

# ExclaveFL: Verifiable Claims for Federated Learning using Exclaves

Jinnan Guo  
Imperial College London  
London, United Kingdom  
jinnan.guo@imperial.ac.uk

Andrew Paverd  
Microsoft Security Response Center  
Cambridge, United Kingdom  
andrew.paverd@microsoft.com

Kapil Vaswani  
Indian Institute of Science Bangalore  
SPARC  
Bengaluru, India  
kapil.vaswani@sparc-labs.in

Peter Pietzuch  
Imperial College London  
London, United Kingdom  
prp@imperial.ac.uk

## Abstract

In federated learning (FL), data providers jointly train a machine learning model without sharing training data. This makes it challenging to provide verifiable claims about the trained model, e.g., related to the employed training data, data sanitization, or training algorithm—a malicious data provider can simply deviate from the correct training protocol without detection. Prior FL training systems have used trusted execution environments (TEEs) to protect the training computation, relying on the confidentiality and integrity of TEEs. TEE confidentiality, however, has been shown to be vulnerable to various attacks, e.g., via side channels.

We describe EXCLAVEFL, a system for training FL models that establishes verifiable claims about models through fine-grained runtime attestation proofs. Since such proofs only require integrity, EXCLAVEFL generates them using *exclaves*, which are integrity-only execution environments without software-managed secrets, making them immune to data leakage attacks. EXCLAVEFL uses exclaves to attest data transformations during FL training, and the runtime attestation proofs then form an attested dataflow graph of the entire training computation. The graph is checked by an auditor to ensure that the trained model satisfies claims, e.g., about the use of data sanitization by data providers or aggregation by the model provider. Since exclaves require minor yet proprietary firmware changes, we evaluate a variant with similar performance but slightly weaker security guarantees on existing AMD SEV-SNP hardware. We show that EXCLAVEFL introduces less than 12% overhead compared to unprotected FL model training.

## CCS Concepts

• **Security and privacy** → *Systems security*; • **Computing methodologies** → *Machine learning*; *Distributed artificial intelligence*.

## Keywords

Federated learning, trusted execution, verifiable computing

## ACM Reference Format:

Jinnan Guo, Kapil Vaswani, Andrew Paverd, and Peter Pietzuch. 2026. ExclaveFL: Verifiable Claims for Federated Learning using Exclaves. In *ACM Symposium on Cloud Computing (SoCC '26)*, November 18–20, 2026, Singapore, Singapore. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3815789.3827953>

## 1 Introduction

As machine learning (ML) models are used in new and diverse settings, there is an increasing impetus towards providing information about how models were trained. For example, the EU AI Act [21] requires providers of general-purpose ML models to provide “*information on the data used for training, [...] including the type and provenance of data and curation methodologies (e.g. cleaning, filtering etc.), the number of data points, their scope and main characteristics; how the data was obtained and selected as well as all other measures to detect the unsuitability of data sources and methods to detect identifiable biases*”. In response to these requirements, *datasheets for datasets* [26] describe the composition of training datasets, and *model cards* [54] offer information about model training. Providers of open-source models, e.g., Apertus [31], publish the training datasets and code; providers of closed-source models also make claims: e.g., the model card for OpenAI’s GPT-5 describes how the training data was pre-processed to reduce personal information [60]. Such claims are also relevant to model providers: e.g., to demonstrate to regulators that a model has *not* been trained on personal or copyrighted data, or to show that a model *has* been trained on certain content, e.g., under-represented languages [31].

Currently, such claims about datasets and model training are made without proof. For closed-source models, it is impossible to verify claims when interacting with the model via an inference API. Even for open-source models, verifying that a given model is indeed the result of executing the provided training code on a given dataset would require repeating the entire training process, and the trained model may still differ due to non-determinism. Therefore, model claims are typically accepted based on reputational trust.

With *federated learning* (FL) [34, 37, 49], the situation becomes worse, because training is performed by (potentially many) *data providers*, rather than a single model provider. Claims about models trained using FL thus require reputational trust in all data providers, as well as the model provider. Yet FL is still appealing in many



This work is licensed under a Creative Commons Attribution 4.0 International License. SoCC '26, Singapore, Singapore

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2735-1/26/11

<https://doi.org/10.1145/3815789.3827953>

settings because it allows data providers to retain control over their datasets. Its use has been proposed for applications ranging from training predictive language models on user input [30, 67], to training fraud detection models while protecting transaction data of competing financial institutions [85].

In this paper, we explore whether it is possible to *provide verifiable claims about how an FL model was trained*. We observe that this would require systems support during training to discover deviations from the agreed training protocol that would invalidate specific claims. For example, the system would have to discover if a data provider skips a pre-processing step that removes personal information from a dataset. There could be various reasons for such deviations, ranging from accidental (e.g., bugs in the training code) to actively malicious (e.g., skipping steps to reduce costs).

A strawman design for creating an FL training system with verifiable claims would be to execute the training computation inside *trusted execution environments* (TEEs), such as Intel SGX enclaves [5] or AMD SEV-SNP confidential virtual machines (VMs) [3]. TEEs ensure that the integrity and confidentiality of computation are preserved by the hardware: TEE memory is isolated, and TEEs offer attestation proofs, which contain a signed measurement of the TEE memory taken at deployment time. Indeed, prior designs of FL training systems have used TEEs to offer claims about trained models through attestation proofs [28, 65, 68].

We argue, however, that TEEs are the wrong abstraction to create verifiable claims about FL model training for two reasons:

**(1) Focus on static attestation proofs.** In TEE-based FL training systems [28, 65, 68], TEEs take memory measurements at deployment time to produce *static* attestation proofs, which establish that the TEE runs the expected FL system and training code. A model provider can check each data provider’s static attestation, e.g., when establishing an initial authenticated network connection [68]. Such static attestation proofs, however, are hard to relate to claims about the final trained FL model: verifiable claims depend on how the training process was carried out at runtime, but a static attestation proof only captures the state at the start of the training process.

**(2) Reliance on confidentiality.** TEE implementations typically require confidentiality to provide unforgeable attestation proofs. Therefore, a distributed TEE-based FL training system that relies on attested TLS connections [28] would be vulnerable to an adversarial data provider that leaks the attestation-endorsed TLS session key from their TEE: this would allow them to deviate from the correct training protocol without detection by providing a correct attestation. TEEs strive for confidentiality by isolating TEE memory from external access. In practice, this is challenging due to advanced confidentiality attacks, e.g., using side channels [45, 72, 80].

We conclude that FL training should instead rely on *runtime* attestation proofs about *data transformations* during FL training. The fact that data transformations, such as data sanitization [61], local training [37], poisoning checks [24, 36, 86], and model aggregation [86], have been carried out faithfully can be related to claims about the final trained FL model. Importantly, these runtime attestation proofs only require guarantees about the *integrity* of data transformations and not their confidentiality. Our approach can thus avoid the more challenging security model of TEEs, which must protect against both confidentiality and integrity attacks.

To realize the above approach, our work makes the following two contributions:

**(1)** We describe a new abstraction called an **exclave**, which is a hardware-based *integrity-only execution environment*. The security guarantees of exclaves do not depend on keeping data secret from the exclave operator. Unlike in TEE implementations, the integrity of exclaves is still preserved *even if the exclave operator reads all data within the exclave*, either directly or through a side channel. Exclaves support attestation through a hardware-based mechanism (e.g., a security co-processor [3]) to measure code integrity and sign attestation proofs at runtime, which can include runtime data. The attestation signing keys are used and maintained by the hardware only, and are never accessible to software, making exclaves more resilient to data leakage attacks. We show how exclaves can be realized through minor modifications of existing TEE implementations.

**(2)** With the help of exclaves, we describe the design and implementation of **EXCLAVEFL**, a system for executing FL training jobs that provides verifiable claims about trained FL models. EXCLAVEFL uses exclaves to generate runtime attestation proofs of all executed data transformations in the FL job: this covers the transformations performed by data providers and the aggregations performed by the model provider in each training round. Each attestation proof, which we call an *exclave data record* (EDR), binds together (through hashes): (i) the data inputs to a transformation; (ii) the code performing a transformation; and (iii) the resulting data output of a transformation. EDRs are collected and used to construct an end-to-end *exclave dataflow graph* (EDG) that then establishes global properties. The EDG provides a complete and integrity-protected representation of all data transformations that led to a final trained FL model. It can be used by an auditor to verify claims about the FL model, e.g., that data pre-processing and model training have been carried out faithfully by data providers and that the model provider has aggregated the contributions of all data providers without bias.

We create a prototype implementation<sup>1</sup> of EXCLAVEFL that extends an existing FL framework, NVFlare [70], to execute all FL data transformations inside exclaves. EXCLAVEFL achieves this without requiring changes to the framework implementation. Since our exclave design requires minor modifications to proprietary TEE firmware, we implement an exclave variant on AMD SEV-SNP using a virtual TPM [53], which offers similar performance but with slightly weaker security guarantees.

Our experiments show that EXCLAVEFL can produce verifiable claims with low overhead: the execution of an FL training job within exclaves and the generation of runtime attestation proofs reduce training throughput by less than 12% compared to a baseline without any security guarantees.

## 2 Verifiable Claims in Federated Learning

We provide background on federated learning (§2.1) and discuss the challenges when providing verifiable claims about trained models (§2.2). We then introduce trusted execution environments (§2.3) and survey whether such approaches (and other techniques) can provide suitable verifiable claims (§2.4). Finally, we describe the threat model and security requirements for our desired solution (§2.5).

<sup>1</sup><https://github.com/microsoft/exclave-fl>

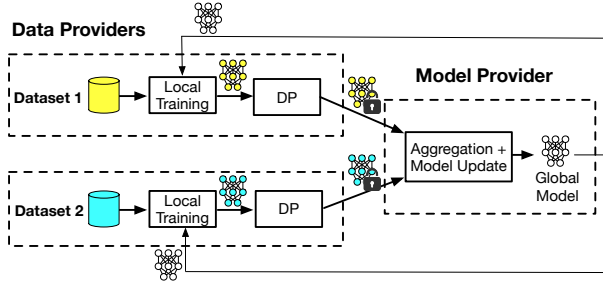


Figure 1: Overview of federated learning

## 2.1 Federated learning

Federated learning (FL) trains machine learning (ML) models while allowing data providers to retain control over their data. While FL applies to many scenarios, we consider the specific case of collaboration between multiple participants who wish to train models jointly using proprietary datasets, either for commercial gains, or to obtain models for their own use. As shown in Fig. 1, FL participants jointly train a model: *data providers* perform training locally using their own datasets and periodically send updates (either gradients or models) to a centralized *model provider*. The model provider aggregates the updates and distributes updated model versions to the data providers. This process repeats for a fixed number of rounds until the model converges. Techniques exist to protect the datasets or the quality of the trained FL model. For example, data providers may apply *differential privacy* (DP) [20] to bound the contribution of individual data records in the training dataset. To ensure the quality of the trained FL model, FL jobs may include algorithmic approaches that detect or mitigate poisoning attacks [77], e.g., using outlier detection [77] or robust aggregation [8, 86].

## 2.2 Claims about trained FL models

It is an increasing requirement to provide information about aspects of the training process for ML models. Proposals for providing such information, now used in practice, include *datasheets for datasets* [26] and *model cards* [54]. Analogous to their use in electronics, datasheets for datasets describe the composition and intended use of a training dataset; model cards perform a similar function for a trained model. Model cards should ideally include information about the training algorithm and the training datasets (e.g., by referring to the datasets' datasheets), as well as details of any dataset pre-processing (e.g., to filter certain types of content).

In fully open-source models, the exact training code and dataset are made publicly available. In open-weight or closed-source models, the training code and dataset may be confidential, but model providers may still make public claims about the training code, dataset, and pre-processing. For example, the card for OpenAI's GPT-5 model claims that their data processing pipeline "includes rigorous filtering to maintain data quality and mitigate potential risks" and uses "advanced data filtering processes to reduce personal information from training data" [60]. For all types of models, information about training may need to be provided to regulators. For example, Article 53 of the EU AI Act requires model providers to provide this information, "upon request, to the AI Office and the national competent authorities" [21].

Making claims about FL training, e.g., that particular datasets or attack mitigations have been used, is even more challenging. Claims about a trained FL model require trust in all participants who contributed to the training. There could be various reasons for any of these participants to deviate from the agreed training protocol, either unintentionally or deliberately. For example, a data provider may accidentally omit or misconfigure steps when pre-processing training data, leading to undesirable data being used for training. Alternatively, they may promise to train on a particular dataset but only train on a subset of this data to reduce costs.

It is an open challenge how to verify such claims about model training, even with access to the training code and datasets. For closed-source models that are only accessible via inference APIs, it is impossible to verify claims about model training; for open-source models, one could attempt to repeat the training process, but depending on the model size and training duration, this may require significant hardware resources, thus incurring substantial costs. Even if this were feasible, there is no guarantee that the trained model would match the original model due to the inherent non-determinism in the training process. Therefore, claims about model training typically have to be accepted based on *reputational trust* in the entity making the claims. In an FL scenario, however, this reputational trust would have to extend to all participants, i.e., the model provider and all data providers.

## 2.3 Trusted execution environments

Trusted execution environments (TEEs) [3–5] try to ensure both the confidentiality and integrity of execution in a verifiable manner. Code and data in TEE memory are isolated, which prevents even privileged attackers, such as malicious administrators, operating system (OS) kernels, or hypervisors, from accessing or tampering during execution. TEEs also support *attestation* [11] for external parties to verify a measurement taken over TEE memory, including both code and data. TEEs, including attestation mechanisms, are implemented in hardware, e.g., relying on memory encryption, memory access checks, and hardware signatures for attestation.

TEEs exist at different granularities: ARM TrustZone [59] provides a separate *secure world*; Intel Software Guard Extensions (SGX) offers isolation at a process granularity using *enclaves*; more recently, *VM-based* TEEs, such as AMD SEV-SNP [3], Intel TDX [12, 84], and ARM CCA [4, 47], isolate entire VMs from the hypervisor and other privileged system software.

Hardware support for TEEs has also been integrated into GPUs. For example, Nvidia's H100 GPUs support a unified TEE that spans the CPU and GPU [17, 29]. In particular, it protects sensitive GPU state, including GPU memory and configuration registers, from external access. Communication between the CPU-based TEE and the GPU is encrypted and integrity-protected. Graphcore IPUs provide similar TEE functionality [81].

## 2.4 Approaches for verifiable claims

Since TEEs protect computation at a hardware level and provide attestation proofs, an FL model provider could require all data providers to perform their model training inside TEEs and use remote attestation to demonstrate this in a provable manner. Existing approaches therefore propose hosting FL training jobs inside

TEEs: CFL [28] demonstrates how to run FL frameworks in TEEs using SEV-SNP VMs coupled with NVIDIA Confidential GPUs; SecureFL [42] is an FL training system that integrates SGX-based TEEs with ARM TrustZone on the edge to ensure privacy of both models and data; PPFL [55] uses TEEs to prevent model and data providers from observing the model during training, which avoids membership inference attacks; SecFL [65] includes a policy manager to enforce attestation policies agreed between all clients; OpenFL [68] uses SGX-based TEEs during FL execution to prevent model leakage; GradSec [52] targets FL privacy threats by protecting a subset of the model layers inside TEEs; CrowdGuard [69] mitigates model poisoning attacks by employing TEEs for client-side cross-validation on local models; Laminator [19] and Atlas [75] use TEEs to attest certain properties of centralized ML training and inference.

Using TEEs to provide verifiable claims about a trained FL model gives rise to two challenges: (1) the attestation proofs offered by TEEs are based on measurements of the TEE memory at deployment time. Such *static* attestation proofs include a hash over the FL job and framework code. Typically, they do not include a hash over the datasets, because the data may be unavailable at deployment time. This makes it challenging to relate such an attestation proof to any claims about the final FL model: for an auditor to verify claims linked to such proofs, they would have to re-run the attested FL training job and observe if they obtain the same final FL model.

In addition, (2) TEE attestation relies on the confidentiality of TEE memory, because typical attestation implementations, e.g., Intel SGX quoting enclaves, rely on attestation keys stored in TEE memory. Since TEEs guarantee confidentiality, such an approach is sound, unless confidentiality is compromised. Here, TEEs have witnessed a long sequence of data leakage attacks that can disclose secrets from TEE memory, often by exploiting difficult-to-mitigate architectural and micro-architectural side channels [40, 45, 46, 82]. If such attacks are used to leak the attestation key from a TEE, the integrity of attestation proofs can be compromised by an adversary. This would enable the adversary to produce bogus attestations, which they could use as evidence for false model claims.

Such vulnerabilities extend beyond leaking the attestation key itself. Even for TEEs such as AMD SEV-SNP, which keep the root attestation signing keys in a separate security co-processor, attestation mechanisms use the root keys to attest other keys, which then reside in TEE memory (e.g., TLS session keys). If such an attested key is extracted from TEE memory, an adversary could use it to send arbitrary model updates via the attested TLS connection while still appearing to be attested correctly.

Prior work has observed that guarantees about confidentiality and integrity could be separated in trusted execution. Sealed-glass proofs (SGP) [78] use SGX-based TEEs as “transparent enclaves” to provide attestation proofs of computation over secret inputs, while allowing a host to inspect the TEE memory. Transparent enclaves, however, still assume that attestation keys remain secret: with SGX enclaves, the attestation key is held inside a separate quoting enclave (QE) residing in TEE memory, which makes it vulnerable to side-channel attacks [80].

Instead of relying on hardware mechanisms, other approaches have used algorithmic and cryptographic techniques to establish claims about trained models. *Proof of learning* approaches [1, 35] build on techniques such as proof-of-work (PoW) [56], verified

computation [27], and zero-knowledge proofs [25, 63]. These approaches, however, either incur prohibitively high overheads or can be subverted [22, 89]. For example, zero-knowledge proofs such as zkSNARK [63] introduce orders-of-magnitude computational overheads [25]; PoW-based mechanisms [10] require the majority of the participants to follow the protocol correctly. This assumption does not hold in FL because participants are mutually distrusting.

## 2.5 Threat model and security goals

We assume that any participant in the FL training job (i.e., a data provider or the centralized model provider), or a set of colluding participants, could be adversarial. This also includes the case in which honest participants have been compromised by an adversary.

**Adversary’s goal.** The adversary aims to deviate from the predefined FL training protocol in a way that modifies or influences the final FL model, *and* for this deviation to remain undetectable by other participants and/or third-party auditors. This goal encompasses many specific attacks that participants could attempt: e.g., an adversarial data provider may perform fewer training rounds to reduce resource costs, skip a pre-processing step that would otherwise filter out specific data, or alter their dataset during training to bias the model; similarly, an adversarial model provider may omit updates from specific data providers during aggregation to bias the model, or send different models to different data providers to infer information about a particular provider’s private dataset [62]. More generally, any adversarial participant may want to deviate from the predefined FL training protocol by changing their training code, or modifying or falsifying data transferred during the FL training job.

**Adversary’s capabilities.** We assume a strong adversary with privileged access to a participant’s system. This includes full control of the OS and hypervisor of a participant’s machine, as well as the ability to intercept and modify all inbound and outbound network communication. In addition, the adversary can carry out side-channel attacks to leak memory contents, including TEE memory. We assume, however, that the adversary does not carry out advanced physical attacks against the hardware, such as physically extracting keys from a separate on-chip security co-processor, due to the high costs and expertise required.

**Security guarantees.** EXCLAVEFL aims to ensure that the aforementioned adversary cannot deviate from the predefined FL training protocol, in a way that modifies or influences the final model, without detection. EXCLAVEFL thus enables any participant or third-party auditor to verify claims about the FL training process. We define a *verifiable claim* to be any statement about the FL training process that could have been made by a (hypothetical) trusted omniscient observer with full visibility of the training process. Claims could include statements about the executed code for training and aggregation, the number of participants and rounds, or any other runtime property of the FL job. It is also possible to make statements about the precise data used for training, but sometimes it is desirable for the data to remain private to the data providers. In such cases, statements can be made about a *representation* of the data, e.g., a blinded hash to confirm that the data has not been changed during FL training. In addition, claims may make statements about what pre-processing (e.g., data filtering) has been performed.

**Security non-goals.** Given our assumed adversary’s capabilities, we cannot *prevent* deviation from the FL training protocol, but only require detection. Similarly, we cannot guarantee the *availability* of verifiable claims—there are various ways in which our strong adversary can prevent such claims from being made (e.g., through a denial-of-service attack). In addition, we do not provide *model confidentiality* from data providers. Instead, we assume FL scenarios in which the trained model is ultimately shared with all participants, e.g., a group of hospitals that is jointly training a model on patient data. Finally, we do not aim to detect the poisoning of a training dataset *before* training begins. There is a large body of related work on the detection or prevention of such attacks [83]. EXCLAVEFL supports such mitigations by (i) ensuring that they have been applied correctly by the respective participants; and (ii) allowing third parties to verify that they were in place during training.

### 3 Exclaves

We introduce *exclaves*, a new abstraction that overcomes the challenges described above. We first explain the enclave concept (§3.1) and then describe how exclaves can be implemented (§3.2).

#### 3.1 Exclave concept

Current TEE implementations use attestation to establish a trust relationship between the TEE and a verifier. For example, an attested TLS network connection [79] can prove that the verifier is communicating with a TEE and that all messages received via the TLS connection were sent by that TEE. An adversary with the ability to leak data from a TEE (e.g., through side channels), however, can undermine the security guarantees of current TEE implementations when used to secure FL training (see §2.5). In the above example, the adversary could leak the session key of the attested TLS connection and masquerade as the TEE. In the presence of such an adversary, we therefore cannot rely on secrecy of data within the TEE.

**Exclaves.** We introduce the *exclave*, which is an *integrity-only* execution environment that does not contain data that must be kept secret from an adversary. Similar to TEEs, an exclave ensures that code is executed without external interference, and that the data within the exclave can only be modified by code within the exclave. An exclave thus offers *integrity* guarantees for its code and data, even against privileged software running on the same host.

Unlike TEEs, exclaves do not provide confidentiality guarantees—the only piece of data that must be kept secret is a single attestation signing key. As we explain in §3.2, the attestation key can be held by a security co-processor, which is hardened against side-channel attacks by being physically separate [23]. While TEEs can execute the same workloads as exclaves, the attestation mechanisms of TEEs assume confidentiality of TEE memory.

Code that executes inside an exclave can request the security co-processor to produce a signed attestation report, which contains (i) a measurement of the exclave’s code and data at deployment time and (ii) user-defined data supplied by the application code inside the exclave at runtime. This allows the attestation report to prove that a particular claim was made by the exclave at runtime.

**Exclave Data Records (EDRs).** The attestation report can be used to produce an unforgeable record of the *data transformations* carried out by the exclave, i.e., any computation that takes a set of

data inputs and produces a set of data outputs. Exclaves can thus be used to attest individual data transformations at runtime. After performing a data transformation within an exclave, the exclave can produce a runtime attestation proof that binds the data transformation to the specific code and data of the transformation. We refer to such an attestation proof as an *exclave data record* (EDR).

More formally, an EDR is an existence proof that at least one execution of the specified code carried out the stated data transformation. In terms of its implementation, an EDR is a signed statement that contains (i) a measurement of the exclave’s code, defining the computation; and (ii) a representation of the data transformation, containing two key/value structures: one for inputs and another for outputs. The keys uniquely identify inputs and outputs (e.g., `model_weights` or `training_data`); the values are fixed-size cryptographic hashes of the input and output data. Analogous to TEEs, an exclave’s code measurement is a hash of the code pages loaded at deployment time. Given access to the code, a verifier can thus recompute the hash and compare it against the measurement.

#### 3.2 Exclave implementation

An exclave can be implemented with minor modifications of a TEE implementation, such as AMD SEV-SNP [3], by reusing its isolation and hardware-based attestation mechanisms. With AMD SEV-SNP, exclaves can rely on the memory isolation enforced by the memory management unit (MMU): during address translation, the MMU checks the access policy stored in the *reverse map table* (RMP), which is managed by the CPU and the *platform security processor* (PSP) [3, 7, 38]. The exclave uses the PSP to generate hardware-signed attestation reports at runtime. The signing key for attestation reports, i.e., the *versioned chip endorsement key* (VCEK) [3, 11], is stored and managed by the PSP only.

Exclaves, however, must generate runtime attestation reports by passing data securely to the PSP, without relying on shared software secrets—any software secret would violate their secret-free nature. In AMD SEV-SNP, each TEE maintains *VM platform communication keys* (VMPCKs) to protect communication with the PSP. An exclave implementation using AMD SEV-SNP must therefore protect PSP communication through an MMU access check instead of a shared secret: the exclave must use a special *protected page*, accessible only by the exclave and the PSP, to create attestation reports.

An exclave implementation using AMD SEV-SNP would thus require the following firmware changes: with exclaves as CVMs, a protected page could be implemented as a runtime-accessible analogue of SNP’s existing *secrets page*, which is RMP-protected, guest-owned, but PSP-writable only on provisioning. This change would require: (1) defining a new guest-owned protected page, whose RMP entry has the same metadata as the secrets page (e.g., exclave’s guest ASID, validated-bit, etc.); (2) extending the firmware with functionality for the PSP to read/write this protected page at runtime and not only at provisioning time; and (3) recording this page type as `PAGE_TYPE_PROTECTED` and provisioning it during exclave creation (similar to secrets page provisioning).

With these changes, access to the protected page would be enforced by SNP’s RMP check, enabling secret-free attestation. These firmware changes would be feasible, because they reuse the existing implementation of the secrets page protection. Moreover, they

would not affect the verification of the CVM's initial state post-boot: the expected hash of the CVM firmware image would need to be updated, but this is the same as for a standard firmware update.

For other TEE implementations, achieving secret-free attestation would require the removal of privileged software components, such as *security monitors* (SMs), from the runtime attestation process. For example, the Keystone TEE [44] would require a secure hardware signer (e.g., HSM/TPM [88]), making its SM an untrusted proxy for attestation requests to that hardware signer. Compared to other TEEs, AMD SEV-SNP already provides hardware-signed runtime proofs by the PSP, which matches an exclave's requirements.

Since we cannot modify the proprietary AMD SEV-SNP firmware, we emulate equivalent signing functionality for exclave attestation. As we describe in §6.3, our exclave implementation on unmodified AMD SEV-SNP hardware emulates the modified attestation approach using a *virtual TPM* as a stand-in for the PSP's attestation functionality, which yields comparable performance.

## 4 Exclave Federated Learning

We describe the design of *Exclave Federated Learning* (EXCLAVEFL), a system that produces verifiable claims about an FL training process and the resulting model. EXCLAVEFL uses exclaves to produce runtime attestation proofs, in the form of EDRs, of all data transformations leading to a final trained FL model. The collected set of EDRs thus provides a trustworthy end-to-end record of the training process over all FL computation rounds. An auditor can use the EDRs to verify claims about the trained FL model (see §5).

The use of exclaves in an FL training system introduces two challenges: (i) the integrity of the claims must not depend on the confidentiality of the FL training process, because, unlike TEEs, exclave execution is visible to FL participants; and (ii) to keep the TCB small, the claims should only require the minimum amount of code to be integrity-protected (i.e., executed inside exclaves). In particular, the claims should not rely on the integrity of the FL framework code, as long as the EDRs attest the data transformations to establish correctness.

As shown in Fig. 2, EXCLAVEFL executes an FL training *job* as a sequence of FL *tasks* that define the data transformations. Data providers and the model provider execute tasks in each training round. An FL framework such as NVFlare [70] provides the task implementations, orchestrates task execution and offers APIs for data movement between tasks. EXCLAVEFL then executes each task within an exclave, which ensures execution integrity and produces EDRs for each data transformation.

In an FL job, the tasks executed by data providers may include: (i) a *local training task* that takes a global FL model, reads a dataset from storage, computes a local model update, and returns the updated local model; and (ii) a *differential privacy (DP) task* that inputs a local model update, adds DP noise to protect privacy, and outputs a DP-protected local model update. A model provider may execute: (iii) an *aggregation task* that receives DP-protected local model updates from data providers and outputs an aggregated model; and (iv) a *model update task* that converts the aggregated model to a global FL model. The updated global FL model is sent to the data providers ahead of the next training round. Note that the rest of

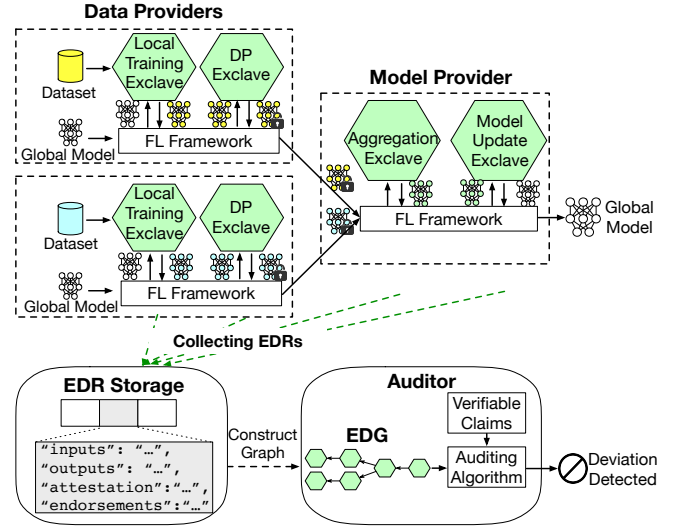


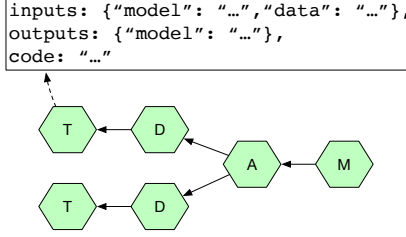
Figure 2: Design of EXCLAVEFL (Exclaves are shown in green.)

the FL framework executes outside of exclaves, thus keeping the TCB limited to the data transformations that affect the FL model.

Since EXCLAVEFL executes tasks inside exclaves, the produced EDRs create a trustworthy record of the executed data transformations. For example, an EDR produced by the aggregation task contains hashes of (i) the local model updates received by the task in a round and (ii) the computed aggregated model; an EDR produced by the model update task has hashes of (i) the aggregated model, (ii) the previous global model, and (iii) the updated global model; and an EDR produced by the local training task contains hashes of (i) the received global model, (ii) the training dataset, and (iii) the updated local model. If the FL job executes faithfully, the EDRs must show that the correct code for the task implementations was run. In addition, the hashes of the data inputs/outputs in the EDRs must match between EDRs produced by different tasks: e.g., the output hash of the updated global model in the EDR generated by the model update task (executed by the model provider) must match the input hashes of the global model in the EDRs generated by the local training tasks (executed by the data providers).

When the local training task reads a dataset from storage, EXCLAVEFL must ensure that the input hash in the EDR remains valid. Therefore, a data provider must provide the training dataset as an integrity-protected disk image [39], protected by a Merkle tree [51] to preserve the integrity of the blocks. When the dataset is accessed by the local training task in the exclave, EXCLAVEFL checks the dataset blocks against the Merkle tree. The root hash of the Merkle tree (combined with a random salt) serves as a non-revocable *data commitment* to the training dataset, which is included in the EDR.

When exclaves produce EDRs, the EDRs are sent to an *EDR storage service*. This service maintains all generated EDRs during job execution and makes them available for auditing. Note that the EDR storage service does not need to be trusted: since EDRs are integrity-protected by a cryptographic signature, any tampering is detectable. If individual EDRs are withheld by an adversary from the EDR storage service, this is revealed by the introduced inconsistencies between EDRs (see below). As we explain in §5, an *auditor* can check the EDRs for mismatches and inconsistencies, which indicate



**Figure 3: Example of an Exclave Dataflow Graph** (Edges show reverse dataflows; nodes show Training, DP, Aggregation, and Model update tasks.)

a deviation from the faithful execution of the FL job and invalidate the claims about the trained FL model. The auditor can be an FL participant, a consumer of the trained FL model who needs to verify correct training, or a regulator who audits the training process.

## 5 Auditing of Verifiable Claims

Auditing verifiable claims involves constructing a graph based on the EDRs (§5.1) and then verifying the claims against the graph (§5.2).

### 5.1 Constructing exclave dataflow graphs

The EDRs collected by the EDR storage service are used to construct a directed acyclic graph, which we refer to as the *exclave dataflow graph* (EDG). Fig. 3 shows a sample EDG: each vertex  $v \in V$  is an EDR, which is defined by the tuple  $(in, code, out)$ , which refers to the data input, code, and data output hashes from that exclave execution, respectively. Each edge  $e \in E$  represents the (reversed) data flow between exclaves; it points from the downstream (consumer) vertex  $v_y$  to the upstream (producer) vertex  $v_x$ :

$$E = \{(v_y, v_x) \mid v_y, v_x \in V \text{ and } v_y.in \cap v_x.out \neq \emptyset\} \quad (1)$$

The edge label represents the data that flows between the two exclave executions. It is defined as the non-empty intersection of the outputs of  $v_x$  and the inputs of  $v_y$  (since exclaves may have multiple inputs and outputs for a single execution):

$$l(v_y, v_x) = v_y.in \cap v_x.out \quad (2)$$

We define the  $EDG = (V, E, l)$  as a collection of vertices  $V$ , reversed dataflow edges  $E$ , and the edge label function  $l$ . For auditing, Alg. 1 shows how EXCLAVEFL constructs an EDG from a collection of EDRs  $v$ , which are attested as  $\sigma(v)$ . For each EDR, EXCLAVEFL first verifies the attestation with the hardware manufacturer, e.g., using AMD’s Key Distribution Service [38] (line 1). If the verification succeeds, EXCLAVEFL adds the EDR to the EDG as a vertex; if it fails, e.g., due to an invalid signature or other tampering, the vertex is ignored (see Fig. 4a).

EXCLAVEFL adds edges to the EDG by iterating over vertices.<sup>2</sup> If the inputs of an EDR vertex match the outputs of another, an edge is added between the vertices, labeled with the dataflow between the exclaves (lines 10–14). If the data was modified outside the exclave, e.g., by an adversary during transit, there is no match, and no edge is added due to the input/output mismatch (see Fig. 4b).

### Algorithm 1 Constructing an EDG for auditing

**Input:** Collection of vertices  $v \in V'$ , attestation reports  $\sigma(v) \in \Sigma$

**Output:**  $EDG = (V, E, l)$

```

1:  $V \leftarrow \{v \mid \text{verify\_attestation\_report}(\sigma(v)) = \text{true}\}$ 
2:  $EDG \leftarrow \{V, E = \emptyset, l = \emptyset\}$ 
3: // Step 1: Index vertices by input
4: for each  $\{v_y \in V\}$  do
5:   for each  $\{input \in v_y.in\}$  do
6:     append  $v_y$  to  $vertexOf[input]$ 
7: // Step 2: Create edge when input and output match
8: for each  $\{v_x \in V\}$  do
9:   for each  $\{output \in v_x.out\}$ 
10:    if  $\{vertexOf[output] \neq \emptyset\}$ 
11:      for each  $\{v_y \in vertexOf[output]\}$ 
12:         $e \leftarrow (v_y, v_x)$ 
13:         $l(e) \leftarrow v_y.in \cap v_x.out$ 
14:         $E \leftarrow E \cup \{e\}, l \leftarrow l \cup l(e)$ 
15: return  $(V, E, l)$ 

```

### 5.2 Checking verifiable claims

The constructed EDG is audited to verify claims about the FL model. Next, we describe the verifiable claims using examples of training-time deviations and explain how an auditor would detect them:

**Claim 1: Correct FL training algorithm implementation.** Any participant could modify the code of tasks before deployment. For example, a data provider could modify the training and DP tasks to skip data integrity checks, ignore certain training data, inject too much noise, or modify model updates. A model provider could modify the aggregation or model update exclaves to omit inputs, change aggregation, modify model weights, or distribute different models to participants. The modified code would be measured by exclaves and recorded in the EDR. Since the auditor traverses the EDG and checks each vertex against a list of acceptable code measurements for tasks in the FL job, they would detect the claim violation.

**Claim 2: Correct transmission of training data.** A malicious participant could modify the data in transit, either between exclaves (Fig. 4b), or when sent by the FL framework between participants. This would result in a mismatch between EDR outputs and inputs, omitting the edge from the EDG. Due to the missing output edge from a vertex, the auditor would identify the claim violation.

**Claim 3: Correct use of DP/unbiased data aggregation.** As shown in Fig. 4c, a malicious participant could bypass the DP task execution and send the training task’s output directly without DP noise to the model provider, e.g., to increase the impact of their harmful data records. Similarly, a model provider could introduce bias by dropping contributions from certain data providers. This is more subtle than the previous attacks, because it still results in a complete EDG. The shape of the resulting EDG, however, would differ: it would miss vertices/edges, or have edges that span multiple training rounds, which would be detectable by the auditor.

**Claim 4: Integrity of training dataset.** A malicious data provider could add, remove, or modify a dataset after training has commenced (e.g., performing an adaptive poisoning attack [87]). If the data is modified while being read by an exclave, the verification against the Merkle tree fails; if the dataset is modified between

<sup>2</sup>In practice, the FL job structure can be used to ascertain the vertices that *should* be connected, thus avoiding naive iteration.

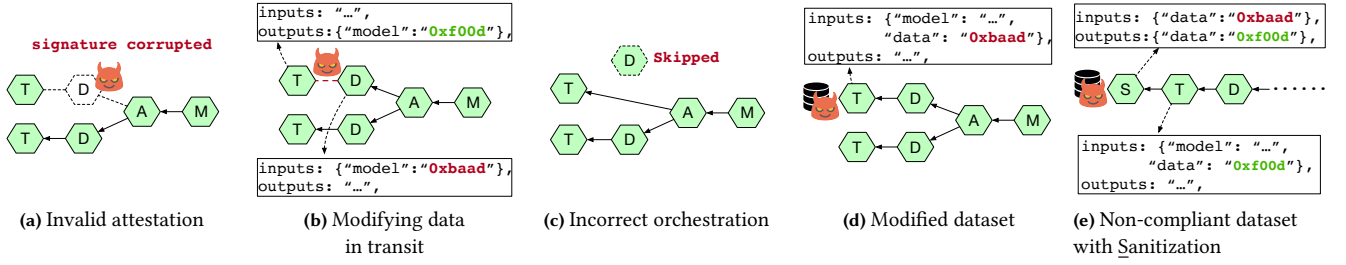


Figure 4: Variations of the EDG from Fig. 3 with deviations

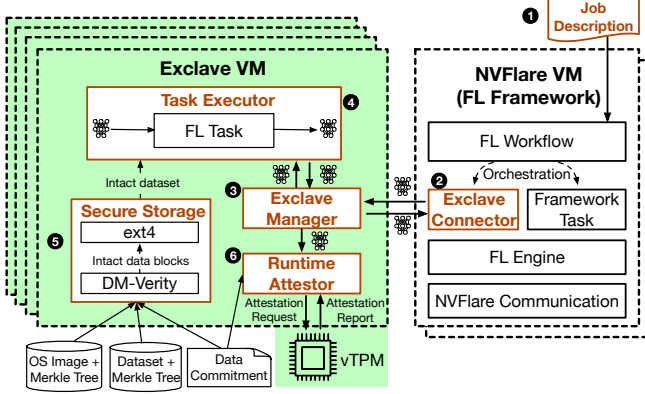


Figure 5: EXCLAVEFL implementation

training rounds (see Fig. 4d), the hash of the modified dataset is recorded in the EDG, allowing the auditor to discover the change.

**Claim 5: Appropriate dataset sanitization.** A data provider may use a non-compliant dataset without sanitization, e.g., with harmful content or personal information. If the FL job includes a sanitization task, its correct execution is recorded by the EDRs. Therefore, an auditor can compare the training task’s input dataset commitment against the set of data commitments produced by the sanitization task (Fig. 4e). An adversary may skip sanitization, or excessively filter their data to contribute less; both are detectable due to the code measurement of the data sanitization task in the EDG. Note that the auditor does not require access to the plaintext dataset to detect whether the dataset was exposed to a given sanitization task: EDGs only contain the data hashes, because EXCLAVEFL audits the training process rather than the training data.

## 6 EXCLAVEFL Implementation

In this section, we describe the EXCLAVEFL implementation (§6.1) and its components (§6.2), as well as our implementation of exclave attestation (§6.3) and integrity-protected data storage (§6.4).

### 6.1 Overview

A goal of the EXCLAVEFL implementation is to be compatible with different FL frameworks and to support existing FL frameworks with minimal code changes. To achieve this, EXCLAVEFL adopts an orchestrator/worker model in which an untrusted FL framework orchestrates the FL job and task execution inside exclaves.

Table 1: FL tasks deployed in exclaves

| Exclave      | Task API                          | Description                                      |
|--------------|-----------------------------------|--------------------------------------------------|
| Training     | StartTrain(...)<br>GetWeight(...) | Receives global model<br>Returns local model     |
| DP           | AddNoise(...)                     | Performs local DP                                |
| Aggregation  | Accept(...)<br>Aggregate(...)     | Receives local models<br>Aggregates local models |
| Model Update | ModelUpdate(...)                  | Applies diff to full model                       |

Fig. 5 shows how EXCLAVEFL integrates with an FL Framework: a model provider submits an FL job description to the FL Framework ①. The framework orchestrates the job according to its description, and allocates FL tasks to exclaves via Exclave Connectors ②, which are EXCLAVEFL components that invoke exclaves. When an exclave receives a task, an Exclave Manager ③ forwards its payload (e.g., global model) to a Task Executor. During task execution ④, the task can access persistent storage via a Secure Storage module ⑤, which provides integrity protection for on-disk datasets. When the task completes, a Runtime Attestor ⑥ attests the EDR of the exclave. The execution result (e.g., a local model) is then returned.

### 6.2 Components

Our EXCLAVEFL implementation has the following components:

**FL Framework.** EXCLAVEFL uses NVFlare [70], which is an open-source distributed FL framework by Nvidia. It provides FL task implementations for many FL algorithms, e.g., federated averaging, federated optimization, and secure aggregation, and other building blocks, e.g., model trainers, model aggregators, and task filters. Tab. 1 shows a set of FL tasks used in a typical FL job. EXCLAVEFL executes all of these tasks inside exclaves.

**Exclave Connectors** are gRPC clients [32] that NVFlare uses to execute FL tasks in exclaves. They are implemented as NVFlare components, which makes them compatible with the NVFlare API. To add an exclave, EXCLAVEFL includes an Exclave Connector in NVFlare’s job description file. This approach of using Exclave Connectors avoids direct modifications to the NVFlare code.

**Exclaves** are implemented as confidential VMs (CVMs) using AMD SEV-SNP [3]. Within each CVM, gRPC servers can execute FL tasks.

**Exclave Managers** expose exclave interfaces to NVFlare to support the orchestration of FL jobs. Exclave Managers receive FL tasks from NVFlare, deserialize payloads, and forward payloads to the Task Executor. They then take the execution results from the Task Executor and pass them to NVFlare.

**Task Executors** execute tasks inside exclaves. They receive FL tasks from the Exclave Manager, load the data/code, and execute the tasks. For example, the Task Executor for LLM training loads the model, the training code, and the training dataset. EXCLAVEFL reuses FL task execution code from NVFlare with minimal changes.

The **Runtime Attestor** constructs EDRs and interacts with a virtual Trusted Platform Module (vTPM) [53] for attestation. During task execution, it records the storage accesses and monitors requests/responses passing through the Exclave Manager to get exclaves' input and output data. It stores a hash of each data item together with metadata describing the intended usage (e.g., input/output) and properties (e.g., model/dataset). After an input/output pair is collected, it builds a nested JSON object: keys contain the data usage and properties, and the values contain data hashes. The Runtime Attestor invokes the vTPM's signing API with a SHA-256 digest of this object, which the vTPM signs together with a system measurement to produce an EDR.

### 6.3 Attestation

To prototype exclave attestation without firmware modifications, we use a vTPM [53] and virtual machine privilege levels (VMPL) [3]. We run the vTPM in VMPL 0 (highest privilege), and exclaves in VMPL 2 (lower privilege); VMPL 1 remains unused, following existing AMD SEV-SNP deployment practices [14]. The MMU and RMP enforce isolation: lower privilege levels (i.e., exclaves) cannot interfere with higher ones (i.e., vTPM), protecting the attestation process. The vTPM provides attestation primitives (measurement and signing), and its signing key is endorsed by the PSP on CVM launch. Attestation requests/responses pass through the RMP-protected, VMPL 2 memory region, writable by the vTPM and exclaves. The vTPM reads the runtime claim's hash, extends it into PCR23, signs the PCRs, and writes the signed report.

This approach makes the vTPM an emulation of AMD's PSP with our proposed firmware modifications. While this provides a significant degree of security, there is residual risk because the vTPM runs on the main CPU and shares micro-architectural state (e.g., CPU caches) with attacker-controlled processes. While having the vTPM at a higher privileged VMPL increases attack difficulty, moving the attestation signing key to a separate co-processor, as done by exclaves, would eliminate this risk. This also aligns with the planned evolution of TEEs: e.g., in future generations, Intel TDX [12] proposes to create attestation quotes using separate security hardware [33].

### 6.4 Secure storage with commitment

The Secure Storage module offers integrity-preserving, verifiable storage for exclaves, which protects both datasets and task code.

**Datasets.** EXCLAVEFL uses Linux's *dm-verity* [39] for integrity protection, which checks accessed disk blocks against a Merkle tree. Data providers use the *dm-verity* tool to package datasets with a Merkle tree. The generated data commitment is a unique measurement over the dataset, which is included in EDRs by the Runtime Attestor. When setting up an exclave, EXCLAVEFL mounts the data disk and checks the data commitment against the Merkle tree.

**Code.** EXCLAVEFL also uses *dm-verity* protection for the exclave's code image by using the Merkle tree root hash as a code measurement. When an exclave is launched, the vTPM includes the code measurement in the PCR11 field of the attestation report. Since it uniquely identifies the exclave code, it can be used by the auditor to determine if an FL task was executed by the correct exclave.

## 7 Evaluation

We evaluate EXCLAVEFL to answer: (i) what is EXCLAVEFL's end-to-end training performance compared to existing solutions? (§7.2); (ii) how does EXCLAVEFL scale with more data providers? (§7.3); (iii) what is the latency breakdown of EXCLAVEFL? (§7.4); (iv) what overhead does EXCLAVEFL's runtime attestation add? (§7.5); (v) what is the impact of secure storage on throughput and latency? (§7.6); and (vi) what is the auditing overhead as EXCLAVEFL scales? (§7.7)

### 7.1 Experimental setup

**Testbed.** We deploy confidential VMs (CVMs) on Azure: CVMs without GPUs run on AMD EPYC 7763v CPUs; CVMs with GPUs run on AMD EPYC 9V84 CPUs, both with SEV-SNP and vTPM enabled. The GPUs are NVIDIA H100 with Hopper Confidential Computing extensions [58]. We deploy 5 CVMs (EC4as\_v5) to execute 1 model update exclave and 4 NVFlare clients; 5 CVMs (EC8as\_v5) to execute 4 DP exclaves and 1 NVFlare server; and 1 CVM (EC16as\_v5) as the aggregation exclave. Training exclaves are deployed on 4 CVMs with CPUs (DC32as\_v5) or 4 CVMs with GPUs (NCC40ads\_H100\_v5). Each training exclave uses an Azure P30 data disk (200 MB/s of throughput with 5k IOPS).

**Implementation.** We implement EXCLAVEFL in Python and C++. It uses PyTorch v2.5.0 with CUDA 12.4 for the FL tasks, Azure Guest Attestation Library v1.0.2, and NVFlare v2.2.1 [70]. As described in §6.3, we emulate the attestation behavior of exclaves using a vTPM implementation [53] running on the AMD SEV-SNP platform.

**Baselines.** We compare EXCLAVEFL with two baselines: (i) NVFlare (No security) executes each NVFlare node (server and clients) in its own regular VM without SEV-SNP features; and (ii) NVFlare (CVM) [28] executes each NVFlare node in its own CVM. To facilitate comparison, both NVFlare (CVM) and EXCLAVEFL use the vTPM for attestation and *dm-verity* for secure storage.

**Datasets.** We use 3 public datasets: (i) Wikitext103 [50] consists of raw text with over 100 M tokens from Wikipedia articles; (ii) NCBI-disease [18] is a disease text mining dataset with 793 fully annotated PubMed abstracts; and (iii) CIFAR10 [41] has annotated  $32 \times 32$  pixel images with 50,000 training and 10,000 testing samples.

**Models.** We use 6 widely-used models, covering three different categories: (i) decoder-only transformers—we use GPT-2 [66] with 124 M parameters, and a distilled version of GPT-2, DistilGPT2 [71], with 82 M parameters; (ii) encoder-only transformers—we employ BERT [16] with 110 M parameters, and MobileBERT [76] with 25 M parameters; and (iii) convolutional neural networks (CNN)—we use LeNet [43], a 5-layer CNN, and VGG9 [74], a 9-layer CNN.

### 7.2 End-to-end throughput

We evaluate the end-to-end performance of EXCLAVEFL in terms of training throughput. We run EXCLAVEFL with 1 FL server and

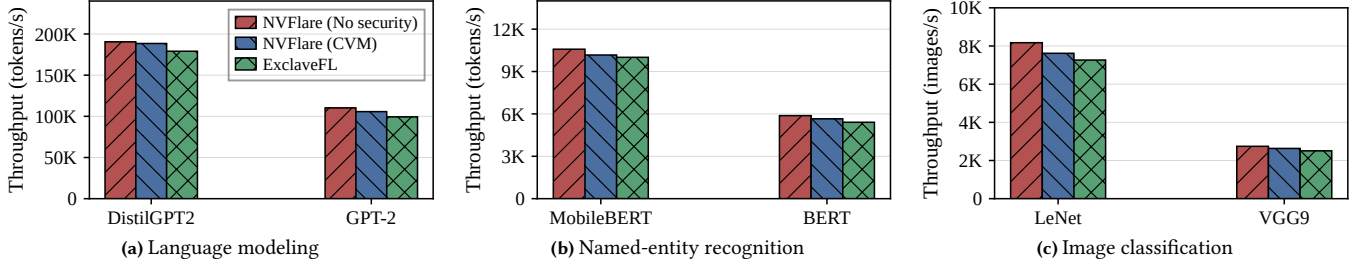


Figure 6: End-to-end training throughput

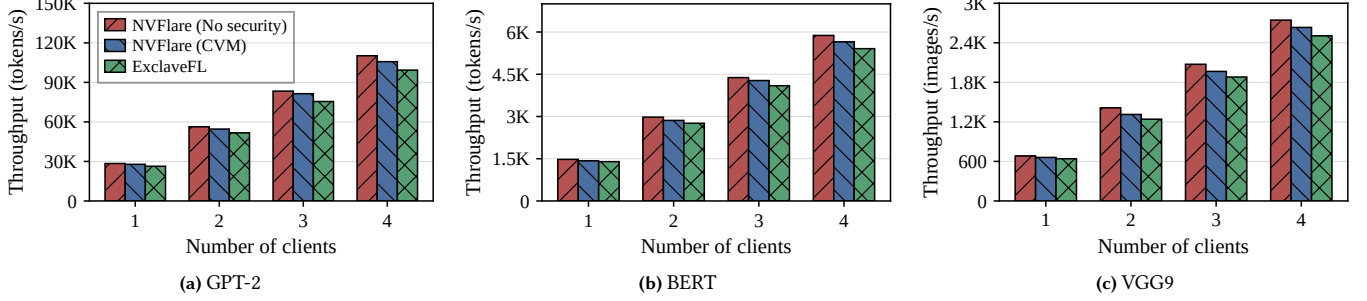


Figure 7: Throughput with different numbers of clients

4 FL clients. Each client performs model training with either a CPU or GPU CVM, depending on the workload. After each training round, the FL clients use the sparse vector differential privacy technique (SVT-DP) [48] to add DP noise to the locally trained model. EXCLAVEFL runs each FL job for 10 rounds with federated averaging (FedAvg) aggregation. We measure EXCLAVEFL’s performance for our three ML categories: (i) language modeling; (ii) named-entity recognition; and (iii) image classification. Fig. 6 shows the measured throughput results.

**Language modeling.** In this experiment, each client has a training enclave that executes the language modeling task in a GPU CVM (40 vCPUs; 320 GB of RAM; 1 NVIDIA H100 GPU). We train both GPT-2 and DistilGPT2 on the Wikitext103 dataset, which is partitioned into four subsets, one for each client.

Fig. 6a shows the training throughput. With DistilGPT2, EXCLAVEFL has a 6% overhead compared to the NVFlare (No security) baseline; NVFlare (CVM) has a 1% overhead. With GPT-2, when compared to NVFlare (No security), EXCLAVEFL and NVFlare (CVM) incur less than 10% and around 4% overhead in training throughput, respectively. We observe that, in both cases, EXCLAVEFL is slower than NVFlare (CVM): in EXCLAVEFL, enclaves introduce runtime overhead due to the runtime attestation; and the distributed deployment of enclaves adds network overhead when (de)serializing the model and transmitting the serialized model.

**Named-entity recognition.** Training enclaves execute in CPU CVMs (32 vCPUs; 128 GB of RAM). We use the MobileBERT and BERT models on the NCBI-disease dataset, split into 4 client shards.

Fig. 6b shows the training throughput. With MobileBERT, EXCLAVEFL and NVFlare (CVM) are 5% and 4% slower than NVFlare (No security), respectively. With BERT, EXCLAVEFL incurs an 8% overhead compared to NVFlare (No security), and NVFlare (CVM) has a 4% performance cost. We conclude that, when training MobileBERT, EXCLAVEFL has less performance impact compared to the

BERT training: since MobileBERT is 4.4× smaller than BERT, there is less network communication between enclaves in EXCLAVEFL.

**Image classification.** We also execute the image classification task with CPU CVMs under the same setup as before. Here, we use the LeNet and VGG9 models with the CIFAR10 dataset. Since the CIFAR10 dataset is small, we replicate it four times before sharding, ensuring that each client has sufficient training data.

Fig. 6c shows the performance results. For VGG9 training, EXCLAVEFL and NVFlare (CVM) compared to NVFlare (No security) incur 9% and 4% overhead, respectively. Compared to other workloads, EXCLAVEFL has a higher overhead for LeNet training: it is 11% slower than NVFlare (No security). Even with NVFlare (CVM), LeNet training overhead is 7%. Here, the CVM overhead and EXCLAVEFL’s extra scheduling/communication impact performance.

For moderate and large models (i.e., VGG9, MobileBERT, BERT, DistilGPT2, and GPT-2), EXCLAVEFL’s overhead is below 10% compared to NVFlare (No security), which is slightly higher than NVFlare (CVM)’s overhead of up to 4%. While LeNet shows higher overhead than other models, the overhead stays low across all models.

### 7.3 Scalability

Next, we investigate EXCLAVEFL’s scalability. We increase the number of FL clients and measure training throughput. Note that adding FL clients results in a corresponding increase in data parallelism. We use the same testbed settings and FL tasks as in §7.2.

First, we measure EXCLAVEFL’s scalability with the GPT-2 model for language modeling (Fig. 7a). Here, we scale EXCLAVEFL from 1 to 4 clients, each with a GPU. With 4× the number of FL clients, the throughput of EXCLAVEFL, NVFlare (CVM), and NVFlare (No security) increases to 3.8×, 3.8×, and 3.9×, respectively.

We also consider the BERT (Fig. 7b) and VGG9 (Fig. 7c) models. In these experiments, we increase the CPU count when scaling FL clients. With BERT, scaling the clients 4× leads to 3.9×, 4.0×, and

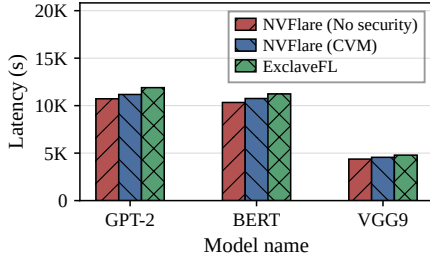


Figure 8: End-to-end FL latency

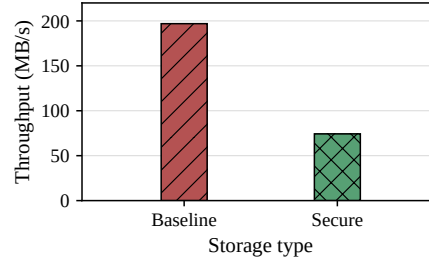


Figure 9: Storage read throughput

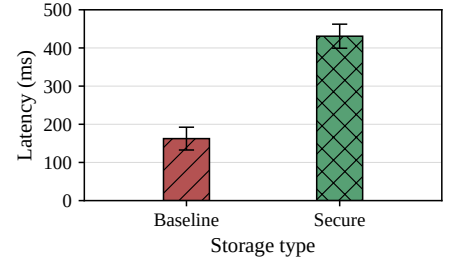


Figure 10: Storage read latency

Table 2: Latency for unmodified NVFlare (No Sec) and EXCLAVEFL

| Component    | GPT-2    |          | BERT     |          | VGG9    |         |
|--------------|----------|----------|----------|----------|---------|---------|
|              | No Sec   | ExFL     | No Sec   | ExFL     | No Sec  | ExFL    |
| Training     | 8,387 s  | 8,659 s  | 8,745 s  | 9,164 s  | 4,195 s | 4,641 s |
| DP           | 1,035 s  | 1,176 s  | 699 s    | 763 s    | 20 s    | 21 s    |
| Aggregation  | 9 s      | 50 s     | 6 s      | 29 s     | <1 s    | 1 s     |
| Model update | 5 s      | 29 s     | 2 s      | 18 s     | <1 s    | 1 s     |
| FL framework | 1,286 s  | 1,983 s  | 883 s    | 1,257 s  | 158 s   | 124 s   |
| Total        | 10,721 s | 11,897 s | 10,335 s | 11,232 s | 4,373 s | 4,789 s |

4.0× the throughput for EXCLAVEFL, NVFlare (CVM), and NVFlare (No security), respectively. For VGG9, we observe that the throughput for EXCLAVEFL, NVFlare (CVM), and NVFlare (No security) increases to 3.9×, 4.0×, and 4.0×, respectively, when we quadruple the client count. Both results show that EXCLAVEFL scales effectively with more FL clients, similar to the baselines. Since all clients run in parallel, runtime and I/O overheads do not accumulate.

## 7.4 Latency breakdown

We report EXCLAVEFL’s end-to-end latency in Fig. 8. While latency is inversely related to throughput, it makes the system overhead explicit and can be decomposed into per-component contributions. We provide such a latency breakdown for NVFlare (No security) and EXCLAVEFL in Tab. 2. Since exclaves use dm-verity-protected secure storage, the storage overhead is included in both the end-to-end latency results (Fig. 8) and the latency breakdown (Tab. 2).

Among all components, the training exclave dominates execution time (97% for VGG9; 82% for BERT; 73% for GPT-2). Note that GPT-2 trains with GPU exclaves, while VGG9 and BERT train with CPU exclaves. Thus, the GPT-2 training task occupies a smaller proportion of overall time.

The time spent executing the aggregation and model update exclaves increases linearly with model size, because the averaging/addition operations have linear time complexity. The DP exclave has a worst-case quadratic time complexity, because the SVT-DP algorithm iterates over all model parameters (potentially multiple times) to calculate the noise and apply the noise threshold. The total time spent on the DP, aggregation, and model update exclaves is less than 1% for VGG9 and up to 11% for GPT-2.

The execution time for the FL framework, including the time spent on orchestration and communication, also increases linearly with model size. Frequent communication and orchestration make the FL framework the second largest contributor to EXCLAVEFL’s execution time: 17%, 11%, and 3% for GPT-2, BERT, and VGG9, respectively. This overhead comes from EXCLAVEFL’s orchestration

of tasks that execute in exclaves: it is necessary for EXCLAVEFL to deploy exclaves in separate CVMs to provide the fine-grained runtime attestation of each exclave.

Compared to NVFlare (No security), EXCLAVEFL’s overhead comes from two sources. First, running FL workloads across disaggregated exclaves adds extra communication and orchestration overheads that inflate the FL framework cost. These overheads are significant for BERT and GPT-2, because these models are large. For VGG9, the communication overhead is small.<sup>3</sup> Second, each exclave incurs runtime costs from running the workload inside a CVM with a gRPC server, together with model (de)serialization/hashing, secure storage, and attestation. This appears as the per-component overhead of EXCLAVEFL over NVFlare (No security) in Tab. 2.

In addition, we evaluate the overhead of EXCLAVEFL’s data sanitization exclave, which a data provider runs prior to the FL training on an Azure NCC40ads\_H100\_v5 CVM. The data sanitization exclave uses a classification model for harmful text, granite-guardian-hap-38m [61], to detect toxic or abusive content and then removes the identified content to sanitize the dataset.

We measure the execution time of the data sanitization exclave when sanitizing each data provider’s Wikitext103 dataset. The result shows that the exclave takes 257 seconds to sanitize each data provider’s dataset, which adds 2% overhead to EXCLAVEFL’s GPT-2 training. We conclude that enforcing dataset properties using standard sanitization techniques is therefore feasible in practice.

## 7.5 Overhead of runtime attestation

Next, we measure the overhead due to runtime attestation. We use a single CVM with 4 vCPUs and 16 GB of RAM to generate a SHA-256 digest over 1 GB of input data and include the resulting hash digest as the runtime claim in an attestation request.

We observe that, on average, creating and signing an attestation report with the vTPM takes 4 ms. This step has constant time complexity, because both creation and signing operations take a fixed-size hash digest as input. Generating hash digests is the most time-consuming operation during attestation: it takes 687 ms to calculate a SHA-256 hash over the 1 GB of input data, and the time grows linearly with the input data size.

We also measure the attestation overhead with AMD’s PSP: the CVM requires less than 7 ms to fetch a signed attestation report from the PSP. Although we use vTPM attestation to emulate PSP attestation, the performance difference between them is marginal (under 3 ms per attestation), making it negligible compared to EXCLAVEFL’s end-to-end training time (hours).

<sup>3</sup>The residual difference is due to cloud placement noise: for NVFlare (No security), the FL framework spends more time waiting for straggling clients.

**Table 3: Overhead of auditing**

| #Clients ×<br>#Rounds | Signature<br>Verification | EDG<br>Creation | Total     | Storage |
|-----------------------|---------------------------|-----------------|-----------|---------|
| 10 × 10               | 26 ms                     | 3 ms            | 29 ms     | <1 MB   |
| 10 × 100              | 196 ms                    | 13 ms           | 209 ms    | 2 MB    |
| 10 × 1,000            | 1,886 ms                  | 131 ms          | 2,017 ms  | 17 MB   |
| 100 × 10              | 218 ms                    | 20 ms           | 238 ms    | 2 MB    |
| 100 × 100             | 1,802 ms                  | 146 ms          | 1,948 ms  | 16 MB   |
| 100 × 1,000           | 17,404 ms                 | 1,837 ms        | 19,241 ms | 162 MB  |

## 7.6 Overhead of secure storage

We also explore the performance impact of EXCLAVEFL’s secure storage for datasets. We mount a disk protected by dm-verity on an Azure DC32as\_v5 CVM (P30 data disk with up to 200 MB/s and 5,000 IOPS). We then use the FIO benchmark [6], which sequentially reads data from the ext4 file system on the secure disk. We compare against insecure storage as a baseline, which uses a standard D32as\_v5 VM on an unprotected Azure P30 disk.

As Fig. 9 shows, the sequential read throughput of the integrity-protected disk is 62% lower compared to regular storage. This is due to the cost of checking the block against the Merkle hash tree, together with CVM overhead. Latency also increases accordingly (Fig. 10): the secure storage module has 2.7× the baseline latency (reported latencies include queuing time from the in-flight I/Os). The resulting read amplification impacts both throughput and latency, which contributes to EXCLAVEFL’s performance overhead.

## 7.7 Overhead of auditing

We evaluate the overhead that the auditing process incurs in real-world settings, when the number of clients scales from 10 to 100, and the number of rounds varies from 10 to 1,000. We generate synthetic EDRs and measure the auditor’s computation and storage overheads in a standard VM with 4 vCPUs and 16 GB of RAM.

As shown in Tab. 3, when 100 clients train for 1,000 rounds, it takes the auditor 19 seconds to verify all EDR signatures and construct the EDG, which is negligible in comparison to the FL training time.<sup>4</sup> The incurred storage cost during auditing is 162 MB. We conclude that EXCLAVEFL’s auditing process introduces a small overhead, even with a large number of clients and training rounds.

## 8 Related Work

**Mitigating attacks in FL.** Aggregation protocols can be designed to mitigate privacy/integrity attacks in FL. *Secure aggregation* [9, 73] addresses FL privacy concerns when aggregators are untrusted through cryptographic means. For example, Bonawitz et al. [9] implement secure aggregation using pairwise Diffie-Hellman key agreement and t-out-of-n Shamir’s secret sharing; Sharma et al. [73] use additive secret sharing. These protocols are complementary to EXCLAVEFL, because they ensure privacy, while EXCLAVEFL provides verifiable integrity guarantees that the correct computation was performed. Secure aggregation protocols could be used with EXCLAVEFL to provide both privacy and integrity guarantees. EXCLAVEFL is compatible with a range of FL-specific techniques: *robust aggregation* [64] makes the FL training process more resilient

<sup>4</sup>Verifying each exclave signing key against its endorsement is a one-time, per-exclave startup cost, and thus not included in the costs measured for each EDR.

against integrity attacks; Tolpegin et al. [77] detect data poisoning attacks using outlier detection. To protect FL training against model poisoning attacks, Krum [8] and Trimmed-mean [86] are *Byzantine-robust aggregation* protocols. While such protocols require at least a majority of trusted data providers, EXCLAVEFL’s verifiable claims hold with any number of malicious participants.

**Trusted execution for FL.** Mo et al. [55] assume that both the FL server and clients are honest-but-curious. They use TEEs to prevent participants from observing the model during training, thus preventing inference or data reconstruction attacks. Messaoud et al. [52] address client-side privacy threats by protecting gradient updates inside TEEs. Since a subset of the model layers is protected from gradient leakage, this mitigates inference attacks. Both of the approaches assume a weaker threat model, without EXCLAVEFL’s detection of integrity attacks. CrowdGuard [69] uses TEEs for integrity-preserving FL. In each round, data providers evaluate the locally trained models of other data providers on their own data inside TEEs and vote on the poisoned model. Unlike EXCLAVEFL, CrowdGuard assumes TEE confidentiality, making it more vulnerable to side-channel attacks. Laminator [19] and Atlas [75] use TEEs to attest certain properties of centralized ML training and inference. Indeed, the claims produced by EXCLAVEFL during FL training could be used by Laminator when performing attested inference. With minor modifications, both Laminator and Atlas could be adapted to use exclaves to remove reliance on TEE confidentiality.

**Ensuring integrity.** *Proof-carrying code* [57] and *model checking* [13] can show that code complies with integrity policies by verifying it against a specification. Since EXCLAVEFL relies on execution integrity, it can be combined with such approaches. Other techniques enforce *control-flow integrity* during execution: C-FLAT [2] and LO-FAT [15] attest intra-program control flow to ascertain the current execution state of a single node. In contrast, EXCLAVEFL attests task-level data transformations across distributed participants, allowing for verifiable claims about a final trained FL model. While there are similarities (e.g., the use of hashes), the attestation objective is different, e.g., two FL training runs with identical control-flow attestations could still produce different final models.

## 9 Conclusions

We described EXCLAVEFL, a new system for executing FL training jobs that enables an auditor to verify claims about the final trained FL model. EXCLAVEFL introduces *exclaves* as a new abstraction for the execution of FL tasks *with integrity only*, which attest all performed data transformations at runtime. Since executed FL tasks are attested through exclave data records (EDRs), participants cannot deviate from the correct FL training protocol—thus violating model claims—without detection. Exclaves produce EDRs, which can be combined to create an exclave dataflow graph (EDG) that spans the entire FL job. An auditor uses the EDG to verify end-to-end claims about the FL job. We showed that EXCLAVEFL can be integrated with a state-of-the-art FL framework (NVFlare) to detect a wide range of deviations with low runtime overhead.

**Acknowledgments.** We thank the reviewers and our shepherd for their insightful comments and feedback. Jinnan Guo is supported by the President’s PhD Scholarship at Imperial College London.

## References

- [1] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. 2024. Zero-knowledge proofs of training for deep neural networks. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*. 4316–4330.
- [2] Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. C-FLAT: Control-Flow Attestation for Embedded Systems Software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) (CCS '16). Association for Computing Machinery, New York, NY, USA, 743–754. doi:10.1145/2976749.2978358
- [3] AMD. 2020. Strengthening VM isolation with integrity protection and more. (2020).
- [4] ARM. 2024. Arm Confidential Compute Architecture. <https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture>
- [5] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O’Keeffe, Mark L Stillwell, et al. 2016. SCONe: Secure Linux containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 689–703.
- [6] Jens Axboe. 2026. Flexible I/O Tester (fio). <https://github.com/axboe/fio>. Accessed: 2026-07-06.
- [7] Alexis Bagia, Vincent Quentin Ulitzsch, Daniël Trujillo, Mengyuan Li, Mengjia Yan, and Jean-Pierre Seifert. 2025. A Close Look at RMP Entry Caching and Its Security Implications in SEV-SNP. In *Proceedings of the 14th International Workshop on Hardware and Architectural Support for Security and Privacy*. 10–18.
- [8] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems* 30 (2017).
- [9] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.
- [10] Felipe Bravo-Marquez, Steve Reeves, and Martin Ugarte. 2019. Proof-of-learning: A blockchain consensus mechanism based on machine learning competitions. In *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*. IEEE, 119–124.
- [11] Robert Bühren, Christian Werling, and Jean-Pierre Seifert. 2019. Insecure until proven updated: analyzing AMD SEV’s remote attestation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1087–1099.
- [12] Pau-Chen Cheng, Wojciech Ozga, Enriquito Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel TDX demystified: A top-down approach. *Comput. Surveys* 56, 9 (2024), 1–33.
- [13] Edmund M Clarke. 1997. Model checking. In *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings* 17. Springer, 54–56.
- [14] Coconut SVSM community. 2026. Coconut Secure VM Service Module. <https://github.com/coconut-svsm/svsm>.
- [15] Ghada Dessouky, Shaza Zeitouni, Thomas Nyman, Andrew Paverd, Lucas Davi, Patrick Koeberl, N Asokan, and Ahmad-Reza Sadeghi. 2017. LO-FAT: Low-overhead control flow attestation in hardware. In *Proceedings of the 54th Annual Design Automation Conference* 2017. 1–6.
- [16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.
- [17] Gobikrishna Dhanuskodi, Sudeshna Guha, Vidhya Krishnan, Aruna Manjunatha, Michael O’Connor, Rob Nertney, and Phil Rogers. 2023. Creating the First Confidential GPUs: The team at NVIDIA brings confidentiality and integrity to user code and data for accelerated computing. *Queue* 21, 4 (2023), 68–93.
- [18] Rezarta Islamaj Doğan, Robert Leaman, and Zhiyong Lu. 2014. NCBI disease corpus: a resource for disease name recognition and concept normalization. *Journal of biomedical informatics* 47 (2014), 1–10.
- [19] Vasisht Duddu, Oskari Järvinen, Lachlan J. Gunn, and N. Asokan. 2025. Laminator: Verifiable ML Property Cards using Hardware-assisted Attestations. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy (CODASPY '25)*. doi:10.1145/3714393.3726492
- [20] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.
- [21] EU. 2024. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689>. Accessed: 2025-09-22.
- [22] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. 2023. Proof-of-Learning is Currently More Broken Than You Think. In *2023 IEEE 8th European Symposium on Security and Privacy*. 797–816. doi:10.1109/EuroSP57164.2023.00052
- [23] Jessie Frazelle. 2019. Securing the Boot Process: The hardware root of trust. *Queue* 17, 6 (2019), 5–21.
- [24] Edoardo Gabrielli, Dimitri Belli, Zoe Matrullo, Vittorio Miori, and Gabriele Tolomei. 2023. Protecting federated learning from extreme model poisoning attacks via multidimensional time series anomaly detection. *arXiv preprint arXiv:2303.16668* (2023).
- [25] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujif, Mohammad Mahmood, Guru-Vamsi Policharla, and Mingyuan Wang. 2023. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1880–1894.
- [26] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for Datasets. arXiv:1803.09010 [cs.DB] <https://arxiv.org/abs/1803.09010>
- [27] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. SafetyNets: Verifiable execution of deep neural networks on an untrusted cloud. *Advances in Neural Information Processing Systems* 30 (2017).
- [28] Jinnan Guo, Peter Pietzuch, Andrew Paverd, and Kapil Vaswani. 2024. Trustworthy AI Using Confidential Federated Learning. *Commun. ACM* 67, 9 (2024), 48–53.
- [29] Krishnaprasad Hande. 2023. Announcing Azure Confidential VMs with NVIDIA H100 Tensor Core GPUs in Preview. <https://techcommunity.microsoft.com/t5/azure-confidential-computing/announcing-azure-confidential-vm-with-nvidia-h100-tensor-core/ba-p/3975389>
- [30] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).
- [31] Alejandro Hernández-Cano, Alexander Hägele, Allen Hao Huang, Angelika Romanou, Antoni-Joan Solergibert, et al. 2025. Apertus: Democratizing Open and Compliant LLMs for Global Language Environments. arXiv:2509.14233 [cs.CL] <https://arxiv.org/abs/2509.14233>
- [32] Kasun Indrasiri and Danesh Kuruppu. 2020. gRPC: up and running: building cloud native applications with Go and Java for Docker and Kubernetes. O’Reilly Media.
- [33] Intel. 2026. Intel® TDX Module Extension for Quoting. <https://www.intel.com/content/www/us/en/content-details/874303/intel-tdx-module-extension-for-quoting.html>. Accessed: 2026-06-13.
- [34] Malhar S Jere, Tyler Farnan, and Farinaz Koushanfar. 2021. A taxonomy of attacks on federated learning. *IEEE Security & Privacy* 19, 2 (2021), 20–28.
- [35] Hengrui Jia, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. 2021. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1039–1056.
- [36] Ehsanul Kabir, Zeyu Song, Md Rafi Ur Rashid, and Shagufta Mehnaz. 2024. FLShield: a validation based federated learning framework to defend against poisoning attacks. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2572–2590.
- [37] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.
- [38] David Kaplan. 2024. Hardware VM Isolation in the Cloud. *Commun. ACM* 67, 1 (2024), 54–59.
- [39] Linux kernel development community. 2024. dm-verity. <https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/verity.html>.
- [40] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 1–19.
- [41] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
- [42] Eugene Kuznetsov, Yitao Chen, and Ming Zhao. 2021. SecureFL: Privacy Preserving Federated Learning with SGX and TrustZone. In *Proceedings of the Sixth ACM/IEEE Symposium on Edge Computing (SEC)*. ACM, San Jose, CA, USA, 55–67. doi:10.1145/3453142.3491287
- [43] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
- [44] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.
- [45] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. 2022. A systematic look at ciphertext side channels

- on AMD SEV-SNP. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 337–351.
- [46] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. 2021. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *30th USENIX Security Symposium (USENIX Security 21)*. 717–732.
- [47] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. 2023. Enabling realms with the arm confidential compute architecture. *USENIX; login* (2023).
- [48] Min Lyu, Dong Su, and Ninghui Li. 2016. Understanding the sparse vector technique for differential privacy. *arXiv preprint arXiv:1603.01699* (2016).
- [49] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.
- [50] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* (2016).
- [51] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*. Springer, 369–378.
- [52] Aghiles Ait Messaoud, Sonia Ben Mokhtar, Vlad Nitu, and Valerio Schiavoni. 2021. GradSec: a TEE-based Scheme Against Federated Learning Inference Attacks. In *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning* (Virtual Event, Germany) (ResilientFL '21). Association for Computing Machinery, New York, NY, USA, 10–12. doi:10.1145/3477114.3488763
- [53] Microsoft. 2026. Virtual TPMs in Azure confidential VMs. <https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-tpms-in-azure-confidential-vm>. Accessed: 2026-07-07.
- [54] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*. 220–229.
- [55] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: Privacy-Preserving Federated Learning with Trusted Execution Environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 94–108. doi:10.1145/3458864.3466628
- [56] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>. Accessed: 2026-08-05.
- [57] George C Necula. 1997. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 106–119.
- [58] Rob Nertney. 2023. Confidential Compute on NVIDIA Hopper H100. <https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1.0.pdf>
- [59] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. 2016. TrustZone Explained: Architectural features and use cases. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 445–451.
- [60] OpenAI. 2025. GPT-5 System Card. <https://cdn.openai.com/gpt-5-system-card.pdf>. Accessed: 2025-09-22.
- [61] Inkfit Padhi, Manish Nagireddy, Giandomenico Cornacchia, Subhajit Chaudhury, Tejaswini Pedapati, Pierre Dognin, Keerthiram Murugesan, Erik Miehl, Martin Santillán Cooper, Kieran Fraser, et al. 2024. Granite Guardian. *arXiv preprint arXiv:2412.07724* (2024).
- [62] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. 2022. Eluding secure aggregation in federated learning via model inconsistency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2429–2443.
- [63] Maksym Petkus. 2019. Why and how zk-SNARK works. *arXiv preprint arXiv:1906.07221* (2019).
- [64] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. 2022. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing* 70 (2022), 1142–1154.
- [65] Do Le Quoc and Christof Fetzer. 2021. SecFL: Confidential Federated Learning using TEEs. *CoRR abs/2110.00981* (2021). [arXiv:2110.00981](https://arxiv.org/abs/2110.00981) <https://arxiv.org/abs/2110.00981>
- [66] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* (2019).
- [67] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329* (2019).
- [68] G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, et al. 2021. OpenFL: An open-source framework for Federated Learning. *arXiv preprint arXiv:2105.06413* (2021).
- [69] Phillip Rieger, Torsten Krauß, Markus Miettinen, Alexandra Dmitrienko, and Ahmad-Reza Sadeghi. 2022. CrowdGuard: Federated backdoor detection in federated learning. *arXiv preprint arXiv:2210.07714* (2022).
- [70] Holger R Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, et al. 2022. NVIDIA FLARE: Federated learning from simulation to real-world. *arXiv preprint arXiv:2210.13291* (2022).
- [71] Victor Sanh, L Debut, J Chaumond, and T Wolf. 2019. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
- [72] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-privilege-boundary data sampling. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 753–768.
- [73] Shreya Sharma, Chaoping Xing, Yang Liu, and Yan Kang. 2019. Secure and efficient federated transfer learning. In *2019 IEEE international conference on big data (Big Data)*. IEEE, 2569–2576.
- [74] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
- [75] Marcin Spoczynski, Marcela S. Melara, and Sebastian Szyller. 2025. Atlas: A Framework for ML Lifecycle Provenance & Transparency. In *2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. doi:10.1109/EuroSPW67616.2025.00058
- [76] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2158–2170.
- [77] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursay, and Ling Liu. 2020. 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, 480–501.
- [78] Florian Tramer, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. 2017. Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE Computer Society, Los Alamitos, CA, USA, 19–34. doi:10.1109/EuroSP.2017.28
- [79] Hannes Tschofenig, Yaron Sheffer, Paul Howard, Ionuț Mihalcea, Yogesh Deshpande, Arto Niemi, and Thomas Fossati. 2024. *Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. Internet-Draft draft-fossati-tls-attestation-08. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-fossati-tls-attestation/08/> Work in Progress.
- [80] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient Out-of-Order execution. In *27th USENIX Security Symposium (USENIX Security 18)*. 991–1008.
- [81] Kapil Vaswani, Stavros Volos, Cédric Fournet, Antonio Nino Diaz, Ken Gordon, Balaji Vembu, Sam Webster, David Chisnall, Saurabh Kulkarni, Graham Cunningham, et al. 2023. Confidential Computing within an AI Accelerator. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 501–518.
- [82] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. 2023. PwrLeak: Exploiting Power Reporting Interface for Side-Channel Attacks on AMD SEV. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 46–66.
- [83] Zhibo Wang, Jingjing Ma, Xue Wang, Jiahui Hu, Zhan Qin, and Kui Ren. 2022. Threats to training: A survey of poisoning attacks and defenses on machine learning systems. *Comput. Surveys* 55, 7 (2022), 1–36.
- [84] Hasini Witharana, Debapriya Chatterjee, and Prabhat Mishra. 2024. Verifying Memory Confidentiality and Integrity of Intel TDX Trusted Execution Environments. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 44–54. doi:10.1109/HOST55342.2024.10545349
- [85] Wensi Yang, Yuhang Zhang, Kejiang Ye, Li Li, and Cheng-Zhong Xu. 2019. FFD: A federated learning based method for credit card fraud detection. In *Big Data–BigData 2019: 8th International Congress, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings* 8. Springer, 18–32.
- [86] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*. PMLR, 5650–5659.
- [87] Hangfan Zhang, Jinyuan Jia, Jinghui Chen, Lu Lin, and Dinghao Wu. 2023. A3FL: Adversarially adaptive backdoor attacks to federated learning. *Advances in neural information processing systems* 36 (2023), 61213–61233.
- [88] Huanguo Zhang, Zhongping Qin, and Qi Yang. 2008. Design and Implementation of the TPM chip J3210. In *2008 Third Asia-Pacific Trusted Infrastructure Technologies Conference*. IEEE, 72–78.
- [89] Rui Zhang, Jian Liu, Yuan Ding, Zhibo Wang, Qingbiao Wu, and Kui Ren. 2022. “Adversarial Examples” for Proof-of-Learning. In *2022 IEEE Symposium on Security and Privacy (SP)*. 1408–1422. doi:10.1109/SP46214.2022.9833596