the parameters may restrict the model's accuracy and convergence time (*i.e.,* model's utility) [15]. This trade-off should be carefully studied to understand where to apply DP-based algorithms and how much noise to add. Similarly to the previous primitive, these trade-offs may differ from each algorithm or use case.

The balance between utility, performance, and privacy is a fundamental challenge in PPML. Exploring different cryptographic techniques strives to balance the competing demands of data utility and individual privacy. Moreover, there is still a gap between the solutions focusing only on one of the two final stages of the ML pipeline (*i.e.,* model training and inference) and the ones that focus on both the most computational expensive stages or even in the entire pipeline. For this, we argue that a comprehensive PPML system should answer the following question:

Is it possible to balance privacy, performance, and utility in a PPDML solution?

With this in mind, the main goal of this thesis is to explore the intricate interplay between distributed machine learning and privacy preservation and to dissect the underlying mechanisms of existing privacy-preserving techniques. We further split this goal into three main objectives.

First, by leveraging third-party infrastructures with access to TEE, we aim to understand the trade-offs between privacy and runtime performance in a PPDML solution. This solution must comply with the following: *i)* applicable to different ML algorithms, *ii)* have no impact on the accuracy of the models, and *iii)* should be resilient against the attacks mentioned above.

Our first objective builds upon the idea that computation can be balanced between the enclaves created by TEE and an untrusted environment. This can be achieved by reducing the number of operations done at enclaves, therefore reducing the runtime performance of ML tasks. Nonetheless, this balance must be carefully studied as the computation to split and balance may differ from each use case, as seen in the work depicted in the following chapters.

Our second objective is to apply such a solution to a different use case, where data privacy is critical, for instance, the human genomic data. In detail, by focusing on different data formats, namely Variant Call Format (VCF) and several tasks such as GWAS and Linear Regression (LR) for GWAS, one must consider the different requirements and define a particular balance between the trusted and untrusted code. The demand for this new balance comes from the different data formats and operations performed on each of the algorithms' implementations. Although having the same mathematical intuition, the LR applied genomic data, differs in implementation when compared to financial data as it must be able to address different types of data, find correlations and also provide new insights related to the predicted *p-values*.

Finally, our third objective focuses on environments where TEEs are not available, and several entities collaborate for a central model (*i.e.,* in typical FL environment). As such, we focus on the applicability of DP in a FL setting to the transportation and urban mobility domain, where the data is heterogeneous, the privacy requirements are different, and the hardware availability is limited. With this, we aim to offer a solution to privacy-preserve data while training a model to understand users' mobility patterns and transportation modes in mobile environments. The main goal is to understand the trade-offs between privacy and utility in a real-world use case defined for and by smart cities following a Privacy-Preserving Federated Learning (PPFL) architecture.

These three objectives intend to provide a comprehensive answer to the question stated above.

1.2 Contributions

The following contributions achieve the goals above-mentioned:

Soteria. The first contribution of this dissertation focuses on delivering a secure approach applied over Apache Spark's ML library, MLlib, which already provides both the scalability and the fault tolerance of a distributed system.

Our solution relies on the features of Apache Spark as a distributed system and the security guarantees of TEEs to promote a privacy-preserving distributed approach. The key insight of Soteria is that ML runtime performance could be improved by reducing the number of operations done at enclaves. In fact,

this insight is backed up by previous work exploring the partitioning of computation across trusted and untrusted environments, but in contexts with different security requirements and processing logic than those found in ML workloads (*e.g.*, SQL processing [321], MapReduce [135], distributed coordination [36]). Understanding and defining the set of trusted and untrusted ML operations is a challenging task. Ideally, these operations should significantly reduce the enclaves' overall computational and I/O load for different ML workloads, and doing so should not leak critical sensitive information during the execution of ML workloads.

In summary, Soteria introduces a new computation partitioning scheme for Apache Spark's MLlib, Soteria-P, that offloads non-critical statistical operations from the trusted enclaves to untrusted environments. By providing a security proof and a comprehensive experimental evaluation, we show that Soteria-P reduces the runtime of ML algorithms by up to 41% compared to previous related work.

Gyosa. While the problem of PPDML can be solved with Soteria, applying such a solution to other types of data, such as genomic data, requires the adaptation of the analysis pipeline since genomic data relies on different algorithms. Despite Soteria providing a generally applicable solution for ML algorithms, one must be able to support these application-specific algorithms when dealing with genomic data.

With a focus on Genome-Wide Association Studies (GWAS), Gyosa offers a scalable, distributed, and privacy-preserving solution for GWAS analysis. While Soteria relies on Apache Spark's MLlib, Gyosa relies on *Glow* [99], a framework for genomic data analysis. The intricacies of genomic data and the different algorithms applied lead to a different approach regarding the hybrid computation scheme.

Results show that Gyosa is able to privacy-preserve genomic analysis while maintaining the original results of GWASes. As expected, there is a trade-off between runtime computation and the level of privacy-preserving guarantees similar to Soteria. Gyosa shows a runtime execution overhead ranging from $2.5\times$ for X^2 statistic tests to $10\times$ on regression-based algorithms. These results highlight that it is possible to effectively reduce the computation runtime overhead by distributing the computation across multiple nodes while still offering privacy guarantees for sensitive data.

TAPUS. As the final contribution, we explore how to ensure PPML for collaborative distributed environments where TEEs are not available. With this in mind, we focus on a use case related to smart cities and propose TAPUS. TAPUS is a prototype that explores users' geolocation information to understand their mobility patterns and transportation modes while promoting sustainable travel behaviors and preserving the privacy of this data.

This solution resorts to FL to provide a mobile prototype that averages the gradients from several users to train a model that can be used to understand their mobility patterns and transportation modes. Privacy is preserved by following a PPFL architecture and applying DP, where noise is added in two different ways: *i)* directly to the data (using local DP), or *ii)* to the gradients (by leveraging two different algorithms *DP-FedAVG* and *DP-FedSGD*). The trade-offs between privacy and utility are evaluated by comparing the accuracy of the model with and without the application of DP for the three cases.

The results show the feasibility of using both DP-FL algorithms and local DP mechanisms to protect

users' data. In sum, the results show a decrease of up to 25% of the accuracy when using Local DP with an ϵ of 0.5. When evaluating the scalability of TAPUS, we show that the system can handle several clients while preserving users' data privacy. In these experiments, we show that the system can handle up to 50 clients with up to 16% decrease in accuracy.

1.3 Results

Core publications. The contributions of this thesis have been published or are under submission to distinct international conferences and journals.

- Brito, C., Ferreira, P., Portela, B., Oliveira, R. and Paulo, J. **"SOTERIA: Preserving Privacy in Distributed Machine Learning."** In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, 2023 [38].

  This conference paper presents Soteria, a privacy-preserving distributed machine learning framework that leverages Apache Spark [314] as its underlying computing mechanism and MLlib API as the ML engine. Unlike previous work, where all ML-related computation is performed at trusted enclaves, Soteria introduces a hybrid scheme, combining computation done inside and outside TEE' enclaves. By finding the balance between trusted and untrusted code and what needs to run inside and outside the enclaves, Soteria implements an alternative to run ML workloads with Apache Spark MLlib [192] in a privacy-preserving manner. The experimental evaluation validates that our approach reduces the runtime of ML algorithms by up to 41% compared to previous related work. Soteria is available as a prototype at https://github.com/claudiavmbrito/soteria.

- Brito, C., Ferreira, P., Portela, B., Oliveira, R. and Paulo, J. **"Privacy-Preserving Machine Learning on Apache Spark."** In *IEEE Access*, 2023 [40].

  This journal publication extends Soteria's conference publication [38] by presenting a more in-depth analysis of the security guarantees and the performance overhead of Soteria for new machine learning algorithms. The experimental evaluation validates that our approach reduces the runtime of ML algorithms by up to 41% compared to previous related work. A security proof and a discussion regarding resilience against a wide spectrum of ML attacks accompany our protocol.

- Brito, C., Ferreira, P. and Paulo, J., **"A Distributed Computing Solution for Privacy-Preserving Genome-Wide Association Studies."** Available as a preprint in bioRxiv [37].

  This paper presents Gyosa, a privacy-preserving distributed GWAS framework that leverages Soteria to perform GWAS in a privacy-preserving and scalable way. Gyosa redefines the GWAS pipeline to be executed in a distributed fashion by resorting to *Glow* while ensuring that sensitive data is not leaked during the execution of the pipeline. Following the same paradigm of computation partitioning, Gyosa also defines which computation is performed inside and outside the enclaves, being

mindful of the security guarantees, possible leakage, and performance overhead. In brief, Gyosa allows users to delegate their GWAS analysis to untrusted third-party infrastructures confidentially. The experimental evaluation validates the applicability and scalability of Gyosa, reinforcing its ability to provide enhanced security guarantees. The results show that, by distributing GWASes computations, one can achieve a practical and usable privacy-preserving solution. Gyosa is available as a prototype at https://github.com/claudiavmbrito/gyosa.

- Pina, N., Brito, C., Vitorino, R., Cunha, I. **"Promoting sustainable and personalized travel behaviors while preserving data privacy."** In *Transportation Research Procedia - Proceedings of TRALisbon, 2022* [232].

  This conference paper proposes the implementation and deployment of a FL system for understanding users' transportation modalities while reporting its carbon footprint and raising awareness of its environmental impact. Created under the FranchetAI project, this paper introduces a prototype that can analyze geolocation data while preserving users' privacy. The privacy of users is guaranteed by resorting to DP and implementing a PPFL prototype. The experimental evaluation validates the applicability and scalability of FL in the transportation domain.

- Brito, C., Pina, N., Esteves, T., Vitorino, R., Cunha, I., Paulo, J. **"Promoting sustainable and personalized travel behaviors while preserving data privacy."** Accepted on *Transportation Engineering (TRENG)*, 2024.

  This journal publication extends the conference publication [232] by presenting a more in-depth analysis of the security guarantees and the performance overhead of Federated Learning for the transportation domain. Also, it provides a more thorough background on the topic of FL for the transportation use case and the application of ML/DL algorithms for understanding mobility patterns and transportation modes. The prototype also relies on explainable AI to provide insights to the users regarding their mobility patterns. One of the system's main features is the usage of DP to preserve the privacy of the users' data following a PPFL architecture. The implementation of the solution resorts to the Flower framework [25] and allows the seamless integration of both the explainable AI and the DP mechanisms.

Complementary publications. The following publications were published in collaboration with multiple researchers. Although not directly related to the main contributions of this thesis, these works leverage several topics addressed in it.

- Cepa, B., Brito, C. and Sousa, A. **"Generative Adversarial Networks in Healthcare: A Case Study on MRI Image Generation."** In *IEEE 7th Portuguese Meeting on Bioengineering (ENBENG), pp. 48-51*, 2023 [47].

This conference publication presents preliminary results on the applicability of Generative Adversarial Networks (GANs) in the healthcare domain. This work shows the potential of GANs to generate synthetic MRI images that can be used to train machine learning models. The main goal is to overcome the lack of data in the healthcare domain by generating synthetic data that can be used to train machine learning models. This work resorts to Chainer [49] to implement the prototype and evaluate the performance of GANs in the healthcare domain. This work is publicly available at https://github.com/beatrizcepa26/dissertation.

- Alves, J., Soares, B., Brito, C., and Sousa, A. **"Cloud-Based Privacy-Preserving Medical Imaging System Using Machine Learning Tools."**. In *EPIA Conference on Artificial Intelligence (pp. 195-206)*, 2022 [13].

  This conference publication presents MedCloudCare (MCC), a cloud-based medical imaging system that leverages machine learning tools to provide a more efficient and accurate diagnosis. The main goal was to propose and implement a prototype for researchers and medical staff to share and analyze medical images. Furthermore, this platform ensures the privacy of the patients' data by encrypting data at-rest and in-use as researchers may not have permission to access sensitive data in plaintext. MCC is publicly available at https://github.com/Joao231/Healthcare-App.

- Macedo, R., Correia, C., Dantas, M., Brito, C., Xu, W., Tanimura, Y., Haga, J., Paulo, J. **"The Case for Storage Optimization Decoupling in Deep Learning Frameworks."** In *1st Workshop on Re-envisioning Extreme-Scale I/O for Emerging Hybrid HPC Workloads*, co-located with *IEEE International Conference in Cluster Computing*, 2021 [180].

  This workshop publication emphasizes the limitations of current I/O optimizations implemented in Deep Learning (DL) frameworks. It proposes Prisma, a framework-agnostic middleware that implements a parallel data prefetching mechanism. We validate the applicability and portability of our approach by optimizing the training performance of TensorFlow [4] and PyTorch [225]. Prisma is publicly available at https://github.com/dsrhaslab/prisma.

- Brito, C., Machado, M. and Sousa, A. **"Electrocardiogram Beat-Classification Based on a ResNet Network."** In *MedInfo (pp. 55-59)*, 2019 [39].

  This conference publication presents a deep learning approach for electrocardiogram (ECG) beat classification. This work proposes a ResNet network to classify ECG beats into four classes. The main goal is to provide a more accurate and efficient approach to classify arrhythmic ECG beats and test the generalization of the trained model when applied to new and unforeseen data. By resorting to Apache Spark [314] and Elephas [42], this work builds ground for distributing the training of deep learning models for the healthcare environment.

1.4 Outline

In the subsequent chapters, we delve into the intricacies of privacy-preserving techniques while proposing and evaluating new solutions for privacy-preserving distributed ML. Specifically, the rest of the document is organized as follows:

- **Chapter 2** (§2). We introduce background concepts of ML, DML, and PPML and discuss the state-of-the-art approaches regarding the last topic by their chosen cryptographic primitive.
- **Chapters 3** (§3). We discuss the design, implementation, and evaluation of Soteria, a privacy-preserving distributed ML system that leverages the scalability and reliability of Apache Spark and its ML library (MLlib).
- **Chapters 4** (§4). We present the design, implementation, and evaluation of Gyosa, a tool built upon Soteria that enables GWAS to be performed securely and in a scalable way by resorting to Glow, an API built on top of Apache Spark.
- **Chapters 5** (§5). We present the design, implementation, and evaluation of TAPUS, a PPFL solution, which evaluates the trade-offs between privacy and utility in a real-world use case defined for and by smart cities.
- **Chapter 6** (§6). We discuss the final remarks and future research work.

2

Privacy-preserving Machine Learning Background

The human capacity for dealing with information has been overpowered due to the large amounts of data produced. The need to compute large amounts of information has brought Artificial Intelligence (AI) to the world of Big Data. Machine Learning (ML) techniques have set their mark on data analysis, and with the shift of computation paradigm from Central Processing Units (CPUs) to GPUs, the number of use cases in Deep Learning (DL) has suffered an exponential increase. The high computational complexity demand of these techniques and the increasing size of the datasets have directed new studies to large-scale and distributed learning and raised awareness for the interplay of AI with distributed systems. A distributed machine learning system, although computationally complex, must present the core features of a large-scale system, namely, fault tolerance, resilience, and scalability.

On the other side, the cooperation between different entities to benefit from data collected from distinct sources and the collaborative learning among these entities emphasize the need to protect this sensitive data. The lack of computational resources and the posterior need to outsource computation to third-party entities, like cloud providers, raises a similar need to protect the processed data and the computation. Although data privacy has been making the headlines for the past years, it is still a topic broadly researched, and recent works have shown the feasibility of deploying PPML solutions in real-world scenarios. All these fields are colliding to facilitate the data analysis of large amounts of data in agreement with regulations like HIPAA [118] or GDPR [94, 295].

This chapter provides an overview of the concepts and relevant state-of-the-art results for this work and is divided into four subsections. Section 2.1 presents an overview of ML and DL techniques and is followed by Section 2.2, which presents current strategies deployed in distributed ML systems. Section 2.3 presents the state of the art of PPML systems by the cryptographic primitive they implement, focusing on the most relevant works. Finally, Section 2.4 defines the lessons learned and the open challenges addressed in the following chapters.

2.1 Machine Learning

Large-scale data analytics has shifted the focus to Machine Learning and other learning techniques that simplify the computation of large amounts of data to classify, identify, generate, and organize it. Nonetheless, these techniques are not trivial since they are computationally expensive, with each algorithm having its own complexity. The goal of each task (*i.e.,* objective of the usage of an algorithm, for instance, classification, clustering, or dimensionality reduction), complexity, and data type will also influence which algorithm and technique should be deployed.

While the theoretical and mathematical side of ML must be addressed, several core concepts should not remain unattended. These concepts are the basis of ML and are essential to understanding the overall functioning of the algorithms and techniques. Next, we will address the main concepts of ML, namely the learning methods, the algorithms, the loss functions, and the optimizers.

2.1.1 Learning Methods

First, it is essential to emphasize the four main methods of Machine Learning: *i)* supervised machine learning, *ii)* unsupervised machine learning, *iii)* semi-supervised machine learning, and *iv)* reinforcement learning. These methods are based on how the data is provided to the system, relying on the terminology of labels and features.

Labels: A label is our intended prediction output. The label is usually represented by the *y* in a regression or classification problem, represented by $Y = f(X)$.

Features: On the opposite side, the features are the input of the function, usually represented by *X* ($X_0, X_1, X_2, ..., X_n$).

- **Supervised Learning** rests on existing labeled data; the ML model will learn how to map the input features into the output label. This allows learning the patterns from this data and classifying or performing regression on new and unforeseen labeled data. This method can be seen as a simple linear function $y = f(x)$, where it is given the *y* and *x*, and the goal is to predict the approximation function *f*. This approach is usually applied in classification and regression problems [26].

- **Unsupervised Learning**, contrary to supervised learning, can classify unlabeled data into different groups, pointing towards two different applications: clustering and association. This method takes the unlabeled data and finds relationships between data, and these relationships allow for clustering the data by groups. Considering this, unsupervised learning deals with a collection of unlabeled data $(X, ?)$ and intend to learn $f(X_m) \longrightarrow X_n$, where $m >> n$. Onto this matter, there are several models of *f*, namely dimensionality reduction, Principal Component Analysis (PCA), manifold learning, clustering, and auto-encoders, among others [26].

- **Semi-Supervised Learning** merges supervised and unsupervised learning; labeled data is used to train the models, and unlabelled data is used to predict or validate the outcome. This method can be used for all kinds of applications if correctly applied when in the presence of training datasets.

- **Reinforcement Learning** focuses on finding *y* by reaching an approximation function from an initial linear function, with *x* and *z* being the input and the action performed in the system, respectively. This is accomplished based on a sequence of actions, observations, and rewards. The overall goal is to produce a policy for acting in the environment and adapting itself to new conditions introduced to the environment. Reinforcement learning agents can learn by experience [26].

2.1.2 Algorithms, Loss Functions, and Optimizers

An ML algorithm is used to find patterns and correlations within the data. With a broad spectrum of available methods, their complexity degree varies accordingly, but their goal is to reach, within the available data, the task proposed by the user. In accordance with each task (*e.g.,* classification, dimensionality reduction, regression) to perform, different algorithms are suitable for it, and that is the main criterion on how to choose the correct algorithm [137]. Although having different purposes and characteristics, most algorithms aim to find the best objective function for the task according to its training dataset. These functions are optimized recurring to optimization algorithms, which intend to minimize or maximize the loss functions [324].

The objective function is a mathematical function that learns the internal parameters of the model being trained. These parameters are learned and updated based on the optimization algorithm. As such, optimization algorithms have a leading role in the training process of the machine learning models [102, 324].

For instance, with input samples $(X_1, y_1)...(X_n, y_n)$, where $X_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}$ in a linear model function, defined as,

$$f_{w,b}(x) = w^T x + b, \tag{2.1}$$

where $w \in \mathbb{R}^m$ represents the set of parameters and b is the intercept [307]. The parameters are found by iteratively reducing the training error. This is done by resorting to the **Loss Function**. This function is a mathematical method to measure how well the algorithm behaves for the given task. It allows quantifying how far the predictions are based on the predicted values and true values and depends on the algorithms used. However, three commonly used functions are Mean Squared Error (L_2), Mean Absolute Error (L_1), and Cross Entropy. For example, a linear regression model, such as 2.1, can use Mean Square Error as its cost function,

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - (w^T x_i + b))^2, \tag{2.2}$$

where N is the total number of data points, y_i is the actual value and $w^T x_i + b$ is the predicted value [185, 244].

The flow at which the loss function is minimized depends on the optimizer. The optimizer will aid in the search for the best coefficients or weights. The Stochastic Gradient Descent (SGD) is a standard optimization algorithm used both in ML algorithms or when training DL models. This optimization algorithm minimizes a function by iteratively moving to the lowest point. In ML, the gradient descent relies on the loss function to calculate the partial derivatives for the data points. With the partial derivatives, it is possible to learn the slope of the loss function and to know in which direction the coefficients or the weights should be updated. Algorithm 2.1 presents the overall functioning of Gradient Descent. In brief, a gradient is a vector consisting of the partial derivatives of a function. These derivatives will point to the direction in which the weights should be shifted [116, 326].

Algorithm 2.1: Gradient Descent

1: Initialize weights randomly
2: **while** not convergence **do**
3: Compute gradient, $\frac{\partial J(W)}{\partial W}$
4: Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5: **end while**
return *Weights*

The optimization algorithms rely on the knowledge of gradients, which, as said before, are partial derivatives. These gradients represent a measure of change and can point to which direction the coefficients or the weights should be updated to lower the loss function. In Deep Learning, it is also essential to understand the concept of the *learning rate*, represented by η in Algorithm 2.1. The learning rate allows taking small steps into updating the weights to make the model converge [165, 326].

One of the main concerns of the optimizer and the learning rate is being stuck in a local minimum and unable to find the global minimum, allowing it to converge. This usually happens in high dimensional data, where it is common to find several local minima, thus becoming increasingly difficult to reach the global minimum. With this in mind, the learning rate aids the optimizer in updating the weights. However, the steps shall not be too small or too large, as the first hypothesis will not be able to converge to an optimum, and the second may end up in a local minimum [116, 165].

A last concept regarding the optimizers is the *regularization*, which is added as a hyperparameter in deep learning to prevent model overfitting[1]. The regularization value allows imposing a penalty when the model presents large weights even though its prediction may be correct. This prevents the model from memorizing the training data while allowing it to learn how to generalize [165].

These ML algorithms typically follow a workflow, which is discussed below.

[1] Overfitting is a common challenge in machine learning where the model memorizes the training data and can predict well for this dataset. However, this trained model performs poorly in new and unforeseen data.

2.1.3 Pipeline

The machine learning pipeline is composed of several stages. From data collection to deploying trained models in the wild, each stage has its toll on performance and its own resources necessities for the runtime environment. In detail, the main stages of a machine learning pipeline are as follows. The first stage of any ML pipeline is the collection and generation of data, which goes through the process of Extract, Transform and Load (ETL) to be suitable as the input of the remaining workflow (Figure 2.1-❶). After this transformation, the data is set as a dataset and works as the input for the chosen model. These datasets are usually decoupled into two smaller ones, the training and the testing dataset (Figure 2.1-❷). While the first is used for training the model (Figure 2.1-❹), the second one should present new data to the trained model and intends to test how the trained model behaves with previously unseen data (Figure 2.1-❺). Alongside the creation of the two datasets, there is also the need to define the ML model/algorithm to be trained (Figure 2.1-❸).

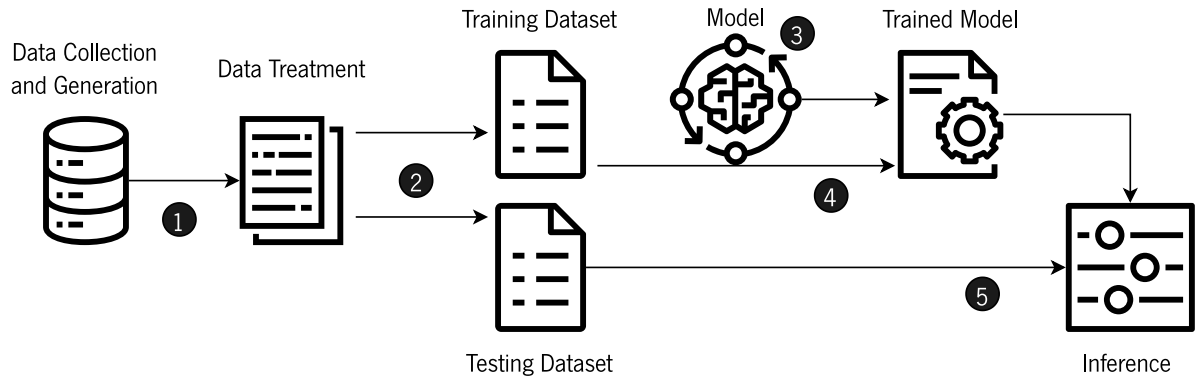


Figure 2.1: A Common machine learning pipeline and workflow, describing the train and inference approaches.

2.2 Distributed Machine Learning

The increasing generation of data, limited by the computational power of a single machine, has led to the development of DML systems. Similar to other distributed processing frameworks, these systems' main goal is to distribute the computation across several machines, decreasing the computation time.

Challenges. Distributing machine or deep learning presents several problems inherent to distributed systems. In brief, there are three goals to developing an architecture for distributed machine learning that meets the standards of distributed computing:

- **Efficiency**: The efficiency of distributed machine learning systems or distributed computing overall can be evaluated by its communication and computational costs. When dealing with distributed

computing, higher communication overhead and higher costs of machine synchronization are expected. This provides a huge opportunity for the development of new and improved optimization algorithms;

- **Ease of Use**: The trade-off between simplicity, flexibility, and utility is one crucial factor determining if the system will be used and further developed. When using distributed machine learning systems, the standard developer may seek an easy-to-use and centralized interface to provide the necessary information regarding its application, leaving on the background information like task partitioning and resource allocation as well as data communication. However, if information regarding resource allocation or the method used for task partitioning is subsequently required, the developers can easily access it and propose new optimizations;

- **Fault Tolerance**: A machine may fail at any given time, and the job that such a machine was performing may be lost. It is vital to safeguard the computation that this machine was performing. Nevertheless, it is essential to notice that the number of failures increases with the number of available machines and computation time. ML is a time-consuming task (*e.g.,* training a machine learning model may take several hours or even days). If the system does not implement a fault-tolerant mechanism, its usage becomes impractical.

Indeed, when focusing on distributed machine learning, one may find two different approaches by which the computation can be parallelized: *i)* model parallelism and *ii)* data parallelism. These two approaches differ in the tasks performed; however, the main goal is the same: it is intended to find the model that better suits the data within the minimum time possible. This goal redirects the research to choose the best methodology for each approach since there are different ways to combine the models, gradients, and weights. While we focus our efforts on ML algorithms, in this setting, the distribution follows a typical architecture of several workers computing over X datasets and broadcasting the gradients, parameters, or other information to a master node or parameter server. Nonetheless, one can follow several strategies to achieve this architecture, and each strategy has advantages and disadvantages [186, 230, 293].

2.2.1 Strategies

Over the years, there have been many procedures developed and implemented to optimize the behavior of common algorithms used in distributed machine learning, which can be decomposed into five main solutions: *i)* MapReduce, *ii)* Parameter Server (PS), *iii)* Gossip Learning, *iv)* All-Reduce and, *v)* Federated Learning (FL).

The common goal of these solutions is to provide a scalable and fault-tolerant system that can be used to train machine learning models. Moreover, besides *Gossip Learning*, these solutions typically follow a similar distributed setting with a master and several workers. The nomenclature of the master may be

different, but the main goal is to aggregate the information from the workers and to provide the necessary information to the workers. Next, each of these solutions is briefly described.

2.2.1.1 MapReduce

The emergence of big data dictated the need to distribute the computation across several nodes to decrease its execution time. MapReduce's programming model was developed to facilitate data distribution and the parallelization of the computation across several nodes while providing fault tolerance and load balancing [70].

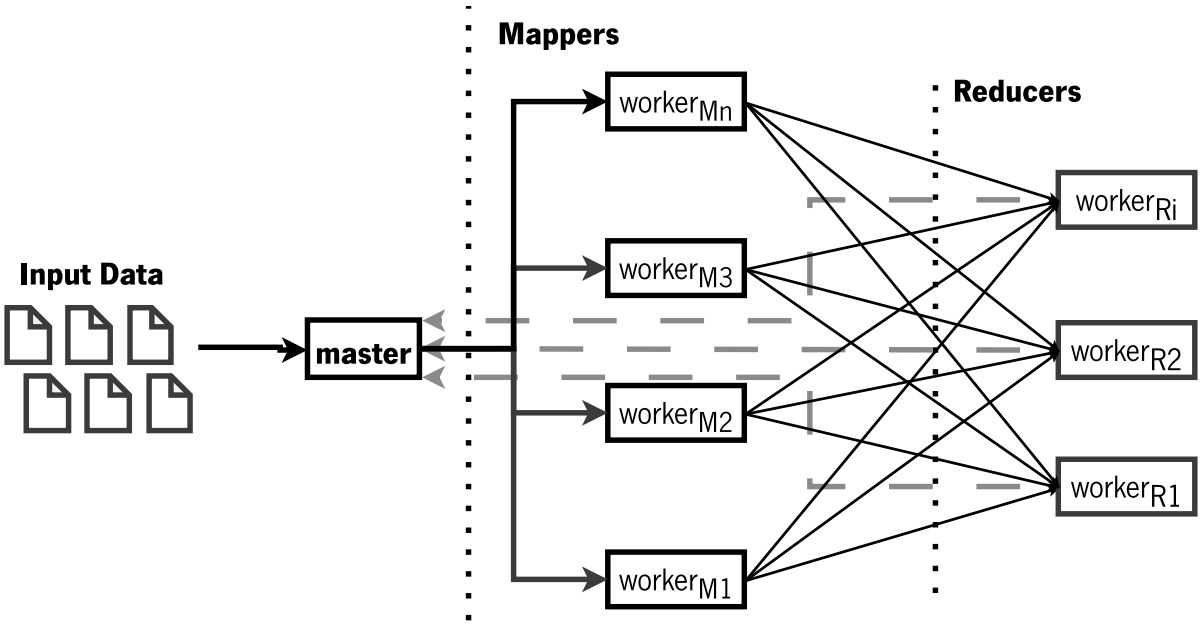


Figure 2.2: Example of a MapReduce architecture. The master M selects a set of workers W to perform the computation. The master M is responsible for distributing the data across the workers W and aggregating the results. The workers W are responsible for the computation of the map function (W_{Mn}) and the reduce function (W_{Ri}).

This model is the basis of *Apache Hadoop* [114] and comprises two main functions: the map function and the reduce function. The map function is responsible for the data distribution across the nodes, and the reduce function is responsible for the aggregation of the computation. The map function is applied to each data point, and the reduce function is applied to the output of the map function. While this model is usually applied to batch processing, and it is unsuitable for iterative algorithms, such as machine learning algorithms, there are some adaptations of this model to support these algorithms, such as in Spark [314].

This paradigm follows a master/worker architecture, where a master M selects a set of workers W to perform the computation. The master M is responsible for distributing the data across the workers W and aggregating the results. The workers W are responsible for the computation of the map function and the reduce function. The master M is responsible for the fault tolerance of the system and the load

balancing of the system, and it can redistribute the computation to another worker W if the current worker W is overloaded or in any point failed [70].

2.2.1.2 Parameter Server

The Parameter Server is a technique that provides an efficient mechanism for aggregating and synchronizing model parameters and statistics between workers.

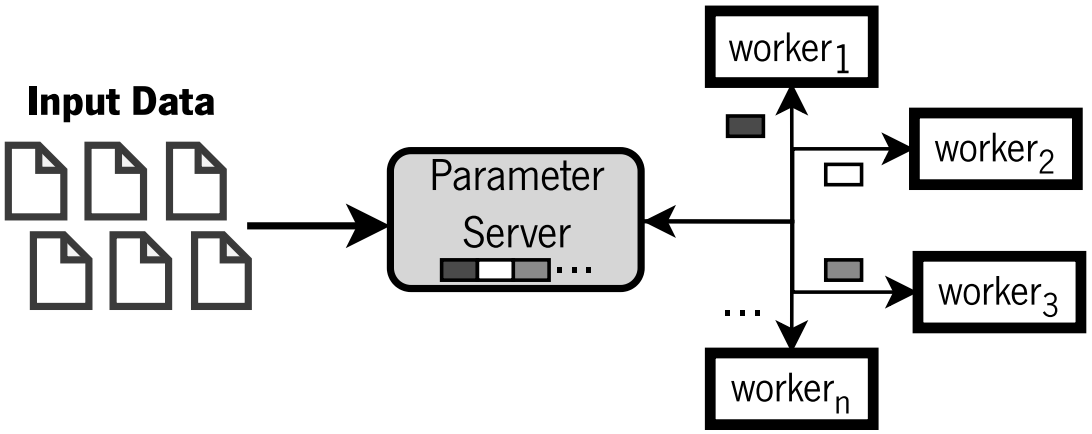


Figure 2.3: Example of a Parameter Server architecture. The parameter server sends a model replica to all its workers, and the workers send the gradients back to the parameter server to be aggregated or averaged. The new gradients are sent to the workers to compute over the model once again until it reaches convergence. These gradients are depicted as small boxes in the figures.

In ML, this strategy follows a simple structure where the parameter server sends a model replica to all its workers, and the workers send the gradients back to the parameter server to be aggregated or averaged, depending on the algorithm used, see Figure 2.3. The new gradients are sent to the workers to compute over the model once again until it reaches convergence. Depending on the methodology followed, synchronous or asynchronous, the parameter server should wait or not for all the workers to calculate the new gradients and update the model. In other words, the parameter server is a distributed shared memory system from which the worker can access the global model parameters with a key-value interface. It is possible to create a cluster of parameter servers while each manages its cluster of workers [69, 158, 270, 300, 306].

2.2.1.3 Gossip Learning

Fully decentralized machine learning is starting to be adopted as the *de facto* standard for collaborative learning. Since 2006, Gossip algorithms have been proposed to increase the opportunity to perform machine learning over distributed data. Known as peer-to-peer communication, Gossip relies on asynchronous communication between entities [33]. In machine learning, each worker is a data owner who chooses a fixed number of workers with whom they want to exchange the gradients, with the intent to

disseminate the gradients across all workers, see Figure 2.4. Gossip learning has several advantages, such as the possibility of creating a fully decentralized system and being fully asynchronous [215].

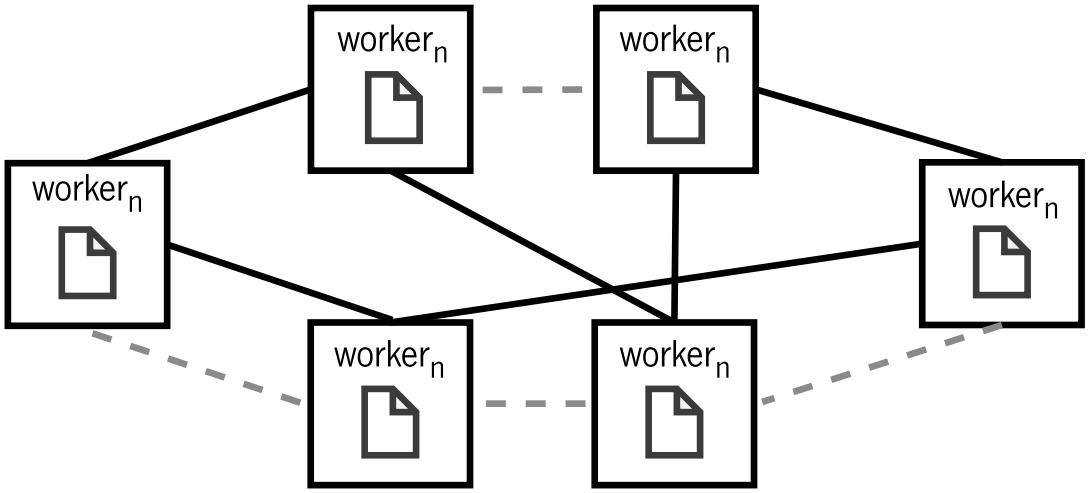


Figure 2.4: Example of a Gossip Learning architecture with a randomized algorithm. Each worker is a data owner who chooses a fixed number of workers with whom they want to exchange the gradients (in this example, this choice is random) to disseminate the gradients across all workers.

This setting has several requirements, such as *i)* the robustness of the model, once it has to still be accurate in a scenario with multiple failures, *ii)* the need for low communication complexity, and *iii)* it must allow the local use of the models for inference. These three design requirements promote the distributed setting where data owners can profit from other data owners with the same type of data without sharing any raw data [64, 215].

Without the need for strong consistency, several gossip-based SGD algorithms have been proposed, which were able to achieve both convergence and faster performance than *All-Reduce* SGD [64, 136, 240].

2.2.1.4 All-Reduce

The acknowledgment of All-Reduce algorithms relies on the knowledge of parallel computing and the fact that these processes can be modeled as directed acyclic graphs (DAG). These graphs are represented by the computations (vertices) and the data dependencies (edges). Conversely, most machine or deep learning computations can be modeled as operations on tensors, creating graphs. In distributed deep learning, these graphs are partially computed in different workers, and all the workers communicate to perform a reduction operation. These operations are based on the combination of values from all processes or workers and the distribution of the results to all processes [226].

The All-Reduce approach, used in Deep Neural Networks (DNNs) and introduced in 2015 by Baidu [305], removes the server concept. This setting usually resorts to the Message Passing Interface (MPI), and the first results, using *InfiniBand* switches, were promising.

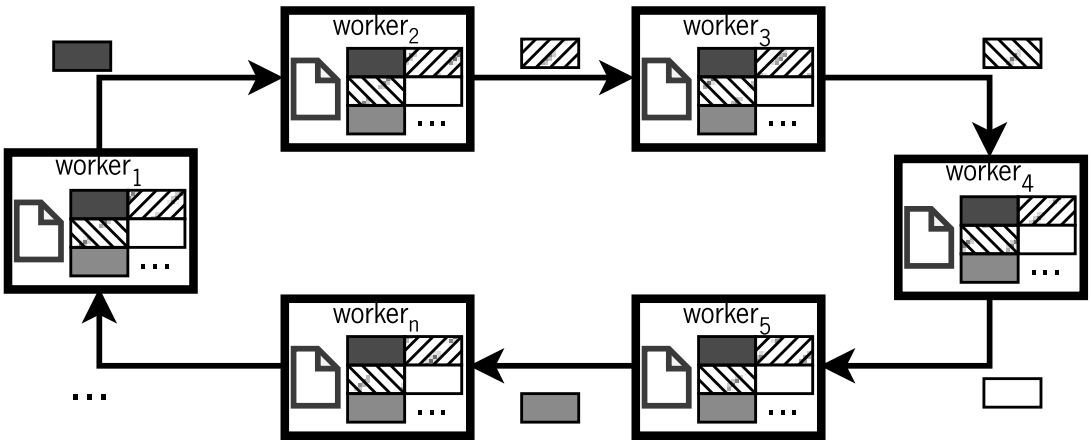


Figure 2.5: Example of an All-Reduce architecture with scatter and reduce phases. Each worker computes over their training data with their model replica and maintains a subset of parameters, here represented as small boxes. In the last communication round, all workers have the aggregated parameters.

This algorithm is commonly applied on GPU processors, and each GPU core is seen as a worker. As seen in figure 2.5, the All-Reduce algorithm follows a simple architecture, where each worker computes over their training data with their model replica and maintains a subset of parameters, i.e., each worker computes its local gradients. Afterward, all workers communicate in a ring-like pattern to compute an aggregate gradient and update their models. This process is repeated until the model converges.

2.2.1.5 Federated Learning

Federated Learning's (FL) primary assumption relies on the fact that it works without the user sending its data to the cloud, providing a distributed mindset with some privacy guarantees. This technique is intended to train a high-quality centralized model while the training is distributed over many clients, who present an untrustworthy and relatively slow network connection, and also highly unbalanced and Non-IID data [27].

As a federated computation environment, this computation scheme has a server coordinating all the devices participating in computing aggregations of its private data. Federated computation can also be seen as a MapReduce setting for decentralized data with privacy aggregation built-in. The main difference between typical MapReduce computation performed at a data center is that, in this particular case, the system has limited computation, intermittent compute node availability, and intermittent data availability [145, 189]. This setting is known to be highly distributed once the training data is stored across many devices.

FL settings allow the usage of other implementation methods such as *Split Learning*. This methodology implies splitting DL models by their layers to different worker nodes, keeping part of the model in the parameter server [282]. Nonetheless, we consider this methodology orthogonal to the work developed in this thesis and highlight that it is in its early stages of development.

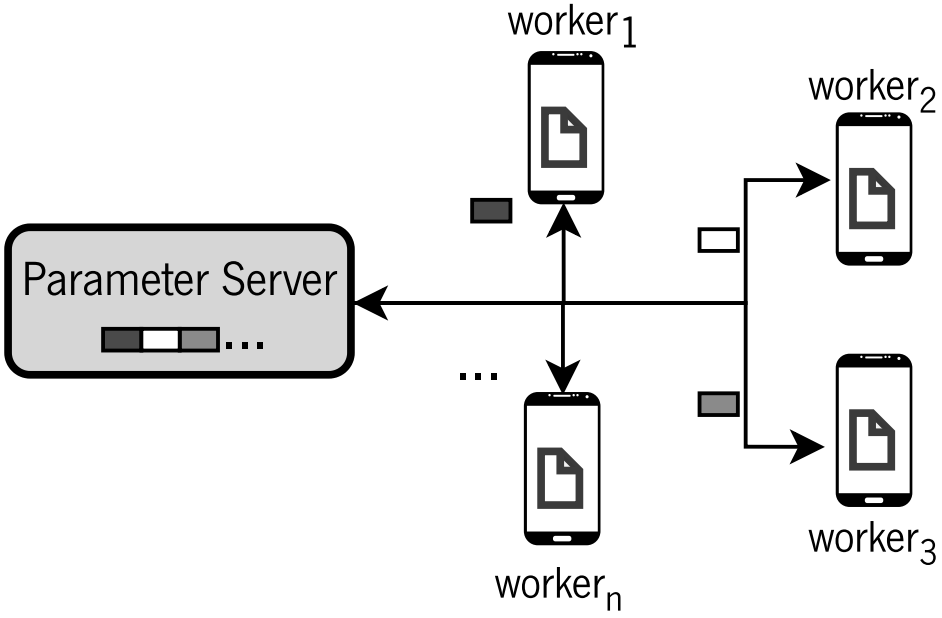


Figure 2.6: Federated Learning architecture. Similar to the parameter server architecture, the server coordinates all the devices participating in computing aggregations of its private data. The devices may choose to participate in the computation or decline the request. The gradients shared between the devices and the server are depicted as small boxes in the figure.

2.2.2 Summary

Each of these strategies works for different environment settings and requirements. With a growing number of distributed machine learning frameworks, understanding each strategy's advantages and disadvantages is vital. Table 2.1 summarizes the pivotal features of each strategy, namely its distribution, efficiency, fault tolerance, privacy, scalability, and applicability. The distribution feature can be split into distributed computation per definition or collaborative computation.

In the first case, distributed computation stems from the goal of decreasing the training or inference time of an ML algorithm by leveraging distributed resources (*e.g.,* from cloud services). Second, in a collaborative computation setting, several nodes from different entities collaborate to train a standard model, each resorting uniquely to their data. The efficiency feature relates to the computation and communication overheads such a strategy may incur. The fault tolerance feature relates to the strategy's ability to withstand failures.

In detail, *MapReduce* presents itself as the most scalable, fault-tolerant, and simple framework. Although its usage can be hindered by its native implementation, optimizations to this architecture model, which are available, for instance, in *Apache Spark*, changed its core and led to in-memory computation. This novel approach removed its bottleneck on batch processing or iterative algorithms and improved its latency by reducing disk-based I/O operations. Regarding fault tolerance, *MapReduce* has been improving this feature over the years, proposing different alternatives to complete the jobs and not affecting its completion time while currently being resilient to byzantine failures [191]. Although these optimizations

Table 2.1: Summary of main strategies for distributed machine learning.

Strategy	Distribution	Efficiency	Fault Tolerance	Privacy	Scalability	Applicability
MapReduce	D	↑	●	○	↑	A, ML, DL
Parameter Server	D	↑	○	○	↑	ML, DL
Federated Learning	H	↑	◑	●	↑	DL
All-Reduce	D	↓	◑	○	↑	DL
Gossip Learning	H	↓	◑	◐	↓	DL

Distribution
D - Distributed
H - Collaborative

Efficiency
↑ - Efficient
↓ - Not Efficient

Fault Tolerance
● - Byzantine
◑ - Crash
○ - No

Privacy
● - Yes
◐ - Partial
○ - No

Scalability
↑ - Scalable
↓ - Not Scalable

Applicability
ML - ML Algorithms
DL - DL Algorithms
A - Analytics (Queries, Stats)

have improved its performance and resilience, one of the drawbacks of this approach is the lack of privacy guarantees.

Similarly, *Parameter Server* presents a simple architecture that is easy to use and implement. However, the parameter server itself can be seen as a single point of failure, and with the increasing number of workers, it can become the scalability bottleneck. With this, research works have been proposed to remove the single point of failure such as the one by Li et al. [157], which proposes having multiple nodes working as parameter servers.

While mainly applied on mobile settings, *Federated Learning* have shown promising results, being applied in production in systems such as *GBoard* from Google. This emphasizes its prevalence in the DML field. However, the hardware limitations, unstable communication, and intermittent compute node availability are seen as the main drawbacks of this approach [139]. Also, this is a setting where privacy is a focal concern since the data is collected directly on the user's device, and sensitive information cannot leave the user's premises.

On the other hand, *Gossip Learning* relies on peer-to-peer communication protocols, which are fully decentralized and asynchronous. However, the communication overhead and the low convergence rate are seen as the main drawbacks of this approach, and its current fault-tolerant mechanisms do not prevail in the presence of byzantine failures, which is a feature shared with FL.

Regarding privacy, FL assumes a threat model where data is not shared among other devices. Similarly, *Gossip Learning* is typically applied in a setting where the entities have some degree of trust. This trust can be improved by resorting to a gossip protocol that provides some degree of privacy. However, the privacy guarantees of these strategies may not suffice as they may not mask the shared gradients or limit the type of computation performed.

Finally, *All-Reduce* is a scalable strategy that reaches high convergence rates. Nonetheless, it can present a bottleneck when the number of workers increases since it is based on a ring-like communication pattern. This issue stems from the increasing communication and synchronization times. In regard to fault tolerance, MPI-based algorithms do not natively provide this property. Nonetheless, recent works have been proposed to tackle this challenge by providing mechanisms that allow for executing a global

summation of gradients when there are failed nodes (*i.e.,* all the remaining active nodes still have the summation of the gradients) [149]. Additionally, this strategy is mainly applied on GPU processors and achieves high throughput levels, a critical factor in the distributed machine learning field [281].

Last, most of these strategies have been primarily applied to DL models. However, *MapReduce* and *Parameter Server* have been leveraged in the training of simple ML algorithms without the usage of GPU.

2.3 Privacy Preserving Machine Learning

Following the success surrounding ML, sensitive data is now being leveraged to improve healthcare services, genomic studies, urban mobility, and financial and insurance proposals. While regulations have been imposed to preserve private data, auditing and regulating it in every setting is still challenging. A solution is to offer privacy-aware alternatives that limit or reduce the leakage of sensitive data [219].

Security. Security in machine learning portrays the need to protect the models from malicious attackers who want to modify the results from the model, making these models misclassify data. Adversarial attacks are imminent, and ML models should be strong enough to endure an attack. It is essential to emphasize the work developed by [222], which addressed several of these problems and has disclosed how a system can be robust to active attacks. In order to assess the vulnerability or robustness of the system to adversarial examples, several open-source projects have been proposed, such as CleverHans [221] or FoolBox [243].

Privacy. One of the major concerns of this subject relies on the amount of sensitive data that entities (*e.g.,* medical or financial institutions, pharmaceutical companies, cities, and stakeholders) want to analyze. This private data presents an obstacle since entities want to maintain control over the data, and therefore, they do not trust third parties to compute their data. Different conditions may jeopardize data privacy, i.e., users without enough computational power must transfer their sensitive data to untrusted parties, losing control over their data. As another example, the cooperation between users creates open communication channels without security guarantees and, therefore, opens to side-channel attacks from malicious adversaries who intend to profit from the users' data. However, several researchers have been focusing on new approaches to privacy-preserving the data.

Considering this, focusing on security and privacy as two different stages is crucial. Security sends us to the problem of securing the machine learning model from attackers, be it by retrieving the parameters from the model or by adding noise to modify its output. On the other hand, privacy in these systems relies on data privacy. User data is susceptible to several attacks, and the lack of protection leaves this sensitive data yielding and open to malicious adversaries. This data can then be corrupted or reconstructed, losing its wealth to the owner.

As seen in Section 2.1, the current machine learning pipeline presents a large attack surface. Next, each attack will be defined and briefly explained, allowing a thorough overview of their goals and the information needed to be successful. It will work as the basis for the solutions and contributions of this

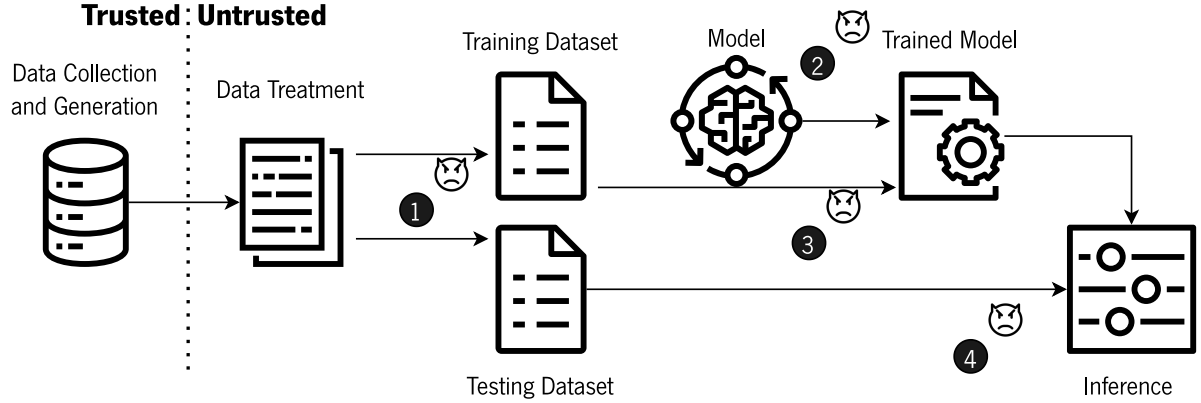


Figure 2.7: Common examples of attacks to the machine learning pipeline. Namely, the workflow shows where adversarial attacks (❶), reconstruction attacks (❷), model inversion and extraction (❸), and finally, membership inference (❹) may occur during its flow.

thesis.

2.3.1 Attacks on the machine learning pipeline

In a machine learning environment, many possible attacks can occur: attacks on the data and its transmission, the model itself, the outputs of the models, and the gradients and parameters of the model [248]. It is essential to disclose the susceptibility to these attacks and why it is vital to preserving data privacy. Another issue is using untrusted environments where data needs to be encrypted to prevent its utilization without permission. Figure 2.7 extends Figure 2.1 by exposing the common surface for each attack to occur.

- **Plaintext Sensitive Data.** Without the necessary computational power, data owners need to transfer their data to a third party. Although this data can be sent through secure channels, the computation will most likely be performed over the data in plaintext format. Without safeguarding mechanisms, this data can be fully leaked.

- **Model Extraction.** The intellectual property of a model relies on its architecture and 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 labels [286]. On the same note, this attack has also been proposed for stealing the hyperparameters of ML models [208, 296]. This attack is of major importance for companies such *Algorithmia* [11], which profits by selling other users' models in a Prediction-as-a-Service setting, or for companies like Google, IBM, or Amazon, which already provide proprietary trained algorithms for classification (Figure 2.7-❷).

- **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 initial train data. Fredrikson et al. [90] have shown that a model used to predict the correct warfarin dosage by the genome of the patient can be inverted and given a patient's warfarin dosage, inferring about a patient's genome, becoming a breach of privacy. In sum, this attack does the inversion of the objective of the trained model; the output is used as input to understand the initial train data and is based on the confidence results compared to the model extraction attack. Following the basis of this attack, researchers have been able to perform the reconstruction of facial images (Figure 2.7-❸) [89, 121].

- **Reconstruction Attacks.** The goal of this attack is similar to that of membership inference. Even so, instead of testing for a specific data point, the adversary intends to reconstruct raw data used for training the model [252]. To be successful, some attacks require the adversary to have model-specific information, namely feature vectors (*e.g.,* Support Vector Machines or K-Nearest Neighbor) [253], others only require black-box access[2] to the model [258]. Nonetheless, in the latter setting, the attacker needs to have access to another dataset with the same distribution as the original training dataset (*i.e.,* local dataset and training dataset are subsets from a larger dataset) (Figure 2.7-❷).

- **Membership Inference.** The membership inference attack implies querying the trained model on adversarial points and recording the answer and, with this, detecting if the sample was used as training data [19, 267]. It is intended to infer whether a sample was in the training set based on the model output. These attacks can be performed in different scenarios. However, it is crucial to notice the tremendous amount of work being done in the last few years regarding this subject (Figure 2.7-❹) [115, 173, 174, 203, 204, 259, 271, 310].

- **Adversarial Attacks.** Adversarial Attacks or Data Poisoning rely on injecting adversarial examples into the machine learning models. These examples are intentionally built to control the models. These adversarial inputs may cause the model to misclassify the raw data (Figure 2.7-❶) [35, 125, 151].

- **Data De-Anonymization.** One of the first mechanisms to protect data is to proceed to its anonymization or the extraction of personal information. However, even anonymized data can allow performing *linkage attacks*, where the combination of different fields and attributes of the anonymized dataset can be linked to a non-anonymized dataset and revert the anonymization. This attack showed its effectiveness when Netflix launched its 1 million challenge and training dataset of movie reviews, which were later de-anonymized by correlating them with IMDB's dataset. These results showed the possibility to de-anonymize data with high accuracy [202].

[2] *I.e.,* an adversary can query any input x and receive the predicted class probabilities $P(y|x)$ for all classes y. This allows the adversary to interact with the trained model without retrieving additional information, *e.g.*, computing the gradients

2.3.2 PPML solutions by Cryptographic Primitives

Privacy-preserving machine learning has been a hot topic for the past few years. However, quoting Papernot,"*an ML model can only be as secure as the system that hosts it*" [219]. This takes us back to the requirement of a trusted entity to host and train these ML models or the demand of having sufficient hardware resources to perform these computations on its premises. On the other side, the use of non-secure ML models can jeopardize its whole purpose (*e.g.,* if a model is not able to endure an attack that makes it wrongly train, this trained model will eventually misbehave which contradicts its first goal). As such, it becomes essential to mitigate the problems with the integrity and robustness of ML models and provide a trusted environment to perform such expensive computations.

There are several limitations when dealing with cryptographic techniques, from the high communication costs in Secure Multi-party Computation (SMPC) to the higher computation costs in Homomorphic Encryption (HE). Although different primitives have been used to deliver a functional, privacy-preserving solution for machine learning, the current state of the art highlights the loss of accuracy or the performance costs of applying such techniques to an already computationally expensive problem. However, the efforts to propose an efficient solution have not yet slowed. As the subject becomes of more importance and broadly applicable, more solutions are being proposed [101, 160, 184, 317, 320]. Even though a wide range of cryptographic primitives is being applied, such as HE, Garbled Circuits, and Secret Sharing (SS), among others, the requirements for a privacy-preserving setting overpass the advantages of these primitives. Therefore, none of the solutions already proposed appear to be optimal.

In this thesis, we intend to mitigate some challenges and limitations of state-of-the-art solutions by leveraging the use of trusted processors and a combination of different cryptographic primitives without disregarding the scale and distribution of the system. Next, we depict the current solutions by the cryptographic primitive (*i.e.,* DP, SMPC and HE, and TEE's) they rely on.

2.3.2.1 Secure Multi-Party Computation and Homomorphic Encryption

Secure Multi-party Computation (SMPC) has the goal to provide a group of participants the opportunity to compute an agreed-upon function over their private inputs without, in any case, exposing any party's sensitive information [58, 84, 214]. Multi-party computation may not rely on any cryptography assumption or protocol if half of the parties are assumed to be honest. However, if there is no assumption of half of the participants being trusted entities, it is necessary to use cryptographic protocols to guarantee that the output is correct, and cheating parties will not be able to learn anything from the honest parties [214].

Following the definition of the intuition of multi-party computation, Yao [309] provided a protocol for two-party computation, although focused on only two parties. This allowed the rise of new SMPC techniques, which were intended to improve communication and computation costs. Despite this, SMPC is still considered impractical for use in applications where real-time performance is needed.

In this section, we cover some SMPC protocols being used by state-of-the-art research in privacy-preserving machine learning, such as Secret Sharing (SS), Garbled Circuits (GC), and Oblivious Transfer

(OT).

Secret Sharing (SS)

Secret Sharing is a primary primitive of MPC and is broadly used in several MPC schemes. In a set of two or more parties, SS is used as a method to distribute a secret between parties. Each share distributed between the parties does not deliver any information. However, the secret can be reconstructed from the shares. Two entities, commonly depicted as Alice and Bob, intend to share a secret between them, where Alice has the secret *D*, and Bob is the recipient. To share its secret, Alice splits *D* into *s* shares, which will be denoted as $D_1, ..., D_s \in Z$ where Z is a finite set. This should guarantee that any *k* of the *s* shares can be used to rebuild the secret *D*. However, any set of *k-1* shares does not reveal anything, making *D* impossible to be reconstructed. The mathematical assumption of this scheme is based on the knowledge that any *k* points can define a *k-1* degree polynomial [31, 84, 262]. SS may be a better solution than GC when dealing with matrix and tensor operations or even with operations that follow a MapReduce paradigm [58].

Garbled Circuits (GC)

As a homomorphic encryption scheme, garbled circuits are also widely used in MPC settings. Yao's Garbled Circuit protocol, deemed as a two-party computation protocol, allows computing any discrete function previously represented as a fixed-size circuit. This protocol can be seen as a Secure Function Evaluation (SFE). Using the same example, Alice and Bob want to compute together a function, where one must be the *garbled circuit generator*, *garbler*, and the other must be the *garbled circuit evaluator*. The function to compute is transformed in a Boolean Circuit *C* with AND, OR, and XOR gates, and a process of garbling is performed by the *garbler*, (Alice), which obfuscates the boolean gate truth table, creating a garbled output table, commonly known as garbled tables or garbled gates. On the other side, Bob, the evaluator, will receive the garbled gates sent by Alice and will also receive the indication of which are the active wire labels[3]. Alice will also send its wire label keys while Bob's inputs are sent via an Oblivious Transfer protocol. In the end, Bob evaluates the garbled circuit and obtains the final output, which will ultimately be sent to Alice for decryption [84, 309].

One of the most known limitations of garbled circuit protocols relies on communication complexity. However, this complexity in Yao's protocol is constant once the rounds do not increase with the circuit depth, which occurs in the GMW protocol [84].

Oblivious Transfer (OT)

The Oblivious Transfer protocol is, alongside Secret Sharing, a central building block of MPC. It was first introduced in 1981 by Michael Rabin and was intended to solve a problem where both Alice and Bob have

[3]The boolean circuit *C* presents wires, and each of these wires has two keys, two possible values. These are denoted wire labels.

a secret and want to exchange secrets without using other third-party *P* and without other safe mechanisms [239]. As such, Alice, or the Sender, has a secret $x_0, x_1 \in \{0,1\}^n$ and Bob, the Receiver, has a selection bit $b \in \{0,1\}$. Bob wants to know any of two strings of the sender, namely x_b, Alice will stay oblivious to any information, and Bob will also stay oblivious to any other value of Alice. There are several OT protocols based on different assumptions, such as the protocols proposed by Naor and Pinkas [201] and Aiello, Ishai, and Reingold [10], which are based on Diffie-Hellman [229]. Although OT protocols are commonly secure when facing honest-but-curious adversaries, the integration of zero-knowledge proofs in these protocols has made them aware and secure against malicious adversaries [194]. Despite these advances, these approaches are performance-wise inefficient and unattainable for real-world applications.

Homomorphic Encryption (HE)

HE is a cryptographic primitive used to compute over encrypted data. When having two parties, Alice and Bob, Alice encrypts her data before outsourcing it to another untrusted party, Bob, to perform operations over it. When the computation is performed, Bob sends the encrypted result back to Alice, who can decrypt it and get it in plaintext. In sum, HE is an operation performed on top of cyphertexts, where the decrypted output should equal the operation's result if performed on top of plaintexts. There are several derivations of HE, specifically, Partially Homomorphic Encryption (PHE), Fully Homomorphic Encryption (FHE), or even Somewhat Homomorphic Encryption (SWHE) [7, 95, 207, 216].

Figure 2.8 presents the schematic of a simple scenario of client-server for homomorphic encryption functions. Homomorphic encryption can be seen as a cycle of 5 main actions with three associated functions (encryption, decryption, and evaluation), starting with generating a set of keys by the client, its private and public keys (P_k, S_k). This set of keys allows the client to encrypt its private message, $Enc_{Sk}(m)$, and send it along with its public key to the server. The server can now be queried to evaluate the encrypted data with the agreed function, following the confirmation with the public key of the client, $Y = f_{pk}(Enc_{sk}(m))$. Afterward, the server returns the result of this computation to the client, only remaining the decryption of the result, $Dec_{Sk}(Y)$.

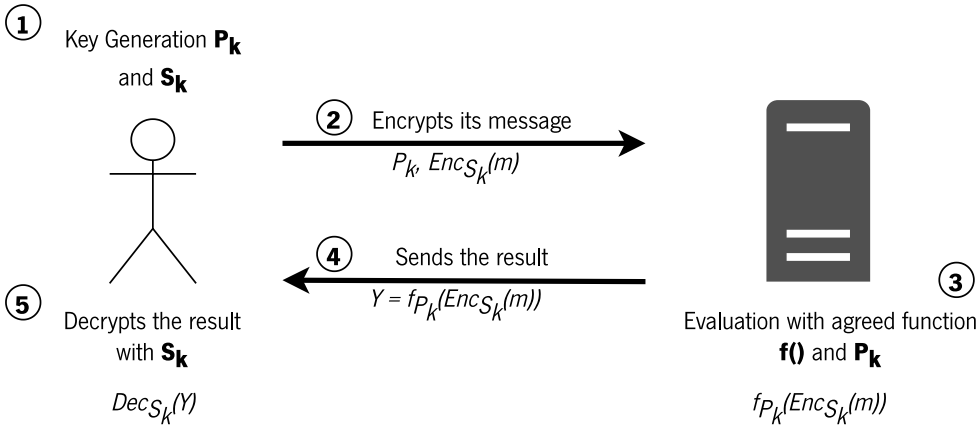


Figure 2.8: Schematic representation of the homomorphic encryption primitive.

Partially Homomorphic Encryption (PHE). This primitive is limited to some operations, such as addition and multiplication, and is only able to perform one of the operations. There are three main PHE cryptosystems: RSA, ElGamal, and Paillier. The first two cryptosystems present a multiplicative homomorphism while the last one has additive homomorphism [79, 216, 249]. In brief, this cryptosystem allows additions over encrypted data, providing a good solution for e-voting schemes, among others [216].

Somewhat Homomorphic Encryption (SWHE). Similarly, SWHE is limited to a specific number of operations or even some circuits. This terminology depicts homomorphic schemes that allow evaluating Boolean and arithmetic circuits [7]. The scheme proposed by [309], where the function to compute is described as a *Boolean circuit*, enables two parties, A and B, to jointly compute this function over their private inputs without revealing anything except the output of the function. Additionally, the SWHE scheme proposed by Boneh, Goh, and Nissim was considered the first step towards an FHE scheme and allowed to do one multiplication and many additions while keeping the ciphertext size constant [7, 30].

Fully Homomorphic Encryption (FHE). Finally, FHE follows the same pattern as the other two approaches. FHE schemes allow a boundless number of evaluation operations to be performed on top of the encrypted data while guaranteeing that the results are within the ciphertext space. Currently, FHE algorithms are not yet practical for real-world applications, but FHE schemes developed for specific applications (*e.g.,* to handle the computation of large numbers, to perform Single Instruction Multiple Data (SIMD) operations) show its practicality [184].

Table 2.2: Operation comparison between the three types of homomorphic encryption.

Primitive	Operations	Schemes
PHE	Addition or Multiplication	RSA ElGamal Paillier
FHE	Searching, Sorting, Max, Min, AND, OR, NOT	Gentry
SWHE	Circuits (AND, OR, NOT), Addition, Multiplication	Garbled Circuits BGN

PHE: Partially Homomorphic Encryption **FHE:** Fully Homomorphic Encryption **SWHE:** Somewhat Homomorphic Encryption

These three types of primitives present different characteristics, being employed according to the operations they enable to perform (Table 2.2). However, although there were several approaches to make FHE a realistic solution for practical implementation, it is still challenging to employ such a technique since it is extremely slow to compute.

According to Acar et al. [7], the performance of any encryption scheme should be evaluated by its security degree, speed, and simplicity. As such, homomorphic encryption schemes must be able to ensure that an attacker cannot obtain any information using any attack while being fast and transparent. The last feature is its simplicity, which allows these cryptosystems to be used in other systems, such as multi-party computation.

Current Solutions

Currently, SMPC and HE are usually combined to deliver secure and private solutions. In this sense, Ma et al. [179] recently proposed a promising new framework for privacy-preserving ML where the obtained results have surpassed previously state-of-the-art solutions. This work shows the feasibility of running both homomorphic encryption and secure multi-party computation for collaborative machine learning. Nonetheless, it is important to notice that similar to other solutions, this framework focuses on specific machine learning algorithms, such as Neural Networks (NNs).

In a different setting, Li et al. [159] uses different keys between the clients and the servers to keep data encrypted but be able to compute over it. The multi-key schemes this work uses present high privacy guarantees, where the sensitive data is privacy-preserved, as well as the intermediate results and the trained model. However, using fully homomorphic schemes has shown low-efficiency results, increasing the computational requirements in the system. In a multi-party computation environment, the *SecureML* [198] framework enables multiple clients to share their data between two non-colluding servers in a two-party computation (2PC). These servers train different models on the joint dataset of all clients without learning any information beyond the trained models. This approach can scale to two or three servers and relies on GCs, HE, and SS. This procedure allows performing arithmetic computations on shared decimal numbers. However, it cannot compute non-linear functions, which are present in NNs as activation functions. *ABY3* [197] is an extension of the previous work, *SecureML*, by focusing on three-party computation (3PC) which incurs several changes and improves the runtime performance when compared with its 2PC counterpart (*i.e.,* SecureML).

Phong et al. [231] create a new version of the Privacy-Preserving Deep Learning (PPDL) methodology with additively homomorphic encryption and asynchronous stochastic gradient descent. They also prove that the approach used by Shokri and Shmatikov [266] is not the best approach to privacy-preserve local data. They achieve a good balance between security and accuracy, being the only trade-off in the increased communication between the learning participants and the cloud server. So, their approach magnifies the trade-off between efficiency/privacy and not accuracy/privacy.

Alternatively, Gupta and Raskar [112] proposes a multi-party computation approach without cryptographic primitives, relying only on the lack of sensitive information the encoded representations exhibit. In this method, two entities, Alice and Bob, with Bob being a High-Performance Computer, perform a two-party computation. Alice and Bob initiate their computation at the same time with random parameters. Alice iterates over its dataset and transmits encoded representations to Bob while Bob then computes losses and gradients and sends them back to Alice.

Ryffel et al. [255] offer the first reliable general framework for privacy-preserving deep learning (*PySyft*). While proposing a standardized protocol for communication between workers, the authors focus their efforts on the federated learning setting. Built over PyTorch, this framework aims to create a chain abstraction model on tensors to override operations while also allowing the implementation of differential privacy and secure MPC primitives.

By following the previous approaches, *BLAZE* [227] and *FLASH* [41] are two frameworks that employ SMPC to allow the execution of LR, Logistic Regression, and NNs. These frameworks main goal is to optimize previous solutions, such as *SecureML* [198], *ABY3* [197] and *ASTRA* [52]. More recently, these solutions were extended to support four-party computation (4PC) in *Tetrad* [147] and *Trident* [53]. The evolution to 4PC allows for a more efficient dot product computation and also the usage of simpler protocols for communication between parties. The applicability of most of these solutions is still limited to specific machine learning algorithms, such as LR, Logistic Regression, DTs, Support Vector Machine (SVM) and NNs [6, 65].

Although novel and more efficient protocols are currently being proposed and made available at the time of writing, a thorough overview of each independent system is yet to be made as it may introduce new insights regarding the limitations and potential of each protocol. Nonetheless, as seen in the previous approaches, one of the limitations of SMPC-based solutions is the number of entities that can participate in the protocols. Most of these solutions are based on 2PC to 4PC protocols, which hinders the scalability of these solutions. Regardless, these solutions do not focus on the scalability per se but on the privacy guarantees offered.

For a further overview of these state-of-the-art solutions regarding SMPC and HE, please refer to Table 2.3. These systems are categorized by the cryptographic primitives they rely on, the machine learning stage they are applied to, the algorithms they support, and the protocol they follow. The latter is translated to the number of parties that can participate in the protocol, which is mainly applied to SMPC. Also, a thorough overview of the current state-of-the-art solutions focusing on these cryptographic primitives can be seen in the recent work from Qin et al. [238].

A different cryptographic primitive that follows the same principles of SMPC and HE is the use of Zero-Knowledge Proof (ZKP). However, we see this work to be orthogonal to the work that is being developed in this thesis, and the current state of the art is in its early stages [172, 319].

2.3.2.2 Differential Privacy and Federated Learning

The intuition behind Differential Privacy (DP) relies on the fact that changing any individual points on the input data will not change the query result, and an attacker cannot deduce private information with high confidence [76]. In this sense, Dwork and Roth [78] were able to compare machine learning and differential privacy as having the same goal: to extract new insights from data without memorizing any individual data points. DP is commonly used when two requirements are met, namely: *i)* the data to be processed is sensitive, and *ii)* the adversaries want to recover this sensitive data. Formally, aiding the anonymization of data, differential privacy introduces randomness or noise in data, limiting the leakage of private information [78].

In 2006, Dwork et al. [77] proposed the concept of ε-differential privacy to define privacy loss when collecting data from a private database. The intuition behind this definition is the same as differential privacy. If one of the elements is removed from the dataset, the analysis returns the same result. Although

Table 2.3: Overview of PPML solutions resorting to Secure Multi-Party Computation and Homomorphic Encryption.

Systems	Crypt. Primitives	ML Stage	Algorithms	Protocol
SecretFlow-SPU [179]	●	●	*DL*	2, 3, n
SecureML [198]	●,◑	●	*ML, DL*	2
ABY3 [197]	●,◑	◐	*ML, DL*	3
Gupta and Raskar [112]	–	●	*DL*	n
Abspoel, Escudero, and Volgushev [6]	●	●	*ML*	3
Tetrad [147]	●	◐	*ML,DL*	4
FLASH [41]	●	◐	*ML,DL*	4
Trident [53]	●,◑	◐	*ML,DL*	4
BLAZE [227]	●	◐	*ML, DL*	3
ASTRA [52]	●	○	*ML*	3
Damgård et al. [65]	●	○	*ML*	n
PySyft [255]	○,●	○	*DL*	?
DeepSecure, CryptFlow [242, 251]	●,◐	○	*DL*	2
XONN, SIRNN [241, 245]	●,◐	○	*DL*	2
Phong et al. [231]	○	◐	*DL*	–
Li et al. [159]	○	◐	*DL*	–
Helen [322]	○	●	*ML*	–
GAZELLE [138]	○	○	*DL*	–
Bost et al. [32]	○	○	*ML*	–

Cryptographic Primitives
○ - HE
◐ - OT
◑ - GC
● - SS

ML Stage
● - Training
○ - Inference
◐ - Both

Algorithms
ML - Machine Learning
DL - Neural Networks

Protocol
N - Number of Parties

this is a valuable technique, this abstraction from specific data points may be difficult to achieve when there are limited and distinct data points. For instance, the application of this technique to the medical context can be harsh to abstract when dealing with rare conditions and these distinct data points [78].

Currently, DP algorithms can be defined by *how* and *where* the noise is added. The *how* can be defined by the noise distribution mechanisms such as *e.g.,* Laplace, Gaussian, or Exponential. Each of these mechanisms applies noise differently and accordingly to a distribution. The *where* can be defined by the location where the noise is added, *e.g.,* the input, the gradients, or the output, which in the ML pipeline translates to adding DP-based algorithms on the data treatment stage, the training stage or the inference stage (prediction phase), as seen in Figure 2.9. In sum, as defined in Ponomareva et al., the common goal of each of these approaches is to protect the original training data [235].

One of the critical concepts in DP is the notion of ϵ, which is the privacy parameter or the privacy budget [235]. The ϵ defines the level of privacy, and smaller values represent stricter privacy levels. The ϵ is inversely related to the noise added to the data, and the lower the ϵ, the higher the noise added to the data. Similarly, this concept increases the trade-off between privacy and accuracy [235].

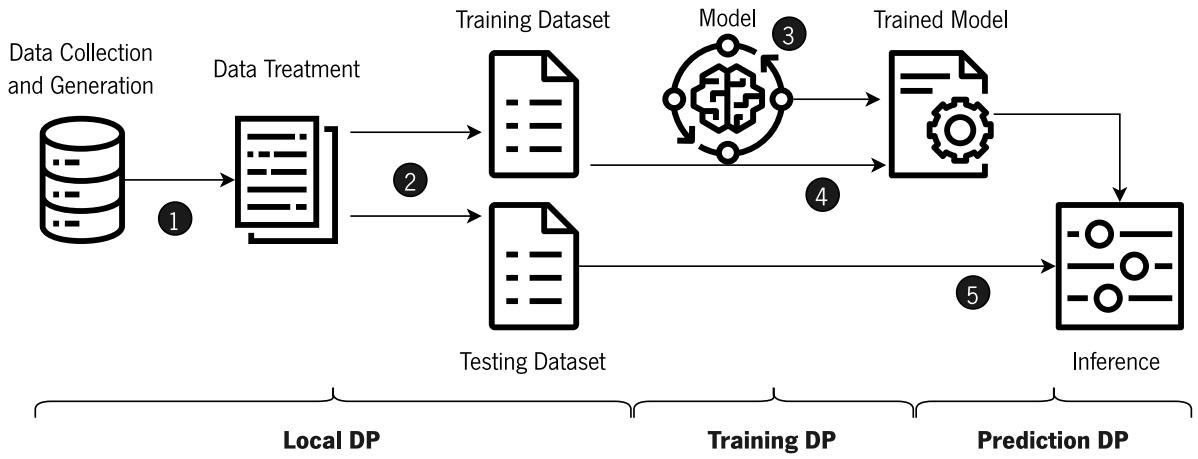


Figure 2.9: Differential privacy application on the machine learning pipeline. The noise can be added to the input data here referred to as Local DP, the gradients as Training DP, or the output as Prediction DP.

In contrast, FL, although not directly depending on differential privacy, may also apply DP-based noise to the data to protect the privacy of the users. In its common usage, FL does not offer strong security guarantees. By resorting to known protocols, such as *FedProx* [162], or *FedAVG* [189], data is not directly shared between the clients and the server. However, the clients need to broadcast their model gradients. Other protocols follow the same principle as seen in Li et al. [160].

In brief, protocols such as *FedProx* and *FedAVG* are based on an initial protocol *FedSGD*. The main difference between these protocols is the addition of a proximal term to the loss function in *FedProx* and the addition of a weighted average of the gradients in *FedAVG* [189]. The proximal term in *FedProx* allows the addition of a regularization term to the loss function, which is a common technique to avoid overfitting [162]. The weighted average of the gradients in *FedAVG* allows the addition of a weighted average of the gradients, which is a common technique to avoid the divergence of the model. However, these techniques do not hinder membership inference or model inversion attacks. With this, and as attacks rise to enable the usage of gradients to reconstruct original data, the interplay between DP and FL has been the focus of recent approaches that aim to privacy-preserve data in these settings [160].

Current Solutions

In general, when focusing on PPML, differential privacy is applied on the gradients that are broadcasted between clients and the parameter server to mitigate the leakage and the inversion of the models.

One of the first approaches was introduced by Rubinstein et al. [254] to SVMs. This approach was based on the addition of noise to the output classifier. Similarly, Chaudhuri, Monteleoni, and Sarwate [54] proposed a differentially private ERM classifier. This approach was based on the addition of noise to the gradient of the objective function.

Shokri and Shmatikov [266] proposed a selective stochastic gradient descent, allowing the selection of which parameters would be broadcast. However, their second solution allowed them to combine this

selective process with DP. In sum, this method suggested a new distributed training technique based on selective stochastic gradient descent with differential privacy. On the same note, Abadi et al. [3] proposed the combination of differential privacy with the stochastic gradient descent, although suggesting that other optimization algorithms could be used, *i.e.,* by adding a specific noise to the computation. A significant contribution of this work was the development of the "*moment accountant*" algorithm, which can track and analyze the privacy loss in these systems. In the same line of work, *Rényi Differential Privacy* was introduced by Mironov [195] as a new privacy definition that allows the analysis of the privacy loss in a more accurate way.

With this in mind and with FL not providing the security guarantees needed to perform secure training, McMahan et al. [190] introduced several changes to the federated algorithm proposed previously and implements the *DP-FedAVG*, which adds DP-noise to the parameters computed by the clients before sending them to the parameter server.

On the other side, Papernot et al. [219] proposed a new algorithm, Private Aggregation of Teacher Ensembles (PATE), with strong privacy guarantees to train models with disjoint datasets. These trained models are known as teachers, and the student model learns by the major voting among all the teachers. An interesting aspect of this approach is that when the teachers agree on a trained model, it is guaranteed that the models found a general pattern and did not memorize the data. The student will then learn by the major voting among all the teachers and cannot access an individual teacher or its parameters or data, protecting both the training dataset and the several teacher models.

A different approach is proposed in Mao et al. [183]. In this work, the authors split the model between the clients and the server, with the clients being edge devices. As such, the model is split on the first convolutional layer and sent to the mobile device. At the end of the computation, the model's output on the user side is privacy-preserved, using Gaussian noise. This output is then used as the model's input on the server side. All the intermediate results are hidden for each side.

A different approach was taken by Lecuyer et al. [154], which introduced *PixelDP*. The goal of this solution was to add noise to the input data and any intermediate layer of the model to increase the robustness of the models to adversarial samples. Unlike the other works, this work shows the feasibility of using DP to increase the models' robustness, providing groundbreaking information for further solutions.

Regarding Decision Trees (DT), several works resorting to DP have been proposed, and an encompassing overview of these solutions can be seen in Fletcher and Islam [87]. The preliminary results date to 2009 and evaluate how the implementation of DP in DT affects the accuracy of the model.

In addition, the medical field has also been a target for the application of DP in ML models, which show the applicability of these solutions to reduce the risk of privacy leakage [9, 123, 260]. These systems differ from the previous ones as they do not propose new algorithms but rather discuss the performance of the current state-of-the-art solutions in a medical setting or apply Gaussian noise by implementing an algorithm similar to *DP-SGD*.

Other works have been able to provide a taxonomy for the implementation of DP in ML as seen in Liu et al. and Gong et al. [101, 170]. Similarly, the shortcomings of the usage of DP are mainly focused on

Table 2.4: Overview of PPML solutions resorting to Differential Privacy.

Systems	Algorithm	Distribution	Applicability	Noise	Stage
Shokri and Shmatikov [266]	DP-SGD	*D*	*DL*	○	◑
Abadi et al. [3]	DP-SGD	*C*	*DL*	●	◑
McMahan et al. [189, 190]	DP-FedAVG	*F*	*DL*	●	●
PATE [222]	–	*D*	*DL*	○	◑
Mao et al. [183]	?[a]	*S*	*DL*	●	◐
NbAFL [301]	DP-SGD	*F*	*DL*	●	◑
Triastcyn and Faltings [287]	BayesianDP-FedSGD	*F*	*DL*	●	◐
DP-SCAFFOLD [206]	SCAFFOLD	*F*	*ML,DL*	●	◑
Adnan et al. [9]	DP-SGD	*F*	*DL*	●	◑
PixelDP [154]	PixelDP	*C*	*DL*	?[b]	◐
Sawhney et al. [260]	DP-SGD	*C*	*DL*	●	◑
Splitfed [282]	PixelDP	*S*	*DL*	◑	◑

Distribution
C - Centralized
D - Distributed
F - Federated
S - Split

Applicability
ML - Machine Learning
DL - Neural Networks

Noise
● - Gaussian
○ - Laplace
◑ - Other

Stage
● - Input
○ - Output
◑ - Training
◐ - Mix

[a] The authors do not specify the algorithm used.
[b] The authors do not specify the noise distribution used.

the trade-off between accuracy and privacy, which is still a challenge to be addressed.

Finally, there are several recent surveys worth mentioning. Zhang, Lu, and Liu [317] provide a systematic review of the current state-of-the-art solutions regarding the application of DP in FL, while Demelius, Kern, and Trügler [71] highlights the works on the convergence of DL with DP. In accordance with SMPC and HE, the fusion of FL and DP is growing exponentially with the evolution and development of FL and new thorough works are expected to be made available.

In table 2.4, we highlight some works that are relevant to the current state-of-the-art regarding the application of DP in ML. These works are categorized by the base algorithm they build upon, the system's distribution and applicability, the noise mechanism they follow, and the stage in which DP is added. The different noise mechanisms are categorized by the type of noise they add, *e.g.,* Gaussian, Laplace, or other (*e.g.,* Exponential, Poisson). The stage in which DP is added is categorized by the stage in the ML pipeline, *e.g.,* input, training, or inference/output. The *mix* category is used when the noise is added in more than one stage. The distribution and the applicability of the system are categorized by the learning distribution that the system follows, *e.g.,* centralized, federated, distributed or split, and to which type of learning algorithms it has been applied, *e.g.,* ML or DL.

In sum, most of these works are focused on a DL setting and the training of DL algorithms, which may hinder the broad usage of these solutions in other settings. Nonetheless, the application of DP in ML is still a growing field with several open challenges to be addressed. Although some of these works are categorized as being distributed, the main difference from the FL ones is based on the fact that they do

not rely on typical and defined FL protocols, such as *FedProx* or *FedAVG*.

2.3.2.3 Trusted Execution Environments

The increasing number of hardware-based solutions has potentiated the shift in the paradigm of using only software-based cryptographic techniques. Trusted hardware came to mitigate the still-existing problems with in-memory storage protection. These components can isolate sensitive code and data by providing an isolated environment, protecting this information from any malicious attacker who intends to modify or perform reverse engineering to the code and data sent to this trusted environment. Trusted Execution Environments (TEEs) provide an alternative to SMPC and HE, tapering the costs of performing such expensive computation and allowing the outsourcing of data and code to an untrusted system with trusted hardware.

Some hardware-based solutions have been proposed, with two being the most relevant: *Intel's Software Guard Extensions (SGX)* [188] and *ARM's TrustZone* [14]. These solutions are based on the same principles, although they differ in the implementation and the hardware they are deployed on. Also, *AMD's Secure Encrypted Virtualization* [261] is a hardware-based solution that allows the creation of encrypted virtual machines and is comparable with *Intel's Trust Domain Extensions* (TDX). Academic approaches have also been proposed, such as *Sanctum* [63] and *Sanctuary* [34]. These solutions have already been thoroughly compared in recent literature [134, 320]. In brief, SGX resorts to the intuition of *enclaves*, a region isolated from any other code in the system and is applied to personal computers or servers, similar to *SEV* [105, 134]. As for *Trustzone*, although built for multi-platforms, its development has been focused on mobile devices similar to *Sanctuary* [134].

Although these hardware-based solutions have shown their advantages and trade-offs between performance and security, the following works focus mostly on SGX due to its broad availability and documentation.

Generic Applications on TEEs

While offering an isolated environment where computation can be performed, solutions have been proposing a method where all the data processing is done inside these environments. Solutions such as *SCONE* [17], *RYOAN* [127], *Gramine* [289], *SGX-LKL* [237], and *Panoply* [265] have shown the feasibility of execution full applications (*e.g.,* DNN[129], BWA [303]) inside TEEs without incurring in exponential performance overheads.

These works can be decomposed into two main categories: *i)* sandbox and *ii)* minimal Trusted Computing Base (TCB)/host interface solutions.

Sandbox solutions such as SCONE [17] or RYOAN [127] allow the seamless introduction of a non-secure application to TEEs. *SCONE* presents itself as a portable tool to run applications inside trusted execution environments without the need to modify the application's source code. This tool allows the creation of a secure container, which is then deployed in a trusted execution environment where the

application can be executed. On the other side, *RYOAN* presents a sandbox utility that allows the attestation of a module to be computed in the trusted execution environment. This solution can also be applied to a distributed setting similar to a federated learning one. Nonetheless, it still relies on the assumption that all the computation is performed inside the enclave and that each entity has SGX-enabled hardware. In the same sense, *Enarx* [83] and *Veracruz* [292] emerged as alternative solutions that provide users with a sandboxing environment to run their applications.

Systems like *Gramine* [289], *SGX-LKL* [237], *Occlum* [264] or *Panoply* [265] offer a solution for running unmodified applications inside a trusted execution environment to reduce the host interface or the TCB. Both *Gramine*, *Occlum* and *SGX-LKL* follow a similar approach by deploying partially or fully the Library Operating System (LibOS) inside the enclaves, trying to minimize the host interface. While the latter aims to use a complete LibOS, including network stacks and file system, inside the enclave, the first only deploys a partial LibOS and supports multiprocess applications as it relies on the hosts' threading implementations. On its own, *Occlum* is a lightweight LibOS that provides a secure environment for running legacy Linux applications. Also, by resorting to Rust, this solution is more resilient to memory-safety bugs. On the other side, *Panoply* follows a different approach since it delegates all the C library calls to the host and exposes a larger host interface[265]. Although reducing the TCB, this approach may lead to the leakage of sensitive parameters [237]. However, both *Panoply* and *SGX-LKL* are no longer maintained as opposed to *Gramine*, which Intel and other universities are currently backing. Both *Veracruz* and *Occlum* are based on *Apache Teaclave*'s confidential computing platform [16].

Computation Partitioning on TEEs

Performing all the computations inside enclaves is a costly operation due to the increased TCB. This issue has been addressed by solutions resorting to computation partitioning, *i.e.,* partitioning the computation between trusted and untrusted code/environments. Nevertheless, this partitioning is not a trivial task and requires a thorough analysis of the application to be deployed. For instance, computation partitioning leads to the execution of code inside and outside of enclaves. With this, it is vital to analyze the data and the code to be executed to guarantee that sensitive data is not disclosed to untrusted entities or malicious adversaries.

Uranus [135] and *Montsalvat* [313] propose computation partitioning between the trusted execution environment and the untrusted environment. This approach allows non-sensitive computation to be performed in a non-secure environment while sensitive data is processed inside the trusted execution environment. *Glamdring* [168] proposes a static analysis tool that infers a partition between trusted and untrusted code in an application. It tries to achieve a balanced distribution of partitions to minimize the number of edges crossing between components. Another approach, Civet [290], focuses solely on partitioning Java applications. It provides an annotation-based approach for partitioning Java applications and ensures inter-object communication and consistent garbage collection across the partitioned components. Finally, *Uranus* [135] and *Montsalvat* [313] propose two automatic partitioning tools. Uranus [135]

addresses the challenges of automatic partitioning between trusted and untrusted code in Intel SGX enclaves. However, unlike *Glamdring*, Uranus tries to enforce the trusted and untrusted code partition at runtime. Conversely, Montsalvat [313] provides an automatic partitioning tool for *GraalVM* images that automatically annotates trusted and untrusted code to be computed inside trusted execution environments. More recently, *SecV* [312] extends the previous work proposed in Yuhala et al. [313] and creates a polyglot solution that, based on *GraalVM*, allows the partitioning of trusted and untrusted code for different programming languages.

Current solutions

Although the usage of TEEs has increased to deliver PPML solutions, it is commonly deployed in three learning settings: *i)* centralized; *ii)* collaborative; and, *iii)* distributed learning. The first approach focuses on performing the computation securely in each entity's premises or outsourcing it to cloud environments that perform the computation in a centralized manner. The second one follows the same principles of SMPC where each entity performs its computation and exchanges the results with the other entities. The third approach aims to decrease the execution time of the computation by distributing it among different nodes.

With the increasing usage of TEEs, with a focus on SGX, the applicability of these solutions ranges from simple SQL queries [321] to more complex machine learning algorithms [155].

Outsourcing computation to a third party typically follows a centralized approach, as seen in solutions like *SGX-BigMatrix*, *Slalom*, *MaskAI*, *Privado*, *SkSES* and *TensorScone* [110, 144, 150, 152, 263, 285]. *Slalom* and *Privado* [110, 285] focus on protecting the inference of neural networks by outsourcing it to TEEs. *Slalom* presents one of the first solutions of partitioning DNNs for inference. This solution is based on the partitioning of the DNN into two parts, one that is executed inside the enclave and another that is executed outside the enclave. As for *Privado*, the authors focused their efforts on showing that DNN inference inside SGX is feasible but is vulnerable to access pattern attacks. With this, the authors present a solution that allows the inference of DNNs inside SGX while mitigating the access pattern leakage. This mitigation is achieved by resorting to OT protocols, which create an input-oblivious solution.

MaskAI [152] and *SkSES* [144] target the genomic analysis pipeline for two different tasks, *i.e.,* read mappings and GWAS, respectively. They use distinct approaches as the first masks data with filters before refining alignment score inside the TEEs while the second re-implements GWAS algorithms and X^2 tests to be executed inside the secure environments. Finally, *TensorScone* relies on the *Scone* technology to deploy a secure environment for the execution of DL model and is implemented on top of TensorFlow.

In the collaborative learning setting, Ohrimenko et al. [210] present their data-oblivious version of five of the most used machine learning algorithms, creating an alternate version of K-Means, SVM, NNs, DTs, and matrix factorization. These modified algorithms provide strong privacy guarantees while preventing the exploitation of side channels induced by memory, disk, and network accesses. In brief, this framework

allows multiple clients to collaboratively train a model with the guarantee that their dataset is privacy-preserved.

Following the same assumptions, *Smac* [74], *Occlumency* [155] and both *DyPS* and *I-GWAS* [223, 224] present solutions that focus on centralized and collaborative environments. The two latter approaches, *DyPS* and *I-GWAS*, focus on genomic workloads and on the collaboration for genomic querying and analysis. They also focus on the usage of SGX to protect the data and the code from malicious adversaries by performing this analysis inside the enclaves [223, 224]. Furthermore, *Occlumency* [155] resorts to SGX to protect the data and the code for DL inference stage solely. By outsourcing this computation to cloud computing environments, this solution promotes collaboration between entities that may want to leverage data or models from other entities.

Although initially proposed for centralized settings, some solutions can also leverage distributed computations by deploying multiple enclaves or building on top of distributed computation engines [126, 321]. *Opaque* [321] is one of the first solutions that resort to TEE and builds on top of Apache Spark. By redefining some Apache Spark UDF operators and combining TEEs with OT protocols, this solution allows the execution of some SQL queries while trying to mitigate possible access pattern leakage from SGX.

On another note, *Chiron* [126] allows the training of a machine learning model on a cloud service without revealing their training data, and once trained, only the data owners can query the model. It supports launching multiple enclaves while each operates on different shards of training data, keeping the enclaves synchronized via a parameter server to ensure they all collaboratively converge to the same model. As such, besides providing a privacy-preserving approach, this framework also provides data parallelism based on a parameter server to train the model faster. *Myelin* [129] presents an approach similar to *Chiron* while adding differential privacy and data oblivious protocols.

Moreover, some application-specific solutions for the distributed setting have been proposed to reduce and parallelize execution time [109, 303]. Both works focus on the distributed setting, where the computation is distributed among different nodes. *HySec-Flow* [303] is a framework that allows the execution of read mapping operations done on top of genomic data. In contrast with *SGX-Spark* [109], which tries to create a secure version of Apache Spark [314], *HySec-Flow* is based on a non-scalable framework, namely BWA[4] [156].

Table 2.5 presents an overview of the current state-of-the-art solutions regarding the application of TEEs in PPML. These solutions are categorized by the cryptographic primitives they resort to, the type of learning algorithm they support, the distribution of the system, and the protocol they follow. Although the main focus is on the usage of TEEs, some of these solutions resort to other cryptographic primitives to increase the security guarantees offered. The distribution of the system is categorized by the distribution of the computation, *e.g.,* collaborative or distributed or centralized.

These works are limited by their applicability to specific machine learning algorithms, their proposed distribution, and the cryptographic primitives they rely on. Nonetheless, the usage of TEEs is still a growing

[4]BWA was built for local or remote deployment in a single server, does not natively support parallel computation

field with several open challenges to be addressed.

Table 2.5: Overview of PPML solutions resorting to Trusted Execution Environments.

Systems	Crypt. Primitives	Distribution	Algorithms	Availability
SGX-BigMatrix [263]	●, ◐	*C*	*ML*	○
Slalom [285]	●	*C*	*DL*	●
MaskAI [152]	●	*C*	*A*	○
Privado [110]	●	*C*	*DL*	○
SkSES [144]	●	*C*	*A*	●
TensorScone [150]	●	*C*	*DL*	○
Ohrimenko et al. [210]	●,◐,○	*C, H*	*ML*	○
SMac [74]	●	*C, H*	*A*	●
Occlumency [155]	●	*C, H*	*DL*	○
DyPS, I-GWAS [223, 224]	●	*C, H*	*A*	○
Chiron [126]	●	*C, D*	*DL*	○
Myelin [129]	●,◑	*C, D*	*DL*	○
Opaque [321]	●,◐	*C, D*	*A*	●
HySec-Flow [303]	●	*D*	*A*	◐
SGX-Spark [109]	●	*D*	*ML, A*	●
Uranus [135]	●	*D*	*A (mostly)*	●

Cryptographic Primitives
● - TEE
◐ - OP
◑ - DP
○ - SMPC

Distribution
C - Centralized
D - Distributed
H - Collaborative

Algorithms
ML - Machine Learning
DL - Neural Networks
A - Analytics (SQL queries, etc)

Availability
● - Yes
○ - No
◐ - Partially

2.4 Lessons Learned

We now discuss the key insights provided by this chapter, the open research challenges that remain to be addressed, and the research questions we aim to answer in this thesis.

2.4.1 Discussion

The number of solutions proposed to address the challenge of PPML has been increasing since 2015, which poses a need for a general solution that has yet to be proposed and implemented.

The training and inference stages of machine learning are two very different steps with unique computational and privacy needs. Although some works depicted above propose a solution to one of the stages, a small subset of approaches such as *ABY3* [197] highlight the viability to perform both stages, creating a hybrid solution. However, the small subset of hybrid approaches shows that there are still challenges to be addressed.

The combination of cryptographic primitives allows further exploring the threats that the ML pipeline encompasses. However, adding security guarantees brings overhead to the system's runtime performance. Employing HE and consecutive data encryption and decryption increases the computational time compared with non-cryptographic solutions. The polynomial approximations that are required to use this primitive can be costly to the accuracy of the model, which can be seen as akin to the approximations of deep neural networks to XOR or AND gates when using GC [278]. Nevertheless, in solutions based on HE techniques, such as the works developed in [48, 59, 97, 119], the approximation of non-linear functions to polynomial functions still prevails, leaving its mark on the accuracy, in the execution time and directly on the algorithms that need to be re-implemented. On the same note, systems relying on garbled circuits, such as the ones in [198, 246, 251], also tend to present higher communication costs. New solutions try to combine these different cryptographic primitives to diminish the computation time or the communication complexity. However, this combination is still far from optimal compared to non-privacy-preserving environments.

In light of this, Federated Learning has tried to fulfill all the requirements of a collaborative learning environment, where users collectively work together to train a model without explicitly sharing raw data. However, this architecture lacked the privacy guarantees needed for large-scale performance. Nevertheless, McMahan et al. [190] proposed a way to mitigate the lack of security guarantees of federated learning, introducing noise to the federated averaging algorithm. Moreover, Bonawitz et al. [29] also proposed a protocol for secure aggregation to be applied directly to federated learning. This protocol was made to provide privacy guarantees for this setting and a failure-robust protocol, allowing the clients to drop out of the computation without influencing the overall system. This protocol relies on several primitives such as secret sharing, key agreement, authenticated encryption, and secure signatures. The main issue with this approach relies on its explicit target to edge devices and the small volume of data in each node. These two works tried to deliver a privacy-preserving approach to Federated Learning, relying on different cryptographic primitives.

Despite the broad set of approaches here depicted, and the ones depicted on the several surveys proposed in the last few years, there is still the need to satisfy two conditions simultaneously: *i)* the data owner needs to keep its sensitive data private and *ii)* the model owner wants to protect its intellectual property. Considering every possible attack on the system, the system is secure if these two needs are satisfied.

In this regard, TEEs-based solutions have revealed promising results for balancing performance and accuracy while protecting both the model and the data. However, these solutions follow an approach in which full applications are deployed inside these environments, leading to a high TCB [126, 285]. Based on this, computation partitioning is a promising approach to address this challenge. Nonetheless, as seen by previous solutions, the partitioning of sensitive and non-sensitive computation is dependent on the computation itself (*i.e.,* the algorithms) [135, 313]. For instance, as shown in the following chapters, the partitioning of computation for ML algorithms is dependent on the algorithm itself, and generic solutions may not be directly applied to this context. Another issue with current solutions is that they cannot

scale to large datasets, as the amount of data that can be stored and the computational power of these environments is limited. Nonetheless, scaling these solutions to large datasets is a work in progress, as seen by the recent works in [109, 303].

We note that TEEs do not cover all security attacks, namely solutions based on this hardware are still vulnerable to side-channel attacks, such as page fault attacks, timing attacks, and cache attacks [211]. For this, several works propose solutions that tackle these attacks, such as *LibSodium* [167].

2.4.2 Challenges

Based on this chapter and the previous summary, we highlight some main challenges for PPDML research:

C1: Performance. Many privacy-preserving techniques introduce additional computational overhead, increasing the runtime performance. This can impact the scalability and efficiency of the ML process. For instance, the most secure solutions are based on secure multi-party computation but are also the most inefficient [231]. Similarly, TEEs present high-end security guarantees and high runtime performance overhead as the amount of TCB increases [105].

C2: Applicability. The need to change algorithms to accommodate different privacy-preserving techniques limits these solutions' applicability and versatility. This is a challenge when applying these solutions to different domains. Solutions resorting to HE are typically based on polynomial approximations and often need to reimplement the algorithms, limiting its applicability [6, 65]. Moreover, changing how an algorithm is fundamentally implemented can be error-prone and lead to a loss of accuracy.

C3: Utility. Privacy-preserving methods, such as DP, involve a trade-off between data privacy and model utility. Designing solutions that maintain the accuracy of the original model is a tricky challenge and a special topic when dealing with solutions based on DP. Defining the proper privacy budget and the right amount of noise to be added to the model and data while maintaining the model's accuracy is still a work in progress, as seen by the previous solutions [15].

C4: Trusted Parties. Some solutions require a level of trust in third-party entities, such as the central server in federated learning. This lack of trust can raise concerns about the misuse of data or breaches of trust.

C5: Scalability. As the dataset grows, privacy-preserving techniques' complexities can become more pronounced. In detail, the distribution of computation or the collaboration between entities are two different settings with distinct security requirements. In the first approach, the goal is to diminish the computation time by resorting to third-party infrastructures, while the second intends to leverage data from different entities simultaneously. Ensuring both methods can scale effectively to handle large datasets while privacy-preserving data is an ongoing challenge.

C6: Hardware Requirements. Some privacy-preserving techniques demand specific hardware (*e.g.,* TEEs), which hinders the usage of these solutions for organizations with limited hardware or budget. IOT

and mobile devices usually have limited computational resources or do not possess TEE-enabled hardware for sustaining hardware-specific solutions.

With this thesis, we will explore different solutions that cover one or more of these challenges. Ultimately, we aim to answer the following two research questions:

- How to design PPDML solutions that are efficient, secure, and practical?
- How can these solutions be adapted to different scenarios regarding hardware and workloads?

3

Preserving Privacy in Distributed Machine Learning

The ubiquitous environment provided by cloud computing providers offers a scalable, reliable, and performant environment to deploy compute-intensive ML workloads. However, many of these workloads operate over users' sensitive information (*e.g.*, medical records, financial information). In addition, regulations like HIPAA and GDPR enforce strong security policies when processing or storing sensitive data at untrusted third-party infrastructures [118, 295]. As such, outsourcing data storage and computation to third-party services leaves users vulnerable to attacks that may compromise the integrity and confidentiality of their data [131, 279].

As explained in Sections 2.1.3 and 2.3.1, the ML pipeline encompasses several stages, both for model training and inference, in which users' data is known to be susceptible to different attacks such as *adversarial attacks*, *model extraction*, and *inversion*, and *reconstruction attacks* [89, 286, 296]. This leads to the increasing requirements of proposing PPDML solutions that can protect users' data and models from these attacks. However, to provide a PPDML solution that is efficient, secure, and practical, one shall consider the following challenges discussed in Section 2.4.2:

- **C1: Performance.** Privacy-preserving techniques introduce additional computational overhead, increasing the runtime performance. This additional overhead can impact the scalability and efficiency of the ML process. It is of utmost importance to balance this trade-off.

- **C2: Applicability.** The applicability of PPDML solutions is a crucial aspect to consider. PPDML solutions should support a wide range of ML algorithms and workloads.

- **C4: Trusted Parties.** The demand to trust a third party to perform computations raises concerns about data misuse or breaches of trust. This can be mitigated by removing the need to trust in a third party to perform computations.

Recent works have addressed these attacks with solutions based on homomorphic encryption or secure multi-party computation schemes [32, 97]. However, these cryptographic schemes impose a

significant performance toll that restricts their applicability to practical scenarios [231]. Nonetheless, since early 2016, several works have been building on top of each other to mitigate this performance penalty while offering higher security guarantees [32, 97, 147, 197, 198, 227]. These solutions have been able to support a wide range of ML algorithms, in most cases with the added need to re-implement them. However, they still impose a significant performance penalty, which can be prohibitive for some applications.

To circumvent this performance penalty, another line of research is that of exploring hardware technologies enabling Trusted Execution Environments TEEs, such as Intel SGX [188]. These technologies allow the execution of code within isolated processing environments (*i.e.,* enclaves) where data can be securely handled in its original form (*i.e.*, plaintext) at untrusted servers.

Initial approaches resorting to SGXs focused on deploying the full ML workload inside the enclaves. However, as the amount of computational and I/O operations performed at the enclaves increases, the performance of ML training and inference is noticeably affected by hardware limitations, limiting the design's applicability in practice. This is also true for the increasing TCB size, which incurs higher performance penalties regarding execution time and the attack surface (*i.e.,* increasing TCB leads to an increase attack surface as discussed in Section 2.3.2.3) [73].

To tackle the aforementioned issue, solutions have been proposed based on the insight that ML runtime performance could be improved by reducing the number of operations done at enclaves. In fact, this insight is backed up by previous work [36, 135, 321] exploring the partitioning of computation across trusted and untrusted environments, but in contexts with different security requirements and processing logic (*e.g.*, SQL processing, MapReduce, distributed coordination) than the ones found for ML workloads. As such, since ML workloads present different processing logic and security requirements, the partitioning of computation across trusted and untrusted environments must be rethought.

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

Our reasoning is twofold: *i.)* the majority of current attacks on the ML pipeline are only successful if the attacker has some knowledge about the datasets and/or models being used [35, 89]; and *ii.)* previous work shows that such knowledge cannot be inferred from the information leaked by statistical operations, such as the calculation of confidence results, table summaries, ROC/AUC curves, and probability distributions for classes, without additional knowledge regarding the dataset or the model [50]. As a result, these operations are ideal candidates to be offloaded from enclaves. We support these claims by analyzing the security and performance implications of different ML workloads and attacks.

Contributions. Based on the previous challenges and current issues, we propose Soteria, a PPDML system. Soteria aims to answer the following questions: *i) How can we design a PPDML solution that is efficient, secure, and practical*; *ii) How can we reduce computation performed inside the TEEs to improve*

the efficiency of such a PPDML solution?

Soteria is an open-source system for distributed privacy-preserving ML that leverages the scalability and reliability of Apache Spark and its ML library (MLlib) [38]. Unlike previous solutions [109, 263], Soteria supports a wide variety of ML algorithms without changing how users build and run these within Spark. Further, it ensures that critical operations, which enable existing attacks to reveal sensitive information from ML datasets and models, are exclusively performed in secure enclaves. This means that the sensitive data being processed only exists in plaintext inside the enclave and is encrypted in the remainder of the dataflow (*e.g.*, network, storage). This solution enables robust security guarantees, ensuring data privacy during ML training and inference.

Soteria introduces a new computation partitioning scheme for Apache Spark's MLlib, Soteria-P, that offloads non-critical statistical operations from the trusted enclaves to untrusted environments. Soteria-P is accompanied by a formal security proof for how data remains private during ML workloads and an analysis of how this guarantee ensures resilience against various ML attacks. Furthermore, Soteria offers a baseline scheme, Soteria-B, where all ML operations are done inside trusted enclaves without a fine-grained differentiation between critical and non-critical operations. Soteria-B provides a performance and security baseline for comparison against our new partitioned scheme.

Evaluation. We compare experimentally both approaches with a non-secure deployment of Apache Spark and a state-of-the-art solution, namely SGX-Spark [109]. Our experiments, resorting to the HiBench benchmark [130] and including seven different ML algorithms, show that Soteria-P, while considering a more significant subset of ML attacks, reduces training time by up to 41% for Gradient Boosted Trees workloads and up to 4.3 hours for Linear Regression workloads when compared to SGX-Spark. Also, compared to Soteria-B, Soteria-P reduces execution time by up to 37% for the Gradient Boosted Trees workloads and up to 3.3 hours for the Linear Regression workloads.

3.1 Key Technologies

This section highlights the key technologies and concepts that work as building blocks for our solution.

3.1.1 Apache Spark

Apache Spark is a distributed cluster computing framework that supports Extract, Transform, and Load (ETL), analytical, ML, and graph processing workloads over large volumes of data. Spark can be deployed on a cluster of servers, at a private infrastructure, or in the cloud and supports different data sources (*e.g.*, HBase, HDFS) to read the data to be processed and store the corresponding output and logs. The framework performs most of the computation in-memory, thus promoting better performance for data-intensive applications when compared to Hadoop's MapReduce [315].

Spark follows a master-worker distributed architecture similar to Hadoop and its MapReduce topology. Users interact with the master node by submitting jobs (*i.e.*, processing workloads) and collecting the

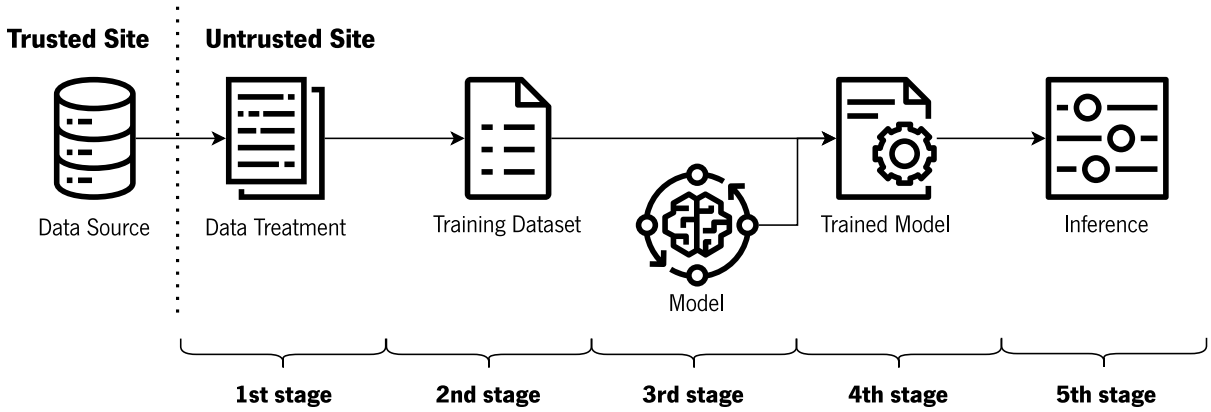


Figure 3.1: Typical ML workflow for data engineers and scientists. This workflow has five different stages: data treatment, dataset creation, algorithm or model choice, algorithm or model training, and model persistence or inference.

corresponding outputs. The master is responsible for submitting jobs to the worker nodes by resorting to a job scheduler and resource negotiator named Spark Driver. This driver splits the jobs into tasks (*i.e.*, map phase) and defines which workers will compute a given set of tasks. Finally, each worker performs the designed task(s) and reports the resulting output to the master, which then aggregates (*i.e.*, reduce phase) and forwards the response back to the user. Further details about Spark's flow are discussed in Section 3.3.1 as SYSA follows its flow.

The ML library of Apache Spark (*MLlib*) [192] enables Spark users to build end-to-end ML jobs. Its workflow is similar to the one found in other ML solutions, with the addition of an initial data treatment stage, which is enabled by Spark's core Application Programming Interface (API). The library also provides a set of tools and utilities for feature extraction and model persistence.

Figure 3.1 shows the typical ML workflow for data engineers and scientists. The first stage is typically known as the process of ETL, where data is collected and extracted in a pre-determined format, treated accordingly, and loaded in the needed format (*e.g.*, CSV, Parquet, Text File). In the second stage, data is split into train and test datasets, and a given ML algorithm is chosen. The third stage is the training stage, where data is iterated to deliver an optimized trained model at the fourth stage. In the fifth stage, the trained model can be saved (persisted) and loaded (accessed) for inference.

3.1.2 Intel Software Guard Extensions

As a trusted execution environment, SGX is a set of new instructions available in an Intel processor that applications can use to create protected memory regions within their space. These regions (enclaves) are isolated from any other code in the system, allowing only the code inside the enclave to access data in the enclave [188, 210]. This also includes the isolation from higher privilege users, host Operating System (OS), hypervisor, and BIOS. The integrity and the results, as well as the identity of the parties, can be attested with the remote attestation provided by SGX, ensuring the computation was correctly executed

[285].

Since SGX protects code and data from privileged access, sensitive plaintext data can be processed at the enclave without compromising its privacy. The computation with unencrypted data and code allows diminishing the execution time compared to other cryptographic techniques (*e.g.*, searchable encryption, homomorphic encryption) [58, 210].

Nonetheless, even though the second generation of SGX has improved the size of the protected memory region, it still defines the Enclave Page Cache (EPC) to 128MB per CPU [120]. Memory swapping occurs when such a limitation is met, which is a performance-costing mechanism [73]. Thus, SGX-based solutions must balance the number of I/O operations, the amount of data handled by enclaves, and the Trusted Computing Base (TCB) to optimize performance.

For this work, we chose Intel SGX over other TEE's (*e.g.*, ARM TrustZone [14]) due to its general use in academia [135, 153, 321] and industry [20], availability, as well as its security guarantees and computing reliability. However, our solution is generic and can be applied to other TEE technology that follows similar design principles to SGX.

3.1.3 Gramine

Porting existing source code into TEEs is a non-trivial and error-prone task. Initially, the application's source code needed to be rewritten in C/C++ to run inside enclaves, thus requiring manual intervention. This inefficient approach led to the proposal of solutions to port unmodified applications to run inside TEEs automatically.

In this work, we resort to Gramine [289] (previously named *Graphene-SGX*), an open-source library OS designed to run unmodified applications inside Intel SGX enclaves. Gramine's architecture (Figure 3.2) is based on Drawbridge's *picoprocess* [236], which provides an isolated address space. Its architecture contains three main components: *i)* the unmodified binary application and corresponding libraries, to be ported into SGX; *ii)* the LibOS, a custom OS implemented by Gramine; and *iii)* the Enclave Platform Adaptation Layer (PAL).

PAL. This is a core component of Gramine, which acts as an intermediary between the LibOS and the underlying hardware platform to provide the necessary abstractions and interfaces that enable Gramine to run efficiently [108]. In detail, this layer:

- Abstracts the low-level hardware details, allowing Gramine to be portable across different hardware platforms without requiring extensive modifications to its core codebase.
- Handles the initialization and management of the SGX enclaves and sets up the enclave memory layout.
- Facilitates communication between the Gramine enclave and the host system.

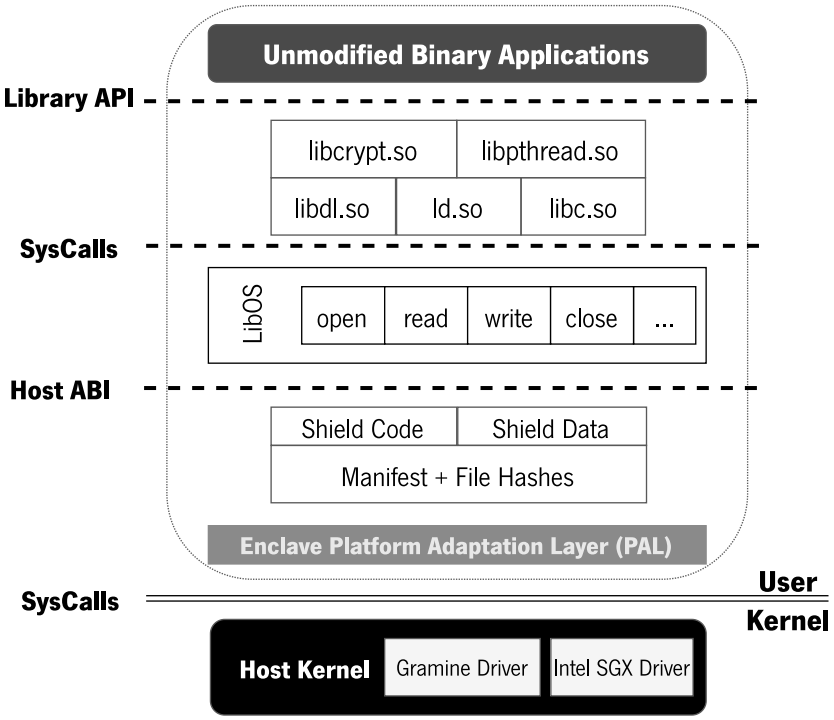


Figure 3.2: Gramine's architecture based on [289]. The enclave includes an OS shield, a library OS, `libc`, and other user binaries.

- Provides mechanisms for securely passing data and invoking system calls from the enclave to the host and vice versa.

- Manages the resources allocated to the enclave, ensuring it has access to the necessary memory, CPU, and other resources.

Manifest File. Through this configuration file, developers specify the application libraries to be deployed inside a secure enclave, the paths for files that the enclave will need to access securely, and trusted system libraries. It also contains the environment's specifications (*e.g.*, the CPU and memory resources for each enclave) [107].

In sum, the Gramine library works similarly to a paravirtualization environment, taking advantage of the standard virtualization benefits such as security isolation, host platform compatibility, and migration while mapping high-level APIs onto a few paravirtual interfaces to the host kernel [289].

By resorting to this framework, one can take full advantage of the application's native performance while potentiating the use of trusted execution environments. Moreover, with a growing community focused on confidential computing with Intel SGX, Gramine is supported by Intel and several universities [106].

3.2 Threat Model and ML Attacks

To fully understand the security requirements of Soteria, we first need to understand the threat model of our solution and the attacks that such an ML system may face. We follow this with a comparison of Soteria with state-of-the-art solutions, which is summarized in Table 3.1).

3.2.1 Soteria Threat Model

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

Our deployment model is depicted in Figure 3.3 and is as follows. The client (data owner) will be trusted and provide input for ML tasks. Then, a Spark master node and N worker nodes will be deployed in an untrusted environment (cloud provider) equipped with Intel SGX technology. Externally, we also consider a distributed data storage backend. The protocol assumes an implicit setup where the client securely stores its input data within this backend, which is also considered untrusted throughout the protocol execution.

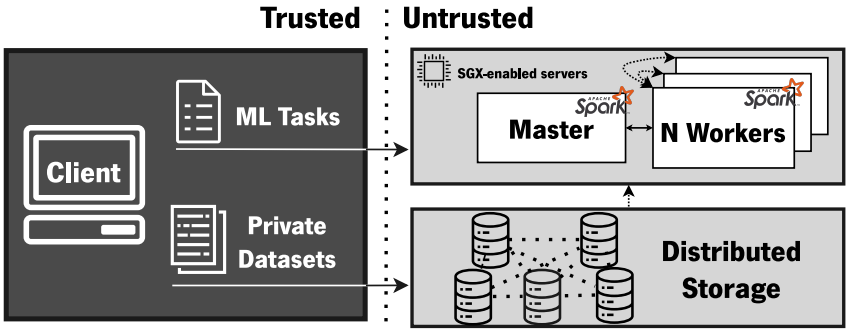


Figure 3.3: Soteria deployment model.

We consider semi-honest adversaries, meaning that security is defined according to a threat that attempts to break the confidentiality of data and model but will not actively deviate from the protocol specification. This is a good fit for cloud-based systems, where data breaches are common and malicious entities can read internal processing data temporarily [131]. In brief, our security goal is to allow clients to provide input data for training and inference in a way that is not vulnerable to confidentiality breaches.

3.2.2 ML Workflow Attacks

Throughout this work, we consider the black-box setting proposed by [56], which is as follows. When an adversary is given *black-box* access to a model, it means that it can query any input x and receive the predicted class probabilities $P(y|x)$ for all classes y. This allows the adversary to interact with the trained model without retrieving additional information, *e.g.*, computing the gradients. Ensuring security against

attacks under this threat model entails including countermeasures against a wide array of attack vectors, further detailed in §2.3.1. Figure 3.4 extends the previous Figure 3.1 by correlating the attacks to the stages at which they occur in the ML pipeline. Given this, we further detail pervasive attacks on the ML pipeline, correlate them with the several stages, and establish their adversarial assumptions.

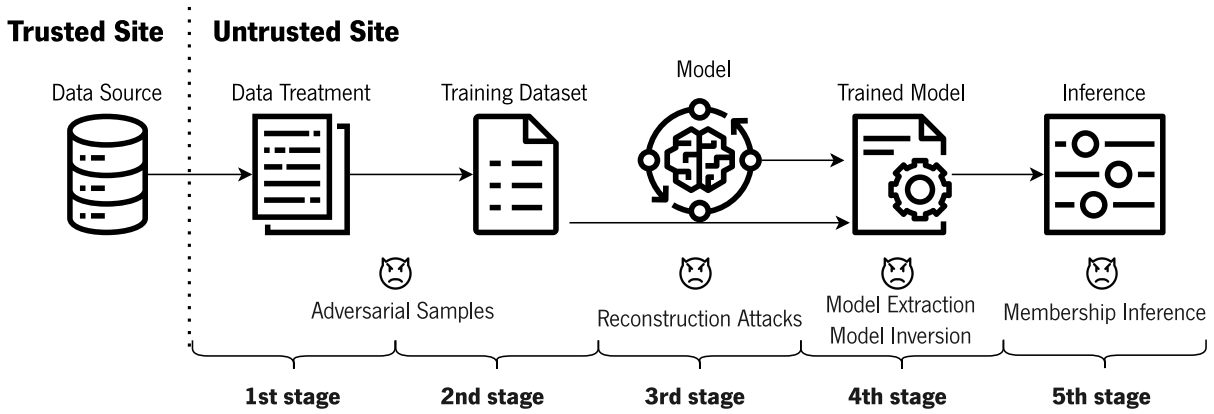


Figure 3.4: ML pipeline attacks and their correlation with the stages of the ML pipeline. Typically, adversarial attacks occur between the first and second stages, reconstruction attacks occur in the third stage, model extraction, and model inversion in the fourth stage, and membership inference attacks occur in the fifth stage.

Adversarial Attacks. These attacks occur between the first and second stages, where data is being treated and prepared for training. They are characterized by injecting malicious data samples to manipulate the model and disclose information about the original data being used for training or inference purposes. Successful attacks in the literature require the attacker to have direct access to the training dataset (data poisoning, transfer-based, and gradient-based attacks), the model and gradients (gradient-based attacks), or the full results (*i.e.,* the output of inference) and class probabilities (score-based attacks) [35, 151].

Reconstruction attacks. The goal of this attack is to reconstruct raw data used for training the model. To be successful, some attacks require the adversary to have model-specific information, namely feature vectors (*e.g.,* Support Vector Machines or K-Nearest Neighbor) [253], others only require black-box access to the model [258]. Nonetheless, in this setting, the attacker needs to have access to another dataset with the same distribution as the original training dataset (*i.e.,* local dataset and training dataset are subsets from a larger dataset). These attacks occur in the third stage while models are training.

Model Extraction. These attacks aim at learning a close approximation to an objective function of the trained model and are deployed in the fourth stage alongside model inversion. This approximation is based on the exact confidence values and response labels obtained by inference. To obtain the desired result, the attacker must know the dimension of the original training dataset (equation-solving attacks), the dimension of the decision trees, the data features, and the final confidence values (path-finding attacks) or hold real samples from the training dataset (class-only attacks and data-free knowledge distillation (DFKD)) [50, 286, 288].

Model Inversion and Membership Inference. These attacks target the recovery of values from the training dataset. Both consider an adversary that queries the ML system in a black-box fashion, and both are currently based on ML services, which publicly disclose their trained models and confidence values. In model inversion attacks, the adversary must have partial knowledge of the training dataset's features to infer and query the model with specific queries [89, 286]. Membership inference aims to test whether a specific data point d was used as training data and requires the adversary to know a subset of samples used to train the model (that does not contain d) [267]. Model inversion attacks occur alongside model extraction attacks, while membership inference attacks occur in the last stage of the pipeline.

Table 3.1: Comparison between state-of-the-art solutions and Soteria regarding the safety against ML attacks.

Attacks		**Systems**				
		[126]	[129]	[263]	[109]*	Soteria
Adversarial	*Gradient-based*	✗	✗	✓	✗	✓
	Score-based	✗	✗	✓	✗	✓
	Transfer-based	✗	✗	✓	✗	✓
	Decision-based	✗	✗	✓	✗	✓
Model Extraction	*Equation-solving*	✓	✓	✗	✓	✓
	Path-finding	✓	✓	✗	✗	✓
	Class-only	✓	✓	✗	✗	✓
	DFKD	✓	✓	✗	✓	✓
Model Inversion		✓	✓	?	✓	✓
Reconstruction Attacks		✓	✓	✓	✓	✓
Membership Inference		✗	✗	✗	✗	✓

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

Summary. Unlike previous works [109, 126, 129, 263], which typically consider a small subset of ML attacks, our proposal aims at providing mechanisms that cover the full range of the exploits mentioned above. Table 3.1 presents relevant state-of-the-art solutions, the security attacks covered by these, and the attacks addressed by Soteria. Intuitively, the resilience of our system is the result of combining several mechanisms, which are only partially ensured by other systems: *i)* authenticity verification of inputs excludes injections necessary for *adversarial attacks*; *ii)* isolation guarantees of our protocol ensure that malicious workers gather no additional information other than statistical data, an essential aspect for preventing most attacks, and *iii)* query input via a secure channel prevents the adversary from performing arbitrary queries to our system, which is also a central requirement for *model inversion* or *reconstruction attacks*. This is analyzed in detail in Section 3.3.5.

TEE-related security issues such as *side-channel* and *memory access pattern* attacks are considered orthogonal and complementary to our design goals. Indeed, mechanisms such as ObliviousRAM [273]

can be layered over Soteria to address these at the cost of additional performance overhead [211].

3.3 Soteria

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

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

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

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

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

3.3.1 Architecture and Flow

As depicted in Figure 3.5, Apache Spark's operational flow is as follows. Before submitting ML tasks (*e.g.*, model training, and/or inference operations) to the Spark cluster, users must load their local datasets and models to a distributed storage backend. Users can then submit ML processing tasks, specified as ML task scripts, to the Spark client, which is responsible for forwarding these scripts to the master node. At the master node, tasks are forwarded to the Spark Driver, which generates a Spark Context that then distributes the tasks to a set of worker nodes.

As workers may be executing different steps of a given task, they need to be able to transfer information among each other through the network. For instance, in a distributed ML training task, this information can contain model parameters that must be exchanged across workers. After finishing the desired computational steps, workers return their outputs to the master node, which merges the outputs and replies to the client.

Similar to the regular flow of Apache Spark, Soteria can be divided into two main environments or sides: the Soteria Client, trusted side, and the Soteria Cluster, untrusted side (*e.g.,* cloud environment). Next, we describe the main modifications required by Soteria to the original Apache Spark's design, depicted in Figure 3.5 by the white boxes.

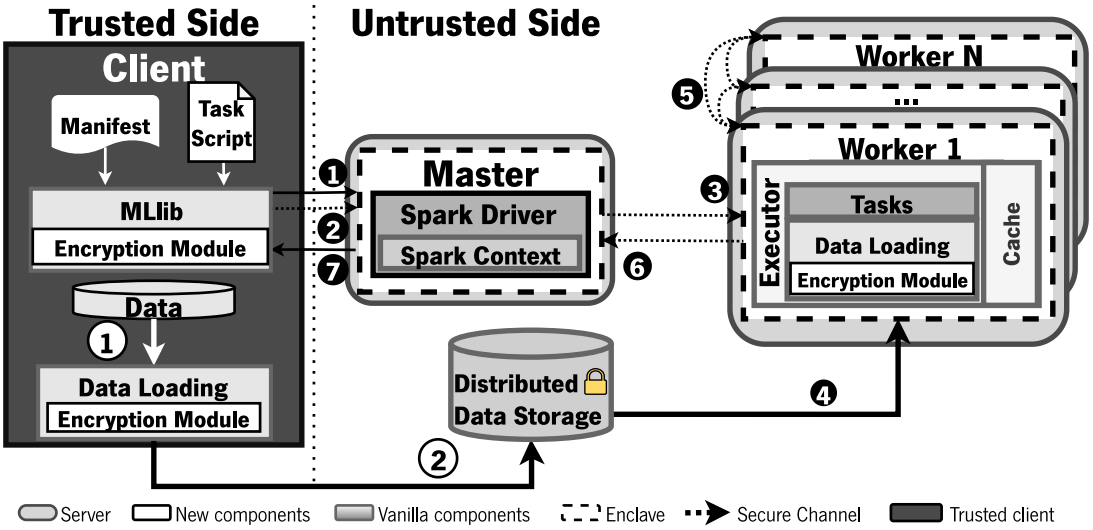


Figure 3.5: Soteria architecture and flow of operations, the latter represented by the numbers. The client encrypts its data and sends it to the distributed data storage backend (①-②). The client also sends the ML task script and the *Manifest* file to the master node (❶-❷). The master node then distributes the ML task script to the worker nodes (❸). The workers fetch the data and process the ML task script (❹-❺) and send the results back to the master node (❻), which merges the results and sends them back to the client (❼).

3.3.2 Client

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

Data Loading. First, the user must specify the data to be loaded to the storage backend for the first operation. However, such data has to be encrypted before leaving the trusted user premises. This step is done by extending Spark's data-loading component with a transparent encryption module (Figure 3.5-①). This module encrypts the data being loaded into the distributed storage backend with a symmetric-key encryption scheme (Figure 3.5-②).

Tasks submission. ML training and inference operations include two main files: the ML task script and the *Manifest* file. The transparent encryption module also integrated within MLlib, is used to encrypt the ML task script (Figure 3.5-❶), which contains sensitive arguments (*i.e.,* model parameters) and the ML's workload processing logic, and to decrypt the outputs (*e.g.*, trained model or inference result) returned by Spark's master node to the client.

The *Manifest* file contains the libraries to be used by the ML task script, as well as the path at the storage backend where the training or inference data for that specific task is kept (Figure 3.5-❷). Briefly, and as explained in the following sections, this file ensures that different Spark components can attest to

the integrity of libraries and data being used/read by them and cannot access other libraries or data that these are not supposed to.

The encryption module is in charge of securely exchanging the *Manifest file* and the user's symmetric encryption key with the SGX enclave on the master node (Figure 3.5-❶❷). This is done once, at the ML task's bootstrapping phase, and requires establishing a secure channel between the client and master's enclave. This channel guarantees the security and integrity of the user's encryption key and the *Manifest file*. At the same time, the encrypted ML task scripts can be safely sent via an unprotected channel.

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

3.3.3 Cluster

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

For inference operations, the master node also needs to access the distributed storage backend to retrieve the stored ML model. The user's encryption key is necessary, so the encrypted model is only decrypted and processed at the secure enclave. The *Manifest* file ensures that only the storage locations specified in the file are accessible to the master node (Figure 3.5-❷).

After processing the ML task scripts, the master's enclave establishes secure channels with the enclaves of a set of workers to send the necessary computational instructions[1] along with the user's encryption key and *Manifest file* (Figure 3.5-❸). The user's encryption key is needed at the worker nodes so that these can read encrypted data (*e.g.,* train dataset or data to be inferred) from the storage backend while decrypting and processing it in a secure enclave environment (Figure 3.5-❹). Once again, the *Manifest file* prevents unwanted access to stored data. Furthermore, the enclaves at the worker nodes establish secure channels between themselves to transfer sensitive metadata information such as model training parameters (Figure 3.5-❺).

Finally, after completing the desired computational tasks, the workers send the corresponding inference or training outputs to the master node through the established secure channel (Figure 3.5-❻). The master node then merges the partial outputs into the final result, which is done inside a trusted enclave, and sends it encrypted, with the user's encryption key, to the trusted client module (Figure 3.5-❼). At

[1]The same metadata sent by a vanilla Spark deployment so that workers know the computational operations to perform.

the latter, the result (*i.e.,* trained model or inference output) is decrypted by the transparent encryption module and returned to the user in plaintext.

3.3.4 Partitioned Design

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

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

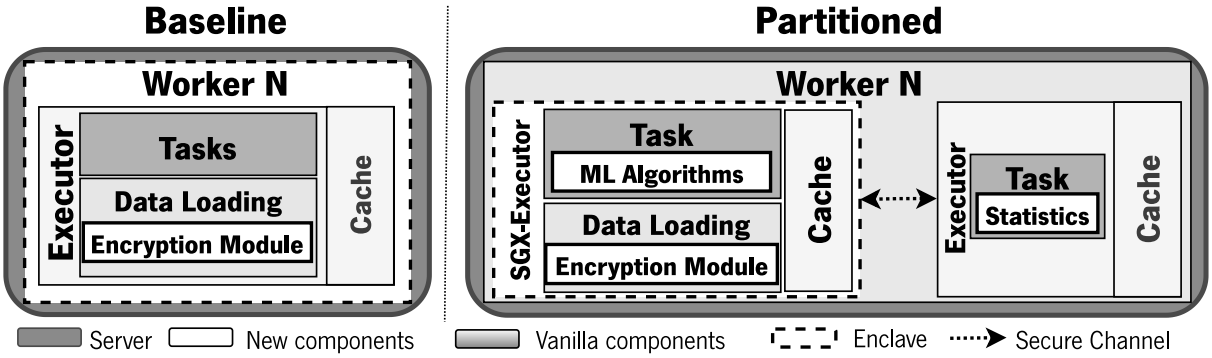


Figure 3.6: Comparison between Soteria-B and Soteria-P schemes.

Soteria Baseline (Soteria-B). In Soteria-B, all computation done by Spark workers is included in a trusted environment. Namely, the executor processes launched by each worker node are deployed inside an enclave, as depicted in Figure 3.6. Outside the enclave, data is always encrypted in an authenticated fashion, allowing the worker to decrypt and validate data integrity within the enclave.

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

Therefore, Soteria-P decouples statistical processing, used for assessing the performance of inference and training tasks, from the actual computation of the ML algorithms done over sensitive plaintext information. This decoupling builds directly upon MLlib and refactors its implementation without requiring changes to how users submit ML tasks. As depicted in Figure 3.6, statistical processing is done by

executor processes in the untrusted environment, while the remaining processing endeavors are done by another set of executors inside a trusted enclave (*SGX-executors*).

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

To provide a more concrete example, Algorithm 3.1 depicts the pseudo-code of a Linear Regression algorithm under the Soteria-P scheme and behaves as follows. The goal is to minimize the loss function, in this particular example, the *Root Mean Squared Error*. First, in the SGX-executor, an instance of Spark loads the dataset, creating a dataframe (X, y). This dataframe is further split into train and test data, $(X_{train}, y_{train}, X_{test}, y_{test})$. After this first pre-processing, an instance of a Linear Regression algorithm (lrM) is trained with the training data, and with the testing data, the first predicted values are inferred (P).

With these values, the Root Mean Squared Error (RMSE) is calculated at the non-secure executor ($rmse$). This computation is depicted in Algorithm 3.2, which intends to find the minimum error value. If no initial error is available, the algorithm returns the calculated RMSE. Otherwise, it returns the newly calculated RMSE ($newRMSE$). This is the only part of the computation within Soteria-P that is done outside trusted hardware, and thus, it is highlighted in orange.

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

Algorithm 3.1: LinearRegression(DS, prms, lrM):

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

Algorithm 3.2: Untrusted($y_{\text{test}}, y_{\text{new}}, \text{rmse}$):

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

The statistical information that can be computed outside the secure enclave may differ and must be decided on a per-algorithm basis. For instance, Principal Component Analysis (PCA), a dimensionality reduction algorithm, performs its computation mostly on top of raw data. However, this algorithm performs an initial validation based on the number of features and principal components to understand if the remaining computation can be done. This validation is done by a statistical function defined in MLlib as $memorycost$, which in Soteria is done outside the enclave.

As with other examples, loss algorithms such as Root Mean Squared Error (RMSE) or Absolute Error (L1), used in Linear Regression, Alternating Least Squares, or Gradient Boosted Trees, are also offloaded from the secure enclaves. Similarly, the optimization algorithms, such as Expectation-Maximization (EM), implemented in Latent Dirichlet Allocation, the cost functions to evaluate the centroids of K-means, and the accuracy calculations in Naive Bayes are decoupled from the main algorithms to run outside of enclaves.

3.3.5 Security Analysis

Our security goal is formally defined using the real-versus-ideal world paradigm, similar to the Universal Composability framework [44]. Succinctly, we prove that Soteria is indistinguishable from an idealized service for running ML scripts in an arbitrary external environment that can collude with a malicious insider adversary. We then use that abstraction to demonstrate how Soteria is resilient to real-world ML attacks. This idealized service is specified as a functionality parameterized with the input data, which executes the tasks described in the ML task script and returns the output to the client via a secure channel.

The full proof of Soteria-P and Soteria-B schemes can be found in Appendix A.1. The outline is as follows. The role played by the master node can be seen as an extension of the client, establishing secure channels, providing storage encryption keys, and receiving outputs. We follow the reasoning of [23] and replace the master node with a reactive functionality performing the same tasks. Similarly, each Soteria worker behaves simultaneously as a processing node and as a client node, providing inputs to the computation of other workers (*e.g.*, model training parameters). This enables us to do a hybrid argument, where worker nodes are sequentially replaced by idealized reactive functionalities executing their roles in the task script.

Finally, all processing is done in ideal functionalities, and all access to external storage is fixed by the

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

3.3.5.1 Security implications of statistical leakage

To show that our system is resilient against ML attacks, we must consider a common prerequisite for such attacks to be successful: the adversary must have black-box access to the model (as per definition on Section 3.2.2). Our result implies that adversaries cannot infer internal data from the workers, and the secure channel between the client and master prevents adversaries from injecting queries into the system. This would intuitively suggest that our adversary is unable to perform queries to the model in a black-box fashion. However, Soteria-P has the aforementioned additional leakage of statistical information.

As such, a crucial security question to answer is: *how does statistical information relate to black-box model access, i.e., does the first imply the second in any way?* Specifically, our argument is by reduction: if an attack based on black-box access to the model occurs in Soteria-P, then the adversary must have been able to extract such access from the statistical information revealed. Indeed, recent work by Chandrasekaran et al. [50] shows that, by only basing the attack on statistical information not directly associated with the raw data (i.e., training and validation dataset, ML model), one cannot perform attacks that disclose sensitive information from such raw information.

Given how statistical data depends on the underlying ML script, consider the concrete example provided in Algorithm 3.1. Here, the leakage can be defined as the sum of all data revealed to the untrusted execution, namely the set of predictive labels y_{test}, and the results of $maxIter$ number of predictions after $lrM.\text{fit}(X_{\text{train}}, y_{\text{train}}, \text{prms})$. Concretely, the leakage l of an execution of Algorithm 3.1 can be defined as:

$$l = (y_{\text{test}}, \sum^{maxIter} lrM.\text{transform}(X_{\text{test}}, y_{\text{test}})) \tag{3.1}$$

Equation 3.1 quantifies the amount of information explicitly revealed to the adversary of Soteria. As such, attacks requiring black-box access to the model can only occur if there exists an efficient algorithm that can take l and produce a sufficient approximation to lrM for black-box attacks to be conducted.

For the general case, extracting model access from statistical data is an ongoing area of research. However, current attacks suggest one is unable to do this in any successful way [50]. This supports our thesis that statistical values *are not sensitive information*, so their leakage does not expose our system to these types of attacks. From this, it follows that Soteria-P *scheme is resilient to any attack requiring black-box access to the model to succeed*.

3.3.5.2 Relation to ML Attacks

We now overview the four types of attacks referred to in Section 3.2.2 on a case-by-case basis. Appendix A.2 contains a more in-depth analysis of these attacks.

Soteria achieves resistance against input forgery through authenticated data encryption. This means that the input dataset is authenticated by the data owner and explicitly defined in the *Manifest file*, allowing the SGX-executors to check the authenticity of all input data. Thus, no forged data is accepted for processing, which is necessary for performing any *adversarial attack*.

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

Finally, *reconstruction attacks* require additional knowledge about internal ML model data. Our security result shows that Soteria is indistinguishable from an idealized ML service, which does not reveal the trained model. This includes the critical feature vectors required for this attack, which cannot be inferred from confidence values and class probabilities alone. Alternatively, *reconstruction attacks* requiring black-box access to the model are strictly stronger, but this, as we have argued, is not possible only with knowledge of confidence values, class probabilities, ROC/AUC curves, and table summaries (the explicit leakage of Soteria-P, as defined in Appendix A.1).

3.3.6 Implementation

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

Our prototype supports both Soteria-B and Soteria-P schemes. For Soteria-P's implementation, Spark's MLlib implementation was decoupled into two sub-libraries, one with the statistical processing (to be executed outside SGX) and another with the remaining ML computational logic (to be executed inside SGX).

Gramine 1.0 was used for the overall management of Intel SGX enclaves' life cycle, for specifying the computation (*i.e.*, internal Spark and MLlib libraries) to run at each enclave, and for establishing secure channels (*i.e.*, with the TLS-PSK protocol) between the enclaves at the master and worker nodes [289]. Soteria's *Manifest* file was also provided by Gramine.

3.4 Evaluation

Our evaluation answers three main questions:

1. How does Soteria impact the execution time of ML workloads?

2. How does the Soteria-P scheme compare, in terms of performance, with state-of-the-art approaches (*i.e.*, Soteria-B and SGX-Spark)?

3. Can Soteria efficiently handle different algorithms and dataset sizes?

We split our evaluation into three different stages to present our results more clearly. In detail, Subsection 3.4.1 presents the methodology used for the testbed, Subsection 3.4.2 summarizes the main evaluation observations, and Subsection 3.4.3 analyzes these observations and provides key insights.

3.4.1 Methodology

3.4.1.1 Environment

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

3.4.1.2 Workloads

As depicted in Table 3.2, we resort to the HiBench benchmark [130] for evaluating seven ML algorithms that are broadly used and natively implemented on top of MLlib. Further, the benchmark suite offers different workload sizes for each algorithm ranging from *Tiny* to *Gigantic* configurations. Next, we describe each of the algorithms in more detail.

Alternating Least Squares (ALS). ALS is an algorithm mainly used in recommendation systems, such as the ones used by Amazon and Facebook, among others. The algorithm is usually formulated as the factorization of an m x n matrix R, where m can be seen as the users and n the items. In this setting, the items' ratings given by the users are not all filled, and the ALS algorithm is used to fill those blank spaces, recommending new items to the users. According to [117], the time complexity of this algorithm is dependent on m, n, and hidden k dimension, which results in a complexity of $O((m+n)k^3 + mnk^2)$.

Linear Regression (LR). Linear Regression is broadly used in statistics to understand the correlation between one dependent variable and one or more independent variables. The outcome of this algorithm is supposed to follow a Gaussian distribution. The computation of a Linear Regression algorithm is based on matrix multiplications and inversions in a matrix of m x n, where m is the number of samples and n is the number of features, the time complexity equals $O(m * n^2 + n^3)$ [75].

K-means clustering (K-means). K-means is an algorithm used in clustering tasks (unsupervised learning tasks). Giving a dataset R ($x_1, ..., x_m$), data should be grouped into clusters. For each data point x_i, a feature vector is given with no labels y_i. Here, for each data point, the goal is to predict k centroids and the label y_i. For this matter, K-means has a quadratic time complexity, $O(n^2)$, with n being the size of the input dataset [217].

Gradient Boosted Trees (GBT). With a fundamental basis in decision trees, GBT merges decision trees with gradient boosting. Overall, an ensemble of trees is used for predicting the target label. A first model is created and followed by a second model that should surpass the first one's results. Combining the best model with the previous models should minimize the error [91]. For each tree, there is a time complexity dependent on the number of training samples, n, and the number of labels, y, $O(nyn_{trees})$ [268].

Latent Dirichlet Allocation (LDA). As a dimensionality reduction algorithm, the goal of LDA is to lower the high-dimensional space, recurring to eigenvalue decomposition, increasing the usability of data and the feasibility of other algorithms. The complexity of this algorithm is given as $O(mnt + t^3)$, where m is the number of samples, n is the number of features, and $t = min(m, n)$ [43].

Principal Components Analysis (PCA). Being a classic method for dimensionality reduction, PCA, in a dataset of n observations and m variables, tries to compute a target dimensionality y based on the eigenvalue decomposition of its covariance matrix [328]. The time complexity of PCA, as similar to other algorithms, depends on the size of matrix nxm, $O(nm * min(n, m) + m^3)$ [80].

Sparse Naive Bayes (Naive Bayes). Used for multi-class classification tasks, Naive Bayes is a highly efficient algorithm that only needs to compute one time over the dataset where, given a matrix of n training examples and m attributes, the time complexity is $O(nm)$ [81].

Table 3.2: Representation of each ML algorithm's tasks, time complexity, and data sizes for different workloads.

Algorithms	Tasks	Time Complexity	Workloads			
			Tiny	Large	Huge	Gigantic
ALS	RS	$O((m+n)k^3 + mnk^2)$ [117]	193KB	345MB	2GB	4GB
PCA	DR	$O(nm * min(n, m) + m^3)$ [80]	256KB	92MB	550MB	688MB
GBT	P	$O(n * y * n_{trees})$ [268]	36KB	46MB	92MB	183MB
LR	C + P	$O(m * n^2 + n^3)$ [75]	11GB	134GB	335GB	894GB
Naive Bayes	MC	$O(nm)$ [81]	-	-	5GB	-
LDA	DR	$O(mnt + t^3)$ [43]	-	-	2GB	-
K-means	CI	$O(n^2)$ [217]	-	-	56GB	-

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

3.4.1.3 Setups and Metrics

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

Moreover, we test SGX-Spark [109], a state-of-the-art SGX-based solution that protects both analytical and ML computation done with Apache Spark. It is designed to process sensitive information inside SGX enclaves, so it can be considered the most similar system to Soteria. However, SGX-Spark can only guarantee that *User Defined Functions (UDFs)* are processed in secure enclaves. This decision leaves a large codebase of Spark outside the protected memory region and, consequently, limits the users to only being able to execute privacy-preserving ML algorithms based on UDFs[2].

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

3.4.2 Performance Overview

Figure 3.7 shows the execution time of all the setups for the seven algorithms when using a huge-sized workload configuration. Moreover, Figures 3.8b, 3.8c, 3.8a and 3.8d present the performance evaluation for *PCA*, *GBT*, *ALS* and *LR* algorithms for different workload sizes. Next, we list our main observations to aid in the characterization of these results. Unless stated otherwise, the performance overhead values discussed in this section correspond to the number of times that the algorithm's execution time increases for a given setup when compared to the Vanilla Spark deployment results. *Observations 1* to *8* correspond to Figure 3.7, whilst *Observations 9* to *12* refer to Figure 3.8.

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

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

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

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

[2]Another similar solution to Soteria is Uranus [135]. Although Uranus is open-source, it currently does not provide enough information on how Apache Spark and all the tested ML algorithms could be deployed using this system, limiting a possible comparison against Soteria.

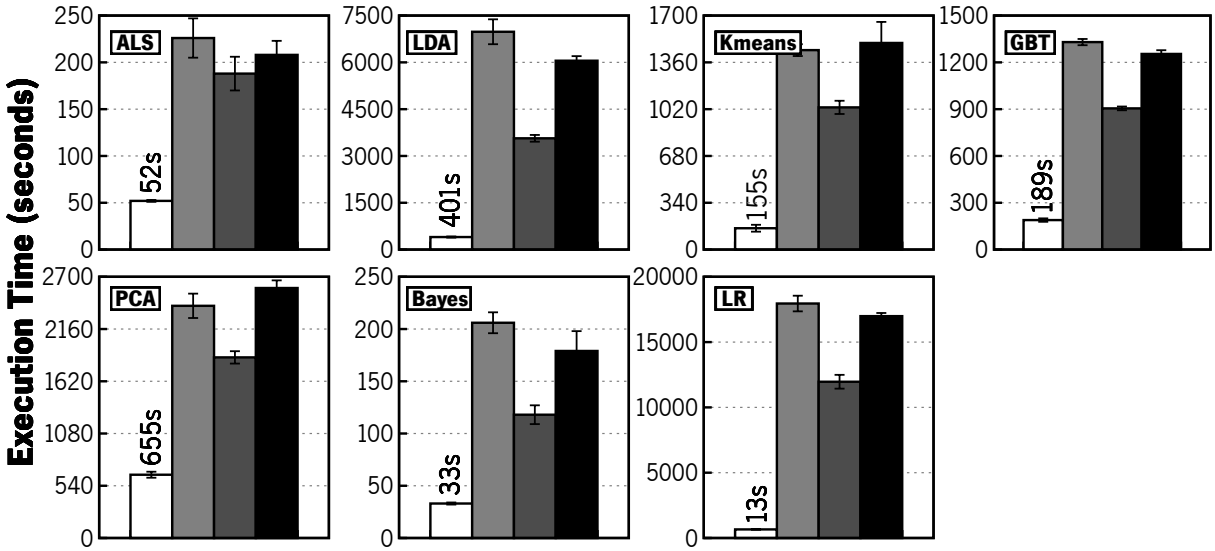


Figure 3.7: Execution time for each algorithm with *Huge* workload. The legend is as follows: □ Vanilla Spark; ■ Soteria-B; ■ Soteria-P; ■ SGX-Spark.

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

Observation 6. The LDA algorithm exhibits higher execution overhead of 17.40x, 8.89x, and 15.08x for Soteria-B, Soteria-P, and SGX-Spark setups, respectively. Soteria-P outperforms SGX-Spark by a difference of 41.5 minutes (*i.e.*, reduces execution time by 41%).

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

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

Observation 9. For *Tiny* and *Large* workloads with the PCA algorithm, Soteria performs similarly for our two schemes while outperforming SGX-Spark. With larger workload sizes, the overhead imposed by our solutions increases. However, Soteria continues to show better performance than SGX-Spark. Soteria-B has an overhead of 1.96x to 5.15x for *Tiny* and *Gigantic* workloads, whilst Soteria-P incurs an overhead of 1.72x to 3.79x. When compared with SGX-Spark, the results show an absolute difference of 4 seconds and 7 minutes (7%) for Soteria-B and 7 seconds and 33 minutes (19% and 31%), respectively, for Soteria-P.

Observation 10. Regarding the GBT algorithm, and the *Tiny* workload, the overhead of Soteria-B, Soteria-P, and SGX-Spark are similar. However, the difference between the three approaches is more

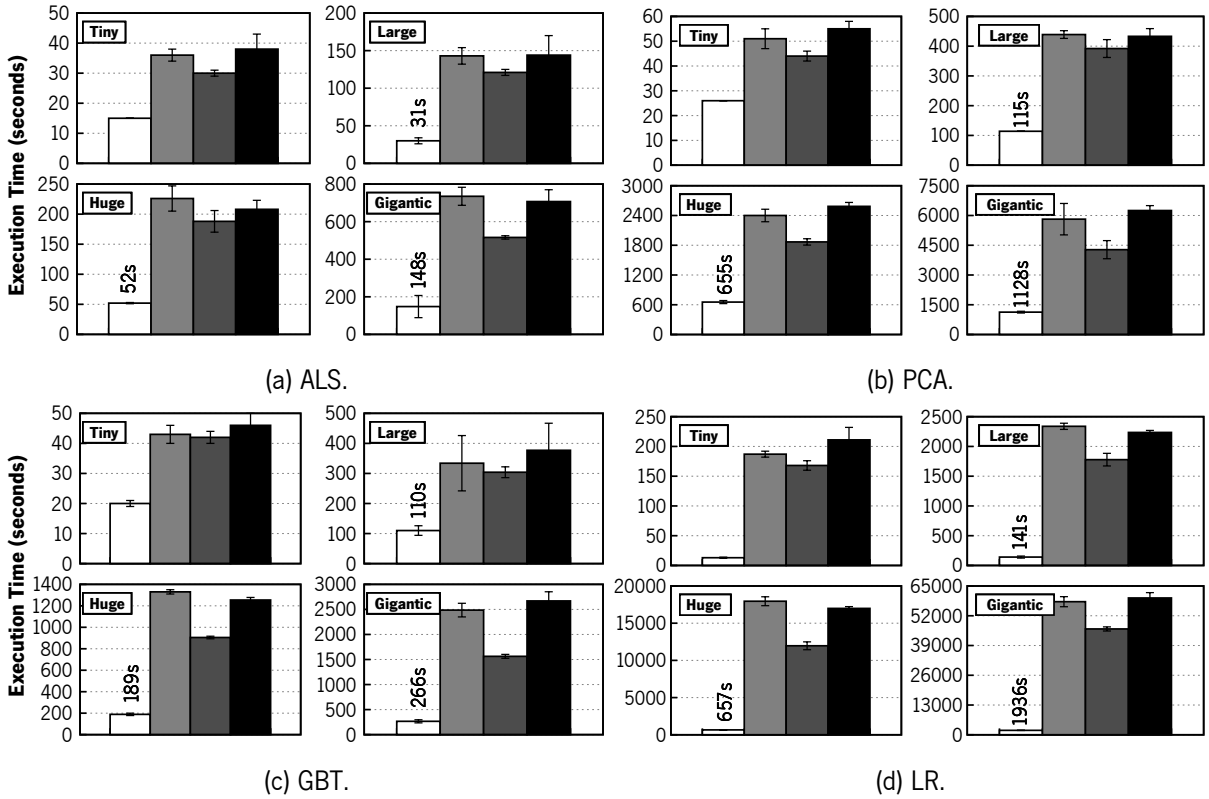


Figure 3.8: Runtime execution for PCA, GBT, ALS, and Linear Regression for *Tiny*, *Large*, *Huge* and *Gigantic* workloads. The legend is as follows: □ Vanilla Spark; ■ Soteria-B; ■ Soteria-P; ■ SGX-Spark.

visible when increasing the workload size. Soteria-P (*Tiny-2.13x and Gigantic-5.88x*) outperforms both approaches, while Soteria-B (*Tiny-2.18x, Gigantic-9.35x*) and SGX-Spark (*Tiny-2.3x, Gigantic-10.34x*) have similar results. Soteria-P surpasses SGX-Spark's execution time in the *Gigantic* workload by up to 41%.

Observation 11. With ALS, Soteria-P shows an execution time overhead of 2.04x and 3.28x, for the *Tiny* and *Gigantic* workloads, respectively. Soteria-P achieves lower overhead than Soteria-B and SGX-Spark for all dataset sizes, with the execution time decreasing by 8 seconds (9%) for the *Tiny* and 191 seconds (27%) for the *Gigantic* workloads.

Observation 12. For LR, with the *Tiny* workload, Soteria-B and Soteria-P increase execution time by 14.39x and 12.95x, respectively. As for the *Gigantic* workload, Soteria-B incurs an overhead of 30.04x and Soteria-P of 23.89x. Compared to SGX-Spark, Soteria-P decreases the execution time by 43 seconds for the *Tiny* workload and by 4.31 hours for the *Gigantic* workload (22.6%).

Observation 13. The overall CPU usage for all the experiments is similar for Vanilla, Soteria-B, Soteria-P, and SGX-Spark. In more detail, Soteria-B with LR presents the upper-bound limit for CPU, showing an increase of 9% compared with Vanilla (20,7% CPU usage) as seen in Table 3.3. The standard deviation is never above 10%.

Observation 14. The mean memory consumption for each algorithm is similar across the different setups. As depicted in Table 3.4, Soteria-B presents an increase of ± 20% when compared to the Vanilla

Table 3.3: Mean CPU usage (in %) for ALS, PCA, GBT, LR, Naive Bayes, LDA, and Kmeans algorithms with the *Huge* workload. Soteria-B shows an increase of up to 9% compared to Vanilla.

	CPU (%)			
Algorithms	**Vanilla**	**Soteria-B**	**Soteria-P**	**SGX-Spark**
ALS	19.7	23.8	22.6	23.1
PCA	19.1	25.5	24.5	27.1
GBT	23.0	28.3	27.5	26.7
LR	20.7	29.7	29.1	33.1
Naive Bayes	18.3	24.6	22.3	23.9
LDA	20.0	28.3	23.3	26.4
K-Means	18.6	26.9	24.5	29.0

Table 3.4: Mean Memory usage (in %) for ALS, PCA, GBT, LR, Naive Bayes, LDA, and Kmeans algorithms with the *Huge* workload. Soteria-B shows an increase of up to 20% compared to Vanilla.

	Memory (GB)			
Algorithms	**Vanilla**	**Soteria-B**	**Soteria-P**	**SGX-Spark**
ALS	3.2	3.8	3.5	3.6
PCA	4.8	5.5	5.1	5.5
GBT	4.2	4.8	4.5	4.5
LR	5.4	6.1	5.9	7.7
Naive Bayes	4.0	4.5	4.3	4.7
LDA	3.6	4.5	4.3	4.6
K-Means	5.5	6.1	5.9	6.6

setup, which is explained by the extra memory needed when fetching data to the enclaves. The standard deviation is never above 9%.

Table 3.5: Mean Network usage (in MB/s) for ALS, PCA, GBT, LR, Naive Bayes, LDA, and Kmeans algorithms with the *Huge* workload. Soteria-P shows a slight increase in network usage.

	Network (MB/s)			
Algorithms	**Vanilla**	**Soteria-B**	**Soteria-P**	**SGX-Spark**
ALS	60.3	65.3	62.3	63.3
PCA	133.0	147.7	139.0	142.3
GBT	48.0	52.7	50.0	53.7
LR	121.7	130.7	133.7	130.0
Naive Bayes	95.0	100.7	103.3	101.3
LDA	80.7	87.7	89.3	93.0
K-Means	106.7	115.0	115.3	116.3

Observation 15. The network shows an upper-bound increase of 10% in Soteria-B with PCA due to extra encrypted data paddings being sent between Spark workers, as seen in Table 3.5. Vanilla Spark shows an upper-bound network usage of 133MB/s for the PCA algorithm. Interestingly, it is possible to see a slight

increase (± 3 MB/s) in the network bandwidth usage on the Soteria-P scheme when compared to vanilla and Soteria-B, which is explained by the extra communication between SGX-executors and non-secure executors. The standard deviation is never above 10%.

Observation 16. Soteria does not impact the accuracy of ML workloads. For all experiments, we measured the corresponding accuracy metrics (*e.g.*, accuracy, root mean square error, or ROC). The results corroborate that both Soteria-B and Soteria-P show accuracy values similar to the vanilla Spark setup.

3.4.3 Analysis

We split the following analysis into three topics, *i)* dataset size; *ii)* algorithm complexity; *iii)* size of trusted computing base (TCB).

3.4.3.1 Dataset size

Figure 3.8 shows the performance degradation for the PCA, GBT, ALS, and Linear Regression algorithms with increasing dataset sizes. Results show that, for PCA, GBT, and ALS workloads with smaller datasets, Soteria-B and Soteria-P perform similarly. However, as the size of the datasets increases, more operations and data must be transferred to the SGX enclave, thus having a more noticeable toll on the overall performance. Indeed, the page swapping mechanism of SGX, which occurs due to its memory limitations, incurs a significant performance penalty [73, 318]. For example, when compared to the vanilla setup, the PCA algorithm overhead for Soteria-B varies between 1.96x for *Tiny* workload and 5.15x for *Gigantic* workload. While for Soteria-P, the execution time increases by 1.78x in the *Tiny* workload and 3.79x in the *Gigantic* workload. Linear Regression is the most expensive algorithm in terms of performance as it processes more data for the distinct workload sizes (Table 3.2). Compared with SGX-Spark, Soteria-P deals better with the data volume increase. Indeed, as seen in *Observations 9-12*, we reduce the execution time from 9% up to 41% when compared to SGX-Spark. Also, compared with our baseline, Soteria-P achieves up to 33% less execution time.

3.4.3.2 Algorithm Complexity

The execution times of ALS and LDA algorithms are very different even though their dataset size is similar. The computational complexity of each algorithm explains these results. For ALS, the synthetic workload data generated by the benchmark has a low hidden k dimension with a low ranking of 10, simplifying the required computation and decreasing execution time. For the LDA algorithm, the computational complexity, and consequently the execution time, are increased due to the higher number of dependencies between values at the generated synthetic workload data. *Observations 2 and 6* emphasize the performance of these two algorithms for a similar workload size. Like LDA, *Observations 3 and 9* show that PCA complexity and performance overhead increase with the processed data volume. Commonly classified as

regression and classification algorithms, Bayes and GBT have similar performance, as seen in *Observation 8* and *Observation 5*. The data sizes of these two algorithms are entirely different, where GBT uses 91.7MB, and Bayes has 5.2GB. However, the Bayes algorithm iterates over the data only once, while GBT iterates over several decision trees to find its best model. Kmeans' performance is highly dependent on the chosen dataset size. This is also true for the Linear Regression algorithm (*Observations 4 and 12*).

3.4.3.3 Size of TCB

The results discussed in Section 3.4.2 show that SGX-Spark outperforms Soteria-B for some of the evaluated algorithms (*Observation 2, 4-6, 8*). As SGX-Spark only protects UDFs, the performance overhead imposed by our solution's larger trusted computing base is naturally higher. Nevertheless, when compared to SGX-Spark, Soteria-B covers a wider range of machine learning attacks while keeping performance overhead below 1.59x. Indeed, for algorithms such as PCA and Kmeans, Soteria-B has a similar or slightly inferior execution time (*Observations 3 and 7*). This happens because, for these algorithms, both SGX-Spark and Soteria-B perform similar computations at the secure enclaves, while the UDF mechanism is not the most optimized choice for running some of these workloads.

Finally, due to the TCB reduction by our second scheme, Soteria-P consistently outperforms SGX-Spark and Soteria-B (*Observations 2-12*). The results show that this solution can reduce the training time by up to 41%, namely for the LDA algorithm with the *Huge* workload (*Observation 6*). Although the statistical information outsourced from secure enclaves differs for each algorithm, in general, Soteria-P outperforms Soteria-B from 1.1x to 1.9x (*Observation 6 and 10*). In detail, for GBT with the *Tiny* workload, the gains of Soteria-P over Soteria-B are low, 1.1x. The small difference in gain is due to the dataset size but also because the computation offloaded from the enclave (*i.e.*, Absolute Error for calculating the model's loss) is only executed once after the entire tree is constructed (*Observation 10*).

Conversely, when running LDA with both Soteria-B and Soteria-P with workload *Huge*, we observe a gain in performance by Soteria-P of 1.9x. In this particular case, the optimization of LDA, which resorts to the EM algorithm and is done outside of the trusted enclave, is performed in several stages of the model training process. This further shows the performance impact of offloading different amounts of statistical computation from secure enclaves.

3.4.3.4 Discussion

The results show that Soteria-P outperforms other state-of-the-art approaches, namely SGX-Spark, for all the considered ML algorithms. Also, Soteria-P achieves better performance than the Soteria-B setup while offering similar security guarantees when considering distinct ML attacks (Section 3.3.5). This is made possible by filtering key operations to be done outside enclaves.

In detail, when compared to Soteria-B, Soteria-P reduces ML workloads' execution time by up to 37%. When compared with SGX-Spark, the execution time is reduced by up to 41%. Interestingly, for the LR algorithm using a *Gigantic* workload (*894GB*), Soteria-P decreases computation time by 4.3 hours and 3.3

hours when compared with SGX-Spark and Soteria-B, respectively. The performance overhead of Soteria-P for the four different algorithms ranges from 1.7x to 23.8x when compared to Vanilla Spark.

3.5 Related Work

This section is split into two subjects. First, we compare Soteria with other solutions for deploying generic applications on trusted execution environments (§3.5.1). Then, we discuss solutions targeting privacy-preserving ML and analytics on TEEs (§3.5.2).

3.5.1 Trusted Execution Environments

The challenges associated with deploying applications on TEEs have been studied since this technology became available, as exposed in §2.3.

In brief, Soteria differs from the solutions that deploy full applications inside TEEs, as it does not aim to offer a generic solution to deploy applications at enclaves. Namely, our system is focused on ML workloads running on top of Apache Spark, which, as shown in Section 3.3, enables specific optimizations in terms of computation partitioning that allow reducing performance overhead. Second, regarding the partitioning of computation between trusted and untrusted environments, our solution does not aim to propose a generic partitioning tool to this end. Namely, Soteria proposes a specific partitioning scheme optimized to balance the security and performance trade-offs for workloads using Apache Spark's MLlib.

3.5.2 Privacy-Preserving Machine Learning and Analytics on TEEs

For this work, we classify PPML solutions into four main groups based on the techniques being used: *i)* encryption-based [32, 97, 231], *ii)* secure multi-party computation [171, 198], *iii)* differential privacy [3, 266] and, *iv)* trusted execution environments (TEEs) [126, 129, 210, 285]. Soteria is included in group *iv)*.

3.5.2.1 Privacy-preserving ML with TEEs

Chiron [126] enables training ML models on a cloud service without revealing information about the training dataset. Also, once the model is trained, only the data owners can query it. It supports launching multiple enclaves while each operates on different shards of training data, keeping the enclaves synchronized via a parameter server to ensure they all collaboratively converge to the same model. Besides providing a privacy-preserving approach, this framework also provides data parallelism based on a parameter server to train the model faster. *Myelin* [129] offers a similar solution to Chiron while adding differential privacy and data oblivious protocols to the algorithms to mitigate the exploits from *side-channels* and the information leaked by the model parameters. Soteria differs from these works as it can cover both the training

and inference phases while providing additional protection against *adversarial samples*, *reconstruction*, and *membership inference* attacks (Table 3.1).

In Ohrimenko et al., five ML algorithms are re-implemented with data oblivious protocols[3]. Combined with TEEs, these protocols ensure strong privacy guarantees while preventing exploiting *side-channel* attacks that observe memory, disk, and network access patterns to infer private information. Unlike this solution, Soteria aims to support all ML algorithms built with MLlib transparently.

3.5.2.2 Privacy-preserving analytics with TEEs

TEEs have also been used to ensure privacy-preserving computation for general-purpose analytical frameworks [263]. In comparison to SGX-Spark [109], detailed in Section 3.4.1, Soteria supports a broader set of algorithms (*i.e.*, any algorithm that can be built with the MLlib API) while protecting users from a more complete set of ML attacks (Table 3.1).

Opaque [321], and Uranus [135] resort to SGX to provide secure general-purpose analytical operations while only supporting a restricted set of ML algorithms. Opaque combines SGX with oblivious protocols and requires the re-implementation and rewriting of the default Apache Spark UDF operators. This solution tries to mitigate the access pattern leakage by sorting and shuffling the entire database, representing a performance overhead of 1.6-46x when adding obliviousness to the system. Uranus is also based on porting UDF processing to SGX enclaves but includes a single ML workload. Differently, Soteria is targeted at ML and is not limited by UDF-based algorithms that, compared with MLlib-based ones, exhibit lower performance for some ML workloads [68]. Therefore, the design, implementation, and security requirements are distinct compared to Soteria.

Like the above solutions, SGX-BigMatrix [263] also relies on SGX to deliver a framework to process large encrypted datasets while hiding their access patterns. This solution proposed a new processing framework based on a generic language optimized for data analytic tasks. Unlike this work, Soteria relies on a widely-used framework, namely Apache Spark, maintaining its easy-to-use interface while increasing its privacy guarantees.

3.5.2.3 Privacy-preserving deep learning with TEEs

TEEs have also been applied to the training and inference of DNNs [110, 150, 155, 285]. However, there is a substantial difference between the internals of ML and DL frameworks and algorithms, thus requiring significantly different privacy-preserving designs for each scenario. Since MLlib does not natively support DL workloads, the focus of Soteria is solely on ML algorithms.

[3] A data oblivious algorithm guarantees that its control flow and memory access patterns are non-dependent of the input data [196].

3.5.2.4 Summary

As shown in Table 3.1, current solutions related to Soteria (*i.e.*, addressing privacy-preserving ML) cover a smaller spectrum of ML exploits. The table does not include Opaque [321] and Uranus [135] since the first only covers SQL queries, while the second only supports a single ML workload. The works in this section are summarized in Table 3.6, categorized by cryptographic primitive, distribution, workload, applicability to Apache Spark, and availability. The type of distribution here refers to centralized, distributed, and collaborative computation. For instance, multi-party computation is depicted as collaborative computation, while outsourcing the computation to the cloud or other third-party infrastructure to perform the intended computation in a single server is named centralized, and approaches similar to the deployment setup of Apache Spark are depicted as distributed.

Table 3.6: Taxonomy of Related Work systems.

Systems	Crypt. Primitives	Distribution	Algorithms	Spark-enabled	Avail.
Chiron [126]	○	*D*	*DL*	○	○
Myelin [129]	○,◑	*D*	*DL*	○	○
SGX-BigMatrix [263]	○, ◐	*C*	*ML*	○	○
SGX-Spark [109]	○	*D*	*ML, A*	●	●
Uranus [135]	○	*D*	*A (mostly)*	◐	●
Opaque [321]	○,◐	*C*	*A*	○	●
Slalom [285]	○	*C*	*DL*	○	●
TensorScone [150]	○	*C*	*DL*	○	○
Privado [110]	○	*C*	*DL*	○	○
Occlumency [155]	○	*C, H*	*DL*	○	○
Soteria [38]	○	*D*	*ML*	●	●

Crypt. Primitives	Distribution	Algorithms	Spark-enabled	Availability
○ - TEE	*C* - Centralized	*ML* - Machine Learning	● - Yes	● - Yes
◐ - OP	*D* - Distributed	*DL* - Neural Networks	○ - No	○ - No
◑ - DP	*H* - Collaborative	*A* - Analytics (SQL queries)	◐ - Partially	

In detail, Chiron [126], SGX-Spark [109], and Myelin [129] aim at protecting sensitive data's privacy during the model training stage while being resilient to model extraction attacks. However, these are vulnerable to adversarial and membership inference attacks because the dataset used for training is not encrypted at rest. Furthermore, SGX-Spark is vulnerable to path-finding and class-only model extraction attacks because only UDF processing is offloaded to trusted enclaves.

Conversely, SGX-BigMatrix [263] supports encryption at rest for the dataset, as in Soteria, but it does not provide encryption for the trained model, thus being susceptible to model extraction and membership inference attacks.

To the best of our knowledge, Soteria is the first PPDML framework that proposes an alternative TEE-based scheme (Soteria-P), which can improve the performance of training and inference workloads by

reducing the number of operations done at secure enclaves. As such, Soteria is the first solution that covers a large spectrum of ML exploits and supports various ML algorithms while not changing how users build and run their algorithms within Spark MLlib.

Moreover, while orthogonal to this work, we acknowledge the increasing focus of the community on privacy-preserving DL-based solutions [110, 150, 155, 285]. To apply Soteria to these algorithms, one would need to profile which operations could be offloaded to run outside the secure enclaves. Also, one would need to support third-party APIs, as Apache Spark (v2.3.4) does not natively support DL workloads. We defer such tasks to future work.

3.6 Summary

This chapter introduces Soteria, a system for PPDML. Our solution builds upon the combination of Apache Spark and TEEs to protect sensitive information being processed at third-party infrastructures during the ML training and inference phases.

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

While Soteria presents a solution for general ML algorithms that are supported by Spark's MLlib, these algorithms do not fulfill the need for application-specific use cases. For instance, when dealing with genomic data and GWAS algorithms, we understand that these computations are not natively supported by Spark's MLlib and resort to third-party APIs (*e.g.,* Glow [99], Adam [8]). Although this is not a limitation of Soteria, it is important to understand how such a use case would benefit from the security guarantees and the computation partitioning scheme provided by Soteria, as well as the scalability provided by Apache Spark. In the next chapter, we present Gyosa, a system that extends Soteria to support GWAS-based algorithms.

4

Privacy-Preserving Machine Learning for Genome-Wide Association Studies

With the advent of next-generation sequencing (NGS) technologies, the cost of genome sequencing has decreased significantly, enabling the generation of large amounts of genomic data in a timely and cost-effective manner [269]. The availability of large-scale datasets opens up new avenues for research on genetic factors, but it also requires computationally efficient algorithms capable of handling the sheer magnitude of data.

Genomic Wide-Association Studies (GWAS) test the association of hundreds of thousands to millions of genetic variants in a cohort of individuals and find the variants that are statistically associated with a specific trait or disease [45, 291]. By 2021, more than 5700 GWASes have been conducted using data from more than one million individuals for more than 3300 traits [291].

However, the feasibility of running these algorithms and statistical methods relies on the existing computational power. When working on a single workstation, commonly referred to as a server, users face limitations imposed by the existing computational capacity. The computational demands of GWAS are directly linked to variables like the number of genetic variants, the number of individuals, and the tested traits and phenotypes. For very large datasets, a single server's computing and storage power may be insufficient. The conventional approach of enhancing server capacity by augmenting core processors, memory, and storage may encounter significant challenges, including exponentially escalating costs and inherent limitations associated with the presently available hardware [247]. Distributed computing, which allows the use of several servers (clusters of servers) in parallel, is a viable solution to reduce the runtime execution of parallelizable and data-intensive algorithms, such as GWAS [70]. Over the years, this paradigm has improved the parallelization and distribution of large-scale computations and their access to large amounts of data. However, acquiring, maintaining, and managing a distributed server infrastructure is a costly and complex task requiring high-end hardware and specialized human resources [247].

A more accessible option involves running GWAS analysis remotely on distributed infrastructures managed by third-party entities, such as those provided by Cloud Computing (*e.g.*, GCP, Azure [21, 60]) and

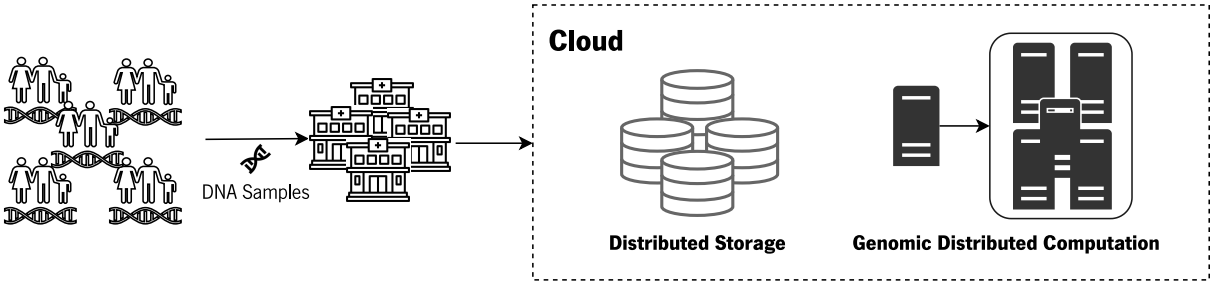


Figure 4.1: An example where a consortium of entities (*e.g.*, a group of Research Hospitals) have access to a large genomic dataset but lack the processing and storage power to analyze it. In this scenario, the entities can offload the processing and storage of the data to a third-party infrastructure (*e.g.*, a Cloud Computing service) and execute the genomic computation remotely.

HPC services (*e.g.*, TACC, AIST [5, 277]). This approach may offer significant benefits, especially in scenarios where: *i*) a single entity, such as a Hospital, possesses a large genomic dataset but lacks the processing and storage power for analysis; and *ii)* a consortium of entities, including Hospitals and research laboratories (see Figure 4.1), aims to collectively analyze their datasets in a unified remote infrastructure, enabling large-scale analysis using shared data.

When offloading data storage and processing to third-party infrastructures, users trust external entities to keep their information secure. However, there have been several reports detailing both external (*e.g.*, done remotely by a hacker) and internal attacks (*e.g.*, led by a malicious system administrator with physical access to the cluster) that have successfully compromised the privacy of sensitive information kept at remote infrastructures, such as the one provided by Cloud Computing services [131]. Indeed, these attacks are one of the major barriers limiting the wider adoption of cloud services for storing and processing clinical data, such as genomic data, since its disclosure can lead to the complete identification of individuals [169, 209]. For instance, early studies showed that only 75 SNPs could help identify an individual [169]. In addition, while carrying information on disease predisposition, the leakage of genomic data may imply privacy risks for the individual and their future and previous generations [128].

When outsourcing a genomic data processing pipeline (*i.e.,* data collection, processing, and sharing) to third-party infrastructures, users increase the attack surface and become susceptible to novel attacks. Namely, in membership inference attacks, the adversary (attacker) can leverage the knowledge it has on a specific individual (*e.g.,* any genetic information, disease predisposition) and query the analysis results explicitly. Specifically, Homer's attack and its variations use the background information of the human genome currently available in the public domain to infer whether an individual's genetic variants' information was used for a specific study [298]. Re-identification attacks aim to reveal the identities of individuals whose data have been anonymized and used in genomic analysis. Recent studies have shown that demographic information can be linked to public genomic data and databases by linking genomic data with genealogical databases [181]. Data poisoning attacks have the intention of manipulating results. It can generate false assumptions and associations when applied to GWASes, introducing bias and yielding erroneous discoveries [200].

To implement a PPDML solution able to address some of these attacks, one should consider some of the challenges addressed in the previous chapter by Soteria, with an added focus on the following:

- **C1: Performance** Genomic data analysis is computationally expensive (*i.e.,* it is typically deployed in single-servers with high-end hardware resources) without the additional complexity of adding privacy measures on top of it. One should be able to balance the trade-off between privacy and performance.

- **C2: Applicability** The need to change algorithms to accommodate different privacy-preserving techniques limits these solutions' applicability and versatility. There is a challenge when applying these solutions to different domains, namely to genomic data and algorithms used in GWAS.

- **C5: Scalability** Large-scale data analytics is becoming a focal point of data analysis. Nonetheless, the primary focus of privacy-preserving solutions for genomic data is on the security and privacy guarantees, not on the ability to scale to large amounts of data. Ensuring the methods can scale effectively to handle large datasets is an ongoing challenge.

Recent works focusing on the usage of TEEs have been typically deployed in a single server, and they have not been designed to handle distributed computation scenarios [57, 256]. Nonetheless, increasing the outlook on this subject is a demand as genomic data grows. Similar to the works depicted in Section 3.5, most of the current solutions are designed to deploy all the computation inside the secure enclave, which leads to an increasing TCB size and increase of the attack surface.

Contributions. Based on the previous challenges, we propose Gyosa, a novel distributed and privacy-preserving framework for securely executing GWAS in untrusted distributed infrastructures. Gyosa is built on top of Apache Spark [315] and uses Glow, a library for genomic processing that includes regression-based algorithms, statistical tests, and population stratification methods to perform GWAS easily [99]. These are combined with TEEs, namely Intel SGX, to provide a secure environment where sensitive genomic information can be efficiently processed in plaintext without disclosing it to internal or external attackers. The Gyosa approach distinguishes itself from recent proposals such [144, 325], as it promotes the outsourcing of computation in a distributed manner in untrusted third-party entities. Also, by resorting to Glow, Gyosa allows the extension of the current genomic analysis pipeline, in which one can add new tasks (*e.g.*, new statistical tests, genomic imputation, and querying). By expanding Soteria, our solution enables fine-grained differentiation between sensitive and nonsensitive information processed by GWAS. By offloading sensitive information to a secure environment, we show that it is possible to run GWAS-based algorithms for large datasets while keeping critical information safe from being leaked by attackers. Finally, Gyosa intends to follow the same design principles as Soteria, namely to be efficient, secure, and practical.

[0]This is further addressed in Section 4.4.3.

Evaluation. To validate our solution, we ran three algorithmic versions based on Logistic Regression, Linear Regression, and X^2 statistical tests. The first two algorithms were run with a variable workload on the benchmarking dataset *"Genome in a Bottle"* [327]. The X^2 test was run against a synthetic dataset simulating $8 * 10^5$ VCF files (one file per individual) with $1 * 10^6$ random unique SNPs. The first two tests evaluate our solution against a vanilla setup without any security guarantees. The third experiment is intended to evaluate the scalability, feasibility, and behavior of Gyosa with increasing servers.

First, the results show the feasibility of offering a privacy-preserving genomic analysis solution that preserves the original results of GWASes. As anticipated, there is a trade-off between runtime computation and the level of privacy-preserving guarantees. Gyosa shows a runtime execution overhead ranging from $2.5X$ for X^2 statistic tests to $10X$ on regression-based algorithms. Importantly, the results highlight that it is possible to effectively reduce the computation runtime overhead by distributing the computation across multiple nodes while still offering privacy guarantees for sensitive data.

4.1 Key Concepts

This section provides an overview of the key concepts and technologies that work as building blocks for our solution. We start by discussing the main concepts of GWAS and the genomic pipeline. Then, we present some statistical methods used in GWAS. Finally, we briefly discuss the *Glow* framework and its integration with Apache Spark.

4.1.1 Genome

The genome refers to the entire genetic material of a living organism, contained within DNA and shared across all cells. In humans, the DNA sequence comprises nearly $3.1\text{x}10^9$ base pairs. On average, between each human, the genome presents a variation of $4\text{x}10^6$ base pairs. These differences are commonly called Single Nucleotide Polymorphism, or SNP for short. Based on the reference and alternative alleles from the two copies of each chromosome, SNPs are often depicted in three configurations based on the alleles: 1) aa: reference-homozygous, 2) aA: heterozygous, and 3) AA: alternative-homozygous. SNPs can be detected with techniques like SNP arrays or genome sequencing [24].

4.1.2 Genomic Pipeline

The need to privacy-preserve genomic data results from the possible attacks on the pipeline of genomic analysis, see Figure 4.2, which can be divided into three main stages: *i)* data collection, *ii)* data analysis, and *iii)* data sharing.

Data Collection. In this stage, the tissue samples are collected from the patient and stored in a biobank. The samples are then processed and analyzed to obtain the genomic data that will be stored in the

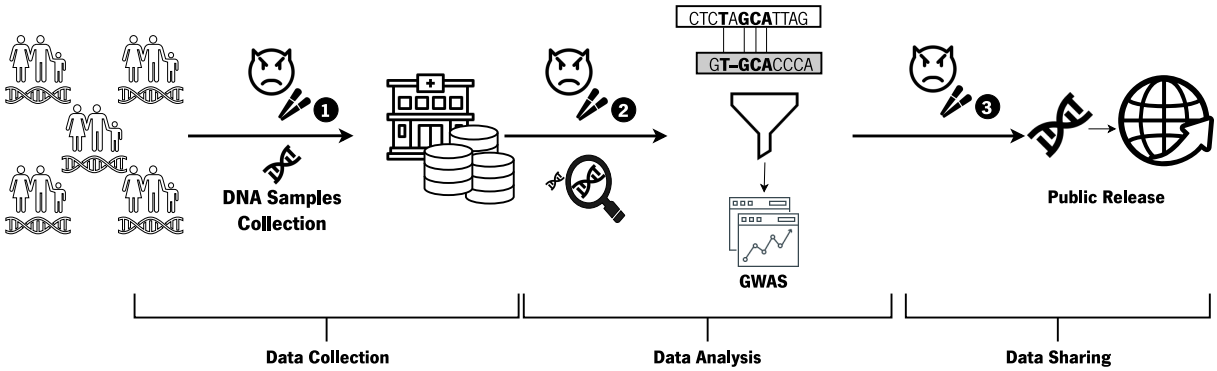


Figure 4.2: A genomic processing pipeline representing the several steps of the genomic analysis. From the collection of the genomic data to its storage, analysis, and public release. It also represents the several stages where data is most vulnerable ❶-❸.

Biocenter[1] or at third-party (*e.g.*, cloud computing infrastructures) premises according to the imposed regulations over such data (Figure 4.2-❶).

Data Analysis. Data is analyzed to derive useful information. This stage is particularly vulnerable to attacks since the data is usually analyzed in plaintext, which means data is fully open to malicious users while being processed (Figure 4.2-❷). This analysis may include several tasks, such as Short-read Sequence Alignment, Genome Imputation, Variant Call, and Genome-Wide Association Studies (GWAS).

Data Sharing. In this stage, the data is shared with the patient or the healthcare provider. This stage is vulnerable since data is often shared in an open format and already provides the association between the patient identity and the genetic information (Figure 4.2-❸).

The existence of sensitive data in a plaintext format at different pipeline steps makes it particularly vulnerable to different types of attacks. We now define these vulnerabilities and address the most common attacks that can be performed in the different stages.

4.1.3 GWAS

Although the genomic pipeline may include several analysis tasks, this work focuses only on GWAS, which tests the correlation between an SNP's allele frequency and the presence or intensity of a certain phenotype. Such studies have been performed for a large number of phenotypes, including many diseases. Tested individuals are separated into case and control groups, and to achieve significant and robust results, large sample sizes are required [291].

4.1.3.1 Statistical Methods

To evaluate the performance of Gyosa, we illustrate its practical application by employing two machine learning (ML) algorithms, Logistic and Linear Regression, and test the association with continuous and

[1]We define a Biocenter as any entity that gathers/collects genomic data (e.g., Hospitals, Medical Research Centers).

binary phenotypes. Following the approach in [144], we used the X^2 test to evaluate the scalability of Gyosa and its performance for different workloads with an increasing number of workers. The choice of such algorithms is corroborated by the current state-of-the-art, which shows that these algorithms are widely used to perform GWASes [291][2]. Next, we analyze the computational complexity of these algorithms using the Big-O notation.

Linear Regression. Such an algorithm is broadly used to fit the relation between one dependent continuous variable (*e.g.*, weight or blood pressure) and multiple independent variables. The algorithm is efficiently implemented based on matrix multiplications and inversions. For a matrix of m x n dimensions, where m is the number of samples and n is the number of features, the time complexity approximates $O(m * n^2 + n^3)$ [75].

Logistic Regression. This algorithm performs binary classification based on dependent variables to be able to distinguish between case and control cases [291]. The time complexity of Logistic Regression can be decomposed into d as the size of the phenotype vector and n as the covariates or features of the phenotype, $O(nd)$ [276].

X^2 **Test.** These tests evaluate if the observed and the expected frequencies (allele counts) in the case-control groups differ significantly. The test can be defined as follows:

$$X^2 = \sum \frac{(observed_i - expected_i)^2}{expected_i} \tag{4.1}$$

, in which we find the observed and the expected frequency of the ith SNP.

The time complexity of the chi-squared test is $O(n)$, where n is the number of samples [233]. Differently from the previous statistical methods, which intend to test for associations, the X^2 focuses solely on the frequency of the alleles.

P-values. *P-values* are a statistical metric used to quantify the differences and incompatibility between the observed results and the null hypothesis. In general, if the P-value is very small, it would be unlikely to observe the data under the null hypothesis. Therefore, the null hypothesis is either not true, or an outlier has been encountered [234]. Indeed, the resulting output of the Linear and Logistic Regression are the p-values.

4.1.4 Apache Spark and Glow

While being built based on the design principles of Soteria, Gyosa also resorts to Apache Spark to enable the distribution of computation. As discussed in §3.1.1, Spark promotes the seamless distribution of computation across multiple servers. It also provides a set of tools and utilities for feature extraction and model persistence. Alongside, Spark's modularity allows easy integration with several third-party

[2]We acknowledge the usage of Principal Component Analysis (PCA), commonly deployed for population stratification. Although addressed in §3.4 for a different use case, PCA is not currently implemented in Glow, and its privacy requirements may differ from the Gyosa current implementation due to its increasing data exchange and convergence rate.

applications. Solutions such as TensorflowOnSpark [308], Glow [99], or ADAM [8] have been built on top of Spark. For this work, we focus our efforts on the *Glow* framework, which provides a set of tools for genomic data analysis.

In brief, Glow offers a solution for performing GWAS at scale while supporting different data formats, including VCF, BGEN, and Plink, by internally converting them into Spark's DataFrames. This framework already includes regression-based algorithms, statistical tests, and population stratification methods to perform GWAS easily [99].

4.2 Threat Model and Genomic Attacks

Like the common ML pipeline, the genomic pipeline is vulnerable to several attacks. In this section, we discuss the most common attacks and the threat model employed in Gyosa.

4.2.1 Attacks on the Genomic Pipeline

When deploying a pipeline, such as the one in Figure 4.2, to a distributed environment, the attack surface will increase. This challenge arises since sensitive data storage and processing are now deployed across multiple servers. However, the critical attacks possible in a distributed pipeline are similar to those in a centralized pipeline (*i.e.,* when data and computation are outsourced to third-party infrastructures). The three most common attacks are *Membership Inference*, *Re-identification*, and *Data Poisoning*.

Membership Inference. These attacks intend to disclose the usage of a specific individual's data in the analysis. An adversary with access to the data and the results usually performs the attack. This adversary then tries to infer which individuals were used in the analysis. This is usually done by querying the analysis results for a specific individual and comparing those from others. If the results differ, the adversary can infer that the individual was not used in the analysis. Specifically, Homer's attack and its variations leverage the background information of the human genome that is currently available in the public domain to query models regarding a specific individual [122, 298].

Re-identification. Such attacks try to reveal the identities of the individuals whose data has been anonymized and used in the analysis. Recent studies have shown that demographic information could be linked to public genomic data and databases. Similarly, it was shown that this could be achieved by linking the genomic data with genealogical databases [181].

Data Poisoning. This attack occurs when malicious users want to poison the data to produce false results or contaminate the final analysis, usually by adding false data. Data poisoning in the context of genomic data and, more specifically, GWAS, can lead to false assumptions and associations that can create bias and release false insights [200].

4.2.2 Threat Model

Gyosa adopts the standard SGX threat model. We consider a scenario where a client seeks to use large amounts of sensitive genomic data and perform computation on top of it at third-party infrastructures. In this model, the client and the hardware are deemed trustworthy, whereas the third-party infrastructure's other components (*i.e.,* host OS, hypervisor) are regarded as untrusted. This assumption represents an honest-but-curious adversary model. In this model, the presupposition is that the adversary is honest and adheres to the protocol but is also curious and seeks to obtain maximum information. This is supported by existing research [74, 132, 144].

We highlight that solutions addressing concerns such as *Denial of Service (DoS)*, *side-channel* attacks, or *memory access patterns* can be employed in Gyosa [211]. However, we consider this research orthogonal to the one proposed here.

4.3 Gyosa

Gyosa is a privacy-preserving distributed GWAS-focused solution. It builds on top of Soteria [38] and resorts to Apache Spark and Glow for delivering a distributed genomic analysis framework. To achieve the proposed security guarantees, Gyosa uses Intel SGX (further detailed in §3.1.2), which provides secure memory regions as enclaves.

4.3.1 Design Goals

Gyosa is split into the client module, deployed on a trusted infrastructure, and the cluster module, deployed on an untrusted site (Figure 4.3). The client module encompasses the encryption of the genomic data and the submission of the GWAS to the untrusted infrastructure. The cluster module includes a distributed Apache Spark and Glow cluster to which the client module will submit the genomic analysis.

4.3.1.1 Client Module

The client module includes three main operations. First, it allows the encryption of VCF files based on authenticated encryption. This mechanism is added to Glow by providing a new class to encrypt this data type. Sensitive data is transparently encrypted before leaving the trusted premises. No changes in the implementation of the analysis scripts are required, thus avoiding any changes to the way users implement or specify their GWAS. Second, it handles the secure outsourcing of the GWAS requests issued by users to the Glow/Spark Cluster. The third operation allows the decryption of the analysis results when returned to users, which are encrypted to avoid revealing sensitive information.

To perform a GWAS, users must specify a task script file where the analysis steps and all the required parameters are defined. This file contains sensitive information about the analysis task that cannot be

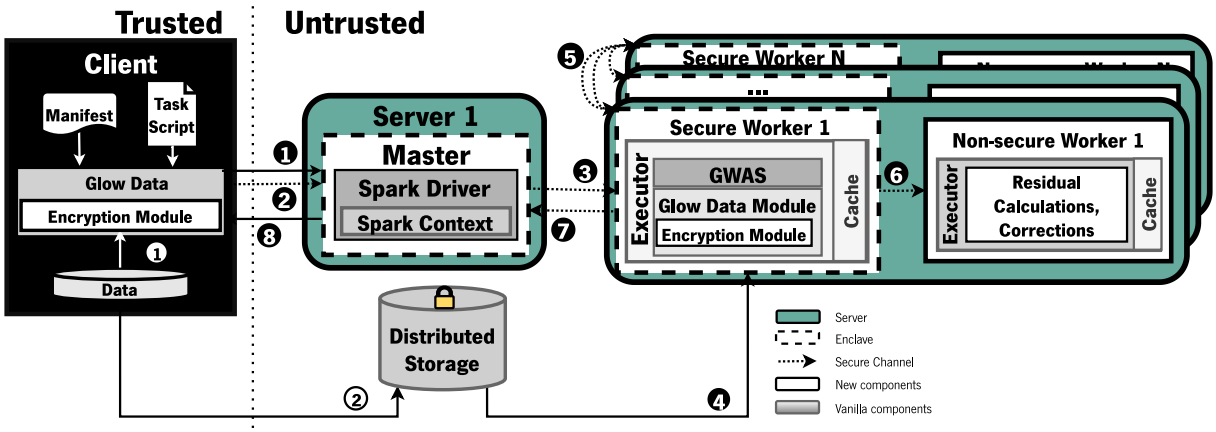


Figure 4.3: A schematic representation of Gyosa architecture. The workflow is as follows. A Biocenter encrypts and sends its data to distributed data storage (①-②). The Biocenter sends the GWAS task script and the *Manifest* file to the master node (❶-❷). The master node then distributes the tasks to the worker nodes (❸). The workers fetch the data (❹) and process the task script between the secure and non-secure workers (❺-❻) and send the results back to the master node (❼), which merges the results and sends them back to the client (❽).

leaked or tampered by malicious adversaries. Thus, the encryption module encrypts the task script, which can be sent via unprotected network channels.

The *manifest* is a predefined file that contains the libraries to run the Glow pipeline and the path for the dataset. To ensure data security and integrity when exchanging this file, a secure channel between the client and the Glow/Spark Cluster is created at the analysis bootstrap phase. This secure channel is also used to transmit the user's encryption key, which is then used by SGX enclaves.

4.3.1.2 Cluster Module

The cluster follows Spark's workflow on the untrusted site, with a master and N workers running on distinct servers. The master is deployed inside an SGX enclave at the untrusted server since the Spark Driver and Spark Context modules require reading plaintext information from the task script to distribute the processing tasks to the workers.

We follow the previous approach (§3.3) and deploy a secure and non-secure worker at each of the remaining cluster servers. The secure worker runs inside an SGX enclave and handles all the computation over sensitive data, while the non-secure worker handles non-sensitive data and runs outside of SGX. The exchange of sensitive information between secure workers and the master and between secure workers is done via secure network channels (Figure 4.3).

4.3.1.3 Partitioned design

Since Soteria targets Machine Learning workloads, which significantly differ from GWAS, Gyosa must redefine how computation is partitioned across secure and non-secure workers.

In Gyosa, non-sensitive computations include the residual values (*e.g.*, matrix calculation of metadata or calculations over single genotype information) and the correction of statistical tests, which in Glow are based on the Firth's approximation algorithm [187]. In detail, this last step is performed as a score test, which compares the predicted values with the observed values to validate the resulting P-values. All the remaining operations (*e.g.*, read operations on top of the VCFs, dataframe transformations, and regression operations, among others) are done over sensitive data and performed in SGX enclaves at the secure workers. Note that all this sensitive information remains fully encrypted when transmitted and stored outside of secure workers to ensure its privacy.

In summary, Gyosa differs from Soteria by considering a different processing pipeline, namely a genomic pipeline for performing GWAS. This implies supporting a new framework (*i.e.*, Glow), including transparent encryption of a new type of dataset file format (*i.e.*, VCF files), and redefining the sensitive analysis steps that must be performed inside secure enclaves. Next, we detail the flow of operations in Gyosa.

4.3.2 Workflow of Gyosa

In Gyosa, the client resorts to Gyosa's encryption module to encrypt the VCF files at the trusted premises (Figure 4.3-①). Then, encrypted data is sent to a distributed data storage shared by various servers on the untrusted infrastructure (Figure 4.3-②). Similarly, the client specifies the studies it wants to run as task scripts and encrypts these before sending them to the untrusted infrastructure (Figure 4.3-❶). The default usage of Gyosa assumes a bootstrapping phase between the client and master in which a secure channel is established and used to share the *manifest* file and the client's key, which was used to encrypt the VCF files and tasks' scripts. This key is also used to decrypt the final results (Figure 4.3-❷).

Following a master–worker architecture, the master, running inside an SGX-enabled server, receives the task the client wants to perform and the path (within the *manifest* file) for the encrypted dataset. The former is sent encrypted through an insecure channel, while the latter is forwarded inside the previously established secure channel. After decrypting the task script inside the secure enclave, the master forwards specific sub-tasks to each secure worker through secure channels established between their enclaves (Figure 4.3-❸). With these sub-tasks, secure workers can fetch the required data from the distributed storage backend and perform the computation. Since data is encrypted at the storage backend, it must be fetched and decrypted at each secure worker enclave to be processed in plaintext (i.e., inside an SGX enclave) (Figure 4.3-❹).

Following a distributed computation paradigm, workers broadcast intermediate results between them (Figure 4.3-❺). In addition, following the partitioning of computation, secure workers broadcast metadata to non-secure workers. Again, after performing their computation, the non-secure workers broadcast the information back to the secure workers (Figure 4.3-❻). Sensitive information shared across secure workers is done through secure channels established between their enclaves. In the final worker-related stage, a consensus regarding the result is reached and sent to the master's enclave through a secure channel

(Figure 4.3-❼). Final results are aggregated and encrypted by the master, with the client's key, and sent to the client for transparent decryption at the trusted premises (Figure 4.3-❽).

4.3.3 Security Analysis

Gyosa combines different mechanisms to safeguard users from attacks, see §4.2.1 for more details on these attacks. It provides transparent authenticated encryption, which protects sensitive data from being disclosed to unwanted parties and ensures anti-tampering properties for clients' data stored in untrusted infrastructures. This feature protects users from poisoning attacks by limiting access to the plaintext data and not allowing the addition of poisoned data. Membership inference and re-identification attacks are subject to an attacker's previous knowledge of the genomic data. By ensuring that private data, while at rest and in transit, is always encrypted and that any sensitive computation is performed on secure SGX enclaves, Gyosa avoids disclosing such knowledge to attackers.

Partitioning the computation across the enclaves and non-secure environments enables performance improvements but increases the attack surface. However, previous work shows that genomic data cannot be inferred from the information leaked from sharing metadata and statistical information [205]. Given this assumption, and by not changing the main security protocol specified by Soteria, Gyosa can keep the information leakage contained to avoid the success of the aforementioned attacks.

4.3.4 Implementation

Gyosa's prototype is built on top of Apache Spark 3.2.1 and implemented using both Java, Scala and Python. Spark's data loading library was extended to include Gyosa's transparent encryption module for VCFs. Similar to Soteria's encryption module, this module also resorts to AES-GCM-128 authenticated encryption cipher mode, to offer data privacy and integrity guarantees.

The prototype supports both Linear and Logistic Regression for GWAS with the X^2 being defined inside the Logistic Regression algorithm. With this, the implementation of *Glow* was decoupled into two sub-libraries, one with the statistical processing (to be executed outside SGX) and another with the remaining computational logic of the algorithms (to be executed inside SGX). This separation allows the non-sensitive operations to be performed outside SGX, while the sensitive operations are performed inside SGX enclaves.

Also, by following Soteria, in Gyosa, Gramine 1.0 was used for the overall management of Intel SGX enclaves' life cycle, for specifying the computation (*i.e.*, internal Spark and Glow library) to run at each enclave, and for establishing secure channels (*i.e.*, with the TLS-PSK protocol) between the enclaves at the master and worker nodes [289]. Gyosa's *manifest* file was also provided by Gramine.

4.4 Evaluation

Gyosa was evaluated to understand the impact of adding security guarantees on top of the application stack composed by Apache Spark and Glow. Two main questions were asked in this evaluation:

1. How does the execution time of Gyosa compares with the non-secure vanilla setup?

2. How does Gyosa behave when increasing the size of the workload and the number of workers?

4.4.1 Methodology

4.4.1.1 Dataset

For the benchmark, we used the data generated by the *Genome in a Bottle* Consortium [327], with genomes sequenced as part of the Human Genome Project. Data from the Ashkenazim Trio family (father, mother, and son) were used in our experiments. The dataset does not contain phenotype data, so additional information was simulated with the PhenotypeSimulator [193]. The algorithms were tested for different workloads by scaling the original dataset by several factors to reach sizes of 1, 4, 16, and 32 GB. Moreover, for the scalability tests, we generated a synthetic dataset consisting of 80,000 VCF files with $1 * 10^6$ unique SNPs, each VCF file representing one individual. We define the workload size as 20k, 40k, 60k, and 80k individuals.

4.4.1.2 Experimental Setup

Tests were performed in a cluster of 4 servers with OS Ubuntu 18.04.4 LTS and Linux kernel 4.15.0. Each machine has a 10Gbps Ethernet card connected to a dedicated local network and 16 GB of memory. Gyosa uses Apache Spark 3.2.1 and was deployed with version 2.6 of the Intel SGX Linux SDK with driver 1.8, with 4 GB of memory. The client and Spark master run on one server, while Spark workers are deployed on the remaining servers.

4.4.2 Secure GWAS

To assess the impact of Gyosa's security mechanisms on the execution time of the algorithms, we compare the results with the ones for the baseline setup.

Figure 4.4 shows the Linear Regression and Logistic Regression algorithm results. In detail, in Figure 4.4a, for a workload of 1 GB, the runtime overhead of the linear regression algorithm is around 4.5x. For 32 GB, the overhead of Gyosa reaches the maximum for the performed tests, with 10x compared to the baseline setup. The comparison of probability values (*p-values*) between the two approaches shows negligible differences (see Figure 4.4b). The slightly different values observed may result from the final approximation since it deals with small values and reverts them to $-log_{10}(p - value)$. Similarly, for the

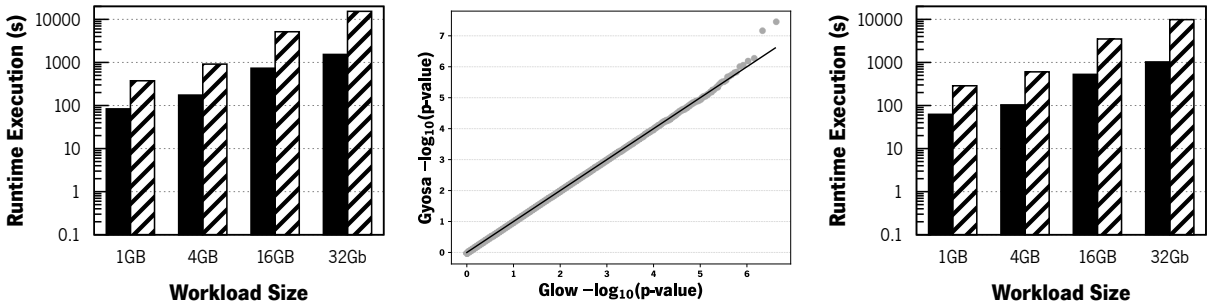


Figure 4.4: The impact of Gyosa on variable workloads (1GB, 4GB, 16GB, and 32GB) with the Linear Regression and Logistic Regression algorithm compared to baseline. *a)* Execution time of Linear Regression in logarithmic scale; *b)* Comparison of p-values between approaches for Linear Regression; *c)* Execution time of Logistic Regression in logarithmic scale. The legend is as follow: ■ Vanilla Glow; ▨ Gyosa.

Logistic Regression algorithm and workload size of 1 GB and 32 GB, Gyosa shows a runtime overhead of 4x and 9.5x (see Figure 4.4c).

For the Logistic Regression algorithm and workload size of 1 GB and 32 GB, Gyosa shows a runtime overhead of 4x and 9.5x.

Gyosa can be used in a cluster setup with multiple servers, each including one secure worker and one non-secure worker. Figure 4.5a shows the results of the overhead imposed for the X^2 frequency test. Gyosa leverages the scalability offered by Apache Spark and Glow, as shown by the experiments with up to 3 servers. We verify a linear decrease in the runtime execution when increasing the number of servers for both Gyosa and baseline setups. Namely, when comparing the execution time obtained by running this experiment over 80,000 VCF files with one and three servers, the runtime execution decreases up to 2.7X or up to 2.4 hours. Regarding the security guarantees, the runtime overhead ranges from 1.3x to 3x for a workload of 40k individuals with three servers.

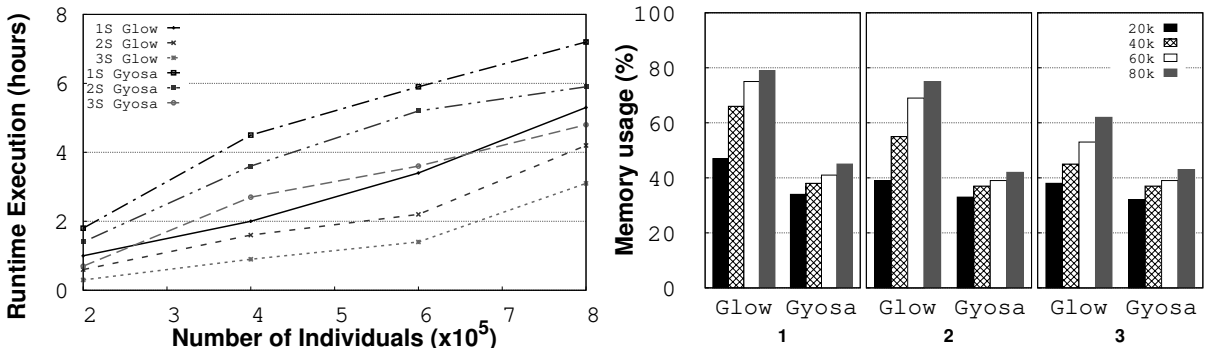


Figure 4.5: Runtime execution (hours) and Memory usage (%) of the X^2 statistic test for 1 to 3 worker nodes and 20k to 80k number of individuals from the synthetic dataset. The setup is represented by M: Master and W: Worker (*e.g.*, 1M+1W represents one Master node with one Worker node). a) Runtime Execution for X^2 statistical test in hours; b) Memory Usage in percentage (%) for X^2 statistical test.

4.4.3 Analysis

The discussion focuses on the performance and scalability of Gyosa.

Performance and Scalability Analysis. By increasing the number of servers, one can distribute and parallelize GWAS computations and substantially reduce the runtime and memory usage, as observed for the X^2 test in Figure 4.5a and 4.5b. Figure 4.5b shows that despite Glow being a memory-intensive solution [99], increasing the number of servers leads to a decrease in the mean memory usage.

Compared with other SGX-based solutions, namely [55], Gyosa shows comparable runtime overhead. Notably, Gyosa distinguishes from all these state-of-the-art solutions [74, 144] by allowing distributed computation across several servers in a cloud environment.

High-end servers used by state-of-the-art solutions (*e.g.*, a configuration with > 40 cores, > 2.0TB of physical memory, and 10TB of disk space [74]) are not widely available and require substantial resources for their setup and maintenance. A cost-efficient alternative is to use cloud environments that allow the distribution of computation by relying on several servers, reducing the execution time of genomic analysis. In Google Cloud [60] with 391.35€ per month, one could opt for *i)* one server with 16 cores and 64 GiB of memory or *ii)* four servers with four cores each and 16 GB of memory. While solution *i)* provides 730 h of computation, solution *ii)* provides a total of 2,920 monthly hours of computation [103].

Currently, the second generation of SGX includes a page cache with 128 MB. Data that does not fit on this cache must be swapped to/from an encrypted memory region, leading to a performance penalty [73, 120]. Also, in our setup, the amount of encrypted memory attributed to SGX is limited to 4GB in each server. For memory-intensive GWAS algorithms, this means additional swapping to/from disk [120, 188]. Note that this SGX technological limitation cannot be solved by simply upgrading the server's hardware but is addressed by Gyosa's distributed design. Namely, by distributing computation across multiple servers, our solution can leverage the aggregated page cache and encrypted memory sizes of multiple servers, which justifies the decreased runtime execution observed for Gyosa in Figure 4.5a.

Our benchmarks show that to deliver a practical privacy-preserving solution, the performance penalty in Gyosa is proportional to the data size, both for regression and classification tasks. Notably, the addition of security guarantees in Gyosa does not alter the results, and therefore, Gyosa presents itself as a viable option for privacy-preserving genomic analysis.

4.5 Related Work

This section is split into three stages. The first two stages present the related work on privacy-preserving genomic analysis that: *i* resorts to software-based cryptographic primitives (§4.5.1); and, *ii)* resorts to hardware-based cryptographic primitives (§4.5.2). Finally, the third stage (§4.5.2) presents a summary and taxonomy of the related work.

4.5.1 Software Approaches

To enable the execution of GWAS at untrusted third-party infrastructures while ensuring the privacy of data, several approaches have been proposed based on standard cryptographic primitives such as HE, SMPC and DP [142, 176]. The implementation of HE-based solutions has shown that for GWAS, this technique still imposes a high-performance execution time penalty and can only support a limited number of algorithmic operations [132]. On the other side, SMPC enables secure data storage and computation even across multiple entities that do not trust each other. However, since this technique resorts to distributed protocols that require several rounds of network communication, it adds a significant delay to the execution of GWAS [325]. In contrast, DP provides a less penalizing solution in terms of performance overhead. However, it lacks robustness since adding noise to the data and computation can compromise the accuracy and undermine the results. Furthermore, DP cannot effectively manage high-dimensional data or cope with growing data volumes [92]. Thus, the presented solutions have limitations that hinder their use.

4.5.2 Hardware Approaches

More recently, TEE's have shown a huge potential as alternative solutions to ensure privacy-preserving computation and storage in untrusted infrastructures for genomic data [74, 144]. While this is a promising technology for running GWAS securely, its application has been typically limited to a single-server mode [57, 256]. By taking advantage of distributed infrastructures, it is possible to enhance the speed and scalability (*i.e.*, the amount of data being analyzed) of GWAS studies. However, as highlighted in this chapter, developing a distributed solution for privacy-preserving GWAS requires a fundamentally new design that differs from previous methodologies.

Summary

In Table 4.1 we present a taxonomy of related work systems that address the problem of privacy-preserving the genomic pipeline. In brief, this table divides these systems into five categories: *i)* by cryptographic primitives; *ii)* by distribution; *iii)* by stages of the genomic pipeline; *iv)* by algorithms; and, *v)* availability. For the first category, we consider the most relevant primitives for genomic data analysis, namely, HE, SMPC, DP, and TEEs. The distribution topology of each system is decomposed into distributed, centralized, and collaborative and considers the outsourcing of computation to third-party infrastructure when referring to the *distributed* and *centralized* sub-categories. The *distributed* sub-category refers to solutions that distribute the computation across multiple servers to parallelize the computation. The *centralized* sub-category refers to solutions that perform the computation in a single server. Conversely, the *collaborative* sub-category refers to solutions that require the collaboration of multiple entities to perform the computation, a setting commonly used when resorting to SMPC or federated environments. For the third

category, we consider the stages of the genomic pipeline where the solution is applied, namely, data analysis and public release. Within *public release*, we also consider the aggregation of the final results from an analysis in a collaborative setup [223]. Finally, for the fourth category, we consider the algorithms that are supported by the solution, namely, GWAS, read mappings, and other tests (*e.g.*, genome imputation).

Table 4.1: Taxonomy of Related Work systems.

Systems	Crypt. Primitives	Distribution	Stage	Algorithms	Avail.
PrivGenDB [132]	○	C, H	A, PR	☼	○
Kim et al. [142]	○	C, H	A	●, ◑	●
Karimi et al. [141]	○	C	A	☼, ◑	○
cGLMM [325]	◐	H	A	●, ◑	●
Drynx [92]	◑, ○	H	A	☼	○
SCOTCH [57]	●, ○	C	A	☼	○
ObliVGen [18]	●,◐	C, H	A	●,○, ◑	○
COLLAGEN [163]	○, ◑, ◐	H	A	●,◑	●
SAFETY [256]	●,○	C, H	A	◑, ☼	○
SkSES [144]	●	C	A	●,○	●
DyPS, *I-GWAS* [223, 224]	●	C,H	A,PR	●,◑	○
MaskAI [152]	●	C	A	◐	○
HySec-Flow [303]	●	D	A	◐	◐
SMac [74]	●	C, H	A	◑	●
Gyosa (this work)	●	D	A	●,○	●

Crypt. Primitives	Distribution	Stage	Algorithms	Availability
● - TEE	*C* - Centralized	*A* - Analysis	● - GWAS	● - Yes
◐ - SMPC	*D* - Distributed	*PR* - Public Release	○ - x^2 tests	○ - No
◑ - DP	*H* - Collaborative		◐ - Read Mappings	◐ - Partially
○ - HE			◑ - Other Tests	
			☼ - Queries	

Most of these solutions are tailored for the collaboration between entities in a federated way, meaning that the data is distributed across multiple entities, and the computation is performed in a collaborative manner [74, 92, 132, 325]. However, these approaches are not suitable for the cloud environment, and Gyosa differs from them as it intends to parallelize the computation among different servers to decrease the execution time. Additionally, another vital aspect relies on the different types of computation to be performed. Privacy-preserving solutions have been applied to a wide arrange of tasks, typically on the data analysis stage, such as read mappings [152, 303], statistical tests [18, 74, 141, 142, 163, 223, 256, 325], queries [57, 92, 132, 141, 256] and even GWAS [142, 144, 224]. Nonetheless, the only distributed solution, *HySec-Flow* [303], focuses on the problem of read mappings. While using a specific library to do this task, the solution does not allow the seamless integration of new and more complex tasks.

Differently from previous solutions and to the best of our knowledge, Gyosa is the first solution to support the execution of GWAS in a distributed environment while ensuring the privacy of data. Finally, by relying on *Glow*, Gyosa stands out from other solutions by enabling the addition of new tasks (*e.g.,* statistical tests, genomic imputation, and querying) in the genomic pipeline, making it easier to extend the secure analysis pipeline.

4.6 Summary

Gyosa offers the first end-to-end privacy-preserving genomic data analytics solution built on Apache Spark and Glow. Distributing GWAS computation across multiple untrusted servers allows researchers to study larger amounts of sensitive genomic data efficiently.

Further, by following a computation partitioning scheme tailored for GWAS, Gyosa decreases the amount of data transferred and processed at secure enclaves, which allows for boosting the algorithms' performance while not compromising security or affecting the quality of the analysis outcomes. The evaluation of Gyosa shows the expected trade-off between runtime computation and the level of privacy-preserving guarantees. In detail, Gyosa shows a runtime execution overhead ranging from $2.5X$ for X^2 statistic tests to $10X$ on regression-based algorithms. Nonetheless, the evaluation also allows us to observe that it is possible to effectively reduce the computation runtime overhead by distributing the computation across multiple nodes (*i.e.,* the runtime execution decreases up to 2.7x when increasing the number of worker nodes from 1 to 3) while still offering privacy guarantees for sensitive data.

Soteria and Gyosa are dependent on the available hardware and require substantial resources for their setup and maintenance. Although cloud services already provide SGX-based servers, when the information cannot be outsourced to third-party entities, and these entities still want to collaborate, they must resort to different solutions. For instance, in IOT scenarios, where the users or sensors generate the data, the data cannot be outsourced to third-party entities. These scenarios are common in smart cities, urban mobility, smart homes, smart health, and mobile environments. Overall, this data should not leave its premises and should be processed locally. In this case, FL-alternatives that privacy-preserve user's data are a viable solution.

In the next chapter, we present TAPUS, a privacy-preserving FL use case that allows users to train ML models locally while preserving their privacy to find mobility patterns and more efficient transportation alternatives.

5

Protecting User's Mobility Patterns with Differential Privacy

According to the World Economic Forum (WEF) [2], *"mobility is a fundamental human need and an essential enabler of prosperity, but the current mobility paradigm is not sustainable"*. Quoting WEF, car travel causes millions of deaths every year, and a significant amount of Greenhouse Gas (GHG) emissions and traffic congestion causing heavy financial loss. The European Commission also acknowledges that transportation is the leading cause of air pollution in cities [61]. With this, cities worldwide have agreed on ambitious goals towards 2030 regarding GHG emissions and carbon neutrality. Given this agenda, the global mobility system is in its early stage of massive transformation. Namely, policymakers are seeking ways to foster smarter, cleaner, and more inclusive mobility. For this to be possible, we argue that one must consider three main challenges.

Carbon footprint awareness. Citizens are not aware of their carbon footprint when using different transportation modes (*e.g.,* walking, bicycle, motorcycle, car, bus). Mobility patterns should be collected and leveraged to provide citizens with information about their impact on the environment.

Sustainable mobility. Cities cannot implement greener strategies for active and shared mobility due to public transportation's low attractiveness. Citizens should be provided with personalized recommendations and incentives for using this type of transport.

Data privacy. Citizens should be aware of how their data is collected and used and, more importantly, that their personal information (*e.g.*, location) is kept private. Without this, the adoption of applications and services that target the two previous challenges will be limited.

These challenges are addressed by the methodology created under the FranchetAI project [88], which promotes personalized and sustainable mobility behavior while preserving data privacy and increasing user trustworthiness. FranchetAI's methodology is built on top of the following pillars: *(i)* AI and GHG estimation models to detect the type of transportation being used by a given citizen, along with its corresponding carbon footprint; *(ii)* state-of-the-art mechanisms that safeguard data collected from user's mobile devices by not sharing private/sensitive data with any external service (*e.g.*, cloud provider); *(iii)* compliance with European best practices in usability, accessibility, and explainable AI to clarify in an understandable way

how users' data is being processed; and, finally, *(iv)* building up on the experience of gamification and habit changing to promote incentives (*e.g.,* rewards, vouchers, among others) to encourage the community to opt for sustainable mobility choices, as well as to create more awareness about sustainability among citizens. As output, users should be informed about their mobility choices' carbon footprint through *carbon digest reports* (daily and weekly) via a mobile application.

Leveraging users' data to provide personalized recommendations is a challenging task. Previous chapters show that personal data is commonly susceptible to privacy breaches and should be tackled likewise. In this sense, preserving sensitive data for collaborative environments with limited hardware resources, such as mobile devices, is challenging. Nonetheless, this is a vital environment for deploying a FL system, as it allows the training of models with data that never leaves the user's premises while leaving the need to resort to specific hardware.

Similar to the ML pipeline, the FL pipeline is also susceptible to privacy breaches. In detail, this pipeline is primarily susceptible to poisoning and membership inference attacks [178, 199, 299]. These attacks tend to manipulate the training process or infer the membership of a specific user in the training dataset. Protecting users' data in these settings has been broadly addressed in several works in the literature [299]. However, most of these works focus on the privacy of the model's parameters, leaving the user's data unprotected.

In specific, transportation mode detection solutions typically require that data is submitted to a central entity for training [140]. However, this approach raises concerns as GPS track data contains sensitive personal information such as location and travel duration. Moreover, heightened privacy awareness and tighter governmental oversight make it challenging for organizations to access raw GPS data directly [140]. Consequently, some researchers have shifted their focus towards privacy-preserving measures, devising secure models for travel mode identification in a FL setting [46, 311]. As such, these works leverage the concept of Privacy-Preserving Federated Learning (PPFL). Nonetheless, these works still lack a comprehensive approach to protect users' data in the context of FL and an evaluation of adding other mechanisms such as DP to the FL workflow [46, 311].

With this in mind, a PPFL solution to tackle the challenges of the FranchetAI project should follow the challenges proposed in Section 2.4.2, namely:

- **C1: Performance.** Many privacy-preserving techniques introduce additional computational overhead, increasing the runtime performance. This can impact the scalability and efficiency of the ML process. In the context of FL, this is particularly important, as the training process is distributed across several clients.

- **C5: Scalability.** The collaboration between entities has distinctive security requirements. In this setting, it is intended to leverage data from different entities simultaneously to train a model. Ensuring this setting can scale effectively to handle large datasets while privacy-preserving data is an ongoing challenge.

- **C6: Hardware Requirements:.** Some privacy-preserving techniques demand specific hardware (*e.g.,* TEEs), which hinders the usage of these solutions for organizations with limited hardware or budget. IOT and mobile devices usually have limited computational resources or do not possess TEE-enabled hardware for sustaining hardware-specific solutions, which is typical for FL environments.

Contributions. In this chapter, we propose TAPUS, a system to protect users' data using DP in the context of FL. It is designed to train models based on users' mobility data to detect the type of transportation a given citizen uses and its corresponding carbon footprint. TAPUS can be split into two main components. First, the AI models, with which one can infer the type of transport for each user after training with its local data. The second component leverages the first model's output to estimate each user's carbon footprint. Our proposal is built on top of the Flower framework [25], which allows deploying federated training both in mobile settings and across data silos. This is an essential feature if cities provide access to other information, such as road topology and traffic congestion, that can be combined with users' data to improve the final models. To provide privacy-preserving guarantees, TAPUS resorts to DP mechanisms to protect users' data. This is done by adding noise to the gradients of the model's parameters before sending them to the server or adding noise directly to the collected data before performing any computation. This is a vital feature of the first component, as it ensures that the user's data is not leaked during the training process.

The main goal of TAPUS is to allow the proposal of a PPFL solution to be leveraged in the FranchetAI project for transportation mode classification and detection.

Evaluation. To implement and evaluate TAPUS we conducted several experiments. The preliminary results comparing Decision Trees, Random Forest, Logistic Regression, and XGBoost algorithms show the impact of different features on predicting the user's mode of transport. Namely, we show that obtaining results with over 80% accuracy is possible when considering the distance and mean velocity of users' trajectories. Such results form the basis for training with more complex algorithms based on neural networks.

The second evaluation is based on the DP mechanisms used to protect users' data. By using `DP-FedAvg` and `DP-FedSGD` strategies (implemented by resorting to Flower [25]), one can introduce random noise on the users' data, specifically on the gradients, mitigating the leakage of their personal information (*e.g.*, location). This is a similar goal to adding noise to the local data before training. However, we note that this latter approach has a greater impact on the model's accuracy than the first two DP-based strategies.

The results show the feasibility of using both DP-FL algorithms and local DP mechanisms to protect users' data distinctly. In sum, the results show a decrease of up to 25% of the accuracy when using Local DP with an ϵ of 0.5. When evaluating the scalability of TAPUS, we show that the system can handle several clients while preserving users' data privacy. In these experiments, we show that the system can handle up to 50 clients with up to 16% decrease in accuracy.

5.1 Key Concepts and Technologies

This Section overviews the key concepts and technologies leveraged in this Chapter. We start by defining a standard protocol for FL, we formalize DP and the algorithms implemented, and then detail the FranchetAI methodology, which provides the building blocks and the use case for this work.

5.1.1 Federated Learning

Following the enormous amounts of collected data from different sources, ML has become the *de facto* solution for analyzing and extracting insights from it. Nonetheless, new regulations (*e.g.*, GDPR) impose new approaches for analyzing data that may contain sensitive information.

FL has emerged as a new ML paradigm targeting Non-Independent and Identically Distributed (Non-IID) data [27, 161], typically generated on the edge, local servers, and mobile devices. Specifically, FL is a type of distributed ML in which models are trained with data from different users, but sensitive information never leaves each user's premises.

In this setting (see Figure 5.1 and 5.2), a centralized server has an initial trained model M_i and broadcasts M_i to every user (Fig. 5.1-❶). Typically, M_i is trained on previously collected, open-source, or private data, resulting in a collection of parameters and hyperparameters. On the user side, the device trains the model based on the user's data (Fig. 5.1-❷).

At each iteration, the centralized server asks N users for their new model parameters (Fig. 5.1-❸). Each user can define whether to participate or not in a given round, and similarly, the centralized server can decline the parameters broadcasted from the decentralized users (Fig. 5.1-❹). At the end of this cycle, the server calculates the average of all obtained parameters (Fig. 5.1-❺) and broadcasts new parameters to every user, which updates locally its own model (Fig. 5.1-❻) [27, 161].

This protocol follows the flow of operations depicted in Figure 5.2. The mobile device joins the FL protocol, and the centralized server replies with the initial or the last model (*i.e.,* if the protocol has been initiated before, the server sends the last model obtained; otherwise, it sends the initial one). The mobile device trains the model with its local data and sends the updated version to the centralized server. The server aggregates the models and sends the updated one to the mobile device. This process is repeated until the model converges or a predefined number of iterations is reached.

Although these protocols do not require the centralized server to have access to the users' data, they still require the server to have access to the users' model parameters. This is a significant limitation, as the server can still infer sensitive information from the users' data [302]. To address this limitation, FL can be combined with DP to provide stronger privacy guarantees.

5.1.2 Differential Privacy

To further strengthen data privacy guarantees, FL may resort to MPC or DP as discussed in Section 5.6.3. In this work, we focus on the latter. The intuition behind DP relies on the fact that changing any individual

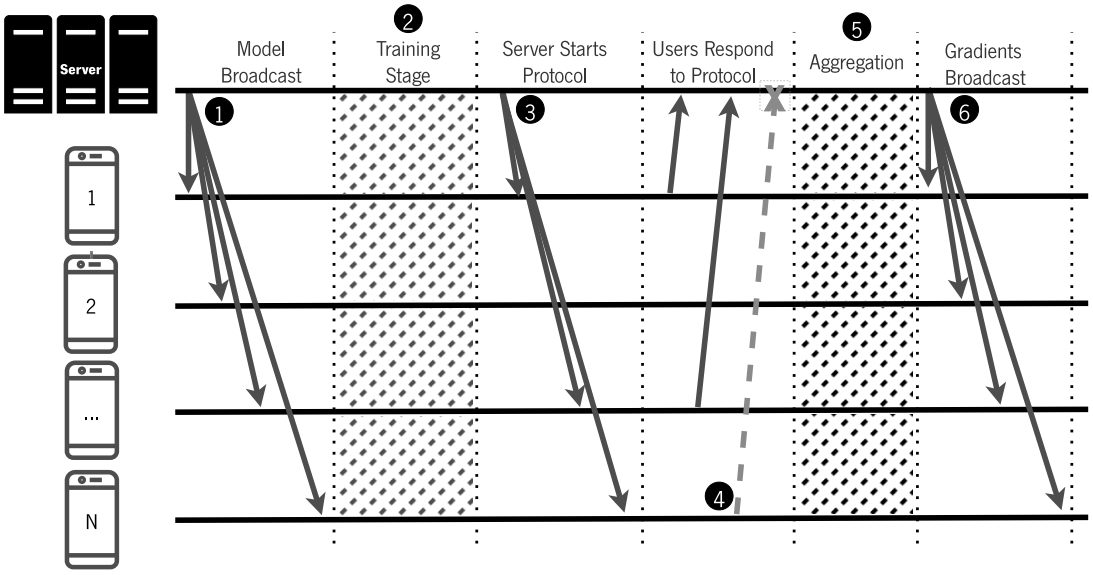


Figure 5.1: An overview of the Federated Learning protocol. The centralized server is responsible for broadcasting the initial model to the users, starting the protocol, aggregating the models, and broadcasting the updated model to the users (❶,❸,❺-❻). The devices are responsible for training the model with their local data and broadcasting the updated model to the centralized server (❷,❹).

point on the input data will not change the query result but will limit the attacker's capabilities for deducing private information from data with high confidence. To this end, DP anonymizes private information by introducing randomness or noise in the original data [78].

Formally, a randomized algorithm $\mathcal{M}$ is $(\epsilon + \delta)$-differentially private if for all datasets D_1 and D_2 that differ in a single element, and for all subsets of the output space $\mathcal{O}$, the following holds:

$$\mathbb{P}[\mathcal{M}(D_1) \in \mathcal{O}] \leq e^{\epsilon} \times \mathbb{P}[\mathcal{M}(D_2) + \delta \in \mathcal{O}] \tag{5.1}$$

Adding noise to sensitive training data increases privacy-preserving guarantees and the models' error rate. In this sense, the trade-off between privacy and accuracy is crucial when using DP [190]. To reduce the error rate, one can apply optimizations such as adaptive clipping [15], which continuously adapts the amount of noise introduced in each individual training sample. Nonetheless, this work does not consider such optimizations, focusing on standard DP algorithms.

In an FL setting, DP can be applied directly to users' data or the parameters being broadcast across users and the centralized server. In both cases, the goal is to guarantee that no private information is leaked when the trained model is queried by third-party entities (*e.g.,* cities, municipalities).

Two different strategies can be used to apply DP in FL: *(i)* adding noise to the users' data before training the model [62], *(ii)* adding noise to the parameters being broadcasted to the centralized server [190, 206]. The first strategy is more straightforward to implement but may lead to a higher error rate (*i.e.,* lower accuracy results and longer times to convergence). The second strategy is more complex to implement but allows for a more fine-grained control of the noise introduced in the training process and a lower accuracy impact.

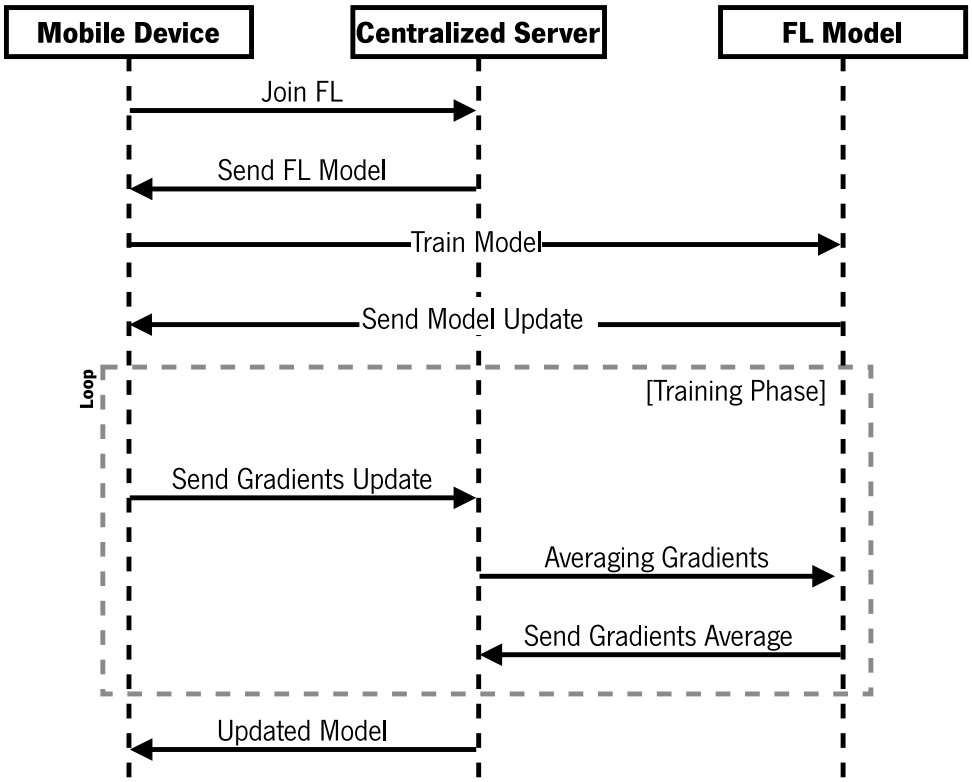


Figure 5.2: An example of the flow of operations in a Federated Learning protocol. The mobile device joins the FL protocol, and the centralized server replies with the model. The mobile device trains the model and sends the updated version to the centralized server. The server aggregates the models and sends the updated one to the mobile device. This process is repeated until the model converges or a predefined number of iterations is reached.

Focusing on the noise added to the gradients, the *DP-FedAvg* strategy [190] is a popular choice. It introduces noise to the aggregated gradients, ensuring that the model does not remember the data it was trained with. This strategy is based on the *Federated Averaging* algorithm, which calculates the average of the aggregated gradients of the model's parameters and broadcasts them to the centralized server. The noise is added to the gradients before the averaging process, ensuring that the model's parameters are not leaked to third-party entities. In brief, in DP-FedAvg (see Algorithm 5.1), each client computes its gradient based on its local data, and then these gradients are aggregated using a secure averaging mechanism to update the global model parameters

Additionally, the *DP-FedSGD* algorithm was also proposed with the same intuition. In DP-FedSGD (see Algorithm 5.2) each client needs to use the stochastic gradient descent (SGD) directly to update its local model with its noisy-local gradients.

As stated by Noble, Bellet, and Dieuleveut, the main difference regarding the usage of DP between both algorithms relies on the fact that DP-FedAvg typically adds noise to the aggregated gradients and, in contrast, DP-FedSGD adds noise directly to the model parameters during updates (this is highlighted in orange in both algorithms 5.1 and 5.2). As the updates can be randomly defined, and a client may not be

Algorithm 5.1: Pseudocode of DP-FedAVG.

Input: Local datasets $D_1, D_2, ..., D_n$ on n devices, learning rate η, target privacy budget ε, noise scale σ
Initialization: Initialize global model parameters θ randomly
for each communication round $t = 1, 2, ...$ **do**
  Sample noise from Laplace or Gaussian distribution: $\mathcal{N}(0, \frac{2q}{n\varepsilon})$
  Broadcast θ to all devices
  for each device $i = 1, 2, ..., n$ **do**
    Compute local gradient: $\nabla f_i(\theta)$ on dataset D_i
    Add noise to the local gradient: $\nabla f_i'(\theta) \leftarrow \nabla f_i(\theta) + \mathcal{N}(0, \frac{2q}{n\varepsilon})$
  end for
  Aggregate noisy gradients: $\hat{g} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \nabla f_i'(\theta)$
  Update global model: $\theta \leftarrow \theta - \eta \cdot \hat{g}$
end for
Output: Trained global model parameters θ

Algorithm 5.2: Pseudocode of DP-FedSGD.

Input: Local datasets $D_1, D_2, ..., D_n$ on n devices, learning rate η, target privacy budget ε, noise scale σ
Initialization: Initialize global model parameters θ randomly
for each communication round $t = 1, 2, ...$ **do**
  for each device $i = 1, 2, ..., n$ **do**
    Sample noise from Laplace or Gaussian distribution: $\mathcal{N}(0, \frac{2q}{n\varepsilon})$
    Broadcast θ to device i
    Compute local gradient: $\nabla f_i(\theta)$ on dataset D_i
    Add noise to the local gradient: $\nabla f_i'(\theta) \leftarrow \nabla f_i(\theta) + \mathcal{N}(0, \frac{2q}{n\varepsilon})$
    Update local model: $\theta_i \leftarrow \theta - \eta \cdot \nabla f_i'(\theta)$
  end for
  Aggregate updated models from all devices: $\theta \leftarrow \frac{1}{n} \sum_{i=1}^{n} \theta_i$
end for
Output: Trained global model parameters θ

selected to participate in a given round, the noise added to the parameters also depends on the algorithm. This randomness of choosing clients may lead to higher privacy-preserving guarantees from the DP-SGD algorithm as each iteration of the model is updated with the noisy gradients, even if they are not updated with gradients from other clients.

5.1.3 FranchetAI Methodology

FranchetAI provides a digital rewarding solution for people opting for sustainable mobility options (*e.g.*, public transportation, electric vehicles), ensuring transparency and trustworthiness between the user and the different stakeholders creating the incentives. Such a solution aims to educate citizens about their carbon

footprint while offering traveling alternatives and rewards for traveling more sustainably. The methodology proposed by FranchetAI and depicted in Figure 5.3 has the end goal of reducing CO_2 emissions. To achieve this, one can split the methodology into several steps. First, a centralized server allows deploying a web platform that processes and visualizes, for instance, traffic flow, road topologies, and public transit networks (Fig. 5.3-❶). Second, a mobile application is deployed on each user's mobile device (Fig. 5.3-❷). This application collects itineraries from GPS, accelerometer, and gyroscope (Fig. 5.3-❸). It also allows users to input additional info on vehicle usage and parameters (*e.g.,* year, class, fuel type) (Fig. 5.3-❹).

Information collected by the mobile app and the web platform is the input for a transportation model. This model, based on AI, infer users' transportation mode by using personal information from their trips and public information from public transit networks (Fig. 5.3-❺). The collected user data is also used to retrain the model to improve its accuracy. This is done in a privacy-preserving manner by resorting to federated learning and guaranteeing that private users' data remain on their premises (*i.e.,* their mobile devices).

The output of the previous model's inference is leveraged by an Life Cycle Assessment (LCA) model that estimates GHG emissions (Fig. 5.3-❻). These GHG emissions will then be leveraged for generating daily and weekly *carbon digest reports* for each user. All these stages are important for the final user to understand the impact of their mobility choices on the environment. Furthermore, the FranchetAI encompasses other stages, for clarity of presentation not here depicted, that aim to increase users' engagement with the application and with greener alternatives.

5.2 Threat Model and Attacks

The FL pipeline follows the same principles as the one in a common ML pipeline but with the added complexity of training models on non-IID data and distributed clients. This pipeline is susceptible to the same attacks as the common ML pipeline, such as poisoning attacks (*i.e.,* adversarial attacks) and membership inference attacks. In this section, we discuss the attacks on the FL pipeline and the threat model used in this work.

5.2.1 Attacks on the FL Pipeline

There are two main types of attacks on the FL pipeline: *poisoning attacks* and *membership inference attacks* [275]. Also, a new taxonomy for these attacks on FL has recently been proposed by Rodríguez-Barroso et al. [250].

Figure 5.4 shows the different attacks on the FL pipeline, namely on the centralized server or the client's side, which are further detailed in the following sections.

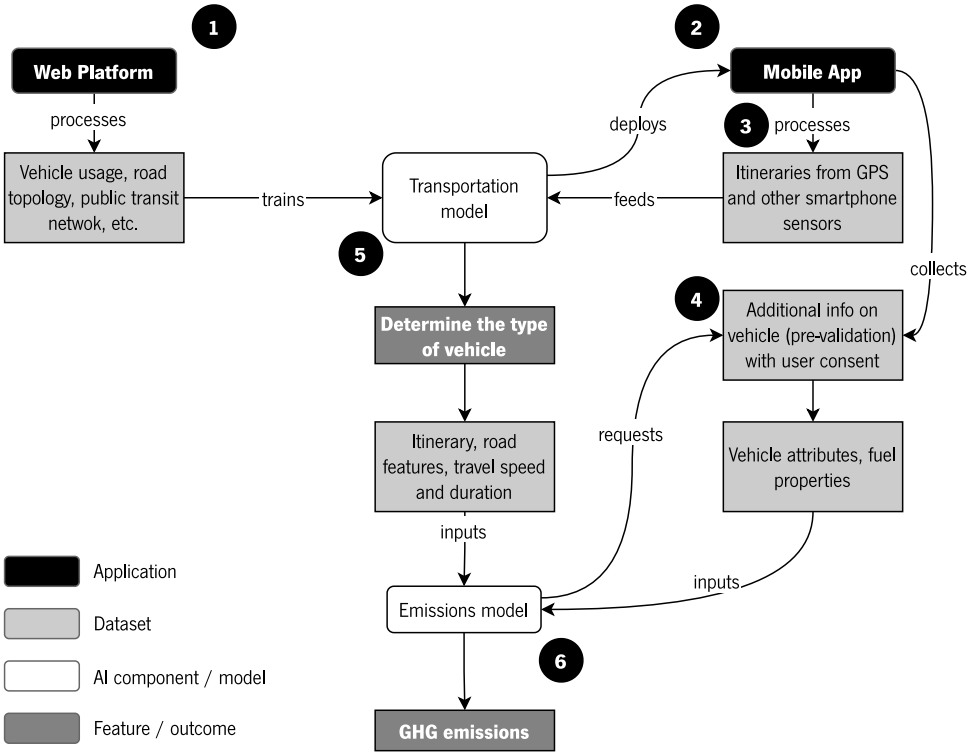


Figure 5.3: Pipeline of the FranchetAI's methodology. Two application services are deployed: a web platform and a mobile application (❶-❷). The mobile application deploys feeds the transportation model (❸). It also allows users to input additional info on vehicle usage and parameters (❹). The collected data is used to train a transportation mode model (❺). The output of the transportation mode model is leveraged by a LCA model that estimates GHG emissions (❻).

5.2.1.1 Poisoning Attacks

Poisoning attacks can be decomposed into two main subgroups: *data poisoning* (*i.e.,* during local data collection) and *model poisoning* (*i.e.,* during local model training) [178, 199, 299].

Data Poisoning. Data poisoning attacks are a common threat to the ML/FL pipeline. As defined in Section 2.3.1, these inject malicious data into the training stage. In the FL setting, this can be done by poisoning the data of several devices and manipulating the global model. With this, a malicious adversary may be able to control the training process and steer the model towards a specific direction (*i.e.,* this is defined as a targeted data poisoning attack) [178, 182, 199, 275, 283, 299].

Such attacks can be mitigated by identifying the malicious users based on their model updates, for instance, by removing outliers [93]. Nonetheless, this is still a challenging and open research problem.

Model Poisoning. Farther from the previous goal of controlling the training stage by injecting poisoned data and, ultimately, the global model, model poisoning attacks aim directly at the global model. This targeting is done by manipulating the local model's parameters and gradients, which are then broadcasted to the centralized server [22, 85, 178, 182, 199, 299].

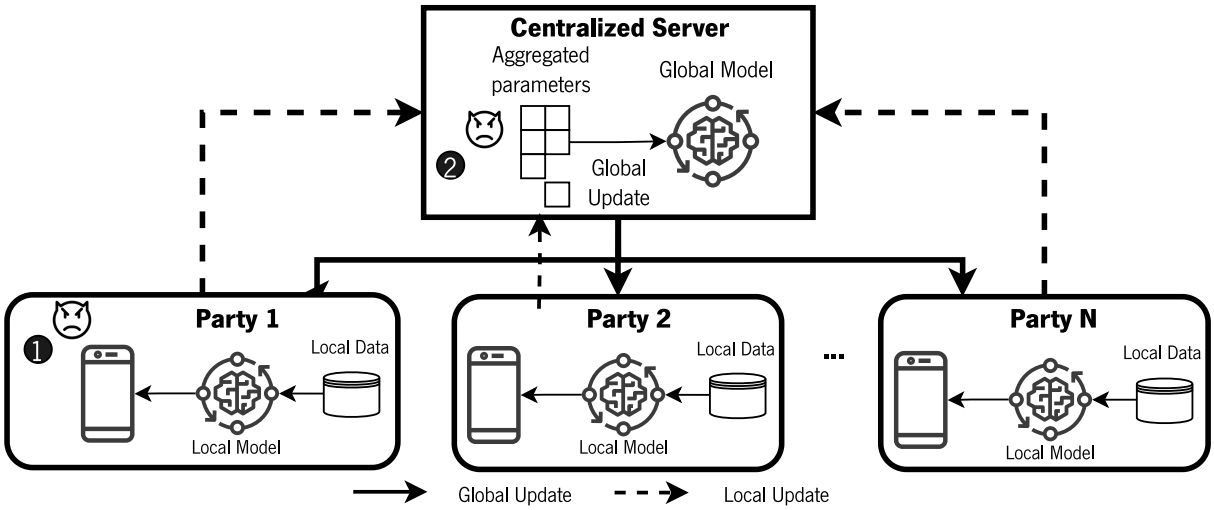


Figure 5.4: Attacks on the FL pipeline. The attacks can be performed on the centralized server (❷) or the client's side (❶).

5.2.1.2 Membership Inference

Similar to the *membership attacks* to the ML pipeline, these attacks, in the FL setting, intend to infer the membership of a specific user in the training dataset. This attack is done by querying the global model and analyzing the model's outputs. If the adversary can infer that a specific user's data was used to train the model, this user's privacy is compromised. In the context of FL, the adversary aims to deduce whether a given sample belongs to the confidential training data of a specific client or of any client [178, 182, 250].

5.2.2 Threat Model

In this work, we consider an honest-but-curious adversary. This adversary adheres to the FL protocol yet endeavors to extract confidential data about other clients from the exchanged information. This adversary is not malicious and does not aim to disrupt the training process.

We acknowledge that this threat model may not hold for all use cases, as it is challenging to ensure that all participants in the FL pipeline are honest. However, this is a common assumption in the literature, and we aim to provide a baseline for the privacy-preserving guarantees of the TAPUS system. In this sense, we highlight that works focusing on byzantine failures and malicious adversaries are still an open research problem and could be employed in TAPUS [178, 250].

5.3 TAPUS

Based on the underlying problems, TAPUS was built to provide a privacy-preserving solution for the FranchetAI mobile application. This system is designed to protect users' mobility patterns while providing accurate and explainable models for transportation and emissions.

5.3.1 Architecture and Flow

TAPUS consists of two main components: the AI component and the GHG component. The former is responsible for choosing and training the models in a privacy-preserving manner. The latter is responsible for estimating the GHG emissions of the users based on the models' predictions.

The architecture of TAPUS is depicted in Figure 5.5. It is composed of three main stages: *(i)* the training of the AI models; *(ii)* the inference of the AI model; and *(iii)* the estimation of the GHG emissions. The flow of TAPUS is as follows. First, an AI model and the GHG estimation function are deployed to each user's mobile device inside the mobile application. Then, the user's sensor-collected data is used to train the local model (Figure 5.5-❶). After the model is trained, the user's data is used to infer the transportation mode (Figure 5.5-❷). The AI model is trained in a privacy-preserving manner by resorting to FL and DP. Namely, the FL protocol employed allows the local training and then the broadcast to the centralized server of the trained gradients (Figure 5.5-❸). The server then aggregates the gradients (Figure 5.5-❹) and broadcasts the updated model parameters back to the users (Figure 5.5-❺). Finally, the GHG emissions are estimated based on the user's transportation mode inferred from the AI model, sensor-collected data (*i.e.,* distance, velocity), and other user input-based data (Figure 5.5-❻). This estimation is performed locally on the user's mobile device, and the report is provided to the user on the mobile application (Figure 5.5-❼).

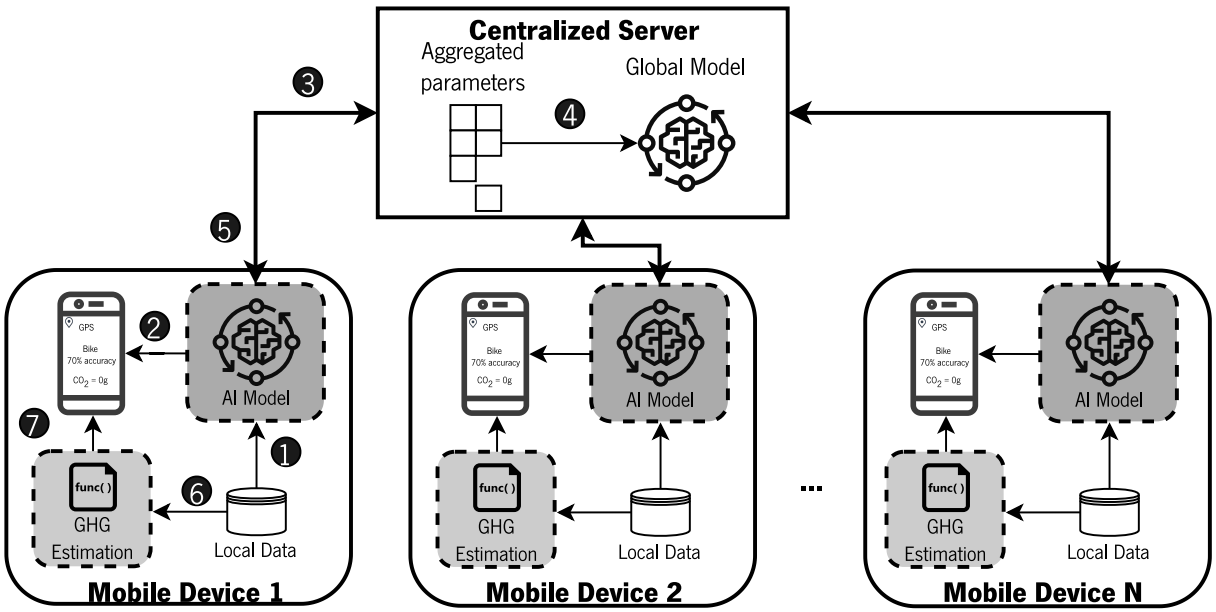


Figure 5.5: Architecture of TAPUS. The system comprises two main components, the AI model and the GHG estimation function. The AI model communicates with the centralized server (which aggregates the parameters ❹), reports to the mobile app and trains with local data (❶-❷, ❸, ❺). The GHG estimation function leverages the output of the AI model and reports to the mobile app (❻-❼).

5.3.1.1 AI Component

Our AI approach is decoupled into two main stages. First, we define the training dataset and how data is preprocessed. Then, we choose the ML models and DL architectures to train with the previous dataset while defining the FL framework.

As explained in Section 5.1 and depicted in Figure 5.6, TAPUS follows the same principles as the common FL protocol. Nonetheless, for increasing the privacy-preserving guarantees, we resort to DP-based algorithms, such as *DP-FedAVG* and *DP-FedSGD* and local DP (*i.e.,* random noise applied directly on the users' premises after the collection of data). Each DP-based algorithm is deployed independently (*i.e.,* each strategy is used separately).

By relying on a highly distributed and privacy-preserving approach, TAPUS can train models on top of sensitive data without sharing it with third-party entities. This is crucial for the FranchetAI mobile application, ensuring users' data is not leaked to third-party entities. Moreover, it allows both ML models and DL architectures to be trained over distributed data.

We focused on the three main crucial parts of the FL pipeline for introducing the noise (*i.e.,* input, gradients, and output). In brief, the input noise is added to the users' data before training the model (Figure 5.6-❶), while the gradient noise is added to the parameters being broadcasted to the centralized server (*i.e.,* the added noise to the gradients is performed by leveraging the *DP-FedSGD* algorithm, Figure 5.6-❷). The output noise is added to the model's parameters, and the model is updated with the aggregated noisy parameters (*i.e.,* by resorting to *DP-FedAVG*, Figure 5.6-❸).

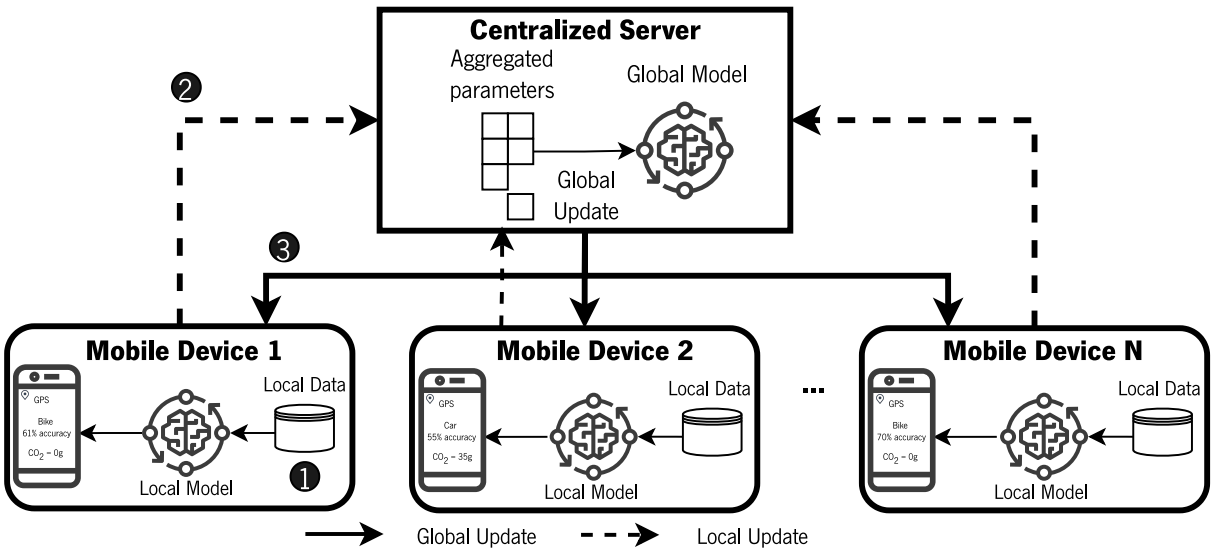


Figure 5.6: Architecture of TAPUS's AI component. The DP-based algorithms are used to introduce noise in the input and gradients(❶-❸).

5.3.1.2 GHG Component

The methodology adopted for estimating GHG and air pollution emissions is based on the *tank-to-wheel* stage of the Well-to-Wheel (WtW) LCA model, which only considers the operation of the vehicle, reducing the information that the user needs to provide to the model [113].

In more detail, emissions are estimated based on the CORe INventory AIR emissions (CORINAIR) system, *i.e.*, the method approved by the European Environment Agency (EEA). CORINAIR adheres to the Intergovernmental Panel for Climate Change (IPCC) guidelines [51] used globally by environmental protection agencies for national and regional evaluations.

According to the IPCC Guidelines for greenhouse gasses, a compiler from the CORINAIR system builds a decision tree to select the appropriate methodology with different complexities and data requirements. As input to the compiler, we apply the Tier 3 methodology from EMEP/EEA emission inventory guidebook [82]. Moreover, the GHG estimations are based on the ultimate CO_2 emissions, which result from different processes (*i.e.,* combustion of fuel, combustion of lubricant oil, and addition of carbon-containing additives in the exhaust). This results in equation 5.2:

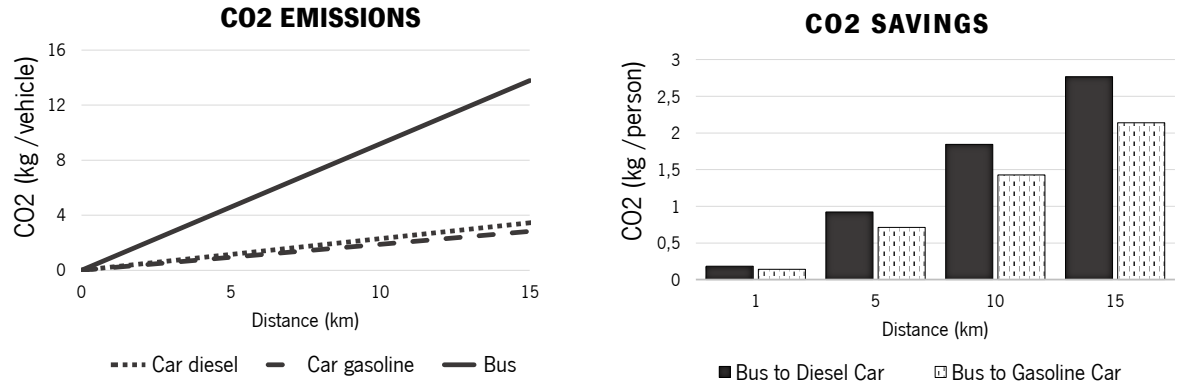
$$E_{ik} = e_{ik}(v) \cdot a_k, \tag{5.2}$$

where E_{ik} is the exhaust emissions of pollutant i induced by a vehicle technology k (in grams); e_{ik} is the emission factor as a function of the vehicle driving speed (in grams per kilometer); a_k is the transport activity in vehicle kilometers traveled for vehicle technology k.

The emissions are calculated individually for each user of the FranchetAI mobile application by considering the average driving speed of the road links that constitute an individual trip. The previous AI model(s) provides the information on traffic data (transportation mode, distance, and traveling speed) necessary to calculate clients' emissions from traffic activity. This information results from the inference of the models trained in the previous stage, and all the calculations are done on the user's mobile device. Also, information on vehicle technology is required, *i.e.*, the Euro Standard information, accessed by considering the age of the vehicle. Therefore, a user is asked to give this detailed information; otherwise, a default technology is used *(Euro 4)*.

In detail, the final emissions model requires the following information: *i)* mean velocity of the trajectory; *ii)* type of fuel of the car; *iii)* category of the vehicle (*i.e.*, passenger, bus, heavy duty and, motorcycle); *iv)* the total distance of the trip; and *v)* the year of the vehicle. The year and category are further used to define the Euro Standard. Still, when the user does not disclose such information, the GHG emissions are estimated based on Euro 4, while the previously trained AI models define the vehicle category.

Figure 5.7a presents an example of using our emissions model. For instance, for a user commuting for 30 min at 30km/h, totaling a distance of 15 Km, one may analyze the CO_2 induced by different vehicle modes (*e.g.*, diesel/gasoline car or bus) in kilograms per vehicle. Also, by assuming that an urban bus will have an occupancy of 20 people, the impact of using such a more environmentally friendly vehicle is presented in Figure 5.7b.



(a) (b)

Figure 5.7: The amount of CO_2 emissions and savings are presented in kilograms per vehicle. The savings are calculated based on the difference between the emissions of the diesel car and the bus. The bus is assumed to have an occupancy of 20 people.

5.3.2 Security Analysis

TAPUS relies on DP-based mechanisms to safeguard users from attacks, see §5.2.1 for more details on these attacks. Leveraging the DP-based algorithms, TAPUS ensures that the models do not remember the data they were trained with, thus preventing the leakage of personal information.

While resorting to *DP-FedAVG* and *DP-FedSGD*, TAPUS introduces noise on the gradients. This guarantees that in any case, the broadcasted gradients do not leak any personal information. With the third approach (*i.e.,* the Local DP) the noise is added to the local data before training the models, ensuring that they do not remember the data they were trained with [62, 143].

Nonetheless, TAPUS does not explore the privacy budget and accounting for any of the approaches as the main goal of TAPUS was to highlight the possibility of using FL and DP for the problem of transportation mode identification. Additionally, works such as Kim, Günlü, and Schaefer which explore the variance bounds of noise (*i.e.,* range of noise values to be defined) can be used to further improve the privacy-preserving guarantees of TAPUS [143].

5.3.3 Implementation

We resort to Flower [25] for the implementation of TAPUS. This framework allows deploying federated training both in mobile settings and across data silos. This is an essential feature if cities provide access to other information, such as road topology and traffic congestion (*i.e.,* similar to the web application proposed in the FranchetAI methodology), that can be combined with users' data to improve the final models.

Flower also provides several averaging algorithms that comprise DP alternatives. These can be used to improve our solution's privacy-preserving guarantees and trustworthiness. Namely, we used the `DP-FedAvg` strategy to introduce random noise on the gradients, mitigating the leakage of their personal

information (*e.g.*, location). We also re-implemented the `DP-FedSGD` algorithm to introduce noise on the updated gradients as a new strategy.

Furthermore, the noise added to the local data before training was implemented in Python by resorting to PyDP [212], a Python wrapper of the Google Differential Privacy library [104]. This wrapper provides a set of tools to apply DP to any project. This library was used to add noise to the local data before training the models, ensuring that they did not remember the data they were trained with.

Finally, the GHG component was implemented in Python, based on the CORINAIR database, and integrated into TAPUS to infer the GHG emissions for each travel on the user's device.

5.4 Methodology

The following section defines the methodology used to develop TAPUS. It is divided into three main subsections: *(i)* the experimental setup, *(ii)* the dataset, GPS data preprocessing, and feature extraction, and *(iii)* the models and algorithms used.

5.4.1 Experimental Setup

The preliminary experiments were run in a single server with OS Ubuntu 18.04.4 LTS and Linux kernel 4.15.0, with a 10Gbps Ethernet card connected to a dedicated local network and 16 GB of memory. This initial setup will be referred to as scenario 1. The dataset, explained in the next section, was stored in a Network File System (NFS) comprising three servers with the exact specifications as the previous one. Further, a similar setup but with four servers was used to develop the FL solution, named scenario 2. The testing setup of FL, depicted as scenario 3, resorted to GCP. In this case, we built a setup with fifty clients and one parameter server. The clients and the parameter server were deployed in GCP's VM instances. The clients were deployed in VM instances with two vCPUs and 8 GB of memory, while the parameter server was deployed in a VM instance with eight vCPUs and 32 GB of memory. The third scenario was used to evaluate the impact of introducing DP in the FL solution, as well as the increase in the number of clients.

5.4.2 Dataset, GPS Data Preprocessing and Feature Extraction

To train the transportation model and the evaluation of TAPUS, we chose the GeoLife GPS Trajectories dataset [323]. It comprises 17,621 GPS trajectory data points from 178 users, each including latitude, longitude, and altitude information. Additionally, data points are labeled with different modes of transportation (classes). In this work, we focus on five classes, including vehicles (comprising individuals' cars and taxis), motorcycles, bikes, buses, and feet (comprising walking and running), which can be seamlessly changed to increase the number of classes.

The preprocessing of input raw data is based on state-of-the-art methods and is used to calculate each user's trajectory's velocity, acceleration, and distance. The distance between two points is calculated using the *Geopy* Python library, which uses the geodesic distance to this end [96]. Formally, let D_i represent the distance for the *ith* GPS point; the distance can be denoted as in Equation 5.3,

$$D_i = Geodesic((lat_i, long_i)(lat_{i+1}, long_{i+1})) \tag{5.3}$$

Based on D_i, the velocity, v_i, is calculated by dividing the distance between two points by the time taken to travel between these. Finally, the acceleration, a_i, is calculated by dividing the velocity across two points with the time taken to travel between these as follows, with T as the number of GPS points in a GPS record:

$$v_i = \frac{D_i}{\triangle t_i}, 1 \geq i \leq T, v_T = v_{T-1} \tag{5.4}$$

$$a_i = \frac{v_{i+1} - v_i}{\triangle t_i}, 1 \geq i \leq T, a_T = 0 \tag{5.5}$$

Input data is then split into training and testing sets. The training set is used to train the model, whereas the testing set is used to evaluate the model. The split is performed using the `train_test_split` function from the SciKit-Learn library [228], using a 70/30 ratio, where 70% of the data is used for training and 30% for testing.

The preprocessed dataset is used to train the first version of the model in a centralized setting, which is then retrained with the data collected on each user's mobile device by resorting to FL. The information collected at each mobile device is identical to the one reported in the GeoLife dataset and goes through the same preprocessing method.

5.4.3 Models and Algorithms

Different ML and DL models were chosen better to understand their trade-offs between accuracy and execution time. For the ML models, we resorted to Random Forest, Decision Trees, Logistic Regression, and XGBoost, provided by the SciKit-Learn library [228]. The DL models were built based on the assumption that training data is provided through CSVs stored in a time-series database. As such, the models were developed with dense and long short-term layers. The literature supports using these architectures for similar time-series use cases [46]. Finally, these models were implemented on top of TensorFlow [280].

5.5 Evaluation

In this section, we highlight and discuss the main results of TAPUS. This section is divided into two main subsections. First (Subsection 5.5.1) presents the preliminary results of the AI models trained with the GeoLife dataset and showcases the viability of using our DL model within an FL setup. The second

subsection explores the trade-offs of introducing DP in the FL set up in the three main crucial parts of its pipeline (*i.e.,* input, gradients, and output).

5.5.1 Preliminary Results

The first tests in scenario 1 with the GeoLife dataset helped limit and create general classes for the transportation modes. By limiting the number of classes, we were able to reduce the tree length and improve the results of Random Forests (RF) and Decision Trees (DT) models by around 5%, reaching an accuracy of 81% (as depicted in Figure 5.8) in less than 3 minutes of training.

Implementing Logistic Regression (LG) with and without cross-validation (LGCV) and XGBoost has shown lower accuracy results, namely 45% for both the logistic algorithms and around 70% for XGBoost. Regarding training times, the convergence of the algorithms took 1 minute and 14.7 hours for LG and LGCV, respectively, and 1.4 hours for XGBoost.

Moreover, the developed DL model based on dense and long-short-term memory layers (LSTM) showed lower accuracy when compared to RF and DT. Yet, its implementation obtained an accuracy of 75% with an execution time of 5 minutes for 50 epochs.

These tests also allowed the perception of the impact of different input features. Namely, the first batch of tests was performed with the traveled distance, the mean velocity, and the mean acceleration. However, the mean acceleration did not impact the accuracy of the trained models positively or negatively.

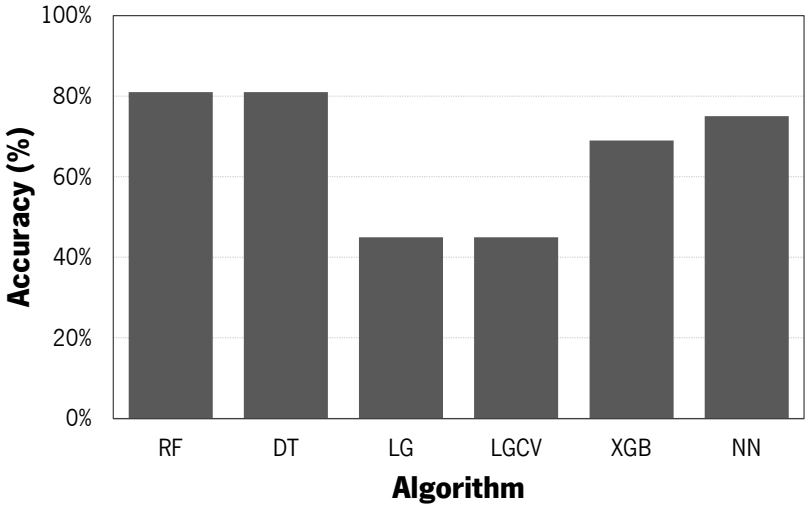


Figure 5.8: Accuracy results for the tested models.

In scenario 2, we sharded the train dataset into ten shards to assess the viability of using an FL setup. Each shard was then attributed to a distinct user's device (mobile application). These experiments focused only on the NN model. Initial results were promising, showing that the model can label the remaining trajectories (*i.e.*, test dataset) with similar accuracy. Also, the retraining of the model on the test dataset was performed in 5 rounds, in which the users' devices trained the models for five iterations and

broadcasted their parameters to the centralized server. In the end, a global model was saved containing the aggregation of all the parameters and kept the initial accuracy of 75%.

5.5.2 Privacy Evaluation

To evaluate the impact of introducing DP in scenario 3, we performed a series of experiments. Namely, we define the usage of the *DP-FedAVG* and *DP-FedSGD* algorithms, as well as the local DP, while comparing with the baseline algorithm *FedAVG*.

We evaluated with ϵ value for the different parts of the pipeline (*i.e.,* 0.1, 0.2, 0.3, 0.4, and 0.5), with a constant clipping gradient of 0.1 for 50 communication rounds or iterations. The results showed that the model's accuracy decreased as the noise multiplier increased, as expected.

Table 5.1: Accuracy results for ten clients with variance in the noise multiplier (*i.e.,* 0.1, 0.2, 0.3, 0.4 and, 0.5).

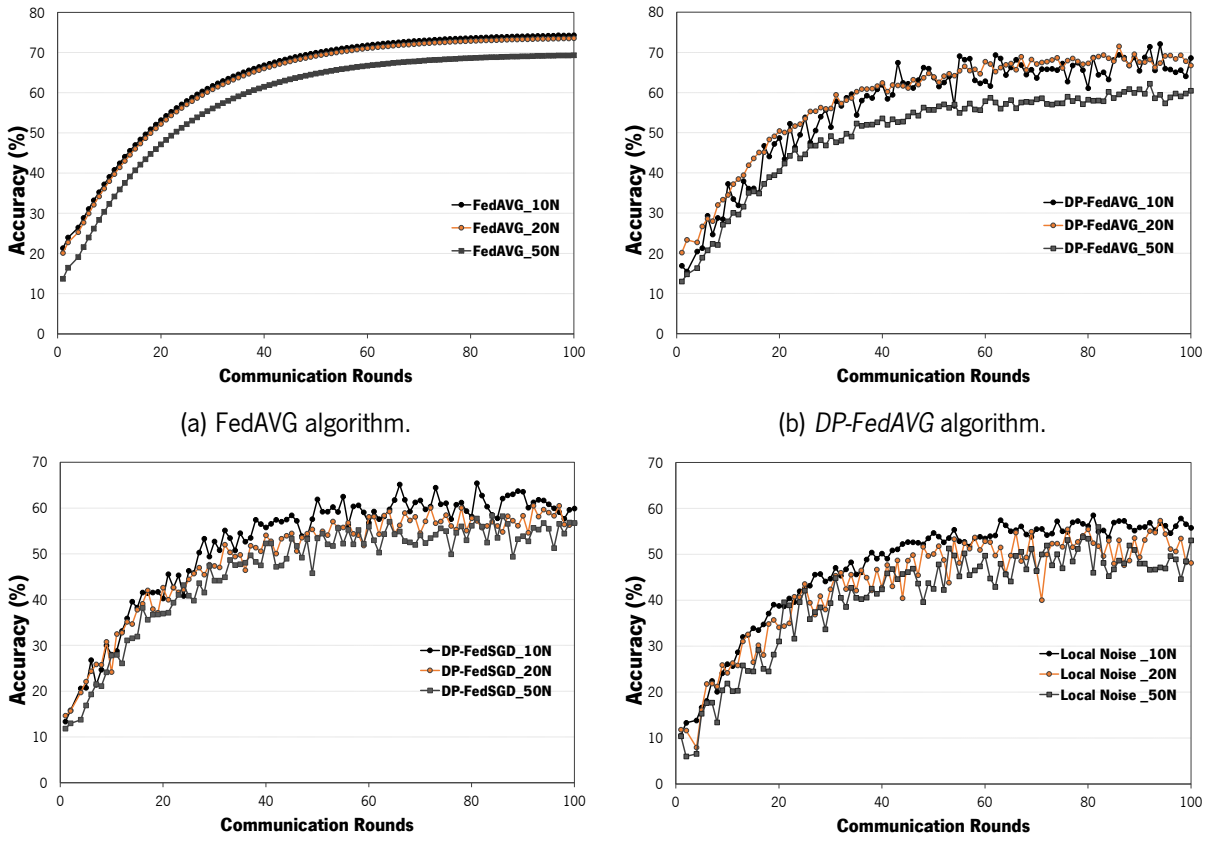
Noise	DP-FedAVG	DP-FedSGD	Local	Baseline
0.1	72,69±0,58	72,2±0,95	66,41±1,89	
0.2	72,13±0,49	69,32±1,53	63,38±1,99	
0.3	71,11±0,61	62,16±0,59	55,30±1,92	74,518±0,57
0.4	69,43±0,39	60,21±1,57	54,40±0,77	
0.5	63,27±0,91	56,67±0,85	50,83±1,25	

The model's accuracy for a noise of 0.5 was 62,3%, 56,7% and 50,1% for *DP-FedAVG*, *DP-FedSGD* and *Local*, respectively. These results are up to 15% lower than the model without DP (75%).

> **Takeaway 1.** *Increasing the amount of noise added to the gradients of the model's parameters decreases the model's accuracy. This behavior is expected, as the noise added to the gradients is directly related to the model's accuracy. Also, the noise added to the data or the model's parameters impacts the model's accuracy. It is important to note that adding noise directly to the input data has a more significant impact on the accuracy and quality of the noise.*

Finally, we evaluate the four settings (*i.e.,* FedAVG, DP-FedAVG, DP-FedSGD and Local) with increasing clients (*i.e.* 10, 20, and 50 clients) and noise of 0.3, maintaining the gradient clipping at 0.1. The results showed that the model's accuracy decreased as the number of clients increased (see Figures 5.9a, 5.9b, 5.9c and 5.9d).

The results showed that the model's accuracy decreased as the number of clients increased. For instance, the model's accuracy for 50 clients was 60,4%, 56,7%, and 53,07% for *DP-FedAVG*, *DP-FedSGD* and *Local*, respectively. These results are up to 16% lower than the model without DP (69,3% for 50 clients).



(a) FedAVG algorithm.

(b) *DP-FedAVG* algorithm.

(c) *DP-FedSGD* algorithm.

(d) Local DP.

Figure 5.9: Impact in the accuracy of TAPUS with an increasing number of clients (10, 20, and 50).

> **Takeaway 2.** *Although the convergence rate of the model is maintained with the increase in the number of clients, the model's accuracy decreases. This behavior is expected, as the noise added to the gradients is directly related to the model's accuracy. Moreover, by increasing the number of clients, more data and parameters are shared, which increases the overall noise and, consequently, decreases the model's accuracy.*

The evaluated solution presents a viable option for mobile settings, other features such as instant velocity and accelerometer and/or gyroscope information, which can be collected through the mobile application, should be made available to train new models and check whether these can improve training speed and the model's accuracy.

5.6 Related Work

Knowing how users interact daily with transportation utilities allows for an understanding of their environmental impact. Solutions targeting such a goal must be able to collect and determine what modes of transportation are being used while pinpointing their carbon footprint. This often requires collecting

sensitive personal information from devices like mobile phones [177]. Therefore, one must also ensure that users' data is not disclosed to unwanted third-parties. With this, state-of-the-art solutions can be decomposed into three main subgroups: *i)* mobility patterns and transportation mode, *ii)* carbon footprint assessment, and *iii)* federated learning for mobility.

5.6.1 Mobility Patterns and Transportation Modes

Mobility pattern studies typically focus on how citizens use urban transportation and how it might be enhanced to better serve their needs and those of communities. Data collected from several sources (*e.g.*, GPS traces, weather conditions, traffic status) can be leveraged to deliver solutions with different purposes [274].

Currently, mobility patterns are classified into two main types: prediction and generation. Regarding mobility pattern prediction, studies have focused on the following topics:

Next-location prediction. It aims to understand the future locations of users based on their previous behavior and historical mobility data. Such information can be beneficial for improving public health, reducing traffic congestion, and providing better travel recommendations. Additionally, it can help urban and public transportation planning, as cities can leverage this information to upgrade their road systems, urban infrastructure, and public transportation systems [177, 213].

Crowd flow prediction. It assists cities in pinpointing areas of possible traffic congestion and infrastructural improvement based on crowd behavior. Once more, these studies are based directly on users' information and how they move around cities. However, they leverage the flow of the crowd itself and do not intend to predict an individual's next geographical position [177, 213].

Mobility pattern generation studies also follow two different approaches:

Trajectory generation. This approach is based on individual GPS traces and trajectories, and the goal is to generate synthetic trajectories that follow distinct traveled distances and predictable human mobility patterns. Generating these trajectories may aid in urban planning and avoid collecting citizens' geographical positions [177, 316].

Flow generation. This approach gathers crowd flow information for a specific geographical region based on users' exact location. This is a crucial process for transport planning and epidemic spread patterns [177, 316].

The prediction and classification of mobility patterns can also be leveraged to define the transportation mode of users. For instance, recent studies have used GPS traces for transportation-mode detection, classification, and prediction. These studies have shown high confidence in defining several classes of transportation (*e.g.*, walk, run, bike, car, bus, train) by using Machine and Deep Learning algorithms

(ML/DL) [124, 166, 297]. Other solutions explore using multiple sensors to detect transportation modes alongside the GPS traces (*i.e.,* accelerometers, gyroscopes, and/or magnetometers).

Further, to improve the accuracy of these models, one can rely on additional information from public transportation networks and urban infrastructure (*e.g.*, roads, streets, highways, etc.). The interplay of these new variables allows for enhancing the output of these prediction and classification models [124].

5.6.2 Carbon Footprint Assessment

Calculating and inferring the carbon footprint of an individual user is complex as it requires the collection of several data points and integrating distinct models [100].

LCA models allow the calculation of the carbon footprints for the entire life cycle of a specific transportation mode, from the manufacturing of the vehicle to its usage and maintenance and, finally, to its disposal (*cradle-to-grave* model). Such models provide a holistic view of vehicles' full life cycle and their impact on the environment [133, 272].

On the other hand, the WtW concept provides an alternative LCA model that focuses only on transport fuels and vehicles' usage. *Well-to-wheel* differs from *cradle-to-grave*, as it does not consider energy and emissions involved in building facilities and the vehicles, nor end-of-life aspects of the latter. The model analysis is often broken down into two stages entitled *well-to-tank* and *tank-to-wheel*. The first stage, known as *upstream stage*, incorporates fuel production, processing, and delivery, while the *downstream* stage deals with vehicle operation. The WtW analysis is commonly used to assess total energy consumption or the energy conversion efficiency and emissions impact of different transport modes [294]. However, when used alone, these models cannot estimate a specific user's daily carbon footprint.

By knowing the transportation mode being used by a given user and by combining information about the user's trajectory (*e.g.,* distance traveled, mean velocity, vehicle class, fuel type) with an LCA model, one can infer the carbon footprint of a commuter [86, 148, 175].

Alternatively, researchers are using ML and DL models to automate the estimation of GHG emissions. These models have achieved high accuracy rates and have been able to infer the carbon footprint of a user based solely on the transportation mode and traveled distance [164].

5.6.3 Federated Learning with Differential Privacy for Mobility

FL is a paradigm that allows ML models to be trained over distributed data without sharing it with third-party entities [189]. FL has been used in several domains, such as healthcare, finance, and transportation.

While FL is based on the assumption that no user data is outsourced to third-party infrastructures, most algorithms do not contemplate attacks that intend to infer sensitive data from the parameters or the trained model. PPFL has emerged to tackle such a challenge. Recent algorithms employ DP as a privacy measure to ensure that sensitive data and gradients are not disclosed to third-party entities [78].

Alongside, SMPC algorithms, namely secure aggregation, is an alternative algorithm applied to FL [28]. While DP trades off accuracy for privacy, MPC trades off performance for privacy.

As an open research problem, the trade-off between privacy and accuracy is crucial when using DP. To reduce the error rate, one can apply optimizations such as adaptive clipping [15], which continuously adapts the amount of noise introduced in each training sample. This is also the case for recently published works such as the one from Guo et al. [111]. This work leverages local DP and a proactive alarming mechanism to increase the resiliency to byzantine attacks and inference attacks in FL. Although crucial, these works are orthogonal to the one proposed here as the aim of TAPUS is to validate the feasibility of employing FL in a transportation field with the added goal of preserving users' data privacy by resorting to DP.

On the transportation field, recent works such as the one proposed by Sakho and Othman [257], resort to Vehicular Ad hoc Networks (VANETs) to create a novel approach to predict trajectories based on a FL architecture. Nonetheless, the focus of this solution is on the vehicles' data, not the users'. Furthermore, only a small subset of works has been proposed to tackle the specific problem of predicting or classifying the transportation mode of users based on GPS traces [46, 311].

We highlight that there are some interesting works to evaluate the possible data leakage from the gradients [302] or to audit if the protocol is truly differentially private [146]. However, TAPUS does not focus on these aspects of PPFL.

Summary. In general, no solution holistically comprises all the previous three main topics. Specifically, in this chapter, we focused on the detection of the transportation modes and the estimation of the carbon footprint of each individual citizen while promoting the preservation of users' data privacy. In contrast to the work proposed in this chapter, there are a few FL solutions that leverage federated algorithms on the mobility use case without the additional privacy measures mentioned above. Moreover, these approaches do not extend further from the transportation mode classification or explore the calculation of GHG emissions.

5.7 Summary

TAPUS is a privacy-preserving FL solution that allows users to train ML models locally while preserving their privacy to find mobility patterns and more efficient transportation alternatives. The results show that the model's accuracy decreases as the number of clients and the noise increase.

Although TAPUS is a preliminary work, it shows that FL can be used to train transportation models while preserving users' privacy. Moreover, it opens two different paths for future work. First, TAPUS shows that an extensive evaluation of the different FL algorithms with the addition of DP is necessary to understand the trade-offs between privacy and accuracy. Furthermore, other techniques that combine the different DP-based solutions could be explored such as the recent work of Girgis et al. [98]. Second, it opens the

possibility of using real-world datasets in a real-world scenario for transportation mode identification of users.

Contrary to Soteria and Gyosa which are dependent on the hardware, TAPUS presents a PPFL alternative that does not rely on TEEs. It emphasizes the feasibility of offering privacy-preserving alternatives without the need to resort to specific hardware.

6

Conclusion

PPML has a key role in the future of AI applications, offering the possibility of leveraging sensitive data in third-party infrastructures (*i.e.,* cloud computing, HPC) or directly in the user's mobile devices. However, as stated in the previous chapters, there are several trade-offs between performance, privacy, and applicability that have been limiting the adoption of these solutions.

In this thesis, we present three contributions that aim to address these limitations and improve the adoption of PPDML solutions. First, we propose Soteria, a novel system that leverages the scalability and reliability of Apache Spark and its ML library (MLlib). This system ensures that critical operations, which enable existing attacks to reveal sensitive information from ML datasets and models, are exclusively performed in secure enclaves provided by Trusted Execution Environments (TEEs). This means that the sensitive data being processed only exists in plaintext inside the enclave and is encrypted in the remainder of the dataflow (*i.e.*, network, storage). This solution enables robust security guarantees, ensuring data privacy during ML training and inference.

Soteria introduces a new computation partitioning scheme for Apache Spark's MLlib, Soteria-P, that offloads non-critical statistical operations from the trusted enclaves to untrusted environments. To highlight the contributions of this novel scheme, Soteria also offers a baseline scheme, Soteria-B, where all ML operations are done inside trusted enclaves without a fine-grained differentiation between critical and non-critical operations. Soteria-B provides a performance and security baseline for comparison against our new partitioned scheme. Our experiments, resorting to the HiBench benchmark [130] and including seven different ML algorithms, show that Soteria-P, while considering a more significant subset of ML attacks, reduces training time by up to 41% for Gradient Boosted Trees workloads and up to 4.3 hours for Linear Regression workloads when compared to SGX-Spark, a state–of–the–art PPML system for Apache Spark. Also, compared to Soteria-B, Soteria-P reduces execution time by up to 37% for the Gradient Boosted Trees workloads and up to 3.3 hours for the Linear Regression workloads.

While Soteria provides a generic PPDML approach for ML workloads, it is not directly suited for other types of data and studies, such as genomic data, namely GWAS, which can also greatly benefit from PPDML. Therefore, we propose Gyosa, a novel distributed computing solution for privacy-preserving genome-wide association studies. This solution is based on Soteria but is tailored to the specificities of

genomic data. By resorting to Glow, Gyosa allows the extension of the current genomic analysis pipeline, in which one can add new tasks (*e.g.*, new statistical tests, genomic imputation, and querying). Built upon the same principles as Soteria, Gyosa enables fine-grained differentiation between sensitive and nonsensitive information processed by GWAS. With this, by offloading sensitive information to a secure environment, we show that it is possible to run GWAS-based algorithms for large datasets while keeping critical information safe from being leaked by attackers. First, the results show the feasibility of offering a privacy-preserving genomic analysis solution that preserves the original results of GWASes. As anticipated, there is a trade-off between runtime computation and the level of privacy-preserving guarantees. Gyosa shows a runtime execution overhead ranging from $2.5X$ for X^2 statistic tests to $10X$ on regression-based algorithms. Importantly, the results highlight that it is possible to effectively reduce the computation runtime overhead by distributing the computation across multiple nodes while offering privacy guarantees for sensitive data.

Although both Soteria and Gyosa are designed to provide a PPML solution for different use cases, these solutions rely on specific hardware, namely TEEs. Nonetheless, the lack of these trusted environments in IOT or mobile devices is a limitation that must be addressed. As such, our final contribution, TAPUS, focuses on the trade-offs between accuracy and privacy for DML environments without specialized hardware. By combining Federated Learning and Differential Privacy, we propose a prototype for understanding users' transportation modality preferences while preserving their privacy and reporting their carbon footprint. To provide privacy-preserving guarantees, TAPUS resorts to DP mechanisms to protect users' data. This is done by adding noise to the gradients of the model's parameters before sending them to the server or adding noise directly to the collected data before performing any computation. This is a vital feature of TAPUS, as it ensures that the user's data is not leaked during training. The results show the feasibility of using both DP-FL algorithms and local DP mechanisms to protect users' data distinctly. In sum, the results show a decrease of up to 25% of the accuracy when using Local DP with an ϵ of 0.5. When evaluating the scalability of TAPUS, we show that the system can handle several clients while preserving users' data privacy. In these experiments, we show that the system can handle up to 50 clients with up to 16% decrease in accuracy.

With the contributions presented in this thesis, we demonstrated that it is possible to develop privacy-preserving and distributed ML systems that can be used to train and infer ML models while preserving the privacy of the data owners. As such, we believe these contributions are a stepping stone towards a wider adoption of privacy-preserving and distributed ML systems.

6.1 Future Work

The field of PPDML is still in its early stages, and there is still a long way to go. The work presented in this thesis opens several avenues for future work.

Adding privacy-preserving mechanisms to any ML algorithm is challenging, and the work presented

in this thesis is no exception. An interesting future work direction is to target the development of new privacy-preserving mechanisms that can be used entirely agnostic to the ML algorithm and framework, which would allow the adoption of PPDML in a broader range of applications and use cases.

Alongside and as seen in some recent works, there should be a focus on combining hardware and software-based cryptographic primitives to improve the security guarantees and balance the trade-offs of both approaches. Namely, both Soteria and Gyosa could leverage data oblivious primitives to improve their security against side-channel attacks. Also, the combination of different DP-based strategies, both local and global, could be used to improve the privacy guarantees of TAPUS. These additional primitives would influence, in both cases, the balance between performance, applicability, and privacy, nonetheless, a novel counterbalance should be found as it would foster collaboration between different entities with sensitive data (*e.g.,* hospitals, financial institutions, user's mobile devices).

Additionally, although this thesis proposed a new overview to the field of PPDML, we acknowledge that the AI field is growing in plain sight with Large Language Models LLMs being proposed almost every day (*e.g.*, LLaMA, BLOOM, BERT, Falcon 180B [12, 72, 284, 304]). As these models are being developed, it is crucial to understand their complexity and what new concerns regarding data privacy they may bring [66].

Modern infrastructures are also evolving, and new hardware is being developed to support the growing computational and storage needs of AI applications (*i.e.,* persistent memory, new GPUs). As such, it is essential to understand how this new hardware can be leveraged to provide privacy guarantees to the data owners.

Concerning FL protocols, it is vital to acknowledge the research community effort to tackle byzantine attacks. Although the work presented in this thesis does not address this issue, it is important to understand how these attacks can be prevented and mitigated in a privacy-preserving manner.

In conclusion, increasing quantities of data and its complexity, new hardware, and new algorithms always introduce new challenges that need further addressing. Any PPDML system must endure these growing challenges and provide privacy guarantees to the entities that own personal and sensitive data.

Bibliography

[1] Amazon Web Services (AWS). *Cloud Computing Services*. https://aws.amazon.com/. (Accessed on 02/06/2023).

[2] World Economic Forum (WEF). *Shaping the Future of Mobility*. https://www.weforum.org/communities/gfc-on-urban-mobility-transitions/. (Accessed on 15/02/2024). 2024.

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. "Deep learning with differential privacy". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 308–318. doi: 10.1145/2976749.2978318.

[4] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "Tensorflow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283.

[5] ABCI. *AI Bridging Cloud Infrastructure*. https://abci.ai/. (Accessed on 02/06/2023).

[6] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. "Secure training of decision trees with continuous attributes". In: *Proceedings on Privacy Enhancing Technologies* 1 (2021), pp. 167–187. doi: 10.2478/popets-2021-0010.

[7] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation". In: vol. 51. 4. New York, NY, USA: ACM, July 2018, 79:1–79:35. doi: 10.1145/3214303.

[8] ADAM. *ADAM User Guide — bdgenomics.adam 0.23.0-SNAPSHOT documentation*. https://adam.readthedocs.io/en/latest/. (Accessed on 05/02/2023).

[9] Mohammed Adnan, Shivam Kalra, Jesse C Cresswell, Graham W Taylor, and Hamid R Tizhoosh. "Federated learning and differential privacy for medical image analysis". In: *Scientific reports* 12.1 (2022), p. 1953. doi: 10.1038/s41598-022-05539-7.

[10] Bill Aiello, Yuval Ishai, and Omer Reingold. "Priced oblivious transfer: How to sell digital goods". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2001, pp. 119–135. doi: 10.1007/3-540-44987-6_8.

[11] *Algorithmia | DataRobot AI Platform*. https://www.datarobot.com/algorithmia/. (Accessed on 08/31/2023).

[12] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. "The Falcon Series of Open Language Models". In: (2023). arXiv: 2311.16867 [cs.CL].

[13] João Alves, Beatriz Soares, Cláudia Brito, and António Sousa. "Cloud-Based Privacy-Preserving Medical Imaging System Using Machine Learning Tools". In: *EPIA Conference on Artificial Intelligence*. 2022, pp. 195–206. doi: 10.1007/978-3-031-16474-3_17.

[14] Tiago Alves. "Trustzone: Integrated hardware and software security". In: *White paper* (2004).

[15] Galen Andrew, Om Thakkar, Brendan McMahan, and Swaroop Ramaswamy. "Differentially private learning with adaptive clipping". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 17455–17466.

[16] *Apache Teaclave*. https://teaclave.apache.org/. (Accessed on 21/11/2020).

[17] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. "SCONE: Secure linux containers with intel SGX". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 689–703.

[18] Aref Asvadishirehjini, Murat Kantarcioglu, and Bradley Malin. "A Framework for Privacy-Preserving Genomic Data Analysis Using Trusted Execution Environments". In: *2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE. 2020, pp. 138–147. doi: 10.1109/tps-isa50397.2020.00028.

[19] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers". In: *International Journal of Security and Networks* 10.3 (2015), pp. 137–150. doi: 10.1504/ijsn.2015.071829.

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

[21] Microsoft Azure. *Cloud Computing Services*. https://azure.microsoft.com/en-us. (Accessed on 01/08/2024).

[22] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. "How To Backdoor Federated Learning". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 2938–2948. url: https://proceedings.mlr.press/v108/bagdasaryan20a.html.

[23] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. "Secure multiparty computation from SGX". In: *International Conference on Financial Cryptography and Data Security*. Ed. by Aggelos Kiayias. Springer. Cham: Springer International Publishing, 2017, pp. 477–497. isbn: 978-3-319-70972-7. doi: 10.1007/978-3-319-70972-7_27.

[24] Reda Bellafqira, Thomas E Ludwig, David Niyitegeka, Emmanuelle Genin, and Gouenou Coatrieux. "Privacy-Preserving Genome-Wide Association Study for Rare Mutations-A Secure FrameWork for Externalized Statistical Analysis". In: *IEEE Access* 8 (2020), pp. 112515–112529. doi: 10.1109/access.2020.3002966.

[25] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. "Flower: A friendly federated learning research framework". In: *arXiv preprint arXiv:2007.14390* (2020).

[26] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[27] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. "Towards Federated Learning at Scale: System Design". In: 1 (2019). Ed. by A. Talwalkar, V. Smith, and M. Zaharia, pp. 374–388. url: https://proceedings.mlsys.org/paper_files/paper/2019/file/7b770da633baf74895be22a8807f1a8f-Paper.pdf.

[28] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. "Practical secure aggregation for federated learning on user-held data". In: *arXiv preprint arXiv:1611.04482* (2016).

[29] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. "Practical secure aggregation for privacy-preserving machine learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 1175–1191. doi: 10.1145/3133956.3133982.

[30] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. “Evaluating 2-DNF Formulas on Ciphertexts”. In: *TCC'05 Proceedings of the Second international conference on Theory of Cryptography*. 2005, pp. 325–341. doi: 10.1007/978-3-540-30576-7_18.

[31] Dan Boneh and Victor Shoup. “A graduate course in applied cryptography”. In: *Draft 0.2* (2015).

[32] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. “Machine learning classification over encrypted data.” In: *NDSS*. Vol. 4324. 2015, p. 4325. doi: 10.14722/ndss.2015.23241.

[33] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. “Gossip algorithms: Design, analysis and applications”. In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 3. IEEE. 2005, pp. 1653–1664. doi: 10.1109/infcom.2005.1498447.

[34] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. “SANCTUARY: ARMing TrustZone with User-space Enclaves.” In: *NDSS*. 2019.

[35] Wieland Brendel, Jonas Rauber, and Matthias Bethge. “Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models”. In: *International Conference on Learning Representations*. 2018.

[36] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. “Securekeeper: confidential zookeeper using intel sgx”. In: *Proceedings of the 17th International Middleware Conference*. 2016, pp. 1–13. doi: 10.1145/2988336.2988350.

[37] Cláudia Brito, Pedro Ferreira, and João Paulo. “A Distributed Computing Solution for Privacy-Preserving Genome-Wide Association Studies”. In: *bioRxiv* (2024). doi: 10.1101/2024.01.15.575678. url: https://www.biorxiv.org/content/10.1101/2024.01.15.575678v2.

[38] Cláudia Brito, Pedro Ferreira, Bernardo Portela, Rui Oliveira, and João Paulo. “SOTERIA: Preserving Privacy in Distributed Machine Learning”. In: *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. SAC '23. Tallinn, Estonia: Association for Computing Machinery, 2023, pp. 135–142. isbn: 9781450395175. doi: 10.1145/3555776.3578591. url: https://doi.org/10.1145/3555776.3578591.

[39] Cláudia Brito, Ana Machado, and António Lui□s Sousa. “Electrocardiogram Beat-Classification Based on a ResNet Network.” In: *MedInfo*. 2019, pp. 55–59. doi: 10.3233/shti190182.

[40] Cláudia V. Brito, Pedro G. Ferreira, Bernardo L. Portela, Rui C. Oliveira, and João T. Paulo. “Privacy-Preserving Machine Learning on Apache Spark”. In: *IEEE Access* 11 (2023), pp. 127907–127930. doi: 10.1109/ACCESS.2023.3332222.

[41] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. “FLASH: fast and robust framework for privacy-preserving machine learning”. In: *Proceedings on Privacy Enhancing Technologies* 2020.2 (2020). doi: 10.2478/popets-2020-0036.

[42] Daniel Cahall. *maxpumperla/elephas: Distributed Deep learning with Keras & Spark*. https://github.com/maxpumperla/elephas. (Accessed on 11/21/2023).

[43] Deng Cai, Xiaofei He, and Jiawei Han. “Training linear discriminant analysis in linear time”. In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE. 2008. doi: 10.1109/icde.2008.4497429.

[44] R. Canetti. “Universally composable security: A new paradigm for cryptographic protocols”. In: *42nd IEEE Symposium on Foundations of Computer Science*. 2001. doi: 10.1109/sfcs.2001.959888.

[45] Rita M Cantor, Kenneth Lange, and Janet S Sinsheimer. “Prioritizing GWAS results: a review of statistical methods and recommendations for their application”. In: *The American Journal of Human Genetics* 86.1 (2010), pp. 6–22. doi: 10.1016/j.ajhg.2009.11.017.

[46] Iago C Cavalcante, Rodolfo I Meneguette, Renato H Torres, Leandro Y Mano, Vinı́cius P Gonçalves, Jó Ueyama, Gustavo Pessin, Georges D Amvame Nze, and Geraldo P Rocha Filho. “Federated System for Transport Mode Detection”. In: *Energies* 15.23 (2022), p. 9256. doi: 10.3390/en15239256.

[47] Beatriz Cepa, Cláudia Brito, and António Sousa. “Generative Adversarial Networks in Healthcare: A Case Study on MRI Image Generation”. In: *2023 IEEE 7th Portuguese Meeting on Bioengineering (ENBENG)*. IEEE. 2023, pp. 48–51. doi: 10.1109/ENBENG58165.2023.10175330.

[48] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. “Privacy-preserving classification on deep neural network.” In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 35.

[49] Chainer. *chainer/chainer: A flexible framework of neural networks for deep learning*. https://github.com/chainer/chainer. (Accessed on 11/21/2023).

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

[51] IPO Change. “IPCC guidelines for national greenhouse gas inventories”. In: *Institute for Global Environmental Strategies, Hayama, Kanagawa, Japan* (2006).

[52] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. “Astra: High throughput 3pc over rings with application to secure prediction”. In: *ACM SIGSAC Conference on Cloud Computing Security Workshop*. 2019, pp. 81–92. doi: 10.1145/3338466.3358922.

[53] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. “Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning”. In: *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. doi: 10.14722/ndss.2020.23005.

[54] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. “Differentially private empirical risk minimization”. In: *Journal of Machine Learning Research* 12.Mar (2011), pp. 1069–1109. url: %5Curl%7Bhttps://jmlr.org/papers/volume12/chaudhuri11a/chaudhuri11a.pdf%7D.

[55] Feng Chen, Shuang Wang, Xiaoqian Jiang, Sijie Ding, Yao Lu, Jihoon Kim, S Cenk Sahinalp, Chisato Shimizu, Jane C Burns, Victoria J Wright, Eileen Png, Martin L Hibberd, David D Lloyd, Hai Yang, Amalio Telenti, Cinnamon S Bloss, Dov Fox, Kristin Lauter, and Lucila Ohno-Machado. “Princess: Privacy-protecting rare disease international network collaboration via encryption through software guard extensions”. In: *Bioinformatics* 33.6 (2017), pp. 871–878. doi: 10.1093/bioinformatics/btw758.

[56] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. “Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models”. In: *10th ACM workshop on artificial intelligence and security*. 2017. doi: 10.1145/3128572.3140448.

[57] Wang Chenghong, Yichen Jiang, Noman Mohammed, Feng Chen, Xiaoqian Jiang, Md Momin Al Aziz, Md Nazmus Sadat, and Shuang Wang. “SCOTCH: Secure Counting Of encrypTed genomiC data using a Hybrid approach”. In: *AMIA Annual Symposium Proceedings*. Vol. 2017. American Medical Informatics Association. 2017, p. 1744. url: %5Curl%7Bhttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC5977689/%7D.

[58] Joseph I Choi and Kevin RB Butler. “Secure Multiparty Computation and Trusted Hardware: Examining Adoption Challenges and Opportunities”. In: vol. 2019. Hindawi, 2019. doi: 10.1155/2019/1368905.

[59] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. “Faster CryptoNets: Leveraging sparsity for real-world encrypted inference”. In: *arXiv preprint arXiv:1811.09953* (2018).

[60] Google Cloud. *Cloud Computing Services*. https://cloud.google.com/. (Accessed on 02/06/2023).

[61] EC (European Commission). *Transport emissions*. https://climate.ec.europa.eu/eu-action/transport-emissions_en. 2016.

[62] Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang. "Privacy at Scale: Local Differential Privacy in Practice". In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD '18. Houston, TX, USA: Association for Computing Machinery, 2018, pp. 1655–1658. isbn: 9781450347037. doi: 10.1145/3183713.3197390. url: https://doi.org/10.1145/3183713.3197390.

[63] Victor Costan, Ilia Lebedev, and Srinivas Devadas. "Sanctum: Minimal Hardware Extensions for Strong Software Isolation". In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 857–874. isbn: 978-1-931971-32-4. url: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[64] Jeff Daily, Abhinav Vishnu, Charles Siegel, Thomas Warfel, and Vinay Amatya. "Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent". In: *arXiv preprint arXiv:1803.05880* (2018).

[65] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. "New primitives for actively-secure MPC over rings with applications to private machine learning". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1102–1120. doi: 10.1109/sp.2019.00078.

[66] Badhan Chandra Das, M Hadi Amini, and Yanzhao Wu. "Security and Privacy Challenges of Large Language Models: A Survey". In: *arXiv preprint arXiv:2402.00888* (2024).

[67] Badhan Chandra Das, M. Hadi Amini, and Yanzhao Wu. "Privacy Risks Analysis and Mitigation in Federated Learning for Medical Images". In: *2023 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2023, pp. 1870–1873. doi: 10.1109/BIBM58861.2023.10385829.

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

[69] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. url: https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf.

[70] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113. doi: 10.1145/1327452.1327492.

[71] Lea Demelius, Roman Kern, and Andreas Trügler. "Recent Advances of Differential Privacy in Centralized Deep Learning: A Systematic Survey". In: *arXiv preprint arXiv:2309.16398* (2023).

[72] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[73] Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, Valerio Schiavoni, Pascal Felber, and Daniel Hagimont. “Everything you should know about Intel SGX performance on virtualized systems”. In: vol. 3. 1. ACM New York, NY, USA, 2019, pp. 1–21. doi: 10.1145/3376930.3376979.

[74] Natnatee Dokmai, Can Kockan, Kaiyuan Zhu, XiaoFeng Wang, S Cenk Sahinalp, and Hyunghoon Cho. “Privacy-preserving genotype imputation in a trusted execution environment”. In: *Cell Systems* 12.10 (2021), pp. 983–993. doi: 10.1016/j.cels.2021.08.001.

[75] Jim Dowling. *Distributed ML and Linear Regression*. https://www.kth.se/social/files/5a040fe156be5be5f93667e9/ID2223-02-ml-pipelines-linear-regression.pdf. (Accessed on 01/12/2020). Nov. 2017.

[76] Cynthia Dwork. “Differential privacy”. In: *Encyclopedia of Cryptography and Security* (2011), pp. 338–340. doi: 10.1007/978-1-4419-5906-5_752.

[77] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. “Calibrating noise to sensitivity in private data analysis”. In: *Theory of cryptography conference*. Springer. 2006, pp. 265–284. doi: 10.1007/11681878_14.

[78] Cynthia Dwork and Aaron Roth. “The algorithmic foundations of differential privacy”. In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407. doi: 10.1561/0400000042.

[79] Taher ElGamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472. doi: 10.1109/tit.1985.1057074.

[80] Tarek Elgamal and Mohamed Hefeeda. “Analysis of PCA algorithms in distributed environments”. In: *arXiv preprint arXiv:1503.05214* (2015).

[81] Charles Elkan. “Boosting and naive bayesian learning”. In: *International Conference on Knowledge Discovery and Data Mining*. 1997. url: https://pages.cs.wisc.edu/~dyer/cs540/handouts/elkan97boosting.pdf.

[82] EMEP/EEA. “Exhaust Emissions from Road Transport. Passenger Cars, Light-Duty Trucks, Heavy-Duty Vehicles Including Buses and Motor Cycles.” In: *European Monitoring and Evaluation Programme (EMEP), Air Pollutant Emission Inventory Guidebook 2019.* 13/2019 (2019).

[83] *Enarx Introduction · enarx/enarx Wiki*. https://github.com/enarx/enarx/wiki/Enarx-Introduction. (Accessed on 21/11/2020).

[84] David Evans, Vladimir Kolesnikov, and Mike Rosulek. “A Pragmatic Introduction to Secure Multi-Party Computation”. In: *Foundations and Trends in Privacy and Security* 2.2-3 (2018), pp. 70–246. issn: 2474-1558. doi: 10.1561/9781680835090.

[85] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. "Local Model Poisoning Attacks to Byzantine-Robust Federated Learning". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1605–1622. isbn: 978-1-939133-17-5. url: https://www.usenix.org/conference/usenixsecurity20/presentation/fang.

[86] João C Ferreira, Vítor Monteiro, José A Afonso, and João L Afonso. "Tracking users mobility patterns towards CO 2 footprint". In: *Distributed Computing and Artificial Intelligence, 13th International Conference*. Springer. 2016, pp. 87–96. doi: 10.1007/978-3-319-40162-1_10.

[87] Sam Fletcher and Md Zahidul Islam. "Decision tree classification with differential privacy: A survey". In: *ACM Computing Surveys (CSUR)* 52.4 (2019), pp. 1–33. doi: 10.1145/3337064.

[88] FranchetAI. *Absorbing Traffic Pollution with AI*. http://franchet.ai. 2022.

[89] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 1322–1333. doi: 10.1145/2810103.2813677.

[90] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. "Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing". In: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 17–32. isbn: 978-1-931971-15-7. url: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/fredrikson_matthew.

[91] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: JSTOR, 2001, pp. 1189–1232. doi: 10.1214/aos/1013203451.

[92] David Froelicher, Juan Ramón Troncoso-Pastoriza, Joao Sa Sousa, and Jean-Pierre Hubaux. "Drynx: Decentralized, secure, verifiable system for statistical queries and machine learning on distributed datasets". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3035–3050. doi: 10.1109/TIFS.2020.2976612.

[93] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. "Mitigating sybils in federated learning poisoning". In: *arXiv preprint arXiv:1808.04866* (2018).

[94] General Data Protection Regulation - GDPR. Online: accessed August 28, 2023. 2023. url: https://gdpr-info.eu/.

[95] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*. Vol. 20. 09. Stanford University Stanford, 2009.

[96] GeoPy. *Welcome to GeoPy's documentation! — GeoPy 2.3.0 documentation*. https://geopy.readthedocs.io/. 2022.

[97] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 201–210.

[98] Antonious Girgis, Deepesh Data, Suhas Diggavi, Peter Kairouz, and Ananda Theertha Suresh. "Shuffled Model of Differential Privacy in Federated Learning". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 13–15 Apr 2021, pp. 2521–2529. url: https://proceedings.mlr.press/v130/girgis21a.html.

[99] Glow. *An open-source toolkit for large-scale genomic analysis*. https://projectglow.io/. (Accessed on 02/06/2023).

[100] Fahimeh Golbabaei, Tan Yigitcanlar, and Jonathan Bunker. "The role of shared autonomous vehicle systems in delivering smart urban mobility: A systematic review of the literature". In: *International Journal of Sustainable Transportation* 15.10 (2021), pp. 731–748. doi: 10.1080/15568318.2020.1798571.

[101] Maoguo Gong, Yu Xie, Ke Pan, Kaiyuan Feng, and Alex Kai Qin. "A survey on differentially private machine learning". In: *IEEE computational intelligence magazine* 15.2 (2020), pp. 49–64.

[102] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[103] Google. *Google Cloud Pricing Calculator*. https://cloud.google.com/products/calculator. (Accessed on 02/23/2023).

[104] Google. *Google's differential privacy libraries*. https://github.com/google/differential-privacy. (Accessed on 02/10/2024).

[105] Christian Göttel, Rafael Pires, Isabelly Rocha, Sébastien Vaucher, Pascal Felber, Marcelo Pasin, and Valerio Schiavoni. "Security, performance and energy trade-offs of hardware-assisted memory protection mechanisms". In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2018, pp. 133–142. doi: 10.1109/SRDS.2018.00024.

[106] Gramine. *Gramine Library OS with Intel SGX Support*. https://github.com/gramineproject/gramine. (Accessed on 01/08/2023).

[107] Gramine. *Manifest syntax*. https://gramine.readthedocs.io/en/stable/manifest-syntax.html. (Accessed on 01/08/2023).

[108] Gramine. *PAL host ABI*. https://gramine.readthedocs.io/en/stable/pal/host-abi.html. (Accessed on 01/08/2023).

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

[110] Karan Grover, Shruti Tople, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. "Privado: Practical and secure DNN inference with enclaves". In: *arXiv preprint arXiv:1810.00602* (2018).

[111] Hanxi Guo, Hao Wang, Tao Song, Yang Hua Ruhui Ma, Xiulang Jin, Zhengui Xue, and Haibing Guan. "SIREN+: Robust Federated Learning with Proactive Alarming and Differential Privacy". In: *IEEE Transactions on Dependable and Secure Computing* (2024). doi: 10.1109/TDSC.2024.3362534.

[112] Otkrist Gupta and Ramesh Raskar. "Distributed learning of deep neural network over multiple agents". In: *Journal of Network and Computer Applications* 116 (2018), pp. 1–8.

[113] S Gupta, V Patil, M Himabindu, and RV Ravikrishna. "Life-cycle analysis of energy and greenhouse gas emissions of automotive fuels in India: Part 1–Tank-to-Wheel analysis". In: *Energy* 96 (2016), pp. 684–698.

[114] Apache Hadoop. https://hadoop.apache.org/. (Accessed on 24/02/2021).

[115] Inken Hagestedt, Yang Zhang, Mathias Humbert, Pascal Berrang, Haixu Tang, XiaoFeng Wang, and Michael Backes. "MBeacon: Privacy-Preserving Beacons for DNA Methylation Data." In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. 2019.

[116] Moritz Hardt, Ben Recht, and Yoram Singer. "Train faster, generalize better: Stability of stochastic gradient descent". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1225–1234. url: https://proceedings.mlr.press/v48/hardt16.html.

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

[118] U.S. Department of Heatlh and Human Services. *HIPAA for Professionals | HHS.gov*. https://www.hhs.gov/hipaa/for-professionals/index.html. Accessed on 21/02/2021. 2019.

[119] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. "CryptoDL: Deep Neural Networks over Encrypted Data". In: (Nov. 2017). arXiv: 1711.05189.

[120] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. "Benchmarking the Second Generation of Intel SGX Hardware". In: *Proceedings of the 18th International Workshop on Data Management on New Hardware*. DaMoN '22. Philadelphia, PA, USA: Association for Computing Machinery, 2022. isbn: 9781450393782. doi: 10.1145/3533737.3535098. url: https://doi.org/10.1145/3533737.3535098.

[121] Briland Hitaj, Giuseppe Ateniese, and Fernando 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.

[122] Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V Pearson, Dietrich A Stephan, Stanley F Nelson, and David W Craig. “Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays”. In: *PLoS genetics* 4.8 (2008), e1000167.

[123] Aron N Horvath, Matteo Berchier, Farhad Nooralahzadeh, Ahmed Allam, and Michael Krauthammer. “Exploratory Analysis of Federated Learning Methods with Differential Privacy on MIMIC-III”. In: *arXiv preprint arXiv:2302.04208* (2023).

[124] Haosheng Huang, Yi Cheng, and Robert Weibel. “Transport mode detection based on mobile phone network data: A systematic review”. In: *Transportation Research Part C: Emerging Technologies* 101 (2019), pp. 297–312.

[125] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. “Adversarial machine learning”. In: *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM. 2011, pp. 43–58.

[126] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. “Chiron: Privacy-preserving machine learning as a service”. In: 2018.

[127] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. “Ryoan: A distributed sandbox for untrusted computation on secret data”. In: vol. 35. 4. ACM New York, NY, USA, 2018, pp. 1–32.

[128] Seoyeon Hwang, Ercan Ozturk, and Gene Tsudik. “Balancing Security and Privacy in Genomic Range Queries”. In: *ACM Transactions on Privacy and Security* (2022).

[129] Nick Hynes, Raymond Cheng, and Dawn Song. “Efficient deep learning on multi-source private data”. In: 2018.

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

[131] Salman Iqbal, Miss Laiha Mat Kiah, Babak Dhaghighi, Muzammil Hussain, Suleman Khan, Muhammad Khurram Khan, and Kim-Kwang Raymond Choo. “On cloud security attacks: A taxonomy and intrusion detection and prevention as a service”. In: Elsevier, 2016. doi: `10.1016/j.jnca.2016.08.016`.

[132] Sara Jafarbeiki, Amin Sakzad, Shabnam Kasra Kermanshahi, Raj Gaire, Ron Steinfeld, Shangqi Lai, Gad Abraham, and Chandra Thapa. “PrivGenDB: Efficient and privacy-preserving query executions over encrypted snp-phenotype database”. In: *Informatics in Medicine Unlocked* 31 (2022), p. 100988.

[133] Szczurowski Jakub, Lubecki Adrian, Bałys Mieczysław, Brodawka Ewelina, and Zar□bska Katarzyna. “Life cycle assessment study on the public transport bus fleet electrification in the context of sustainable urban development strategy”. In: *Science of The Total Environment* 824 (2022), p. 153872.

[134] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. “Trusted execution environments: properties, applications, and challenges”. In: *IEEE Security & Privacy* 18.2 (2020), pp. 56–60. doi: 10.1109/MSEC.2019.2947124.

[135] Jianyu Jiang, Xusheng Chen, TszOn Li, Cheng Wang, Tianxiang Shen, Shixiong Zhao, Heming Cui, Cho-Li Wang, and Fengwei Zhang. “Uranus: Simple, Efficient SGX Programming and its Applications”. In: *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. ASIA CCS ’20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 826–840. isbn: 9781450367509. doi: 10.1145/3320269.3384763.

[136] Peter H Jin, Qiaochu Yuan, Forrest Iandola, and Kurt Keutzer. “How to scale distributed deep learning?” In: *arXiv preprint arXiv:1611.04581* (2016).

[137] Michael I Jordan and Tom M Mitchell. “Machine learning: Trends, perspectives, and prospects”. In: *Science* 349.6245 (2015), pp. 255–260.

[138] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. “Gazelle: A Low Latency Framework for Secure Neural Network Inference”. In: *Proceedings of the 27th USENIX Security Symposium* (Jan. 2018). issn: 1520-5762. arXiv: 1801.05507.

[139] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. “Advances and Open Problems in Federated Learning”. In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210. issn: 1935-8237. doi: 10.1561/2200000083. url: http://dx.doi.org/10.1561/2200000083.

[140] Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. “Reliable federated learning for mobile networks”. In: *IEEE Wireless Communications* 27.2 (2020), pp. 72–80.

[141] Seemeen Karimi, Xiaoqian Jiang, Robert H Dolin, Miran Kim, and Aziz Boxwala. “A secure system for genomics clinical decision support”. In: *Journal of Biomedical Informatics* 112 (2020), p. 103602.

[142] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. “Logistic regression model training based on the approximate homomorphic encryption”. In: *BMC medical genomics* 11.4 (2018), pp. 23–31.

[143] Muah Kim, Onur Günlü, and Rafael F. Schaefer. “Federated Learning with Local Differential Privacy: Trade-Offs Between Privacy, Utility, and Communication”. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, pp. 2650–2654. doi: 10.1109/ICASSP39728.2021.9413764.

[144] Can Kockan, Kaiyuan Zhu, Natnatee Dokmai, Nikolai Karpov, M Oguzhan Kulekci, David P Woodruff, and S Cenk Sahinalp. “Sketching algorithms for genomic data analysis and querying in a secure enclave”. In: *Nature methods* 17.3 (2020), pp. 295–301.

[145] Jakub Konečn, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. “Federated learning: Strategies for improving communication efficiency”. In: *arXiv preprint arXiv:1610.05492* (2016).

[146] William Kong, Andrés Muñoz Medina, Mónica Ribero, and Umar Syed. *DP-Auditorium: a Large Scale Library for Auditing Differential Privacy*. 2023. arXiv: 2307.05608 [cs.CR].

[147] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. “Tetrad: Actively Secure 4PC for Secure Training and Inference”. In: *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022. url: https://www.ndss-symposium.org/ndss-paper/auto-draft-202/.

[148] Maria Kugler, Sebastian Osswald, Christopher Frank, and Markus Lienkamp. “Mobility tracking system for CO2 footprint determination”. In: *Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. 2014, pp. 1–8.

[149] Sameer Kumar and Norm Jouppi. “Highly Available Data Parallel ML training on Mesh Networks”. In: *arXiv preprint arXiv:2011.03605* (2020).

[150] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. “TensorSCONE: a secure TensorFlow framework using Intel SGX”. In: *arXiv preprint arXiv:1902.04413* (2019).

[151] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. “Adversarial Machine Learning at Scale”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017.

[152] Christoph Lambert, Maria Fernandes, Jérémie Decouchant, and Paulo Esteves-Verissimo. “Maskal: Privacy preserving masked reads alignment using intel sgx”. In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2018, pp. 113–122.

[153] Do Le Quoc, Franz Gregor, Jatinder Singh, and Christof Fetzer. “SGX-PySpark: Secure distributed data analytics”. In: *The World Wide Web Conference*. ACM. 2019, pp. 3564–3563.

[154] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. “Certified robustness to adversarial examples with differential privacy”. In: *2019 IEEE symposium on security and privacy (SP)*. IEEE. 2019, pp. 656–672.

[155] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa 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.

[156] Heng Li and Richard Durbin. “Fast and accurate short read alignment with Burrows–Wheeler transform”. In: *bioinformatics* 25.14 (2009), pp. 1754–1760.

[157] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. “Scaling distributed machine learning with the parameter server”. In: *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*. 2014, pp. 583–598.

[158] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. “Communication Efficient Distributed Machine Learning with the Parameter Server”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 19–27.

[159] Ping Li, Jin Li, Zhengan Huang, Tong Li, Chong-Zhi Gao, Siu-Ming Yiu, and Kai Chen. “Multi-key privacy-preserving deep learning in cloud computing”. In: *Future Generation Computer Systems* 74 (2017), pp. 76–85.

[160] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. “A survey on federated learning systems: Vision, hype and reality for data privacy and protection”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[161] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. “Federated learning: Challenges, methods, and future directions”. In: *IEEE signal processing magazine* 37.3 (2020), pp. 50–60.

[162] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. “Federated optimization in heterogeneous networks”. In: *Proceedings of Machine learning and systems* 2 (2020), pp. 429–450.

[163] Wentao Li, Miran Kim, Kai Zhang, Han Chen, Xiaoqian Jiang, and Arif Harmanci. "COLLAGENE enables privacy-aware federated and collaborative genomic data analysis". In: *Genome Biology* 24.1 (2023), p. 204.

[164] Wenxiang Li, Yuanyuan Li, Ziyuan Pu, Long Cheng, Lei Wang, and Linchuan Yang. "Revealing the real-world CO2 emission reduction of ridesplitting and its determinants based on machine learning". In: *arXiv preprint arXiv:2204.00777* (2022).

[165] Yuanzhi Li, Colin Wei, and Tengyu Ma. "Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. url: https://proceedings.neurips.cc/paper_files/paper/2019/file/bce9abf229ffd7e570818476ee5d7dde-Paper.pdf.

[166] Xiaoyuan Liang, Yuchuan Zhang, Guiling Wang, and Songhua Xu. "A deep learning model for transportation mode detection based on smartphone sensing data". In: *IEEE Transactions on Intelligent Transportation Systems* 21.12 (2019), pp. 5223–5235.

[167] LibSodium. *Introduction - Libsodium documentation*. https://libsodium.gitbook.io/doc/. (Accessed on 03/02/2021).

[168] Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, Pierre-Louis Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David Eyers, Rüdiger Kapitza, Christof Fetzer, and Peter Pietzuch. "Glamdring: Automatic Application Partitioning for Intel SGX". In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, July 2017, pp. 285–298. isbn: 978-1-931971-38-6. url: https://www.usenix.org/conference/atc17/technical-sessions/presentation/lind.

[169] Christoph Lippert, Riccardo Sabatini, M. Cyrus Maher, Eun Yong Kang, Seunghak Lee, Okan Arikan, Alena Harley, Axel Bernal, Peter Garst, Victor Lavrenko, Ken Yocum, Theodore Wong, Mingfu Zhu, Wen-Yun Yang, Chris Chang, Tim Lu, Charlie W. H. Lee, Barry Hicks, Smriti Ramakrishnan, Haibao Tang, Chao Xie, Jason Piper, Suzanne Brewerton, Yaron Turpaz, Amalio Telenti, Rhonda K. Roby, Franz J. Och, and J. Craig Venter. "Identification of individuals by trait prediction using whole-genome sequencing data". In: *Proceedings of the National Academy of Sciences* 114.38 (2017), pp. 10166–10171. doi: 10.1073/pnas.1711125114. url: https://www.pnas.org/doi/abs/10.1073/pnas.1711125114.

[170] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. "When machine learning meets privacy: A survey and outlook". In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–36.

[171] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. "Oblivious neural network predictions via minionn transformations". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 619–631.

[172] Tianyi Liu, Xiang Xie, and Yupeng Zhang. “ZkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 2968–2985.

[173] Yunhui Long, Vincent Bindschaedler, and Carl A Gunter. “Towards measuring membership privacy”. In: *arXiv preprint arXiv:1712.09136* (2017).

[174] Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A Gunter, and Kai Chen. “Understanding membership inferences on well-generalized learning models”. In: *arXiv preprint arXiv:1802.04889* (2018).

[175] Oana Lorintiu and Andrea Vassilev. “Transportation mode recognition based on smartphone embedded sensors for carbon footprint estimation”. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2016, pp. 1976–1981.

[176] Wen-Jie Lu, Yoshiji Yamada, and Jun Sakuma. “Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption”. In: *BMC medical informatics and decision making*. Vol. 15. Springer. 2015, pp. 1–8.

[177] Massimiliano Luca, Gianni Barlacchi, Bruno Lepri, and Luca Pappalardo. “A survey on deep learning for human mobility”. In: *ACM Computing Surveys (CSUR)* 55.1 (2021), pp. 1–44.

[178] Lingjuan Lyu, Han Yu, and Qiang Yang. “Threats to federated learning: A survey”. In: *arXiv preprint arXiv:2003.02133* (2020).

[179] Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. “SecretFlow-SPU: A Performant and User-Friendly Framework for Privacy-Preserving Machine Learning”. In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, July 2023, pp. 17–33. isbn: 978-1-939133-35-9. url: `https://www.usenix.org/conference/atc23/presentation/ma`.

[180] Ricardo Macedo, Cláudia Correia, Marco Dantas, Cláudia Brito, Weijia Xu, Yusuke Tanimura, Jason Haga, and Joao Paulo. “The Case for Storage Optimization Decoupling in Deep Learning Frameworks”. In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2021, pp. 649–656.

[181] Bradley Malin. “Re-identification of familial database records”. In: *AMIA annual symposium proceedings*. Vol. 2006. American Medical Informatics Association. 2006, p. 524.

[182] Priyanka Mary Mammen. “Federated learning: Opportunities and challenges”. In: *arXiv preprint arXiv:2101.05428* (2021).

[183] Yunlong Mao, Shanhe Yi, Qun Li, Jinghao Feng, Fengyuan Xu, and Sheng Zhong. “A privacy-preserving deep learning approach for face recognition with edge computing”. In: *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), Boston, MA*. 2018.

[184] Paulo Martins, Leonel Sousa, and Artur Mariano. “A survey on fully homomorphic encryption: An engineering perspective”. In: *ACM Computing Surveys (CSUR)* 50.6 (2017), pp. 1–33.

[185] Dastan Maulud and Adnan M Abdulazeez. “A review on linear regression comprehensive in machine learning”. In: *Journal of Applied Science and Technology Trends* 1.4 (2020), pp. 140–147.

[186] Ruben Mayer and Hans-Arno Jacobsen. “Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools”. In: *CoRR* abs/1903.11314 (2019).

[187] Joelle Mbatchou, Leland Barnard, Joshua Backman, Anthony Marcketta, Jack A. Kosmicki, Andrey Ziyatdinov, Christian Benner, Colm O’Dushlaine, Mathew Barber, Boris Boutkov, Lukas Habegger, Manuel Ferreira, Aris Baras, Jeffrey Reid, Goncalo Abecasis, Evan Maxwell, and Jonathan Marchini. “Computationally efficient whole-genome regression for quantitative and binary traits”. In: *Nature genetics* 53.7 (2021), pp. 1097–1103. doi: 10.1038/s41588-021-00870-7.

[188] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. “Innovative instructions and software model for isolated execution.” In: vol. 10. 1. 2013. url: https://www.eit.lth.se/fileadmin/eit/courses/eitn50/Literature/hasp-2013-innovative-instructions-and-software-model-for-isolated-execution.pdf.

[189] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[190] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. “Learning differentially private recurrent language models”. In: *arXiv preprint arXiv:1710.06963* (2017).

[191] Bunjamin Memishi, Shadi Ibrahim, Marıa S Pérez, and Gabriel Antoniu. “Fault tolerance in MapReduce: A survey”. In: *Resource Management for Big Data Platforms: Algorithms, Modelling, and High-Performance Computing Techniques* (2016), pp. 205–240.

[192] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. “MLlib: machine learning in Apache Spark”. In: vol. 17. 1. JMLR.org, Jan. 2016, pp. 1235–1241.

[193] Hannah Verena Meyer and Ewan Birney. “PhenotypeSimulator: A comprehensive framework for simulating multi-trait, multi-locus genotype to phenotype relationships”. In: *Bioinformatics* 34.17 (2018), pp. 2951–2956.

[194] Silvio Micali, Oded Goldreich, and Avi Wigderson. “How to play any mental game”. In: *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*. ACM New York, NY, USA. 1987, pp. 218–229.

[195] Ilya Mironov. "Rényi differential privacy". In: *2017 IEEE 30th computer security foundations symposium (CSF)*. IEEE. 2017, pp. 263–275.

[196] John C Mitchell and Joe Zimmerman. "Data-oblivious data structures". In: *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2014.

[197] Payman Mohassel and Peter Rindal. "ABY3: A mixed protocol framework for machine learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2018, pp. 35–52.

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

[199] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. "A survey on security and privacy of federated learning". In: *Future Generation Computer Systems* 115 (2021), pp. 619–640.

[200] Nicolas Müller, Daniel Kowatsch, and Konstantin Böttinger. "Data poisoning attacks on regression learning and corresponding defenses". In: *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE. 2020, pp. 80–89.

[201] Moni Naor and Benny Pinkas. "Efficient oblivious transfer protocols". In: *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2001, pp. 448–457.

[202] A Naranyanan and V Shmatikov. "Robust de-anonymization of large datasets". In: *Proceedings of the 2008 IEEE Symposium on Security and Privacy, May 2008*. 2008.

[203] M. Nasr, R. Shokri, and A. Houmansadr. "Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning". In: *2019 2019 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2019, pp. 1021–1035.

[204] Milad Nasr, Reza Shokri, and Amir Houmansadr. "Machine learning with membership privacy using adversarial regularization". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2018, pp. 634–646.

[205] Peter Ney, Luis Ceze, and Tadayoshi Kohno. "Genotype Extraction and False Relative Attacks: Security Risks to Third-Party Genetic Genealogy Services Beyond Identity Inference." In: *NDSS*. 2020.

[206] Maxence Noble, Aurélien Bellet, and Aymeric Dieuleveut. “Differentially Private Federated Learning on Heterogeneous Data”. In: *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. Ed. by Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera. Vol. 151. Proceedings of Machine Learning Research. PMLR, 28–30 Mar 2022, pp. 10110–10145.

[207] Monique Ogburn, Claude Turner, and Pushkar Dahal. “Homomorphic encryption”. In: vol. 20. Elsevier, 2013, pp. 502–509.

[208] Seong Joon Oh, Max Augustin, Mario Fritz, and Bernt Schiele. “Towards Reverse-Engineering Black-Box Neural Networks”. In: *International Conference on Learning Representations*. 2018.

[209] Lucila Ohno-Machado, Xiaoqian Jiang, Tsung-Ting Kuo, Shiqiang Tao, Luyao Chen, Pritham M Ram, Guo-Qiang Zhang, and Hua Xu. “A hierarchical strategy to minimize privacy risk when linking “De-identified” data in biomedical research consortia”. In: *Journal of Biomedical Informatics* 139 (2023), p. 104322.

[210] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. “Oblivious multi-party machine learning on trusted processors”. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 619–636. isbn: 978-1-931971-32-4. url: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko.

[211] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. “Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks”. In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pp. 227–240. isbn: ISBN 978-1-939133-01-4. url: https://www.usenix.org/conference/atc18/presentation/oleksenko.

[212] OpenMined. *The Python Differential Privacy Library. Built on top of: https://github.com/google/differential-privacy*. https://github.com/OpenMined/PyDP. (Accessed on 02/10/2024).

[213] Jonathan Ayebakuro Orama, Assumpció Huertas, Joan Borràs, Antonio Moreno, and Salvador Anton Clavé. “Identification of Mobility Patterns of Clusters of City Visitors: An Application of Artificial Intelligence Techniques to Social Media Data”. In: *Applied Sciences* 12.12 (2022), p. 5834.

[214] Claudio Orlandi. “Is multiparty computation any good in practice?” In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. 2011. isbn: 9781457705397.

[215] Róbert Ormándi, István Hegedüs, and Márk Jelasity. “Gossip learning with linear models on fully distributed data”. In: *Concurrency and Computation: Practice and Experience* 25.4 (2013), pp. 556–571.

[216] Pascal Paillier. “Public-key cryptosystems based on composite degree residuosity classes”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1999, pp. 223–238.

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

[218] Harikumar Pallathadka, Malik Mustafa, Domenic T. Sanchez, Guna Sekhar Sajja, Sanjeev Gour, and Mohd Naved. “IMPACT OF MACHINE learning ON Management, healthcare AND AGRICULTURE”. In: *Materials Today: Proceedings* 80 (2023). SI:5 NANO 2021, pp. 2803–2806. issn: 2214-7853. doi: https://doi.org/10.1016/j.matpr.2021.07.042. url: https://www.sciencedirect.com/science/article/pii/S221478532104894X.

[219] Nicolas Papernot. “A Marauder’s Map of Security and Privacy in Machine Learning”. In: *arXiv preprint arXiv:1811.01134* (2018).

[220] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. “Semi-supervised knowledge transfer for deep learning from private training data”. In: *arXiv preprint arXiv:1610.05755* (2016).

[221] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. “cleverhans v1.0.0: an adversarial machine learning library”. In: *arXiv preprint arXiv:1610.00768* (2016).

[222] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. “Practical Black-Box Attacks Against Machine Learning”. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS ’17. Abu Dhabi, United Arab Emirates: ACM, 2017, pp. 506–519. isbn: 978-1-4503-4944-4.

[223] Túlio Pascoal, Jérémie Decouchant, Antoine Boutet, and Paulo Esteves-Verissimo. “Dyps: Dynamic, private and secure gwas”. In: *Proceedings on Privacy Enhancing Technologies* (2021).

[224] Túlio Pascoal, Jérémie Decouchant, Antoine Boutet, and Marcus Völp. “I-GWAS: Privacy-preserving interdependent genome-wide association studies”. In: *Proceedings on Privacy Enhancing Technologies* 1 (2023), pp. 437–454.

[225] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: 32 (2019). Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. url: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[226] Pitch Patarasuk and Xin Yuan. “Bandwidth optimal all-reduce algorithms for clusters of workstations”. In: *Journal of Parallel and Distributed Computing* 69.2 (2009), pp. 117–124. issn: 0743-7315. doi: 10.1016/j.jpdc.2008.09.002.

[227] Arpita Patra and Ajith Suresh. “BLAZE: Blazing Fast Privacy-Preserving Machine Learning”. In: *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. url: https://www.ndss-symposium.org/ndss-paper/blaze-blazing-fast-privacy-preserving-machine-learning/.

[228] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *J. Mach. Learn. Res.* 12 (2011), pp. 2825–2830. doi: 10.5555/1953048.2078195. url: https://dl.acm.org/doi/10.5555/1953048.2078195.

[229] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. “A framework for efficient and composable oblivious transfer”. In: *Annual international cryptology conference*. Springer. 2008, pp. 554–571.

[230] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. “A survey of methods for distributed machine learning”. In: *Progress in Artificial Intelligence* 2.1 (2013), pp. 1–11.

[231] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. “Privacy-Preserving Deep Learning via Additively Homomorphic Encryption”. In: vol. 13. 5. 2018, pp. 1333–1345. doi: 10.1109/TIFS.2017.2787987.

[232] Noela Pina, Cláudia Brito, Ricardo Vitorino, and Inês Cunha. “Promoting sustainable and personalised travel behaviours while preserving data privacy”. In: *Transportation Research Procedia* 72 (2023), pp. 2768–2775.

[233] Leila Pirhaji, Mehdi Kargar, Armita Sheari, Hadi Poormohammadi, Mehdi Sadeghi, Hamid Pezeshk, and Changiz Eslahchi. “The performances of the chi-square test and complexity measures for signal recognition in biological sequences”. In: *Journal of Theoretical Biology* 251.2 (2008), pp. 380–387.

[234] Matti Pirinen. *GWAS 2: P-values in GWAS*. https://www.mv.helsinki.fi/home/mjxpirin/GWAS_course/material/GWAS2.html. (Accessed on 10/16/2023).

[235] Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Guha Thakurta. “How to dp-fy ml: A practical guide to machine learning with differential privacy”. In: *Journal of Artificial Intelligence Research* 77 (2023), pp. 1113–1201.

[236] Donald E Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen 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.

[237] Christian Priebe, Divya Muthukumaran, Joshua Lind, Huanzhou Zhu, Shujie Cui, Vasily A Sartakov, and Peter Pietzuch. “SGX-LKL: Securing the host OS interface for trusted execution”. In: 2019.

[238] Hong Qin, Debiao He, Qi Feng, Muhammad Khurram Khan, Min Luo, and Kim-Kwang Raymond Choo. “Cryptographic Primitives in Privacy-Preserving Machine Learning: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* (2023).

[239] Michael O Rabin. “How To Exchange Secrets with Oblivious Transfer.” In: *Technical Report TR-81, Aiken Computation Lab, Harvard University* (1981).

[240] S Sundhar Ram, A Nedić, and Venugopal V Veeravalli. “Asynchronous gossip algorithms for stochastic optimization”. In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE. 2009, pp. 3581–3586.

[241] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. “Sirnn: A math library for secure rnn inference”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1003–1020.

[242] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. “Cryptflow2: Practical 2-party secure inference”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 325–342.

[243] Jonas Rauber, Wieland Brendel, and Matthias Bethge. “Foolbox v0.8.0: A Python toolbox to benchmark the robustness of machine learning models”. In: *CoRR* abs/1707.04131 (2017). arXiv: 1707.04131. url: `http://arxiv.org/abs/1707.04131`.

[244] Susmita Ray. “A quick review of machine learning algorithms”. In: *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE. 2019, pp. 35–39.

[245] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. “XONN: XNOR-based oblivious deep neural network inference”. In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 1501–1518.

[246] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. “Chameleon: A hybrid secure computation framework for machine learning applications”. In: *Asia Conference on Computer and Communications Security*. ACM. 2018, pp. 707–721.

[247] Steve Richmond. *Uncovering The True Costs Of IT Infrastructure*. https://www.forbes.com/sites/forbestechcouncil/2021/11/03/uncovering-the-true-costs-of-it-infrastructure/?sh=328de8b67baf. (Accessed on 02/16/2023).

[248] Maria Rigaki and Sebastian Garcia. "A survey of privacy attacks in machine learning". In: *ACM Computing Surveys* 56.4 (2023), pp. 1–34.

[249] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[250] Nuria Rodríguez-Barroso, Daniel Jiménez-López, M Victoria Luzón, Francisco Herrera, and Eugenio Martı nez-Cámara. "Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges". In: *Information Fusion* 90 (2023), pp. 148–173.

[251] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. "Deepsecure - scalable provably-secure deep learning". In: *Proceedings of the 55th Annual Design Automation Conference on - DAC '18*. New York, New York, USA: ACM Press, 2018, pp. 1–6. isbn: 9781450357005.

[252] Mohammad Al-Rubaie and J Morris Chang. "Reconstruction attacks against mobile-based continuous authentication systems in the cloud". In: *IEEE Transactions on Information Forensics and Security* 11.12 (2016), pp. 2648–2663. doi: 10.1109/tifs.2016.2594132.

[253] Mohammad Al-Rubaie and J Morris Chang. "Privacy-preserving machine learning: Threats and solutions". In: IEEE, 2019. doi: 10.1109/msec.2018.2888775.

[254] Benjamin IP Rubinstein, Peter L Bartlett, Ling Huang, and Nina Taft. "Learning in a Large Function Space: Privacy-Preserving Mechanisms for SVM Learning". In: *Journal of Privacy and Confidentiality* 4.1 (2012).

[255] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. "A generic framework for privacy preserving deep learning". In: *arXiv preprint arXiv:1811.04017* (2018).

[256] Md Nazmus Sadat, Md Momin Al Aziz, Noman Mohammed, Feng Chen, Xiaoqian Jiang, and Shuang Wang. "Safety: secure gwas in federated environment through a hybrid solution". In: *IEEE/ACM transactions on computational biology and bioinformatics* 16.1 (2018), pp. 93–102.

[257] TM Sakho and J Ben Othman. "FedVANET-TP: Federated Trajectory Prediction Model for VANETs". In: *2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE. 2023, pp. 1–6.

[258] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and Yang Zhang. "Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1291–1308. isbn: 978-1-939133-17-5. url: https://www.usenix.org/conference/usenixsecurity20/presentation/salem.

[259] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. "ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models". In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. 2019.

[260] Ramit Sawhney, Atula Neerkaje, Ivan Habernal, and Lucie Flek. "How Much User Context Do We Need? Privacy by Design in Mental Health NLP Applications". In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 17. 2023, pp. 766–776.

[261] AMD Sev-Snp. "Strengthening VM isolation with integrity protection and more". In: *White Paper, January* 53 (2020), pp. 1450–1465.

[262] Adi Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[263] Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, and Latifur Khan. "Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors". In: *ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 1211–1228.

[264] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. "Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 955–970. isbn: 9781450371025. doi: 10.1145/3373376.3378469. url: https://doi.org/10.1145/3373376.3378469.

[265] Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. "Panoply: Low-TCB Linux Applications With SGX Enclaves." In: *NDSS*. 2017.

[266] Reza Shokri and Vitaly Shmatikov. "Privacy-preserving deep learning". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 2015, pp. 1310–1321. isbn: 9781509018239.

[267] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. "Membership inference attacks against machine learning models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 3–18.

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

[269] Barton E Slatko, Andrew F Gardner, and Frederick M Ausubel. "Overview of next-generation sequencing technologies". In: *Current protocols in molecular biology* 122.1 (2018), e59.

[270] Alexander Smola and Shravan Narayanamurthy. "An architecture for parallel topic models". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 703–710.

[271] Congzheng Song and Vitaly Shmatikov. “The natural auditor: How to tell if someone used your words to train their model”. In: *arXiv preprint arXiv:1811.00513* (2018).

[272] Christian Spreafico and Davide Russo. “Exploiting the scientific literature for performing life cycle assessment about transportation”. In: *Sustainability* 12.18 (2020), p. 7548.

[273] Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. “Path ORAM: An Extremely Simple Oblivious RAM Protocol”. In: vol. 65. 4. New York, NY, USA: Association for Computing Machinery, Apr. 2018. doi: 10.1145/3177872. url: https://doi.org/10.1145/3177872.

[274] Roelant A Stegmann, Indrė Žliobaitė, Tuukka Tolvanen, Jaakko Hollmén, and Jesse Read. “A survey of evaluation methods for personal route and destination prediction from mobility traces”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.2 (2018), e1237.

[275] Gan Sun, Yang Cong, Jiahua Dong, Qiang Wang, Lingjuan Lyu, and Ji Liu. “Data poisoning attacks on federated machine learning”. In: *IEEE Internet of Things Journal* 9.13 (2021), pp. 11365–11375. doi: 10.1109/JIOT.2021.3128646.

[276] Subham Surana. *Computational Complexity of Machine Learning Models - II | Data Science and Machine Learning | Kaggle*. https://www.kaggle.com/general/263127. (Accessed on 02/06/2023).

[277] TACC. *Texas Advanced Computing Center*. https://www.tacc.utexas.edu/. (Accessed on 02/06/2023).

[278] Hassan Takabi, Ehsan Hesamifard, and Mehdi Ghasemi. “Privacy preserving multi-party machine learning with homomorphic encryption”. In: *29th Annual Conference on Neural Information Processing Systems (NIPS)*. 2016.

[279] Hassan Takabi, James BD Joshi, and Gail-Joon Ahn. “Security and privacy challenges in cloud computing environments”. In: *IEEE Security & Privacy* 8.6 (2010), pp. 24–31.

[280] TensorFlow. *TensorFlow Lite | ML for Mobile and Edge Devices*. https://www.tensorflow.org/lite. 2023.

[281] Truong Thao Nguyen, Mohamed Wahib, and Ryousei Takano. “Efficient MPI-AllReduce for large-scale deep learning on GPU-clusters”. In: *Concurrency and Computation: Practice and Experience* 33.12 (2021), e5574. doi: 10.1002/cpe.5574.

[282] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. “Splitfed: When federated learning meets split learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 8. 2022, pp. 8485–8493.

[283] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. "Data poisoning attacks against federated learning systems". In: *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*. Springer. 2020, pp. 480–501.

[284] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. "Llama 2: Open Foundation and Fine-Tuned Chat Models". In: (2023). arXiv: 2307.09288.

[285] Florian Tramer and Dan Boneh. "Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware". In: *International Conference on Learning Representations*. 2018.

[286] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. "Stealing machine learning models via prediction apis". In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, pp. 601–618.

[287] Aleksei Triastcyn and Boi Faltings. "Federated learning with bayesian differential privacy". In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 2587–2596.

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

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

[290] Chia-Che Tsai, Jeongseok Son, Bhushan Jain, John McAvey, Raluca Ada Popa, and Donald E Porter. "Civet: An efficient java partitioning framework for hardware enclaves". In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 505–522.

[291] Emil Uffelmann, Qin Qin Huang, Nchangwi Syntia Munung, Jantina De Vries, Yukinori Okada, Alicia R Martin, Hilary C Martin, Tuuli Lappalainen, and Danielle Posthuma. "Genome-wide association studies". In: *Nature Reviews Methods Primers* 1.1 (2021), p. 59.

[292] *Veracruz*. `https://github.com/veracruz-project/veracruz/wiki`. (Accessed on 21/11/2020).

[293] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. “A Survey on Distributed Machine Learning”. In: *ACM Comput. Surv.* 53.2 (Mar. 2020). issn: 0360-0300. doi: 10.1145/3377454. url: https://doi.org/10.1145/3377454.

[294] Shrey Verma, Gaurav Dwivedi, and Puneet Verma. “Life cycle assessment of electric vehicles in comparison to combustion engine vehicles: A review”. In: *Materials Today: Proceedings* 49 (2022), pp. 217–222.

[295] Paul Voigt and Axel Von dem Bussche. “The eu general data protection regulation (gdpr)”. In: *A Practical Guide, 1st Ed., Cham: Springer International Publishing* (2017).

[296] Binghui Wang and Neil Zhenqiang Gong. “Stealing hyperparameters in machine learning”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 36–52.

[297] Lin Wang, Hristijan Gjoreski, Mathias Ciliberto, Sami Mekki, Stefan Valentin, and Daniel Roggen. “Enabling reproducible research in sensor-based transportation mode recognition with the Sussex-Huawei dataset”. In: *IEEE Access* 7 (2019), pp. 10870–10891.

[298] Rui Wang, Yong Fuga Li, XiaoFeng Wang, Haixu Tang, and Xiaoyong Zhou. “Learning your identity and disease from research papers: information leaks in genome wide association study”. In: *Proceedings of the 16th ACM conference on Computer and communications security*. 2009, pp. 534–544.

[299] Zhilin Wang, Qiao Kang, Xinyi Zhang, and Qin Hu. “Defense strategies toward model poisoning attacks in federated learning: A survey”. In: *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2022, pp. 548–553. doi: 10.1109/WCNC51071.2022.9771619.

[300] Jinliang Wei, Wei Dai, Aurick Qiao, Qirong Ho, Henggang Cui, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. “Managed Communication and Consistency for Fast Data-parallel Iterative Analytics”. In: *Proceedings of the Sixth ACM Symposium on Cloud Computing*. SoCC ’15. Kohala Coast, Hawaii: ACM, 2015, pp. 381–394. isbn: 978-1-4503-3651-2. doi: 10.1145/2806777.2806778.

[301] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. “Federated learning with differential privacy: Algorithms and performance analysis”. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3454–3469.

[302] Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. “A framework for evaluating client privacy leakages in federated learning”. In: *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*. Springer. 2020, pp. 545–566.

[303] Chathura Widanage, Weijie Liu, Jiayu Li, Hongbo Chen, XiaoFeng Wang, Haixu Tang, and Judy Fox. "HySec-Flow: Privacy-Preserving Genomic Computing with SGX-based Big-Data Analytics Framework". In: *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE. 2021, pp. 733–743.

[304] BigScience Workshop et al. "Bloom: A 176b-parameter open-access multilingual language model". In: *arXiv preprint arXiv:2211.05100* (2022).

[305] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. "Deep image: Scaling up image recognition". In: *arXiv preprint arXiv:1501.02876* (2015).

[306] Eric P. Xing, Qirong Ho, Wei Dai, Jin-Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. "Petuum: A New Platform for Distributed Machine Learning on Big Data". In: KDD '15 (2015), pp. 1335–1344.

[307] Runhua Xu, Nathalie Baracaldo, and James Joshi. "Privacy-preserving machine learning: Methods, challenges and directions". In: *arXiv preprint arXiv:2108.04417* (2021).

[308] yahoo. *TensorFlowOnSpark: TensorFlowOnSpark brings TensorFlow programs to Apache Spark clusters.* `https://github.com/yahoo/TensorFlowOnSpark`. (Accessed on 05/02/2023).

[309] Andrew Chi-Chih Yao. "How to generate and exchange secrets". In: *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE. 1986, pp. 162–167.

[310] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. "Privacy risk in machine learning: Analyzing the connection to overfitting". In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE. 2018, pp. 268–282.

[311] Fuxun Yu, Zirui Xu, Zhuwei Qin, and Xiang Chen. "Privacy-preserving federated learning for transportation mode prediction based on personal mobility data". In: *High-Confidence Computing* 2.4 (2022), p. 100082.

[312] Peterson Yuhala, Pascal Felber, Hugo Guiroux, Jean-Pierre Lozi, Alain Tchana, Valerio Schiavoni, and Gaël Thomas. "SecV: Secure Code Partitioning via Multi-Language Secure Values". In: *arXiv preprint arXiv:2310.15582* (2023).

[313] Peterson Yuhala, Jämes Ménétrey, Pascal Felber, Valerio Schiavoni, Alain Tchana, Gaël Thomas, Hugo Guiroux, and Jean-Pierre Lozi. "Montsalvat: Intel SGX shielding for GraalVM native images". In: *Proceedings of the 22nd International Middleware Conference*. 2021, pp. 352–364.

[314] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. "Spark: Cluster computing with working sets". In: *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*. 2010.

[315] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. "Apache Spark: a unified engine for big data processing". In: vol. 59. 11. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 56–65. doi: 10.1145/2934664. url: https://doi.org/10.1145/2934664.

[316] Yahua Zhang, Anming Zhang, and Jiaoe Wang. "Exploring the roles of high-speed train, air and coach services in the spread of COVID-19 in China". In: *Transport Policy* 94 (2020), pp. 34–42.

[317] Yi Zhang, Yunfan Lu, and Fengxia Liu. "A systematic survey for differential privacy techniques in federated learning". In: *Journal of Information Security* 14.2 (2023), pp. 111–135.

[318] ChongChong Zhao, Daniyaer Saifuding, Hongliang Tian, Yong Zhang, and ChunXiao Xing. "On the Performance of Intel SGX". In: *2016 13th Web Information Systems and Applications Conference (WISA)*. 2016, pp. 184–187. doi: 10.1109/WISA.2016.45.

[319] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. "Veriml: Enabling integrity assurances and fair payments for machine learning as a service". In: *IEEE Transactions on Parallel and Distributed Systems* 32.10 (2021), pp. 2524–2540.

[320] Wei Zheng, Ying Wu, Xiaoxue Wu, Chen Feng, Yulei Sui, Xiapu Luo, and Yajin Zhou. "A survey of Intel SGX and its applications". In: *Frontiers of Computer Science* 15 (2021), pp. 1–15.

[321] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. "Opaque: An Oblivious and Encrypted Distributed Analytics Platform". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 283–298. isbn: 978-1-931971-37-9. url: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng.

[322] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. "Helen: Maliciously secure coopetitive learning for linear models". In: *2019 IEEE symposium on security and privacy (SP)*. IEEE. 2019, pp. 724–738.

[323] Yu Zheng, Xing Xie, and Wei-Ying Ma. "GeoLife: A collaborative social networking service among user, location and trajectory." In: *IEEE Data Eng. Bull.* 33.2 (2010), pp. 32–39.

[324] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021.

[325] Rui Zhu, Chao Jiang, Xiaofeng Wang, Shuang Wang, Hao Zheng, and Haixu Tang. "Privacy-preserving construction of generalized linear mixed model for biomedical computation". In: *Bioinformatics* 36.Supplement_1 (2020), pp. i128–i135.

[326] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. “Parallelized Stochastic Gradient Descent”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc., 2010. url: https://proceedings.neurips.cc/paper_files/paper/2010/file/abea47ba24142ed16b7d8fbf2c740e0d-Paper.pdf.

[327] Justin M. Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher E. Mason, Noah Alexander, Elizabeth Henaff, Alexa B.R. McIntyre, Dhruva Chandramohan, Feng Chen, Erich Jaeger, Ali Moshrefi, Khoa Pham, William Stedman, Tiffany Liang, Michael Saghbini, Zeljko Dzakula, Alex Hastie, Han Cao, Gintaras Deikus, Eric Schadt, Robert Sebra, Ali Bashir, Rebecca M. Truty, Christopher C. Chang, Natali Gulbahce, Keyan Zhao, Srinka Ghosh, Fiona Hyland, Yutao Fu, Mark Chaisson, Chunlin Xiao, Jonathan Trow, Stephen T. Sherry, Alexander W. Zaranek, Madeleine Ball, Jason Bobe, Preston Estep, George M. Church, Patrick Marks, Sofia Kyriazopoulou-Panagiotopoulou, Grace X.Y. Zheng, Michael Schnall-Levin, Heather S. Ordonez, Patrice A. Mudivarti, Kristina Giorda, Ying Sheng, Karoline Bjarnesdatter Rypdal, and Marc Salit. “Extensive sequencing of seven human genomes to characterize benchmark reference materials”. English (US). In: *Scientific Data* 3 (June 2016). Funding Information: National Institutes of Health (R25EB020393, R01NS076465). issn: 2052-4463. doi: 10.1038/sdata.2016.25.

[328] Hui Zou, Trevor Hastie, and Robert Tibshirani. “Sparse principal component analysis”. In: vol. 15. 2. Taylor & Francis, 2006, pp. 265–286.

A

Appendix 1 - Soteria Security Proofs

A.1 Soteria Security Proof

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

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

Formally, this security goal is defined using the real-versus-ideal world paradigm, similar to the Universal Composability [44] framework.

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

A.1.1 Formal Security Model

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

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

Figure A.1: Secure external storage setup.

leakage function $\mathcal{L}$.[1]

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

In the ideal world, (s, m) are provided to dummy Client C, which in turn forwards them to $\mathcal{F}$<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 output size.

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

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

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

A.1.2 Intermediate Result

For convenience, Soteria does not require the Client to provide input data at the time of the ML processing, and instead, the Workers are given access to external storage from which they retrieve the data. When discussing the security in the context of secure outsourced computation for SGX, this is functionally equivalent to classical scenarios where the Client provides these inputs via a secure channel (Theorem 3 in [23]). The reasoning is simply that if a protocol securely realizes a functionality with a given input provided via a secure channel, then the same functionality can be securely realized with the same input fixed in an external storage, securely accessed by the enclave.

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

[2] 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 $\mathcal{F}$.

Consider a protocol Π_1 that securely realises some functionality $\mathcal{F}$ with simulator $\mathcal{S}_1$ according to Theorem 3 of [44]. 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 an encrypted fashion (Figure A.1 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 [23]. Then Π_2, constructed as discussed above, securely realises $\mathcal{F}$ according to Definition A.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: $\mathcal{S}_1.\mathrm{AC}()$ will produce the view for establishing a secure channel, while $\mathcal{S}_1.\mathrm{Send}(l)$ 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 $\mathcal{S}_2.\mathrm{Get}(l)$ to simulate information being retrieved from G, given its length. Our simulator is depicted in Figure A.2.

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

Figure A.2: Simulator for Π_2.

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

- Messages exchanged during the secure channel establishment are exactly the same as in 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

$$\mathrm{Adv}^{\mathrm{Dist}}{}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) = \tag{A.1}$$

$$\Pr[\mathrm{Real}^{G}_{\mathcal{Z},\mathcal{A},\Pi_2} \Rightarrow \mathrm{T}] - \Pr[\mathrm{Ideal}^{G}_{\mathcal{Z},\mathcal{A},\mathcal{S}_2,\mathcal{F}} \Rightarrow \mathrm{T}] \tag{A.2}$$

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

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

$$\leq \mu() \tag{A.4}$$

and Theorem 1 follows.

A.1.3 Soteria Client, Master and Workers

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

Secure channels are established with enclaves. We define $\mathrm{init}(P)$ 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 $\mathrm{receive}$. Untrusted storage is not protected with secure channels and can be accessed using the call $\mathrm{uGet}(G, m)$, which retrieves data from G considering manifest file m. Concretely, this is achieved using the open-source library Gramine, which we assume to implement this mechanism correctly. Finally, the script s defines the actual computation that must be performed by the system and will be executed collaboratively with both Workers. As such, we define s as a stateful object with the main method $\mathrm{Run}(\mathrm{id}, i_1, i_2)$, 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 $\mathrm{Complete}$ that returns T if the task is complete, or F otherwise.

The Soteria components can be analyzed in Figure A.3 and are as follows. The Client C (left of Figure A.3) 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 A.3) 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.[3] Worker W_1 (right of Figure A.3) receives the parameters from M and starts processing the script: retrieves

[3] 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.

encrypted data from G, decrypts, processes and exchanges intermediate results with the other Worker. When the script is concluded, it returns its output to M. The behavior of W_2 is the same, but the connection is established instead with W_1.

Algorithm $C(m, s, k)$	**Algorithm** $M()$	**Algorithm** $W_1()$
$\text{sc} \leftarrow \text{init}(M)$	$\text{sc}_c \leftarrow \text{init}(C)$	$m \leftarrow \epsilon$
$\text{sc.send}(m, s, k)$	$(m, s, k) \leftarrow \text{sc}_c.\text{receive}()$	$\text{sc}_m \leftarrow \text{init}(M)$
$o \leftarrow \text{sc.receive}()$	$\text{sc}_1 \leftarrow \text{init}(W_1)$	$(m, s, k) \leftarrow \text{sc}_m.\text{receive}()$
Return o	$\text{sc}_1.\text{send}(m, s, k)$	$\text{sc}_w \leftarrow \text{init}(W_2)$
	$\text{sc}_2 \leftarrow \text{init}(W_2)$	While $!s.\text{Complete}$:
	$\text{sc}_2.\text{send}(m, s, k)$	$c \leftarrow \text{uGet}(G, m)$
	$o_1 \leftarrow \text{sc}_1.\text{receive}()$	$i \leftarrow \Theta.\text{Dec}(k, c)$
	$o_2 \leftarrow \text{sc}_2.\text{receive}()$	$(o, m) \leftarrow\$\ s.\text{Run}(W_1, i, \epsilon)$
	$\text{sc}_c.\text{send}((o_1, o_2))$	$\text{sc}_w.\text{send}(m)$
		$m \leftarrow \text{sc}_w.\text{receive}()$
		$\text{sc}_m.\text{send}(o)$

Figure A.3: Soteria Components. Client C (left), Master node M (middle), and Worker node 1 W (right).

A.1.4 Full Proof

Given the description of Soteria components in Figure A.3, the Soteria protocol Π_{xyz} is straightforward to describe. Considering a pre-encrypted storage G, the Client C, Master M, and Workers W_1, W_2 execute following their respective specifications. Our theorem for the security of Soteria is as follows.

Theorem 2. Π_{xyz}*, assuming the setup of Figure A.1 and constructed as discussed above, securely realises* $\mathcal{F}$ *according to Definition A.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 A.4 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 A.1. From Theorem 1, we can establish that this adaptation entails negligible advantage to $\mathcal{A}$ provided that the protocol without external

Algorithm $W_1()$	**Algorithm** $W_2()$
$m \leftarrow \epsilon$	$m \leftarrow \epsilon$
$\mathrm{sc}_m \leftarrow \mathrm{init}(M)$	$\mathrm{sc}_m \leftarrow \mathrm{init}(M)$
$(m, s, k) \leftarrow \mathrm{sc}_m.\mathrm{receive}()$	$(m, s, k) \leftarrow \mathrm{sc}_m.\mathrm{receive}()$
$\mathrm{sc}_w \leftarrow \mathrm{init}(W_2)$	$\mathrm{sc}_w \leftarrow \mathrm{init}(W_1)$
While $!s.\mathrm{Complete}$:	While $!s.\mathrm{Complete}$:
$c \leftarrow \mathrm{uGet}(G_1, m)$	$c \leftarrow \mathrm{uGet}(G_2, m)$
$i \leftarrow \Theta.\mathrm{Dec}(k, c)$	$i \leftarrow \Theta.\mathrm{Dec}(k, c)$
$(o, m) \leftarrow\$\ s.\mathrm{Run}(W_1, i, \epsilon)$	$(o, m) \leftarrow\$\ s.\mathrm{Run}(W_1, i, \epsilon)$
$\mathrm{sc}_w.\mathrm{send}(m)$	$\mathrm{sc}_w.\mathrm{send}(m)$
$m \leftarrow \mathrm{sc}_w.\mathrm{receive}()$	$m \leftarrow \mathrm{sc}_w.\mathrm{receive}()$
$\mathrm{sc}_m.\mathrm{send}(o)$	$\mathrm{sc}_m.\mathrm{send}(o)$

Figure A.4: Soteria Workers with split storage.

access realizes the same functionality. However, this is necessarily the case, as it follows the exact structure as the constructions in [23]. We can repeat this process for W_2.[4] As such, using the intermediate result, we can thus upper bound the advantage adversary to distinguish these two scenarios by applying twice the result of Theorem 1.

The third step replaces the Master with an ideal functionality $\mathcal{F}_M$ that simply forwards requests to the Worker functionalities. This one follows the same logic as the previous one, without requiring external storage, as the protocol also follows the exact structure as the constructions in [23].

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 into a single functionality $\mathcal{F}$ for ML script processing. This can be done by constructing a big simulator S that builds upon the simulators for the individual components (S_M, S_{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 A.1.

[4] 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.

Let

$$\mathrm{Adv}^{\mathrm{Dist}}{}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) = \tag{A.5}$$

$$\mathrm{Pr}[\mathrm{Real}^{G}_{\mathcal{Z},\mathcal{A},\Pi_{\mathrm{xyz}}} \Rightarrow \mathrm{T}] - \mathrm{Pr}[\mathrm{Ideal}^{G}_{\mathcal{Z},\mathcal{A},\mathcal{S},\mathcal{F}} \Rightarrow \mathrm{T}] \tag{A.6}$$

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

$$\mathrm{Adv}^{\mathrm{Dist}}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) = 2 \cdot \mathrm{Adv}^{\mathrm{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_{W1},\mathcal{S}_{W1},\mathcal{F}_{W1}}(G) \tag{A.7}$$

$$+ \mathrm{Adv}^{\mathrm{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_{M},\mathcal{S}_{M},\mathcal{F}_{M}}() \tag{A.8}$$

$$\leq \mu() \tag{A.9}$$

and Theorem 2 follows.

A.2 ML Workflow attacks

This section presents the attacks in Section 3.2 in further detail and argues in which circumstances Soteria is secure against each attack. First, we will describe a general adversarial model against Soteria that follows the security restrictions justified in Appendix A.1. Then, we will present an experiment that captures what constitutes a valid attack under each definition, as described in Section 3.1. For each attack, we consider our protocol to be secure if we can demonstrate that one cannot rely on a valid adversary against the experiment of said attack under the constraints of Soteria. In some instances, this will depend on known attack limitations, which we detail case-by-case.

A.2.1 Attacker against Soteria

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

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

Let Π_{xyz} denote the full training protocol of Soteria. It receives external storage G and task script s as inputs, and produces a model m, which can then be queried. Based on the security result of Appendix A.1, the interaction of an adversary with our system can be described in Figure A.5. The adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ can first try and manipulate the input dataset G to G'. This is then used for Π_{xyz}, which will produce the model m and the additional leakage l (Soteria-B has no additional leakage, so $l = \epsilon$). Finally,

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

Figure A.5: Adversary interacting with Soteria.
ff

the adversary can interact black-box with the model until a conclusion r is produced. This will be provided to a $\mathrm{Success}$ predicate, which will state if the attack was successful. This predicate is specific to the attack and allows us to generally describe attacks such as *adversarial samples*, where the goal is to make the resulting model deviate, as well as *membership inference*, where the goal is to retrieve information from the original dataset.

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

A.2.2 Dataset manipulation

Dataset manipulation attacks are defined by an adversary with the capability of inserting, removing or manipulating dataset information. These align with the setting considered for attacks via *adversarial samples*. Figure A.6 is an experiment that describes what constitutes a successful attack for dataset manipulation. The adversary $\mathcal{A}$ is given full knowledge of G[5], and must produce an alternative input dataset G'. We then train the model (protocol Π) over that data to produce model m, and the adversary is successful if said model satisfies some attack success criteria $\mathrm{T/F} \leftarrow \mathrm{Success}$.

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

Figure A.6: Model for dataset manipulation attack.

We now argue that the integrity guarantees of the authenticated encryption used by our external storage G ensure that these attacks do not occur for Π_{xyz}. We do this by showing that any adversary that performs an *adversarial samples* attack on Π_{xyz} can be used to construct a successful attack on the security of the authenticated encryption scheme. First, observe that no attack can be successful if the adversary makes

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

no changes on the input dataset, so if $G = G'$, then $\mathrm{F} \leftarrow \mathrm{Success}$. Furthermore, if Π_{xyz} aborts, then no model is produced, so it naturally follows that the attack is unsuccessful $\mathrm{F} \leftarrow \mathrm{Success}$.[6]
As such, the only cases in which $\mathrm{T} \leftarrow \mathrm{Success}$ are those in which $G' \neq G$ and Π_{xyz} do not abort. But this means that the adversary was able to forge an input that correctly decrypts, breaking the integrity of the underlying encryption scheme. Since the security guarantees of authenticated encryption ensure that the probability of existing such an adversary is negligible, the probability of such an attacker in Soteria will also be negligible.

A.2.3 Black-box Attacks

All the remaining attacks, with the exception of some *reconstruction attacks*, follow a similar setting, where the adversary leverages a black-box access to the trained model, depicted in the experiment of Figure A.7. We begin by running Π to produce our model and leakage and then run an additional procedure $\mathrm{Extract}$ to obtain additional information from the original dataset, which cannot be retrieved by simply querying the model. This procedure captures whatever knowledge regarding the underlying ML training might be necessary for the attack to be successful (e.g., information about data features). We then provide this additional information to the adversary and give it black-box access to the model. The success criteria depends on the specific attack and is validated with respect to the original dataset, model, and the task script being run. E.g., for *model extraction* attack, the goal might be to present a model m' that is similar to m, evaluated by the $\mathrm{Success}$ predicate.
For simplicity, we first exclude all attacks for an external adversary, which does not have black-box access to the model of Soteria. This is true if we can show that one cannot emulate black-box access to the model using confidence values and class probabilities. Albeit an interesting research topic, current attacks are still unable to do this in an efficient way [50]. We now go case-by-case, assuming an adversary can play the role of a genuine client in our system.
Our arguments for Π_{xyz} depend on being able to rely on a successful adversary $\mathcal{A}$ of Figure A.7 to perform the same attack in Figure A.5. As such, the security of our system will depend on the amount of additional information z, on how it can be extracted from the view of the adversary of Soteria.

- For *membership inference*, *reconstruction attacks* based on black-box access to the model, *model inversion*, and *model extraction* via *data-free knowledge distillation*, no additional information z is required. This means that any successful adversary in Figure A.7 will also be successful in Figure A.5, meaning that both for Soteria-B and Soteria-P are vulnerable. Preventing these attacks requires restricting access to the model to untrusted participants.

- *Class-only* attacks for *model extraction* require additional knowledge from the dataset. Specifically, z must contain concrete training dataset samples. This means that to leverage such an adversary

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

$\mathcal{A}$, one must first be able to use $\mathcal{A}_2^m(l)$ to extract such a z. This exactly matches the setting of *model inversion* attacks. This means that Soteria is vulnerable to *class-only* attacks for *model extraction* in Soteria-B or Soteria-P if there is also an efficient attack for *model inversion* in Soteria-B or Soteria-P, respectively.

- *Equation-solving model extraction* requires knowledge of the dimension of the training dataset G. This is additional information z that is not revealed by querying the model, which means no adversary $\mathcal{A}_2^m(\epsilon)$ can retrieve z, and thus Soteria-B is secure against said attacks. However, combining public data with confidence values might allow for $\mathcal{A}_2^m(l)$ to extract a sufficient z to perform the attack, which makes Soteria-P vulnerable to *equation-solving model extraction*.

- *Path-finding model extraction* attacks require information regarding leaf count, tree depth and leaf ID. As such, all this must be encapsulated in z. [286] suggests that such information is not retrievable from only black-box access to the model [286], which means no adversary $\mathcal{A}_2^m(\epsilon)$ can produce z and thus Soteria-B is secure. However, this is information that can be extracted from confidence values, which suggests that an efficient adversary $z \leftarrow\$ \mathcal{A}_2^m(l)$ is likely to exist, and thus Soteria-P is vulnerable to such attacks under these assumptions.

We can generalize the security of our system to these types of attacks as follows. If no additional information z is required, then Π_{xyz} is vulnerable to an adversary that can play the role of an honest client. If z can be extracted from black-box access to the model, then we can still rely on said adversary to attack Π_{xyz}. Otherwise, Soteria-B is secure, as no additional information is leaked. Furthermore, the security of Soteria-P will depend on whether one can infer z from l and from the black-box access to the model. Concretely, if we can show that no (efficient) function F exists, such that $z \leftarrow\$ \mathrm{F}^m(l)$, then Π_{xyz} for leakage l is secure against attacks requiring additional data z.

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

Figure A.7: Model for black-box attacks.

A.2.4 White-box Attacks

White-box attacks capture a scenario where an adversary requires white-box access to the model. These align with the setting of *reconstruction attacks* that explicitly require white-box access to the model. Figure A.8 is an experiment that describes what constitutes a successful *reconstruction attack* in this context.

We begin by training the model (protocol Π) over the original dataset to produce the model. We then provide the trained model directly to the adversary, which will reconstruct raw data r. Finally, the success of the attack is validated with respect to the original dataset.

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

Figure A.8: Model for white-box attacks.

We now argue that these attacks do not occur for Π_{xyz}, as long as it is not possible to extract the model from the confidence values and from black-box access to the model. This is because the attacker of Figure A.8 receives explicitly the model m, whereas the adversary $\mathcal{A}_2$ in Figure A.5 receives the confidence values in l, and black-box access to the model. To rely on such an attacker, $\mathcal{A}_2$ must therefore be able to produce input m from its own view of the system. As such, relying on such an adversary implies there is an efficient way $m \leftarrow\$\ \mathcal{A}^m_{\mathrm{bb}}(l)$ to retrieve the model m from confidence values l and black-box access to the model m, which is exactly the setting of *model extraction* attacks in the previous section. This adversary $\mathcal{A}_{\mathrm{bb}}$ can then be called by $\mathcal{A}_2$ to produce input m, which is then forwarded to the adversary of Figure A.8 to produce a successful attack r. As such, Soteria is vulnerable to white-box *reconstruction attacks* if there exists an efficient adversary $\mathcal{A}'$ that successfully wins the experiment of Figure A.7 for the *model extraction* attack.

A.2.5 Summary

Table A.1 summarizes the attacks discussed. These are divided between all the identified classes of attacks, as well as whether the adversary is external or if it can query the model as a client. In many instances, the security of our system hinges on another argument over specific restrictions assumed for the adversary.
We now list the arguments that propose the security of our system in different contexts.

{1}: an adversary is unable to retrieve the (white-box) model from confidence values {1a}, black-box access to the model {1b}, or both {1c}.

{2}: an adversary is unable to emulate black-box access to the model from confidence values.

{3}: an adversary is unable to retrieve the dimension of the dataset from black-box access to the model {3a} and confidence values {3b}.

{4}: an adversary is unable to retrieve information of leaf count, depth, and ID from black-box access to the model {4a} and confidence values {4b}.

{5}: no *model inversion* attack exists for retrieving dataset samples for Soteria-B {5a} and Soteria-P {5b}.

White-box-based attacks explicitly require additional information, such as feature vectors, over black-box access to the model [258]. This is something that supports {1b} directly, and since confidence values are not computed from feature vectors, so would be {1a} and {1c}. Extracting model access from only confidence values is an active area of research, but current attacks [50] are still unable to do this in an efficient way {2}. Typically, one cannot infer dimension from simply querying the model, which suggests {3a} is true, but this is unclear for confidence values, and thus one might consider {3b} is false. [286] suggests {4a} is true, but {4b} is not. {5} will fundamentally depend on the application [50, 286].

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

Table A.1: Summary of attacks against Soteria. ✓ means Soteria is resilient to the attacks, ✗ means Soteria is vulnerable to the attacks, and {X} means Soteria is secure if argument {X} is also true.