Holding Secrets Accountable: Auditing Privacy-Preserving Machine Learning

Hidde Lycklama, *ETH Zurich;* Alexander Viand, *Intel Labs;* Nicolas Küchler, *ETH Zurich;* Christian Knabenhans, *EPFL;* Anwar Hithnawi, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity24/presentation/lycklama

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

Holding Secrets Accountable: Auditing Privacy-Preserving Machine Learning

Hidde Lycklama[1], Alexander Viand[2], Nicolas Küchler[1], Christian Knabenhans[3], Anwar Hithnawi[1]

[1]ETH Zurich [2]Intel Labs [3]EPFL

Abstract

Recent advancements in privacy-preserving machine learning are paving the way to extend the benefits of ML to highly sensitive data that, until now, has been hard to utilize due to privacy concerns and regulatory constraints. Simultaneously, there is a growing emphasis on enhancing the transparency and accountability of ML, including the ability to audit deployments for aspects such as fairness, accuracy and compliance. Although ML auditing and privacy-preserving machine learning have been extensively researched, they have largely been studied in isolation. However, the integration of these two areas is becoming increasingly important. In this work, we introduce Arc, an MPC framework designed for auditing privacy-preserving machine learning. Arc cryptographically ties together the training, inference, and auditing phases to allow robust and private auditing. At the core of our framework is a new protocol for efficiently verifying inputs against succinct commitments. We evaluate the performance of our framework when instantiated with our consistency protocol and compare it to hashing-based and homomorphic-commitment-based approaches, demonstrating that it is up to $10^4\times$ faster and up to $10^6\times$ more concise.

1 Introduction

Mounting concerns regarding security and privacy in Machine Learning (ML) have spurred interest in Privacy-Preserving Machine Learning (PPML) [42, 68]. These developments aim to address concerns related to user data, whether during inference or training, as well as securing ML models, as organizations seek to maintain a competitive advantage by keeping them confidential. Consequently, secure inference and secure training frameworks have emerged to address various security and privacy concerns inherent in using and training machine learning models [27, 47, 54, 56, 66, 88]. The majority of these frameworks rely on secure computation techniques [6, 50, 51], which offer security guarantees by hiding the data and/or the model during computation. While these techniques are effective in achieving the intended security goals, they also introduce new challenges due to their inherent opacity. To achieve secrecy, these technologies conceal the processes of training and inference, making it difficult to audit PPML pipelines for fairness, transparency, accountability, and other (often legally mandated) objectives. However, a variety of scenarios require both the privacy guarantees of PPML and the ability to audit the ML pipeline. For example, banks interested in collaborating on training better risk assessment algorithms (e.g., for creditworthiness) will need to rely on PPML techniques to protect both customer privacy and commercial information. At the same time, they must also fulfill legal requirements for auditability such as those of the recently enacted EU AI Act [25], which specifically calls out creditworthiness evaluation [79]. While auditability may seem to be in direct conflict with the privacy requirements of PPML, secure computation can, in principle, offer a way forward for verifying these properties while preserving privacy. However, realizing this in an efficient and robust manner is challenging.

Private ML Auditing. ML auditing involves the examination and verification of machine learning models, algorithms, and data to ensure accountability and desired properties such as fairness, transparency, and accuracy during deployment. Approaches to ML auditing can be divided into a priori and post-hoc auditing mechanisms [9]. The former focus on pre-deployment verification techniques that act as predefined sets of verifications on the model, data, or training process [22, 44], such as model and data validation tests [19, 63] or robustness [31] and fairness [7] verification. While covering important use cases of auditing, these remain limited to known prior issues, which, in the case of ML, are hard to exhaustively address given the black box nature of ML. Post-hoc audits, which are triggered in response to detecting undesirable behavior or other triggers, are therefore essential to ensure accountability in real-world deployments of ML [40, 80, 81]. For example, individuals may seek an explanation of a decision to mitigate potential harm or to investigate its fairness [62, 74]. Recent efforts have examined the realization of a priori-auditing techniques in secure settings using a variety of ad-hoc techniques. This includes work for verifying robustness, verifiable fairness, and model and data validation techniques [19, 46, 53, 78]. Post-hoc audits, however, have received scant attention in the secure setting. Due to their on-demand nature, they present a unique set of challenges that a priori audits do not face. In this paper, we, therefore, focus primarily on achieving secure post-hoc audits for PPML.

Secure Post-Hoc Audits. In current practice, auditing of PPML systems generally requires assuming a trusted third party (which can be granted access to training data, models, and predictions) that applies traditional auditing solutions. However, in addition to undermining the privacy-preserving nature of PPML, even a trusted auditor is not sufficient to achieve robust audits. Specifically, parties might inadvertently or maliciously alter their inputs to the auditing phase so that they no longer match their original inputs to the PPML system, distorting the results of the auditing phase. Instead, the auditor would need visibility into the entire training and inference process to ensure the consistency of the audit. One might consider realizing such a trusted auditor cryptographically, by relying on (maliciously secure) Multi-Party Computation (MPC) for the entire pipeline. In practice, however, it is generally not feasible to continuously run large MPC deployments with many parties (e.g., different clients receiving inferences and/or different auditing parties). This is because MPC, in general, scales extremely poorly in the number of involved parties, and due to the complexities of maintaining (and periodically refreshing) a large amount of secret state over extended periods [41, 65, 77]. Note that PPML systems usually sidestep these issues, as training and inference can be realized as distinct phases. As a result, the (usually significant) resources utilized for training do not need to be maintained in order to perform inferences. A practical approach to cryptographic auditing for PPML, therefore, needs to maintain this decoupling while nevertheless ensuring consistent audits.

Contribution. This paper presents Arc, a new framework for privacy-preserving auditing of PPML systems. Arc is highly modular and supports a wide range of efficient PPML approaches and auditing functions, and is the first framework to efficiently implement post-hoc auditing for PPML. Our framework ties together training, inference, and auditing while maintaining consistency via the use of concise cryptographic *receipts*. Arc supports a wide range of PPML approaches, including mixed secure/plaintext settings which are common in practical deployments but pose significant challenges for auditing. The overhead of our framework is primarily determined by the efficiency of the underlying consistency mechanism. We first describe (and prove secure) our auditing protocol using a black-box definition of the consistency layer. We then present a highly efficient instantiation that makes Arc practical for a wide range of PPML deployment scenarios. Finally, we evaluate the performance of our framework when instantiated with our consistency protocol and compare it to hashing-based and homomorphic-commitment-based approaches, demonstrating that it is up to $10^4\times$ faster and up to $10^6\times$ more concise.

In the following, we discuss background and related work in §2. We present the requirements of PPML auditing systems and the design of our PPML auditing framework in §3. In §4, we formalize the Proof-of-Consistency (PoC) and present our consistency check protocol. In §5, we discuss how to realize common auditing functions under MPC. Finally, in §6, we evaluate our framework and compare against related work.

2 Background & Related Work

We briefly introduce relevant background for PPML and ML auditing, and then discuss related work.

Privacy-Preserving Machine Learning. PPML enables parties to securely train and deploy sensitive ML models in environments that involve untrusted or potentially compromised entities. There has been significant progress in PPML in recent years, leveraging advanced cryptographic techniques to ensure data privacy and model integrity [6, 14, 27, 47, 54, 56, 57, 66, 88]. Approaches that rely on MPC typically offer the best performance by distributing trust among n parties. These parties collaboratively execute training or inference computations, all while preserving the privacy of each party's inputs. Protocols are categorized based on the number of parties (t) an adversary can corrupt without breaching security, with distinctions made between a majority of honest parties ($t < \frac{n}{2}$) and a dishonest majority ($t < n$). Moreover, protocols are designed to withstand different adversarial behaviors, ranging from passive corruption, where compromised parties may collude to learn information while following the protocol honestly, to active corruption, allowing adversaries to deviate from the protocol arbitrarily. As of today, the most efficient MPC protocols for PPML rely on homomorphic secret sharing over a field $\mathbb{F}_p$ or ring $\mathbb{Z}_{2^k}$ [6]. This allows them to perform integer arithmetic by adding and scaling shares using the homomorphism of the scheme. Communication among parties is only required during the multiplication of shares. PPML frameworks frequently also offer higher-order primitives essential for machine learning, such as dot products, comparisons, bit extraction, exponentiation, and truncation [6, 27, 51, 56, 66, 88]. Different functionalities might be implemented most efficiently in different fields or rings, in which case we can use share conversion to switch between them, e.g., ring and field-based MPC. In Appendix E in the extended version of the paper [1], we discuss this technique in more detail.

ML Auditing. Auditing of ML systems is an emerging field focused on enhancing the accountability of ML algorithms.

Auditing involves verifying the compliance of organizations' ML models with safety and legal standards, e.g., ensuring they do not infringe on copyright laws. Here, we refer to auditing techniques that analyze an algorithm to offer further insights or assurances regarding the model and its predictions. This includes efforts to enhance transparency by explaining predictions, ensuring fairness, or providing accountability for the contributions of different parties. Depending on the technique, these algorithms may require access to the training data, the model, the prediction, or a combination thereof. Techniques that involve only the training data and the model can often be conducted a priori as part of an internal quality assurance process. However, such a priori techniques are inherently limited: important classes of auditing techniques fundamentally require, or only become practical with, access to the prediction sample. For example, audits for explainability/accountability or fairness, respectively. In addition, due to the nature of ML, we can frequently only identify a prediction as unwanted after the fact, potentially even only after significant time has passed. As a result, we require the ability to perform *post-hoc* auditing in real-world ML deployments.

A wide range of approaches for post-hoc auditing have been proposed in the literature, many of which build upon similar techniques. For example, many algorithms in this space rely on perturbing input data or prediction features to assess the impact of such changes on the model's behavior, effectively treating the model as a black box. These methods find application in a variety of contexts, such as providing explanations for predictions [62, 74], investigating the model's training data for biased or poisoned samples [64, 81, 90], or ensuring fairness by analyzing model predictions under hypothetical scenarios where specific input features are altered [67]. As evaluating these methods can be resource-intensive, alternative techniques employ propagation-based methods, which are more computationally efficient by assuming knowledge of the model's internal structure. These methods attribute importance to model neurons, input features, or training samples based on gradients or activations [5, 86]. These techniques share foundational computational operations with training and inference processes, such as forward passes through the neural network and backpropagation. This similarity in computational models implies that the protocols developed for training and inference can be repurposed, to some extent, for auditing purposes. In §5, we provide a detailed description of the algorithmic aspects of the auditing functions supported in our framework.

Related Work. While this is, to the best of our knowledge, the first framework for (post-hoc) PPML auditing, our work is closely related to efforts aimed at enhancing the reliability of PPML systems. Thus, we briefly discuss the most relevant related work here. Prior research primarily focuses on narrow aspects, enhancing isolated components and instantiations of the PPML pipeline as shown in Table 1. Phoenix integrates randomized smoothing techniques into Fully Homomorphic Encryption (FHE)-based ML inference to guarantee robust and fair model predictions [46]. Holmes improves the quality of MPC training to conduct distribution tests on training data via efficient interactive zero-knowledge proofs before training starts [19]. These works apply and optimize reliability techniques to PPML inference and training but do not allow for retroactive auditing of predictions or training data.

	Mal. Sec.	T	M	I	Co	Ba	St
Phoenix [46]	×	○	●	●	2/4	3/4	–
Agrawal et al. [2]	×	○	●	○	2/4	2/4	4/4
Kilbertus et al. [53]	✓	○	●	●	1/4	1/4	4/4
Segal et al. [78]	✓	○	●	○	1/4	1/4	4/4
Holmes [19]	✓	●	○	○	4/4	4/4	–
Cerebro [93]	✓	●	○	○	1/4	3/4	1/4
Ours (§3)	✓	●	●	●	4/4	4/4	4/4

Table 1: Related work covers different subsets of the PPML pipeline by allowing to audit combinations of the training data (T), the model (M) and the inference (I), and have different overheads for compute (Co), bandwidth (Ba) and storage (St).

Another line of work enables retroactive verification of certain properties, but only for specific components of the PPML pipeline in isolation. Fairness certification allows clients to verify that their private predictions were generated by a certified model [3, 53, 78]. This is usually achieved by having a regulator sign a hash-based commitment of the model. Cerebro [93] extends MPC training by enabling an auditor to conduct post-hoc computation on parties' inputs through a consistency check involving cryptographic commitments. However, their system only allows auditing of parties' datasets individually, which significantly limits the scope of auditing. Additionally, the commitment techniques they employ to ensure the integrity of the training data do not scale to a complete PPML system handling large amounts of training data and potentially many clients.

3 Arc Design

Arc enables private and secure auditing for existing PPML systems by providing end-to-end consistency of data, model, and predictions while preserving the benefits of distinct training, inference, and auditing phases. Our framework is highly modular and supports a wide range of inference and training approaches, including deployments that mix secure computation and plaintext computation. For example, many scenarios permit the release of a differentially private model after a secure training phase, or consider secure inference for a centrally trained model. Our framework allows the execution of arbitrary auditing functions over the original training data, model, and predictions using secure multi-party computation.

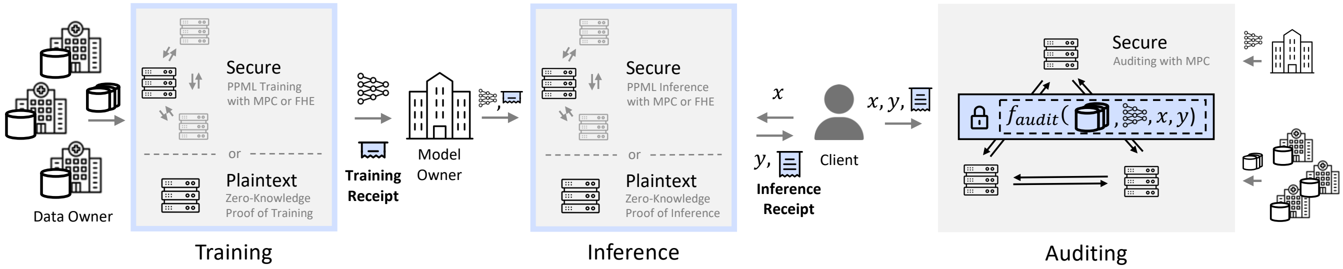

Figure 1: Overview of Arc, which augments existing PPML pipelines with an MPC auditing phase to execute auditing functions. Clients receive a receipt that can later be used to verify the consistency of the training data, model and prediction under audit.

Notably, these audits can be conducted post-hoc, i.e., long after the training or inference phases. In Fig. 1, we illustrate how Arc extends a typical PPML pipeline. Specifically, Arc augments the training phase with a consistency layer that generates a concise cryptographic receipt linking the model received by the model owners and the training data provided by the data owners. During inference, Arc's consistency layer extends this receipt to include the client's prediction sample and result. Using this receipt, the client can later request audit functions to be executed and verify that the data owners and model owners provided their original inputs. This design allows our framework to scale independently of the number of clients and predictions, as the only additional state necessary is the (concise) prediction held by the client.

Threat Model. We consider an actively malicious adversary that can (statically) compromise parties across the training, inference, and auditing phases. The adversary can observe and modify all inputs, states and network traffic of the parties it controls. We assume that at least one party that provides inputs (i.e., training data, model, or prediction sample) or receives outputs is honest. Note that because not all parties are involved in each phase, it is possible that all parties interacting in a phase are malicious. If a phase involves secure computation, we assume at least one of the computational parties is honest. Note that, certain instantiations of secure computation might impose additional constraints on the adversary. For example, Arc can be used with MPC protocols that assume an honest majority of computing parties, which are frequently significantly more efficient than their dishonest majority counterparts.

3.1 Modeling Cryptographic Auditing

A privacy-preserving auditing system must achieve secrecy, correctness, and soundness. For *secrecy*, the system must preserve the privacy guarantees of the underlying PPML systems, except for what can be inferred from the output of auditing[1]. However, to prevent unexpected leakage from malicious audit requests, the system must also be restricted to serving only valid auditing requests, i.e., those corresponding to actual predictions made by the system. Moreover, in order to allow us to rely on the results of auditing, the system must be *correct & sound*, i.e., the audit must be correctly computed even in the presence of malicious parties. In particular, the system must ensure the audit is performed on the original training data and model corresponding to the prediction that is being audited. The system must also have the ability to detect malicious disruptions of the audit process. Specifically, we want to prevent malicious parties from surreptitiously aborting the audit computation and therefore require Identifiable Abort (ID-Abort) security for the auditing phase. Otherwise, malicious actors could prevent auditing without fear of repercussion, fundamentally undermining the concept of auditability. While *publicly identifiable* abort (i.e., everyone, including input parties, learns the identity) would potentially be desirable in the secure outsourced computation (SOC) setting, protocols to achieve this introduce prohibitive overhead [26, 70] and we therefore only require traditional ID-Abort. For training and inference, in comparison, we require only security with abort, as is common in practical protocols for PPML training and inference [18, 21, 38, 66, 71, 88, 94].

As we prove our protocol secure in the real-ideal paradigm [15], we begin by modeling the ideal functionality that a privacy-preserving auditing protocol for PPML should achieve based on the requirements set out above. Despite the separation of the training, inference and auditing phases, we model the intended behavior as a single reactive ideal functionality $\mathcal{F}_{\text{Arc}}$ (cf. Fig. 2) as this directly implies input consistency. We annotate parts only relevant in the plaintext training and/or inference settings in which the adversary can generate local models and predictions (in olive and blue, respectively).

Training & Inference. The functionality allows training models on the input data (assuming the adversary does not choose to abort the computation), adding the resulting model to a list L_{M}. We assume that there is an out-of-band communication channel for learning which models exist and assume the adversary learns about any models that have been trained, even if all parties involved in the training were honest. In the functionality, we model this by leaking the model identifier to

[1] Special care should be taken when choosing auditing functions to ensure their output presents an acceptable privacy-utility trade-off.

Functionality $\mathcal{F}_{\text{Arc}}$

The functionality is parameterized by a learning algorithm $\mathcal{T}$, a set of allowed auditing functions $F_{\texttt{audit}}$, $N_{\texttt{DH}}$ data owners $\texttt{DH} = \{\texttt{DH}_1, \dots, \texttt{DH}_{N_{\texttt{DH}}}\}$, $N_{\texttt{M}}$ model owners $\texttt{M} = \{\texttt{M}_1, \dots, \texttt{M}_{N_{\texttt{M}}}\}$, $N_{\texttt{C}}$ clients $\texttt{C} = \{\texttt{C}_1, \dots, \texttt{C}_{N_{\texttt{C}}}\}$, $N_{\texttt{TC}}$ training computers $\texttt{TC} = \{\texttt{TC}_1, \dots, \texttt{TC}_{N_{\texttt{TC}}}\}$, $N_{\texttt{IC}}$ inference computers $\texttt{IC} = \{\texttt{IC}_1, \dots, \texttt{IC}_{N_{\texttt{IC}}}\}$, $N_{\texttt{AC}}$ audit computers $\texttt{AC} = \{\texttt{AC}_1, \dots, \texttt{AC}_{N_{\texttt{AC}}}\}$. We denote the set of all parties as $\mathcal{P} = \texttt{DH} \cup \texttt{M} \cup \texttt{C} \cup \texttt{TC} \cup \texttt{IC} \cup \texttt{AC}$, The functionality is reactive and its state consists of a set L_{M} of models and corresponding datasets, and a set L_{P} of inference samples and corresponding predictions, Operations only relevant to plaintext training are marked in olive and those only relevant to plaintext inference are marked in blue.

Training: On input $(\texttt{InputData}, D_i)$, store $(\texttt{DH}_i, D_i)$, in the plaintext setting: if $\mathcal{A}$ controls any $\texttt{TC}$, send D_i to $\mathcal{A}$. When the functionality has input D_i from all $\texttt{DH}_i$, clear all $(\texttt{DH}_i, D_i)$ and proceed with:

1. Set $J \xleftarrow{\$} \mathcal{J}$ and compute $M \leftarrow \mathcal{T}(D_1, \dots, D_{N_{\texttt{DH}}}, J)$. In the plaintext setting: if $\mathcal{A}$ controls any $\texttt{TC}$, send (J, M) to $\mathcal{A}$
2. If $\mathcal{A}$ controls any model owners, send M to $\mathcal{A}$. Otherwise, send $\perp$ to $\mathcal{A}$. Wait for a response $a \in \{\texttt{Abort}, \texttt{Deliver}\}$.
3. If a is $\texttt{Deliver}$, choose a random identifier id_M, and append $((D_1, \dots, D_{N_{\texttt{DH}}}), M, \mathsf{id}_M)$ to L_{M}, and send the model $(\texttt{Output}, \mathsf{id}_M, M)$ to each model owner $\texttt{M}_j$, and send (id_M) to $\mathcal{A}$. Otherwise, send $(\texttt{Output}, \perp)$ to all parties in $\mathcal{P}$.

Inference: On input $(\texttt{Predict}, \texttt{M}_j, \mathsf{id}_M, x)$ from $\texttt{C}_i$, the functionality does the following:

1. In the plaintext setting: if $\mathcal{A}$ controls $\texttt{IC}_i$, find M in L_{M} using id_M and send x, M to $\mathcal{A}$.
2. If $\mathcal{A}$ controls $\texttt{M}_j$, all $\texttt{DH}_i$, and all $\texttt{TC}_i$:
   - ask $\mathcal{A}$ for an alternative model M' and $N_{\texttt{DH}}$ datasets D'_k and training randomness J'.
   - If $\mathcal{A}$ returns $\perp$, continue with Step 3.
   - Else, if $M' = \mathcal{T}(D'_1, \dots, D'_{N_{\texttt{DH}}}, J')$, choose a random id'_M, add $((D'_1, \dots, D'_{N_{\texttt{DH}}}), M', \mathsf{id}'_M)$ to L_{M}, set $\mathsf{id}_M = \mathsf{id}'_M$ and send (id_M) to $\mathcal{A}$.
   - Otherwise, send $\texttt{Abort}$ to all parties.
3. Find M in L_{M} using id_M and compute $y \leftarrow M(x)$.
4. If $\mathcal{A}$ controls $\texttt{C}_i$, send (id_M, y) to $\mathcal{A}$.
5. If $\mathcal{A}$ controls $\texttt{M}_j$ or any $\texttt{IC}$, send (id_M) and (y) to $\mathcal{A}$.
6. Wait for a response $a \in \{\texttt{Abort}, \texttt{Deliver}\}$. If a is $\texttt{Abort}$, send $(\texttt{Output}, \perp)$ to all parties in $\mathcal{P}$.
   Else, if a is $\texttt{Deliver}$, send $(\texttt{Output}, \mathsf{id}_M, y)$ to $\texttt{C}_i$ and add $(\texttt{M}_j, \mathsf{id}_M, x, y)$ to L_{P}.

Auditing: On input $(\texttt{Audit}, \texttt{M}_j, \tilde{\mathsf{id}}_M, f_{\texttt{audit}}, \tilde{x}, \tilde{y}, \mathsf{aux})$ from $\texttt{C}_i$, the functionality does the following

1. If $f_{\texttt{audit}} \notin F_{\texttt{audit}}$, send $(\texttt{Malicious}, \texttt{C}_i)$ to all parties in $\texttt{AC}$ and halt.
2. If $(\cdot, \cdot, \mathsf{id}_M) \in L_{\text{M}}$, get $((D_1, \cdots, D_N), M, \mathsf{id}_M)$ from L_{M}. Else, send $(\texttt{Malicious}, \texttt{C}_i)$ to all parties in $\texttt{AC}$ and halt.
3. If $\mathcal{A}$ controls $\texttt{M}_j$ and all $\texttt{IC}_i$: append $(\texttt{M}_j, \mathsf{id}_M, \tilde{x}, \tilde{y})$ to L_{P} if $M(\tilde{x}) = \tilde{y}$, else, send $(\texttt{Malicious}, \texttt{C}_i)$ to all $\texttt{AC}$ and halt.
4. If $(\texttt{M}_j, \mathsf{id}_M, \tilde{x}, \tilde{y}) \notin L_{\text{P}}$ send $(\texttt{Malicious}, \texttt{C}_i)$ to all parties in $\texttt{AC}$ and halt.
5. Evaluate $o \leftarrow f_{\texttt{audit}}(\tilde{x}, \tilde{y}, M, D_1, \cdots, D_N, \mathsf{aux})$.
6. Send (o) to $\mathcal{A}$ if $\mathcal{A}$ controls $\texttt{C}_i$ and send $(\texttt{M}_j, \mathsf{id}_M, f_{\texttt{audit}}, \mathsf{aux})$ to $\mathcal{A}$ otherwise.
   Wait for a response $a \in \{(\texttt{Abort}, \texttt{P}), \texttt{Deliver}\}$, where $\texttt{P} \in \mathcal{P}$.
7. If a is $\texttt{Deliver}$, send $(\texttt{Output}, o)$ to $\texttt{C}$. If a is $(\texttt{Abort}, \texttt{P})$, send $(\texttt{Malicious}, \texttt{P})$ to all parties in $\texttt{AC}$.

Figure 2: Arc's Ideal Functionality.

the adversary after training. In the case of plaintext training, the functionality additionally leaks the input data and model to the adversary if it controls any training computer. Inference proceeds similarly, storing the inference result in the list L_{P}. However, we must take care to also model the special case that occurs if an adversary has control over all data owners, all training computers and at least one model owner. In this case, the adversary could locally generate a valid combination of data and model $((D_1, \cdots, D_N), M)$. In the ideal world, we model this by extending the functionality to accept such locally trained models and, if consistent with the training data, append them to L_{M} before proceeding with the inference.

Auditing. Any client that has received a prediction, can request audits of that prediction using any auditing function from a set of allowed auditing functions. During auditing, the lists L_{M} and L_{P} are used to verify that an auditing request is consistent. Similarly to the special case we had to consider during inference, we must model the ability to locally generate valid inferences if the adversary controls all inference computers and a model owner. In this case, the functionality appends such locally generated predictions to L_{P} (if internally consistent) before verifying the auditing request. During auditing, the adversary can still choose to abort the computation, but as we require auditing to achieve identifiable abort, the adversary must reveal the identity of at least one malicious party to the functionality. In the case of inconsistencies in the audit inputs, the functionality also aborts and identifies the (uniquely determined) party at fault to the audit computers.

3.2 Arc Protocol

Our protocol lifts existing PPML systems to the cryptographic auditing setting by augmenting training and inference to produce receipts that can later be used to verify the consistency of the training data, model, and prediction under audit. Supporting real-world deployments with many potential inference clients requires scaling independently of the number of inferences and clients in the system. At the same time, we want to minimize the state that clients need to store beyond the received predictions. In Arc, we achieve this through concise cryptographic receipts for training and inference, which allows clients to efficiently store all material necessary to later verify the consistency of training data, model, inference sample, and prediction during auditing. At the same time, receipts are cryptographically bound to specific inferences, i.e., clients cannot generate audit requests for predictions they did not receive. In the following, we give an overview of our protocol, Π_{Arc} (c.f. Fig. 3) and its building blocks. We prove the security of our protocol in the real/ideal world paradigm [15] in Appendix B in the extended version of the paper [1].

Building Blocks. We construct our protocol from several cryptographic primitives, including a reactive arithmetic black-box (ABB) interface to abstract PPML protocols. In addition, we require secure point-to-point channels between parties that participate in the same phase and, in the auditing phase, a secure broadcast channel in order to achieve identifiable abort. We assume that all parties (except for clients) have a cryptographic identity which is set up through a public-key infrastructure (PKI), and that clients can access the (public) identities of the other parties through the PKI. Our protocol makes use of standard signatures (cf. Definition A.5 in Appendix A [1]) and a Proof-of-Consistency (PoC) (cf. Definition 4.1) which acts like a commitment, but admits significantly more efficient instantiations in the secure computation setting, as we discuss in the next section. In the following, we will use commitment and PoC interchangeably.

Proof of Training/Inference. When training and inference are realized via secure computation, computational integrity follows directly from the guarantees of the underlying MPC protocol. However, supporting real-world deployment scenarios where either training or inference are computed centrally on plaintexts, requires explicit proofs of computational integrity. There has been significant work on increasingly efficient proofs for both training [36, 85] and inference [20, 48], making use of advances in SNARKs and related proof techniques. We model these as proof-of-training (PoT) or proof-of-inference (PoI) (cf. Definitions A.6 and A.7 in Appendix A [1]). Recently, Proof-of-Learning (PoL) [44] has emerged, aiming to provide an easier-to-generate alternative to PoT by relying only on heuristic assumptions rather than strong cryptographic assumptions [34]. However, PoL *verification* requires access to the original training data and is computationally expensive, requiring many epochs of (re-)training. Therefore, PoL is less attractive for our setting, where verification must be computed under MPC.

Training & Inference. The training and inference phase of Π_{Arc} both provide a layer of consistency around calls to the underlying PPML implementation (i.e., $\mathcal{F}_{\text{ABB}}.\mathsf{Train}_{\mathcal{T}}$ or $\mathcal{F}_{\text{ABB}}.\mathsf{Predict}$). While the inference computation is fully deterministic, training also requires randomness which we securely sample (i.e., $\mathcal{F}_{\text{RAND}}$ [10]) and commit to in order to prevent reordering attacks [83]. Otherwise, the two phases proceed nearly identically:

- *Input Commitments* (T.1/I.1). In addition to their inputs, parties must provide a commitment to their inputs and the protocol verifies the consistency of these commitments before proceeding. In the secure computation setting, this uses PoC.Check which involves an (efficient) multi-party computation. In the plaintext setting, the computing parties can simply locally recompute the commitments.
- *Computation Integrity* (T.2/I.2). After computing the underlying ML training or inference, the computing parties commit (in the secure setting, collaboratively under MPC) to the result and provide the result, commitment, and associated decommitment randomness to the output-receiving parties. As we later need to show that these outputs were the result of a valid computation, the computing parties attest to the integrity of the computation. In the secure setting, this can be achieved via a distributed signature, as at least one of the computing parties must be honest. In the plaintext setting, this requires a PoT or PoI, as we cannot rely on a split-trust assumption for integrity.
- *Input Integrity* (T.3/I.3). While the signature or proof tie the result to a valid computation, they do not provide sufficient guarantees about the inputs. Therefore, the input parties verify their inputs were used and provide signatures to attest to this.
- *Output Consistency* (T.4/I.4). The output from the computation is provided to the output-receiving parties (in the plain) along with the associated receipt data. These parties then verify that the signatures and proofs are valid, and that the outputs (and decommitment randomness) they received matches the commitments in the receipt.

Note that providing a plaintext inference result to the client is essential for a useful inference service. However, one could consider a variant of the protocol in which the model owners only receive shares of the model, rather than the plaintext model. While our protocol could be trivially extended to support this, this would both unnecessarily complicate the notation and would require the long-term storage of secret shares and potentially complicated operations such as secret-share

Figure 3: Protocol Π_{Arc}

Π_{Arc} is a protocol between $N_{\texttt{DH}}$ data owners $\texttt{DH} = \{\texttt{DH}_1, \ldots, \texttt{DH}_{N_{\texttt{DH}}}\}$, $N_{\texttt{M}}$ model owners $\texttt{M} = \{\texttt{M}_1, \ldots, \texttt{M}_{N_{\texttt{M}}}\}$, $N_{\texttt{C}}$ clients $\texttt{C} = \{\texttt{C}_1, \ldots, \texttt{C}_{N_{\texttt{C}}}\}$, $N_{\texttt{TC}}$ training computers $\texttt{TC} = \{\texttt{TC}_1, \ldots, \texttt{TC}_{N_{\texttt{TC}}}\}$, $N_{\texttt{IC}}$ inference computers $\texttt{IC} = \{\texttt{IC}_1, \ldots, \texttt{IC}_{N_{\texttt{IC}}}\}$, and $N_{\texttt{AC}}$ audit computers $\texttt{AC} = \{\texttt{AC}_1, \ldots, \texttt{AC}_{N_{\texttt{AC}}}\}$. Π_{Arc} is parameterized by a learning algorithm $\mathcal{T}$, a set of allowed auditing functions $F_{\texttt{audit}}$, a proof-of-consistency PoC as in Definition 4.1 [1], a signature scheme SIG as in Definition A.5 [1]. Protocol parts only relevant to plaintext training are marked in olive, and those only relevant to plaintext inference are marked in blue. In the case of plaintext training, Arc is also parameterized by a proof of training POT as in Definition A.6 [1] and, in the case of plaintext inference, a proof of inference POI as in Definition A.7 [1]. Π_{Arc} assumes access to an MPC protocol represented by instances of $\mathcal{F}_{\text{ABB}}$ and, in the case of plaintext training or plaintext inference, a distributed randomness source $\mathcal{F}_{\text{RAND}}$. Π_{Arc} also assumes access to a broadcast channel $\mathcal{F}_{\text{BC}}$ and an MPC protocol $\mathcal{F}_{\text{ABB [ID]}}$ with ID-Abort, also used by PoC internally.

Input: Each $\texttt{DH}_i$ holds their training dataset $D_i \in \mathbb{F}^{l \times d_i}$ consisting of a vector of d_i input feature vectors of size l. Each client $\texttt{C}_j$ holds a list of prediction samples $[x]$ where $x \in \mathbb{F}_p^l$ and a set of audit inputs which is a subset of $[x]$.

Initialize: All parties except the clients receive signing keys from $\mathcal{F}_{\text{PKI}}$. All parties receive all corresponding verification keys from $\mathcal{F}_{\text{PKI}}$. The parties also receive public setup parameters for PoC $\mathsf{pp_{poc}} \leftarrow \mathsf{PoC.Setup}(1^\lambda, d)$ where d is the maximum of all d_i and m, and (in the case of plaintext training) $\mathsf{pp_{pot}} \leftarrow \mathsf{POT.Setup}(1^\lambda)$ and (in the case of plaintext inference) $\mathsf{pp_{poi}} \leftarrow \mathsf{POI.Setup}(1^\lambda)$.

Training: The protocol proceeds as follows with training computers $\texttt{TC}$, data owners $\texttt{DH}$ and model owners $\texttt{M}$, using a new instance of $\mathcal{F}_{\text{ABB}}$:

T.1 Each data owner $\texttt{DH}_i$ samples a random decommitment value $r_{D_i} \xleftarrow{\$} \mathcal{R}$ and:

- Inputs D_i to $\mathcal{F}_{\text{ABB}}$ or sends (D_i, r_{D_i}) to all $\texttt{TC}$.
- Computes a commitment to the training dataset $c_{D_i} = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, D_i, r_{D_i})$ and sends (c_{D_i}) to all $\texttt{TC}$.
- Executes PoC $\mathsf{PoC.Check}(\mathsf{pp_{poc}}, c_{D_i}, [\![D_i]\!]; D_i, r_{D_i})$ with all $\texttt{TC}$ or each $\texttt{TC}$ verifies that $c_{D_i} = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, D_i, r_{D_i})$ for all $\texttt{DH}_i$.

T.2 Each training computer $\texttt{TC}_j$:

- Samples $[\![r_M]\!], [\![r_J]\!], [\![J]\!]$ using $\mathcal{F}_{\text{ABB}}.\mathsf{RAND}$ or all $\texttt{TC}$ and $\texttt{M}$ receive r_M, r_J, J from $\mathcal{F}_{\text{RAND}}$.
- Invoke $\mathcal{F}_{\text{ABB}}.\mathsf{Train}_{\mathcal{T}}([\![D_1]\!], \ldots, [\![D_{N_{\texttt{DH}}}]\!], [\![J]\!])$ to compute the model $[\![M]\!]$ or compute $M \leftarrow \mathcal{T}(D_1, \ldots, D_{N_{\texttt{DH}}}, J)$.
- Using $\mathcal{F}_{\text{ABB}}$, commit to the model $[\![c_M]\!] = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, [\![M]\!], [\![r_M]\!])$, randomness $[\![c_J]\!] = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, [\![J]\!], [\![r_J]\!])$ and open c_M, c_J to all $\texttt{TC}$, $\texttt{DH}$ and $\texttt{M}$ or compute $c_M \leftarrow \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, M, r_M)$ and $c_J \leftarrow \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, J, r_J)$.
- Compute $[\![\sigma_{\texttt{TC}}]\!] \leftarrow \mathsf{SIG.DistSign}(\mathsf{sk}_{\texttt{TC}_j}, c_{D_1} \| \ldots \| c_{D_{N_{\texttt{DH}}}} \| c_M \| c_J)$ and open $\sigma_{\texttt{TC}}$ to $\texttt{M}$, $\texttt{DH}$ using $\mathcal{F}_{\text{ABB}}$ or $\pi_{\text{T}} \leftarrow \mathsf{POT.Prove}(\mathsf{pp_{pot}}, (c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}, c_M, c_J); D_1, \ldots, D_{N_{\texttt{DH}}}, M, J, r_{D_1}, \ldots, r_{D_{N_{\texttt{DH}}}}, r_M, r_J)$.
- Send $(c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}})$ and open $[\![M]\!], [\![r_M]\!]$ to $\texttt{M}$ using $\mathcal{F}_{\text{ABB}}$ or send $(c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}, c_M, c_J, \pi_{\text{T}}, M, r_M, r_J, J)$ to all $\texttt{M}$.
- Send $(c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}, c_M, c_J, \pi_{\text{T}})$ to all data owners $\texttt{DH}$.

T.3 Each $\texttt{DH}_i$ checks that it received the same $(c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}, c_M, c_J, \pi_{\text{T}})$ from all $\texttt{TC}$, $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{TC}}, c_{D_1} \| \ldots \| c_{D_{N_{\texttt{DH}}}} \| c_M \| c_J, \sigma_{\texttt{TC}})$ or $\mathsf{POT.Verify}$ $(\mathsf{pp_{pot}}, c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}, c_M, c_J, \pi_{\text{T}})$, and its c_{D_i} is contained in $c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}$ and aborts otherwise. Then, each computes $\sigma_{\text{T}}^i \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}_{\texttt{DH}_i}, c_{D_1} \| \ldots \| c_{D_{N_{\texttt{DH}}}} \| c_M \| c_J)$ and sends σ_{T}^i to all $\texttt{M}$.

T.4 Each model owner $\texttt{M}_k$ checks each of the following and aborts if any fail:

- Verify that the $(c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}, \sigma_{\texttt{TC}})$ (and $c_M, c_J, M, r_M, r_J, J, \pi_{\text{T}}$) received from each $\texttt{TC}$ are consistent with each other.
- $c_M = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, M, r_M)$ and $c_J = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, J, r_J)$.
- The list of signatures $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{DH}_i}, c_{D_1} \| \ldots \| c_{D_{N_{\texttt{DH}}}} \| c_M \| c_J, \sigma_{\text{T}}^i)$ for each $\texttt{DH}_i$.
- $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{TC}}, c_{D_1} \| \ldots \| c_{D_{N_{\texttt{DH}}}} \| c_M \| c_J, \sigma_{\texttt{TC}})$ or $\mathsf{POT.Verify}$ $(\mathsf{pp_{pot}}, c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}, c_M, c_J, \pi_{\text{T}})$.

Inference: The protocol proceeds as follows between inference computers $\texttt{IC}$, client $\texttt{C}_i$ and model owner $\texttt{M}_k$, using a new instance of $\mathcal{F}_{\text{ABB}}$:

I.1 $\texttt{C}_i$ sends c'_M (identifying the requested model) to all $\texttt{IC}$, and inputs a prediction sample x to $\mathcal{F}_{\text{ABB}}$ or sends x to all $\texttt{IC}$, then:

- All $\texttt{IC}$ ask $\texttt{M}_k$ to send $(c, \sigma_{\text{T}}, \sigma_{\texttt{TC}}$ or $\pi_{\text{T}})$ where $c = (c_{D_1}, \ldots, c_{D_{N_{\texttt{DH}}}}, c_M, c_J)$ to $\texttt{IC}$, and input the model M to $\mathcal{F}_{\text{ABB}}$ or send M, r_M to $\texttt{IC}$.
- The inference computers abort if $c_M \neq c'_M$.
- $\texttt{M}_k$ executes $\mathsf{PoC.Check}$ $(\mathsf{pp_{poc}}, c_M, [\![M]\!]; M, r_M)$ with all $\texttt{IC}$ or each $\texttt{IC}$ checks $c_M = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, M, r_M)$ and aborts if it fails.

I.2 Each inference computer $\texttt{IC}_j$:

- Computes $[\![y]\!]$ by invoking $\mathcal{F}_{\text{ABB}}.\mathsf{Predict}([\![M]\!], [\![x]\!])$ or computes $y \leftarrow M(x)$.
- Samples $[\![r_x]\!], [\![r_y]\!]$ using $\mathcal{F}_{\text{ABB}}.\mathsf{RAND}$ or all $\texttt{IC}$ and $\texttt{DH}_i$ receive r_x, r_y $\mathcal{F}_{\text{RAND}}$.
- Computes $[\![c_x]\!] = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, [\![x]\!], [\![r_x]\!])$ and $[\![c_y]\!] = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, [\![y]\!], [\![r_y]\!])$ and opens c_x and c_y to $\texttt{IC}$, $\texttt{C}_i$ and $\texttt{M}_k$ using $\mathcal{F}_{\text{ABB}}$ or computes $c_x = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, x, r_x)$ and $c_y = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, y, r_y)$ and send (c_x, c_y) to $\texttt{C}_i$ and $\texttt{M}_k$.
- Computes $[\![\sigma_{\texttt{IC}}]\!] \leftarrow \mathsf{SIG.DistSign}(\mathsf{sk}_{\texttt{IC}_j}, c \| c_x \| c_y)$ & opens $\sigma_{\texttt{IC}}$ to $\texttt{C}_i, \texttt{M}_k$ using $\mathcal{F}_{\text{ABB}}$ or $\pi_{\text{I}} \leftarrow \mathsf{POI.Prove}(\mathsf{pp_{poi}}, c_M, c_x, c_y; M, x, y, r_M, r_x, r_y)$.
- Sends $(c, \sigma_{\text{T}}, \sigma_{\texttt{TC}}$ or $\pi_{\text{T}})$ to $\texttt{C}_i$ and open $([\![y]\!], [\![r_x]\!], [\![r_y]\!])$ with $\mathcal{F}_{\text{ABB}}$ to $\texttt{C}_i$, or send $(c, \sigma_{\text{T}}, \sigma_{\texttt{TC}}$ or $\pi_{\text{T}}, \pi_{\text{I}}, y, r_x, r_y)$ to $\texttt{C}_i$. Sends (c, π_{I}) to $\texttt{M}_k$.

I.3 The model owner checks that it receives the same (c, c_x, c_y), that $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{IC}}, c \| c_x \| c_y, \sigma_{\texttt{IC}})$ or $\mathsf{POI.Verify}$ $(\mathsf{pp_{poi}}, c_M, c_x, c_y, \pi_{\text{I}})$, aborting otherwise, and computes $\sigma_{\text{I}} \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}_{\texttt{M}_k}, c \| c_x \| c_y \| \sigma_{\text{T}} \| \sigma_{\texttt{TC}}$ or $\pi_{\text{T}} \| \sigma_{\texttt{IC}}$ or $\pi_{\text{I}})$ and sends σ_{I} to $\texttt{C}_i$.

I.4 The client $\texttt{C}$ checks each of the following and aborts if any fails:

- Verify that the $(c, c_x, c_y, \sigma_{\text{T}}, \sigma_{\texttt{TC}}$ or $\pi_{\text{T}}, \sigma_{\texttt{IC}}$ or $\pi_{\text{I}}, y, r_y, r_x)$ received from each $\texttt{TC}$ are consistent with each other.
- $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{M}_k}, c \| c_x \| c_y \| \sigma_{\text{T}} \| \sigma_{\texttt{TC}}$ or $\pi_{\text{T}} \| \sigma_{\texttt{IC}}$ or $\pi_{\text{I}}, \sigma_{\text{I}})$ is a valid signature by $\mathsf{pk}_{\texttt{M}}$.
- Verify that $c_x = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, x, r_x)$ and $c_y = \mathsf{PoC.Commit}(\mathsf{pp_{poc}}, y, r_y)$.
- The list of signatures $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{DH}_i}, c, \sigma_{\text{T}}^i)$ for each $\texttt{DH}_i$, and $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{IC}}, c \| c_x \| c_y, \sigma_{\texttt{IC}})$ or $\mathsf{POI.Verify}$ $(\mathsf{pp_{poi}}, c_M, c_x, c_y, \pi_{\text{I}})$.

Figure 3: Protocol Π_{Arc} (cont.)

Auditing: The protocol proceeds as follows on a new instance of $\mathcal{F}_{\text{ABB [ID]}}$ between computing parties $\texttt{AC}$, a client $\texttt{C}_j$ and the model owner $\texttt{M}_k$:

A.1 The client $\texttt{C}_j$ inputs (x, y) to $\mathcal{F}_{\text{ABB [ID]}}$ and broadcasts $(c, c_x, c_y, \sigma_\text{I}, \sigma_\text{T}, \sigma_{\texttt{TC}}$ or $\pi_\text{T}, \sigma_{\texttt{IC}}$ or $\pi_\text{I}, \mathsf{pk}_{\texttt{M}_k}, f_{\texttt{audit}}, \mathsf{aux})$ to all parties using $\mathcal{F}_{\text{BC}}$.

A.2 All parties check that $\mathsf{pk}_{\texttt{M}_k}$ is a valid identity from $\mathcal{F}_{\text{PKI}}$, verify the model owner signature with $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{M}_k}, c \| c_x \| c_y \| \sigma_\text{T} \| \sigma_{\texttt{TC}}$ or $\pi_\text{T} \| \sigma_{\texttt{IC}}$ or $\pi_\text{I}, \sigma_\text{I})$ and check that $f_{\texttt{audit}} \in F_{\texttt{audit}}$. Otherwise, each party aborts marking $\texttt{C}_j$ as malicious.

A.3 **Verify Audit Requester:** The client $\texttt{C}_j$ runs $\mathsf{PoC.Check}_{[\text{ID}]}(\mathsf{pp}_{\mathsf{poc}}, c_x, [\![x]\!]; x, r_x), \mathsf{PoC.Check}_{[\text{ID}]}(\mathsf{pp}_{\mathsf{poc}}, c_y, [\![y]\!]; y, r_y)$ with the audit computers to prove to the $\texttt{AC}$ that its inputs x and y are consistent with c_x and c_y. If any of the checks fail, $\texttt{AC}_i$ aborts marking $\texttt{C}_j$ as malicious.

A.4 **Verify Inference:** The model owner inputs the model M to $\mathcal{F}_{\text{ABB [ID]}}$:

- The model owner $\texttt{M}_k$ runs $\mathsf{PoC.Check}_{[\text{ID}]}(\mathsf{pp}_{\mathsf{poc}}, c_M, [\![M]\!]; M, r_M)$ with the audit computers acting as the verifiers to proof that its model input is consistent with c_M from $\texttt{C}_j$. Each $\texttt{AC}_i$ aborts marking $\texttt{M}_k$ as malicious if verification fails.
- Each audit computer computes $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{IC}}, c \| c_x \| c_y, \sigma_{\texttt{IC}})$ or $\mathsf{POI.Verify}(\mathsf{pp}_{\mathsf{poi}}, c_M, c_x, c_y, \pi_\text{I})$.

A.5 **Verify Training:** Each data owner $\texttt{DH}_i$ inputs their dataset D_i to $\mathcal{F}_{\text{ABB [ID]}}$.

- Each data owner $\texttt{DH}_i$ performs $\mathsf{PoC.Check}_{[\text{ID}]}(\mathsf{pp}_{\mathsf{poc}}, c_{D_i}, [\![D_i]\!]; D_i, r_{D_i})$ with the audit computers acting as verifiers to proof that its input $[\![D_i]\!]$ is consistent with c_{D_i}. $\texttt{AC}_j$ also checks $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{DH}_i}, c, \sigma_\text{T})$ for each $\texttt{DH}_i$. If verification fails, $\texttt{AC}_j$ aborts marking $\texttt{M}_k$ as malicious.
- Each audit computer computes $\mathsf{SIG.Verify}(\mathsf{pk}_{\texttt{TC}}, c, \sigma_{\texttt{TC}})$ or $\mathsf{POT.Verify}(\mathsf{pp}_{\mathsf{pot}}, c, \pi_\text{T})$.

A.6 The audit computers compute $[\![o]\!] \leftarrow \mathcal{F}_{\text{ABB [ID]}}.\mathsf{Audit}(f_{\texttt{audit}}, [\![D_1]\!], \ldots, [\![D_{N_{\texttt{DH}}}]\!], [\![M]\!], [\![x]\!], [\![y]\!], \mathsf{aux})$ and use $\mathcal{F}_{\text{ABB [ID]}}$ to open o at $\texttt{C}_j$.

maintenance and re-sharing to new sets of entities [41, 65, 77], a complexity which we aim to avoid in our design.

The receipt received by the model owners after training comprises commitments to the training data, the training randomness and the resulting model; the data owners's signatures; and either the signature from the computing parties or a proof of training. During inference, the model owners provide this training receipt instead of merely the model commitment. As a result, the inference receipt is essentially an extension of the training receipt and includes the equivalent commitments and signatures (or proofs, where applicable) for both training and inference. Therefore, the conciseness of the underlying commitments (i.e., PoC) is crucial to ensuring that the overhead imposed upon the client due to the need to store this receipt is minimized.

Auditing During auditing, the client provides the receipt and inputs the prediction sample and result into the MPC computation. Meanwhile, the model owners and data owners need to provide their respective inputs. The protocol first confirms that the signatures (or proofs, where applicable) in the receipt are valid, in reverse order, i.e., beginning with the last signature generated at the end of inference. Then, it uses $\mathsf{PoC.Check}$ to verify the consistency of the provided inputs with the commitments in the receipt. Finally, after all checks have passed, the protocol computes the audit function (i.e., $\mathcal{F}_{\text{ABB}}.\mathsf{Audit}$). Should any of the checks fail, the protocol aborts and identifies the party at fault. During the auditing phase, we can rely on the fact that an honest client will only ever accept valid receipts, as the proof of computational integrity (or, in the secure computation setting, signatures) allow them to verify that the receipt was generated correctly. At the same time, these checks prevent the adversary from constructing a valid malicious receipt that would incriminate an honest party. As a result, any inconsistency between the receipt and the input data provided to the audit phase can be uniquely attributed to the party providing the corresponding auditing input. For more details, we refer to the proof in Appendix B [1].

4 Proof of Consistency

In our auditing protocol Π_{Arc}, we assume access to a Proof-of-Consistency protocol PoC that allows a party to commit to their (secret) inputs and later allows the parties to collaboratively check that a given (set of) secret shared[2] values is consistent with the provided commitment. Note that PoC might seem related to Verifiable Secret Sharing (VSS), which guarantees that parties receive a valid sharing of a given value. However, this is orthogonal to our requirement of ensuring consistency of *inputs* across different phases and, therefore, different sharings of that value. In theory, the consistency guarantees required for Arc could also be achieved straightforwardly by using standard commitments. However, as we show in our evaluation, such approaches incur significant performance overheads, especially during verification, making practical deployment infeasible.

4.1 Defining Proof-of-Consistency

In the following, we provide the formal definition of PoC and the properties it needs to achieve before discussing several approaches based on existing literature and highlighting their inherent limitations with regard to efficiency and succinctness.

Definition 4.1 (Proof-of-Consistency Protocol)**.** A valid Proof-of-Consistency is an interaction between a Prover $\mathbb{P}$ and a set of $N-1$ Verifiers $\mathbb{V}$. This protocol allows the verifiers to check that a vector $[\![\mathbf{x}]\!] = ([\![\mathbf{x}_1]\!], \ldots, [\![\mathbf{x}_d]\!])$ stored in an ideal functionality $\mathcal{F}_{\text{ABB}}$ is consistent with a commitment c to $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_d) \in \mathbb{F}_p^d$. A Proof-of-Consistency is defined

[2] More precisely, in the representation used by $\mathcal{F}_{\text{ABB}}$.

as a set of protocols (PoC.Setup, PoC.Commit, PoC.Check) where:

- PoC.Setup$(1^\lambda, d) \rightarrow$ pp$_{\text{poc}}$: prepares public parameters pp supporting inputs of size d.
- PoC.Commit(pp$_{\text{poc}}, \mathbf{x}, r) \rightarrow c$: An algorithm in which the prover generates a commitment to (a vector of) inputs $\mathbf{x}$ with randomness r.
- PoC.Check(pp$_{\text{poc}}, c, [\![\mathbf{x}]\!]; \mathbf{x}, r) \rightarrow \{0,1\}$: A protocol where the prover convinces the verifiers that the commitment c is consistent with $[\![\mathbf{x}]\!]$. Only the prover knows $\mathbf{x}$ and r.

A valid Proof-of-Consistency should satisfy correctness, soundness and zero-knowledge, which we formally define in Appendix C. In addition to the formal requirements, a PoC instantiation should yield succinct commitments and its protocols should be efficiently computable (specifically, require minimal MPC operations in PoC.Check) in order to be practical for ML. Succinctness is crucial for ensuring efficient communication and storage, especially when dealing with large input sizes and resource-constrained clients. In the following, we discuss existing approaches and how they fall short in our setting. We refer to Appendix D in the extended version of the paper [1] for formal definitions of the corresponding protocols.

Direct Commitments [2, 53, 78]. A straightforward approach to PoC is to use a cryptographic commitment scheme to instantiate PoC.Setup and PoC.Commit with COM.Setup and COM.Commit, respectively. In PoC.Check, the commitment is verified with respect to the secret shared inputs $[\![\mathbf{x}]\!]$ and decommitment $[\![r]\!]$ by computing COM.Verify using $\mathcal{F}_{\text{ABB}}$. This typically requires recomputing the commitment under MPC, because the usual implementation of COM.Verify is to re-compute the commitment $c' \leftarrow$ COM.Commit$([\![\mathbf{x}]\!], [\![r]\!])$ and checking that $c' = c$. Related work has suggested to use this protocol with commitments based on a collision-resistant hash function, such as SHA-2 [53], SHA-3 [78] and MPC-friendly constructions such as LowMCHash-256 [2]. The advantage of this approach lies in its succinct commitment size which is typically constant. However, despite its efficient storage needs, the hash-based approach incurs significant computational costs during verification. This is primarily due to hashes relying on non-linear operations, which are expensive to compute in MPC.

Homomorphic Commitments [93]. To mitigate the MPC cost of PoC.Check, one can rely on homomorphic commitments such as Pedersen commitments instantiated using an elliptic curve group [72, 93]. Instead of calling COM.Verify for the full input vector $[\![\mathbf{x}]\!]$, parties use the homomorphism to compute a linear combination of commitments to individual elements x_i, trading off MPC overhead with local computation. As a result, parties only compute a single commitment $\tilde{c}' =$ COM.Commit$(\sum_i \gamma^i \cdot [\![x_i]\!])$ with $\mathcal{F}_{\text{ABB}}$ in PoC.Check and compare the result with $\tilde{c}' = \sum_i \gamma^i \cdot c_i$. Unfortunately, a downside of this approach is that commitments to individual elements must be stored, resulting in a size that is linear in $|\mathbf{x}|$. This approach results in a PoC.Check that is asymptotically more efficient than the hash-based approach. In practice, hash-based approaches remain more concretely efficient for very small inputs. However, the Pedersen commitment approach becomes more efficient already for moderate input sizes.

4.2 Arc PoC Protocol

The key insight of our efficient PoC protocol is that we do not actually need to compute COM.Commit in order to verify that the $[\![\mathbf{x}]\!]$ in $\mathcal{F}_{\text{ABB}}$ matches the input $\mathbf{x}$ of COM.Commit. Instead, we propose a protocol that allows the prover $\mathbb{P}$ to convince the verifiers of this fact with a polynomial identity test. Towards this, we first define a polynomial $g(B) := \sum_{i=1}^{d} x_i \cdot B^i$, i.e., interpreting the elements of $\mathbf{x} \in \mathbb{F}_p^d$ as the coefficients of the polynomial. The prover commits to g using a (homomorphic) polynomial commitment scheme [49] to obtain a constant-size commitment c. In PoC.Check, the parties then first collaboratively sample a point $\beta \xleftarrow{\$} \mathbb{F}_p$ and then evaluate the polynomial at β (using $\mathcal{F}_{\text{ABB}}$) by computing $\rho := \sum_{i=1}^{d} [\![x_i]\!] \cdot \beta^i$ and opening ρ. This is cheap in MPC because it only involves addition and scaling operations on the secret shares $[\![\mathbf{x}]\!]$ which can be executed locally. The prover, who originally committed to $\mathbf{x}$ with c, can now do a polynomial commitment opening proof to show that $g(\beta)$ equals ρ. The other parties verify this evaluation proof, which, if valid, implies that the polynomial in $\mathcal{F}_{\text{ABB}}$ is (with high probability) equal to the one committed to with c. One caveat with this approach is that ρ reveals information about $\mathbf{x}$. We can overcome this by generating and committing to a random polynomial g_ω at the beginning of PoC.Check and use it to (additively) mask g. We can then evaluate and open $g(\beta) + g_\omega(\beta)$ which is now indistinguishable from random. The prover and verifiers proceed with the evaluation proof as before, but on the combined commitment of g and g_ω, using the homomorphic property of the polynomial commitment scheme. In the following, we provide a formal definition:

Protocol 4.2 (Consistency Check)**.** Let $\mathcal{F}_{\text{ABB}}$ be an instance of an ideal MPC functionality over a field $\mathbb{F}_p$, let $\mathcal{F}_{\text{RAND}}$ be an ideal functionality that returns a random element from $\mathbb{F}_p$ and let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{F}_p^d$ by the input of prover $\mathbb{P}$. Let $[\![\mathbf{x}]\!] = ([\![x_1]\!], \ldots, [\![x_d]\!])$ be the input of the prover $\mathbb{P}$ to $\mathcal{F}_{\text{ABB}}$. Let PC be a polynomial commitment scheme as in Definition A.2 [1] that is also homomorphic as in Definition A.4 [1]. The protocol Π_{cc} works as follows:

- CC.Setup$(1^\lambda, d) \rightarrow$ pp: Run pp $\leftarrow$ PC.Setup(d), where d is the number of elements in the input.
- CC.Commit(pp$, \mathbf{x}, r) \rightarrow c$: The prover computes a polynomial commitment $c \leftarrow$ PC.Commit$(\mathtt{pp}, g, r)$ where g is defined as $g(z) = \sum_{i=1}^{d} x_i \cdot z^i$. The prover outputs c.

- $\mathsf{CC.Check}(\mathsf{pp}, c, [\![\mathbf{x}]\!]; \mathbf{x}, r) \to \{0,1\}$: The protocol proceeds as follows:
  1. The prover samples a masking value $\omega \xleftarrow{\$} \mathbb{F}_p$ and commitment randomness $r_\omega \xleftarrow{\$} \mathbb{F}_p$ and computes a polynomial commitment $c_\omega \leftarrow \mathsf{PC.Commit}(\mathrm{pp}, g_\omega, r_\omega)$ to a degree-0 polynomial $g_\omega(z) = \omega$. The prover sends c_ω to all parties and inputs ω to $\mathcal{F}_{\mathrm{ABB}}$.
  2. The parties invoke $\mathcal{F}_{\mathrm{RAND}}$ to obtain a random challenge $\beta \xleftarrow{\$} \mathbb{F}_p$.
  3. The parties invoke $\mathcal{F}_{\mathrm{ABB}}$ to compute $[\![\rho]\!] := [\![\omega]\!] + \sum_{i=1}^{d} [\![x_i]\!] \cdot \beta^i$ and subsequently open ρ.
  4. The prover $\mathbb{P}$ generates a proof $\pi \leftarrow \mathsf{PC.Prove}(\mathsf{pp}, c + c_\omega, g + g_\omega, r + r_\omega, \beta, \rho)$ and sends π to each verifier $\mathbb{V}$.
  5. Each verifier runs $\mathsf{PC.Check}(\mathsf{pp}, c \cdot c_\omega, \beta, \rho, \pi)$. If verification passes, they output 1, otherwise 0.

Intuitively, security follows from the fact that if the committed polynomial is not equal to the polynomial evaluated on the secret shares, then the prover can only open the commitment to ρ with negligible probability. We provide a formal security proof in Appendix C (Lemma C.1). The protocol Π_{cc} can be extended to provide ID-Abort, denoted as $\Pi_{\mathrm{cc\ [ID]}}$, by using a broadcast channel for the prover in Step 4 and using an MPC protocol that provides identifiable abort ($\mathcal{F}_{\mathrm{ABB\ [ID]}}$).

Cost Analysis. Many polynomial commitment schemes have a constant storage overhead independent in the input size, resulting in each party having to store only a single, constant-sized commitment for each input vector. Our protocol can be instantiated with any homomorphic polynomial commitment scheme and inherits the efficiency profile of the underlying scheme. If instantiated with KZG polynomial commitments [49], we achieve a constant storage overhead independent of the input size and a constant verification time. Although the public setup parameters of KZG are of size $O(d)$, we can consider them as system parameters and reuse them for the input of each party [49]. Hence, our protocol only requires a storage overhead linear in the size of the input and the number of parties, i.e., $O(N+d)$. If used with an inner-product argument-based polynomial commitment, the commitment size could also be made constant (e.g., a single Pedersen Vector Commitment (PVC)). However, the verification time would be linear in the input size [11].

EC-MPC. Prior work has observed that most secret-sharing-based MPC protocols for finite field arithmetic generalize to arithmetic circuits involving elliptic curve points [70, 84]. Using such protocols with additional support for computations over an elliptic curve group $\mathbb{G}$ of order p (which we denote as $\mathcal{F}_{\mathrm{ABB}}^{[\mathrm{EC}]}$) when instantiating PoC offers a significant improvement to performance. Specifically, we can accelerate the execution of our PoC.Commit algorithm under secure computation (i.e., the creation of output commitments).

Batch verification Our protocol Π_{cc} allows the verifier to check the integrity of the prover's input by verifying one pairing equation. However, the verifier still needs to perform this check for each input party. We can optimize this further for KZG commitments by leveraging their homomorphic property [35, 49] (Definition A.4 [1]). Let $c_1, \ldots, c_N$ be the set of commitments and $\rho_1, \ldots, \rho_N$ the set of target evaluations for each prover $\mathbb{P}_1, \ldots, \mathbb{P}_N$ at a common random point $\beta \in \mathbb{F}_p$ from Step 3 of the consistency check. The verifier first computes a random linear combination of the commitments as $\tilde{c} := \sum_i^N \gamma^i c_i$ for a randomly sampled $\gamma \in \mathbb{F}_p$, as well as the corresponding evaluation $\tilde{\rho} =: \sum_i^N \gamma^i \rho_i$ and aggregate proof $\tilde{\pi} =: \sum_i^N \gamma^i \pi_i$. The verifier can then check this aggregated commitment using $\mathsf{PC.Check}(\mathsf{pp}, \tilde{c}, \beta, \tilde{\rho}, \tilde{\pi})$. This allows the verifier to check, in the optimistic case, only one pairing equation instead of N at the cost of a negligible statistical error. Security follows from the fact that the aggregated polynomial commitment $\tilde{c}$ will only agree with the aggregated evaluation point $\tilde{\rho}$ at a random point β with negligible probability due to the Demillo-Lipton-Schwartz-Zippel Lemma ([29]). If verification passes, this implies that all commitments open to the correct evaluation point with overwhelming probability. If verification fails, this must mean that at least one of the commitments is inconsistent with high probability. In this case, the verifier can proceed to check the commitments and proofs individually.

	T	M	I
Data Validation			
Input Checks [19, 63]	●	○	○
Sample Attribution [39, 45, 52, 55, 81]	●	●	●
Party attribution [64]	●	●	●
Model Validation			
Validation Sets [22]	○	●	○
Feature Attribution [43, 62, 74]	◐	●	●
Certification [46, 53, 78]	○	●	●
Process Validation			
Algorithm Verific. [36, 44, 48, 85]	●	●	○
Constraint Verific. [80]	●	●	○

Table 2: A priori and post hoc algorithms from the ML interpretability and safety literature along with whether they require the training data (T), the model (M) and the inference (I) as input.

5 Auditing Functions

Until now, our discussion has centered around the cryptographic protocol that enables robust and secure audits of private models. We now focus on how we realize the audit functionality enabled by our framework. The algorithmic side of auditing for ML is an active area, and alternative instantiations that enable different properties exist or are actively being developed (see Table 2 for an overview). In our work, we focus on post-hoc audits, specifically accountability (sample/party attribution), explainability (feature attribution), and robustness & fairness (certification). While Arc's design is highly extensible with additional auditing functionalities (similar to how we support a wide range of PPML systems), here we focus our discussion on auditing functions currently implemented in our framework. In Appendix G in the extended version of the paper [1], we provide a more detailed description of each function.

5.1 Robustness & Fairness

The community has devised a range of techniques to show that a model is robust against adversarial examples [37, 91], i.e., that the model is stable to small variations in the input. While *global* robustness guarantees are more naturally realized as a-priori checks, *local robustness* [24], which certifies that a model consistently produces the same prediction y for all inputs within a radius R around the original input x, naturally suits the post-hoc auditing setting we consider. In Arc, we adapt the algorithm proposed by Jovanovic et al. [46] for FHE to the MPC setting. The algorithm samples n perturbed inputs around the input x by adding Gaussian noise and obtaining predictions for these samples. Finally, a statistical check is conducted to assess whether the obtained prediction y remains invariant to these perturbations with high probability. The output of the auditing function is a boolean indicating whether the model is locally robust with confidence $1-\alpha$. We can extend the same technique to achieve fairness guarantees, as there is a well-established connection between robustness and individual fairness [32, 46, 76, 92]. It is sufficient to change the closeness metric of the sampling procedure to generate perturbed inputs that are close to x in terms of fairness (i.e., inputs that should be treated similarly have a small distance).

5.2 Accountability

We consider two flavors of accountability mechanisms that attribute responsibility for decision made by a model: *sample* attribution identifies the influence of individual data samples on a prediction, while *party* attribution merely provides the relative influence of each data owner's dataset, providing auditability while revealing less information. A variety of methods to identify the impact of individual data samples exist [39, 45, 52, 55, 81], however, some (e.g., influence functions [55]) require substantial computational resources (e.g., inverting the Hessian matrix of the loss function) which makes them prohibitively expensive under secure computation. In Arc, we leverage an approach using KNN-Shapley values [45], i.e., Shapely values of a KNN classifier on the training data's latent space representation. As there exists a closed-form formulation of the Shapley values for KNN [45], this allows an efficient MPC realization. For party attribution, Arc uses an efficient unlearning approach [64]. The key idea here is that if a suspicious prediction (x, y) was (at least partially) the result of data provided by a data owner, then excluding that party's data will lead to the absence (or weakening) of the suspicious prediction. Rather than recomputing leave-out models from scratch, we use an algorithm [64] that uses unlearning of the data of a party from the original model [81], which requires only a small number of training epochs until sufficient differences are detectable.

5.3 Explainability

A wide range of methods has been proposed to explain the predictions of complex models [62, 74, 82]. Of these, we consider additive feature attribution methods [62] as particularly suitable for privacy-preserving auditing as they provide an attractive trade-off between leakage and utility. These methods highlight which features of a prediction sample are most influential for the prediction, even for complex ML models. They achieve this by approximating the target model's behavior locally (around the given prediction) with a simple and explainable (e.g., linear) model. In Arc, we leverage KernelSHAP [62] to approximate the local decision boundary of the classifier, through a linear regression on the features. We sample points around the prediction sample and weigh them based on their distance to the sample as measured by the Shapely kernel. As a result, the regression coefficients directly correspond to the Shapley values of the features.

6 Evaluation

In this section, we evaluate the performance of Arc in the training, inference and auditing phases for different workloads and auditing functions. We evaluate the overhead of our protocol when instantiated with different approaches to the consistency layer PoC. For training and auditing, we focus on the MPC versions of our protocol, as these are the most established forms of verifiable ML computation.

Implementation. Our implementation is based on MP-SPDZ [50], a popular framework for MPC computation that supports a variety of protocols. We extend MP-SPDZ with protocols for share conversion and elliptic curve operations on the pairing-friendly BLS12-377 curve [12] provided by the `libff` library [23]. We use ECDSA signatures on the secp256k1 curve [17] for which a distributed signing protocol was previously implemented in MP-SPDZ [28]. For the evaluation proofs of the polynomial commitments, we use the implementation of the KZG polynomial commitment scheme [49] provided by Arkworks' `poly-commit` library [4]. We perform share conversion (cf. Appendix E [1]) to convert between $\mathbb{Z}_{2^{64}}$ and the scalar field $\mathbb{F}_{\text{BLS12-377}}$. The MPC computations for ML training, inference, and auditing functions are expressed in MP-SPDZ's domain-specific language. We rely on the higher-level ML primitives that MP-SPDZ provides that use mixed-circuit computation. Note that we perform exact truncation instead of probabilistic truncation for fixed-point multiplication because the latter has recently been shown to be insecure [61].

To compare the performance of our consistency layer to other approaches (cf. 4.1), we additionally implement a version of PoC based on the SHA3-256 cryptographic hash function denoted by PoC_{SHA3} that internally uses the Bristol-fashion circuit implementation of the Keccak-f sponge function [69]. We also implement a version PoC_{PED} based on Cerebro [93] that uses Pedersen commitments by adapting the open-source implementation provided by the authors. We make the implementation of our protocol and the various consistency layers that we evaluate available as open-source[3].

Experimental Setup. We run Arc on a set of AWS `c5.9xlarge` machines running Ubuntu 20.04, each equipped with 36 vCPUs of an Intel Xeon 3.6 Ghz processor and 72 GB of RAM. For the BERT transformer model, we use `c5.24xlarge`, featuring 96 vCPUs of an Intel Xeon 3.6 Ghz processor and 192 GB of RAM, as it provides sufficient memory to fit the significantly larger model. The machines are connected over a local area network (LAN) through a 12 Gbps network interface with an average round-trip time (RTT) of 0.5 ms. We additionally perform our experiments in a simulated wide area network (WAN) setting using the `tc` utility to introduce an RTT of 80 ms and limit the bandwidth the 2 Gbps. We report the total wall clock time and the total communication cost in terms of the data sent by each party. This includes the time and bandwidth required for the online phase and the preprocessing phase that sets up the correlated randomness necessary for the MPC protocol. We also report the storage overhead for which we apply log scaling as the overhead varies significantly between different PoC approaches and settings. In experiments in the WAN setting and those involving maliciously secure protocols, we estimate the ML training operations based on 5 and 50 batches of gradient descent, respectively. For the BERT transformer model, we extrapolate further due to the significant size of the model and training data. For the related work, which has overhead linear in the size of the input, we also extrapolate results for some of the larger instances.

We evaluate the computational phases in the 3-party computation (3PC) setting with a maliciously secure-with-abort protocol that combines SPDZ-wise redundancy with replicated secret sharing over a 64-bit ring [27]. We also evaluate the performance of a semi-honest protocol based on replicated secret sharing. These protocols are representative of the most efficient MPC protocols in the malicious and semi-honest settings for ML workloads. We apply an optimization for the auditing phase that uses the fact that all inputs in this phase are authenticated using commitments. This allows us to optimistically use a security-with-abort protocol and, only, if the protocol aborts, restart the computation with a less efficient identifiable-abort protocol with the guarantee that this

[3] github.com/pps-lab/arc

	Storage	Prover Comp.	Verifier Comp.
PoC_{SHA3}	N	-	$d \cdot N$ (mpc)
PoC_{PED}	$d \cdot N$	-	$d \cdot N$ (local) $+ N$ (mpc)
Ours	N	d (local)	N (mpc)

Table 3: Asymptotic computational complexity (in big-O notation) and storage requirements of the consistency approaches we evaluate for a computation with N input parties each with d input elements. We differentiate between local and MPC computation. For PoC_{SHA3}, we consider Keccak-f operations. For PoC_{PED} and ours, we consider group operations.

execution uses the same inputs. We choose the 3PC setting because it allows for the most efficient MPC protocols, favoring PoC_{PED} and PoC_{SHA3} whose Check relies more heavily on MPC computation. Other settings such as 2-party computation (2PC) or non-optimistically executing the auditing would require more expensive MPC protocols, resulting in a higher overhead for the computation. This would increase the relative overhead of the related approaches compared to ours.

Scenarios. We evaluate four auditing functions from the previous section, (i) Robustness, (ii) Fairness (cf. §5.1), (iii) KNN-Shapley (cf. §5.2) and (iv) Kernel-SHAP (cf. §5.3), on the following models and datasets.

(W1:Adult): A logistic regression model with 3k parameters trained on the Adult [8] binary classification task for 10 epochs to predict whether a person's income exceeds $50k per year.

(W2:MNIST): A LeNet model consisting of 431K parameters, referred to as 'model C' in prior work [51, 87] trained on the MNIST image classification task [60] for 20 epochs.

(W3:CIFAR-10): A variant of AlexNet [59] as used in Falcon [88], comprising 3.9 million parameters trained on the CIFAR-10 image classification task [58] for 40 epochs.

(W4:QNLI): A BERT transformer model [30] with sequence length 128, comprising 85 million parameters, finetuned for one epoch on 2500 samples of the Stanford Question Answering reading comprehension dataset [73, 89].

6.1 Evaluation Results

We evaluate the overhead that Arc imposes on the training, inference and auditing phases. The main overhead of the consistency layer in training and inference consists of two parts: Verifying the inputs of the computation using PoC.Check and, afterwards, computing the output commitments using PoC.Commit. Other components, such as those related to the signatures are negligible in comparison: distributed signing takes at most 300ms for the WAN and verifying a signature is a local operation taking 1ms. Clients only have to store a single ECDSA signature of 64 bytes for the model owner and each data owner, and a joint signature of 64 bytes for the

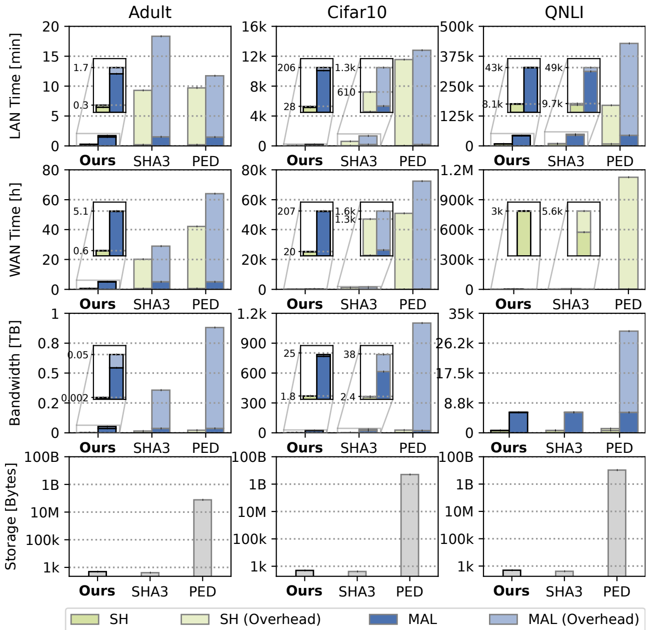

Figure 4: Evaluation of Arc comparing the approaches relative to a single epoch of PPML training.

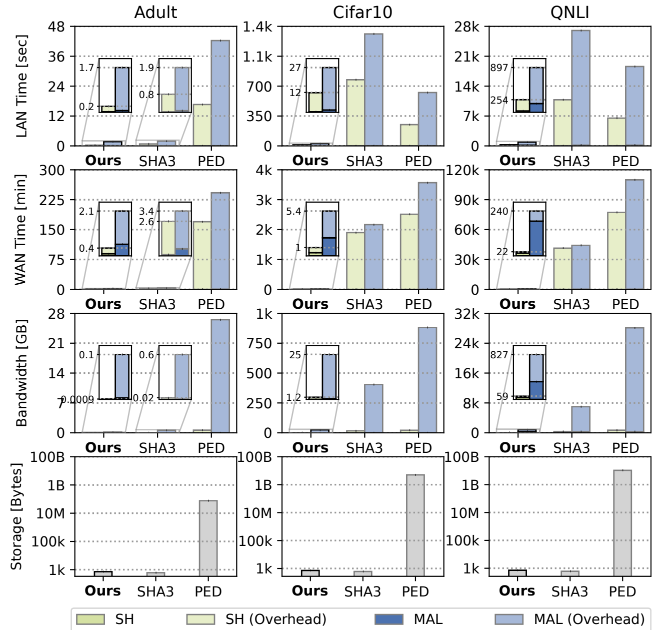

Figure 5: The overhead of our system's consistency protocol relative to a single PPML inference for our three scenarios.

inference computers and the training computers. We provide an analysis of the asymptotic complexity of the different approaches in Table 3 and focus on the concrete performance numbers for the rest of this section.

Training. We show the wall-clock time and bandwidth of a single training epoch in Fig. 4, differentiating between the overhead induced by our framework and the cost of the underlying PPML training. In total, PPML training (W1:Adult) takes 119 seconds, with consistency adding 5 seconds for ours, 545 seconds for PoC_{SHA3}, and 570 seconds for PoC_{PED}. For (W2:MNIST), this is 123 minutes (+ 1.3 minutes, 2.9 hours, or 22 hours), for (W3:CIFAR-10), 15 hours (+ 4.1 minutes, 9.8 hours, or 8 days), and for (W4:QNLI), this is 5.4 days (+ 11 minutes, 1.1 days, or 16 weeks), in the semi-honest LAN setting. We refer to Appendix F [1] for a full report of end-to-end training results and further results for (W2:MNIST) which have been omitted here due to space constraints. Note that we do not evaluate (W4:QNLI) training in the malicious WAN setting, as this is beyond the current state of the art for PPML. Since the bandwidth overhead is not significantly affected by network delays, we only present the bandwidth results once. As storage is, additionally, also independent of the chosen MPC protocol, we do not differentiate between semi-honest and malicious settings for storage.

We observe that the overhead of the baseline approaches varies significantly for training. The timing overhead induced by PoC_{PED} is 66-500x compared to training but only 0.1-26x with PoC_{SHA3} in the LAN setting. In the WAN setting the relative overhead further increases to at most three orders of magnitude because of the large number of MPC round-trips required to compute the operations related to hash functions and elliptic curve operations. In comparison, our consistency check protocol, which outperforms the related approaches across all configurations, introduces only 0.001-1.35x overhead in the LAN and less than 1.02x in the WAN setting. This is because our protocol features more local computation than MPC computation and is therefore less impacted by the slowdown induced by network delays. Overall, we conclude that the overhead of our approach, and also of PoC_{SHA3}, is effectively negligible in the context of PPML training. In contrast, we observe that the PoC_{PED} approach is infeasible for all but the simplest models. The primary cost of PoC_{PED} is the time required to compute the individual Pedersen commitments to the model parameters; the overhead during verification is much smaller as this only involves computing a commitment for each of the three input parties. Finally, the bandwidth and storage required for PoC_{PED} are significant as it needs to commit to each input element individually. Meanwhile, the storage overhead of our approach is independent of the dataset and model sizes and similar to that of the hash-based approach with 496 bytes compared to 416 bytes for PoC_{SHA3}.

Inference. Model inference is a significantly smaller operation than training, resulting in a larger relative consistency overhead As shown in Fig. 5, the consistency operations are at least an order of magnitude slower than the inference itself, even for our approach. However, in absolute numbers, the overhead introduced by our approach is very small across all configurations, ranging from a few seconds to a few minutes for the large transformer model. In contrast, the overhead for PoC_{SHA3} quickly becomes prohibitive for all but the smallest

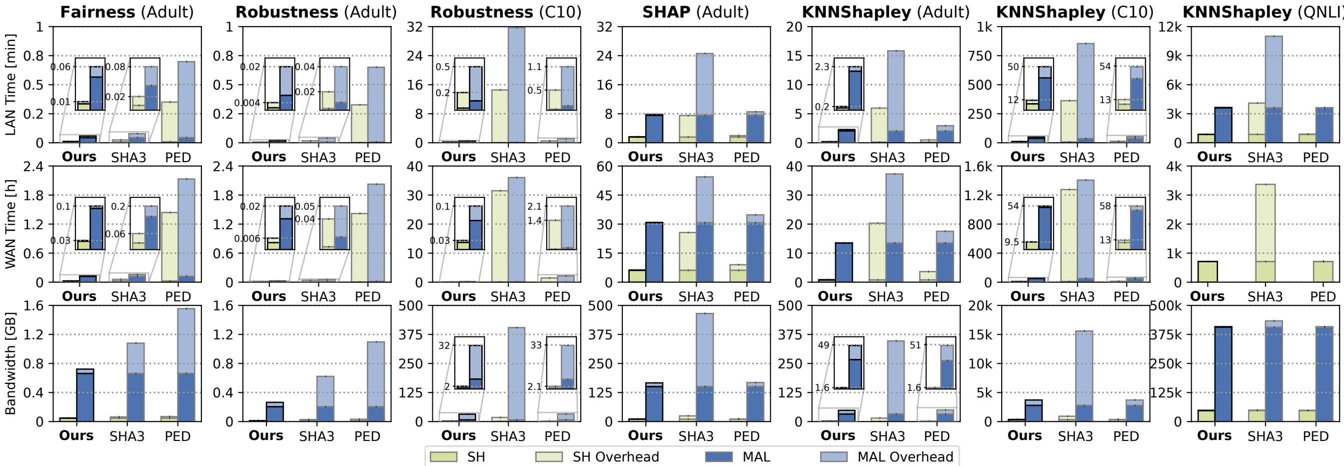

Figure 6: The overhead of Arc's consistency layer relative to the cost of the auditing function computation in MPC for four different auditing functions across our three scenarios.

models, already requiring over thirty hours in the WAN setting for a single (W3:CIFAR-10) inference. $\mathsf{PoC}_{\mathsf{PED}}$ outperforms $\mathsf{PoC}_{\mathsf{SHA3}}$ in some cases, it induces similarly prohibitive overheads across all configurations.

A significant fraction, 35-66%, of the overhead in our approach is the result of the share conversion from the PPML protocol's computation domain $\mathbb{Z}_{2^{64}}$ to the scalar field domain $\mathbb{F}_{\mathrm{BLS12\text{-}377}}$. The conversion requires a bit decomposition and re-composition for each input parameter which is expensive and scales linearly in the input size. Although this overhead is significant, it is small concretely, with 25 seconds in the active security setting for (W3:CIFAR-10). In the case that lower latency is required, the $\mathbb{Z}_{2^{64}}$ secret shares of the model can be cached on the inference servers after a single conversion to $\mathbb{F}_{\mathrm{BLS12\text{-}377}}$ and verification with PoC.Check. When scaling to larger models, $\mathsf{PoC}_{\mathsf{PED}}$ and $\mathsf{PoC}_{\mathsf{SHA3}}$ become prohibitively expensive concretely with a 250-6000x slowdown compared to a single inference. As in training, we observe that the storage required for the client receipts is comparable for our approach and $\mathsf{PoC}_{\mathsf{SHA3}}$, resulting in only 720 bytes and 608 bytes, respectively, per prediction. Most importantly, these are independent of the model and training data size. Meanwhile, $\mathsf{PoC}_{\mathsf{PED}}$ requires storage that is linear in the number of input elements, e.g., requiring 5GB for (W3:CIFAR-10) and 11GB for (W4:QNLI). This results in $\mathsf{PoC}_{\mathsf{PED}}$ inducing prohibitive storage requirements for all but the simplest models.

Auditing. We present the wall-clock time and bandwidth overhead for different auditing functions in Fig. 6. Note that we do not evaluate auditing on (W4:QNLI) for the malicious WAN setting, as the complexity of evaluating auditing functions on such a complex model in this setting is pushing the boundaries of what is possible with current PPML techniques. Across all settings, Arc significantly outperforms related approaches in terms of runtime, with a storage overhead comparable to the hash-based approach. As we move to larger input sizes, for instance in the case of KNN-Shapley that considers the full training dataset, the main cost of our approach after share conversion is the multi-scalar multiplication (MSM) required to compute the opening proof of the polynomial commitment. Each prover party must compute an MSM that is linear in the size of its input. Due to the properties of KZG, the other parties only have to check one pairing equation per prover, which we can further reduce to a single pairing equation due to the batch verification (c.f. §4.2). We also observe that, for larger models, $\mathsf{PoC}_{\mathsf{PED}}$ outperforms $\mathsf{PoC}_{\mathsf{SHA3}}$ and approaches the performance of our approach. This is primarily because there is no need for computing and storing commitments to the output during the auditing phase, sidestepping the major weaknesses of the $\mathsf{PoC}_{\mathsf{PED}}$ approach. In fact, as model sizes increase, the high constant overhead of the Pedersen commitments becomes less noticeable. However, our approach continues to outperform $\mathsf{PoC}_{\mathsf{PED}}$ both asymptotically (cf. Table 3) and also concretely across all configurations.

In conclusion, we observe that our approach outperforms the related work both asymptotically and concretely across all configurations. While $\mathsf{PoC}_{\mathsf{PED}}$ and $\mathsf{PoC}_{\mathsf{SHA3}}$ approach the performance of our solution in some phases (for certain configurations), they remain prohibitively expensive from an end-to-end perspective. Meanwhile, we demonstrate that Arc instantiated with our efficient PoC protocol is highly practical across a wide range of settings, including scaling up to large and complex models that push the boundary of the current state of the art in privacy preserving machine learning.

Acknowledgements

We thank the anonymous reviewers for their insightful input and feedback. We would also like to acknowledge our sponsors for their generous support, including Meta, Google, and SNSF through an Ambizione Grant No. PZ00P2_186050.

References

[1] Hidde Lycklama, Alexander Viand, Nicolas Küchler, Christian Knabenhans, and Anwar Hithnawi. Holding Secrets Accountable: Auditing Privacy-Preserving Machine Learning, 2024. Extended Version, arXiv:2402.15780.

[2] Nitin Agrawal, James Bell, Adrià Gascón, and Matt J Kusner. MPC-Friendly commitments for publicly verifiable covert security. September 2021.

[3] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. QUOTIENT: Two-Party secure neural network training and prediction. CCS '19, New York, NY, USA, November 2019.

[4] arkworks contributors. `arkworks`, 2022.

[5] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On Pixel-Wise explanations for Non-Linear classifier decisions by Layer-Wise relevance propagation. PLoS One, July 2015.

[6] Assi Barak, Daniel E Escudero, Anders Dalskov, and Marcel Keller. Secure evaluation of quantized neural networks. Proceedings on Privacy Enhancing Technologies, 2020:355–375, 2020.

[7] Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. Probabilistic verification of fairness properties via concentration. December 2018.

[8] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996.

[9] Abeba Birhane, Ryan Steed, Victor Ojewale, Briana Vecchione, and Inioluwa Deborah Raji. AI auditing: The broken bus on the road to AI accountability. January 2024.

[10] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. ACM SIGACT News, 15(1):23–27, January 1983.

[11] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient Zero-Knowledge arguments for arithmetic circuits in the discrete log setting. In EUROCRYPT, 2016.

[12] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. IACR Cryptology ePrint Archive, 2018:962, 2018.

[13] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In 2018 IEEE Symposium on Security and Privacy (SP), May 2018.

[14] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. FLASH: Fast and robust framework for privacy-preserving machine learning. Proc. Priv. Enhancing Technol., 2020(2):459–480, April 2020.

[15] Ran Canetti. Security and composition of multiparty cryptographic protocols. J. Cryptology, 13(1):143–202, January 2000.

[16] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Security and Cryptography for Networks, 2010.

[17] Certicom. Standards for efficient cryptography 2 (SEC 2). Technical report, 2010.

[18] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. EzPC: Programmable, efficient, and scalable secure two-party computation for machine learning. In IEEE European Symposium on Security and Privacy, February 2019.

[19] Ian Chang, Katerina Sotiraki, Weikeng Chen, Murat Kantarcioglu, and Raluca Ada Popa. HOLMES: Efficient distribution testing for secure collaborative learning. 2023.

[20] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. ZKML: An optimizing system for ML inference in Zero-Knowledge proofs. 2024.

[21] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In CRYPTO, 2018.

[22] Dami Choi, Yonadav Shavit, and David Duvenaud. Tools for verifying neural models' training data. July 2023.

[23] SCIPR Lab & Clearmatics. Libff: C++ library for Finite Fields and Elliptic Curves. Online: `https://github.com/clearmatics/libff`, January 2024.

[24] Jeremy M Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. February 2019.

[25] European Commission. Artificial Intelligence Act, 2021. COM/2021/206 final.

[26] Robert Cunningham, Benjamin Fuller, and Sophia Yakoubov. Catching MPC cheaters: Identification and openability. In Information Theoretic Security, pages 110–134. Springer International Publishing, 2017.

[27] Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-Majority Four-Party secure computation with malicious security. In USENIX Security, 2021.

[28] Anders Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In ESORICS, 2020.

[29] Richard A Demillo and Richard J Lipton. A probabilistic remark on algebraic program testing. Inf. Process. Lett., 7(4):193–195, June 1978.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. North Am Chapter Assoc Comput Linguistics, 2019.

[31] Samuel Drews, Aws Albarghouthi, and Loris D'Antoni. Proving Data-Poisoning robustness in decision trees. December 2019.

[32] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold,

and Rich Zemel. Fairness through awareness. April 2011.

[33] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In CRYPTO, 2020.

[34] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-Learning is currently more broken than you think. In EuroS&P 2023. IEEE, July 2023.

[35] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical non-interactive arguments of knowledge. 2019.

[36] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. 2023.

[37] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In IEEE S&P, May 2018.

[38] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: Efficient and scalable MPC in the honest majority setting. In CRYPTO, 2021.

[39] Zayd Hammoudeh and Daniel Lowd. Identifying a Training-Set attack's target using renormalized influence estimation. In ACM CCS 2022, January 2022.

[40] Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved problems in ML safety. September 2021.

[41] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Advances in Cryptology — CRYPT0' 95, pages 339–352. Springer Berlin Heidelberg, 1995.

[42] White House. Executive order on the safe, secure, and trustworthy development and use of artificial intelligence, October 2023. E.O. 14110.

[43] Dimitar Jetchev and Marius Vuille. XorSHAP: Privacy-Preserving explainable AI for decision tree models. 2023.

[44] Hengrui Jia, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In 2021 IEEE Symposium on Security and Privacy (SP). IEEE, May 2021.

[45] Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nick Hynes, Nezihe Merve Gurel, Bo Li, Ce Zhang, Dawn Song, and Costas Spanos. Towards efficient data valuation based on the shapley value. February 2019.

[46] Nikola Jovanović, Marc Fischer, Samuel Steffen, Zurich Eth, and Martin Vechev. Private and reliable neural network inference. In ACM CCS, 2022.

[47] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: a low latency framework for secure neural network inference. In USENIX Security, August 2018.

[48] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. Scaling up trustless DNN inference with Zero-Knowledge proofs. October 2022.

[49] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-Size commitments to polynomials and their applications. In ASIACRYPT 2010, 2010.

[50] Marcel Keller. MP-SPDZ: A versatile framework for Multi-Party computation. In ACM CCS, CCS '20, November 2020.

[51] Marcel Keller and Ke Sun. Secure quantized training for deep learning. In ICML, volume 162, 2022.

[52] Rajiv Khanna, Been Kim, Joydeep Ghosh, and Sanmi Koyejo. Interpreting black box predictions using fisher kernels. In AISTATS, volume 89. PMLR, 2019.

[53] Niki Kilbertus, Adrià Gascón, Matt J Kusner, Michael Veale, Krishna P Gummadi, and Adrian Weller. Blind justice: Fairness with encrypted sensitive attributes. June 2018.

[54] Dongwoo Kim and Cyril Guyot. Optimized Privacy-Preserving CNN inference with fully homomorphic encryption. IEEE Trans. Inf. Forensics Secur., 2023.

[55] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. March 2017.

[56] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: Super-fast and robust Privacy-Preserving machine learning. In USENIX Security 21, 2021.

[57] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively secure 4PC for secure training and inference. 2022.

[58] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. NeurIPS, 2021.

[60] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proc. IEEE, 86(11):2278–2324, November 1998.

[61] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference. 2023.

[62] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In NeurIPS, volume 30. Curran Associates, Inc., 2017.

[63] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. RoFL: Attestable robustness for secure federated learning. 2023.

[64] Hidde Lycklama, Nicolas Küchler, Alexander Viand, Emanuel Opel, Lukas Burkhalter, and Anwar Hithnawi. Cryptographic auditing for collaborative learning. In NeurIPS ML Safety Workshop, 2022.

[65] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: Dynamic-Committee proactive secret sharing. In ACM CCS,

November 2019.

[66] Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. In CCS, October 2018.

[67] Debarghya Mukherjee, Mikhail Yurochkin, Moulinath Banerjee, and Yuekai Sun. Two simple ways to learn individual fairness metrics from data. In ICML, volume 119, 2020.

[68] National Science and Technology Council Committee on Technology. Preparing for the future of artificial intelligence, October 2016.

[69] NIST. SHA-3 standard: Permutation-Based hash and Extendable-Output functions. Technical report, 2016.

[70] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-Knowledge proofs for distributed secrets. In USENIX Security, 2022.

[71] Arpita Patra and Ajith Suresh. BLAZE: blazing fast privacy-preserving machine learning. 2020.

[72] Torben Pryds Pedersen. Non-Interactive and Information-Theoretic secure verifiable secret sharing. In CRYPTO. Springer Berlin Heidelberg, 1992.

[73] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. arXiv preprint arXiv:1806.03822, 2018.

[74] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. February 2016.

[75] Dragos Rotaru and Tim Wood. MArBled circuits: Mixing arithmetic and boolean circuits with active security. In Progress in Cryptology – INDOCRYPT 2019, pages 227–249. Springer International Publishing, 2019.

[76] Anian Ruoss, Mislav Balunović, Marc Fischer, and Martin Vechev. Learning certified individually fair representations. February 2020.

[77] David Schultz, Barbara Liskov, and Moses Liskov. MPSS: Mobile proactive secret sharing. ACM Trans. Inf. Syst. Secur., 13(4):1–32, December 2010.

[78] Shahar Segal, Yossi Adi, Benny Pinkas, Carsten Baum, Chaya Ganesh, and Joseph Keshet. Fairness in the eyes of the data: Certifying Machine-Learning models. September 2020.

[79] European Parliamentary Research Service. Briefing Artificial intelligence act. https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf, march 2024. Briefing 2021.

[80] Ali Shahin Shamsabadi, Sierra Calanda Wyllie, Nicholas Franzese, Natalie Dullerud, Sébastien Gambs, Nicolas Papernot, Xiao Wang, and Adrian Weller. Confidential-PROFITT: Confidential PROof of FaIr training of trees. In ICLR, 2022.

[81] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. Traceback of data poisoning attacks in neural networks. October 2021.

[82] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh, editors, ICML, volume 70, 2017.

[83] Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A Erdogdu, and Ross Anderson. Manipulating SGD with data ordering attacks. NeurIPS, April 2021.

[84] Nigel P Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. In Cryptography and Coding, pages 342–366. Springer International Publishing, 2019.

[85] Haochen Sun and Hongyang Zhang. ZkDL: Efficient zero-knowledge proofs of deep learning training. July 2023.

[86] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, ICML, volume 70, 2017.

[87] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. PETS, 2019.

[88] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. 2021.

[89] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In EMNLP Workshop BlackboxNLP 2018, 2018.

[90] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. S&P, April 2019.

[91] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. In NeurIPS, volume 33, 2020.

[92] Mikhail Yurochkin, Amanda Bower, and Yuekai Sun. Training individually fair ML models with sensitive subspace robustness. 2020.

[93] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. Cerebro: A platform for multi-party cryptographic collaborative learning. In USENIX Security, 2021.

[94] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. In IEEE S&P 2019, 2019.

We refer to the extended version of the paper [1] for the remaining appendices.

C Consistency Check

In the following, we proof that our approach (Π_{cc}) fulfills the properties for a Proof-of-Consistency (Definition 4.1), which are defined below. We then provide formal definitions for the strawman constructions discussed in §4.1. In addition, we briefly discuss how to efficiently realize $\mathcal{F}_{\text{ABB}}^{[\text{EC}]}$ for Pedersen Vector Commitments.

A valid Proof-of-Consistency as in Definition 4.1 satisfies the following properties:

- **Correctness:** If $[\![\mathbf{x}]\!]$ is a valid input of $\mathbf{x}$ to $\mathcal{F}_{\text{ABB}}$ and c is a valid commitment to $\mathbf{x}$ computed as $\mathsf{PoC.Commit}(\mathsf{pp}_{\mathsf{poc}}, \mathbf{x}, r)$ for all public parameters $\mathsf{pp}_{\mathsf{poc}} \leftarrow \mathsf{PoC.Setup}(1^\lambda, d)$ and randomness $r \xleftarrow{\$} \mathcal{R}$, then $\mathsf{PoC.Check}(\mathsf{pp}_{\mathsf{poc}}, c, [\![\mathbf{x}]\!]; \mathbf{x}, r) = 1$ with overwhelming probability.
- **Soundness:** If there exists no $\mathbf{x}$ such that $\mathcal{F}_{\text{ABB}}$ holds $[\![\mathbf{x}]\!]$ and $c = \mathsf{PoC.Commit}(\mathsf{pp}_{\mathsf{poc}}, \mathbf{x}, r)$ for all public parameters $\mathsf{pp}_{\mathsf{poc}}$ and for randomness $r \xleftarrow{\$} \mathcal{R}$, then for all $\lambda \in \mathbb{N}$, and for all polynomial-time adversaries $\mathcal{A}$ on input $(\mathsf{aux}_{\mathcal{A}}, [\![\mathbf{x}]\!], r, c))$, the probability that $\mathsf{PoC.Check}(\mathsf{pp}_{\mathsf{poc}}, c, [\![\mathbf{x}]\!]; \mathbf{x}, r) = 1$ is negligible in λ.
- **Zero-knowledge:** For every probabilistic polynomial-time interactive machine $\mathbb{V}'$ that plays the role of the verifiers, there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that for any $[\![\mathbf{x}]\!]$, randomness r and $c = \mathsf{PoC.Commit}(\mathsf{pp}_{\mathsf{poc}}, \mathbf{x}, r)$ the transcript of the protocol between $\mathbb{V}'$ and $\mathbb{P}$ and the output of $\mathcal{S}$ on input ($[\![\mathbf{x}]\!]$, r, c) are computationally indistinguishable.

We now prove that our protocol for input consistency satisfies these properties.

Lemma C.1. Π_{cc} is a Proof-of-Consistency (Definition 4.1).

Proof. The protocol Π_{cc} in Protocol 4.2 satisfies the properties of a PoC:

- **Completeness:** From the correctness of the MPC protocol, it holds that $\rho = \omega + \sum_{i=1}^{d} x_i \cdot \beta^i$. Further, the opening proof of the polynomial commitment $(c \cdot c_\omega)$ also evaluates to ρ at β due to the homomorphic property of the scheme. The verifiers accept because of the completeness of the polynomial commitment scheme.
- **Soundness:** Let ω be a random value, $\hat{f} = f + f_\omega$ be the polynomial defined as in the protocol as $\hat{f}(z) = \omega + \sum_{i=1}^{d} x_i \cdot z^i$, let $[\omega]$ be a secret-sharing of ω and let $[\mathbf{x}]$ be a secret-sharing of $\mathbf{x}$. If the verifiers do not hold a valid secret-sharing $[\omega]$ or $[\mathbf{x}]$, then the MPC protocol in Step 3 aborts. Otherwise, the correctness of the MPC protocol guarantees that a valid secret-sharing of $[\omega]$ and $[\mathbf{x}]$ implies that ρ equals $\omega + \sum_{i=1}^{d} x_i \cdot \beta^i$ and that $\hat{f}(\beta) = \rho$ or the protocol aborts. Let c' be a polynomial commitment such that $c' \neq \mathsf{PC.Commit}(\mathtt{pp}, \hat{f}, r + r_\omega)$. Then, from the polynomial binding property of the polynomial commitment scheme, either c' is a commitment to a different polynomial f' or the verifiers reject the proof in Step 5 with overwhelming probability. In the case that c' is a commitment to a different polynomial f', the verifiers only accept $\mathsf{PC.Check}(\mathsf{pp}, c', \beta, \rho, \pi)$ if f' agrees with $\hat{f}$ at point β because of the evaluation binding property of the polynomial commitment. Because β was sampled uniformly at random, from the Demillo-Lipton-Schwartz-Zippel Lemma [29], it holds that:

$$\Pr\left[\mathsf{PC.Check}(\mathsf{pp}, c', \beta, \rho, \pi) = 1\right] \leq \frac{d}{p} \quad ,$$

  which can be made negligible by choosing a suitably large p. Thus, the verifiers reject with overwhelming probability.
- **Zero-knowledge:** The simulator $\mathcal{S}$ works as follows: It samples d random coefficients that define the polynomial f and one random coefficient to define the polynomial f_ω. Then, it samples a random point $\beta \xleftarrow{\$} \mathbb{F}_p$ and runs the MPC simulator to produce the transcript for the computation of ρ. Finally, it samples $r, r_\omega \xleftarrow{\$} \mathbb{F}_p$ and computes $c = \mathsf{PC.Commit}(\mathsf{pp}, f, r)$, $c_\omega = \mathsf{PC.Commit}(\mathsf{pp}, f_\omega, r_\omega)$ and $\pi = \mathsf{PC.Prove}(\mathsf{pp}, c \cdot c_\omega, f + f_\omega, r + r_\omega, \beta, \rho)$ and outputs $(c, c_\omega, \pi, \rho, \beta)$. The indistinguishability with the real execution follows from the fact that ρ is uniformly distributed in $\mathbb{F}_p$ because of ω, the properties of the MPC protocol and the hiding property of the polynomial commitment scheme.

□