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, have been hard to utilize due to privacy concerns and regulatory constraints. Simultaneously, there is a growing emphasis on enhancing the transparency and accountability of machine learning, including the ability to audit ML deployments. While ML auditing and PPML have both been the subjects of intensive research, they have predominately been examined in isolation. However, their combination is becoming increasingly important. In this work, we introduce Arc, an MPC framework for auditing privacy-preserving machine learning. At the core of our framework is a new protocol for efficiently verifying MPC inputs against succinct commitments at scale. 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). 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 [23, 39, 46, 48, 59, 78]. The majority of these frameworks rely on secure computation techniques [5, 42, 43], 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 challenging to fulfill other desirable and often legally mandated objectives in ML, such as transparency and accountability. While these objectives 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.

Verifiable Claims and Accountability in ML. 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 [8]. The former focus on pre-deployment verification techniques that act as predefined sets of verifications on the model, data, or training process [19, 36], such as model and data validation tests [17, 56] or robustness [26] and fairness [6] 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 [33, 70, 71]. For example, individuals may seek an explanation of a decision to mitigate potential harm or to investigate its fairness [55, 65]. Recent efforts have examined the realization of a priori-auditing techniques in secure settings using a variety of ad-hoc techniques [17, 38, 45, 69]. This includes work for verifying robustness, verifiable fairness, and model and data validation techniques [17, 38]. 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

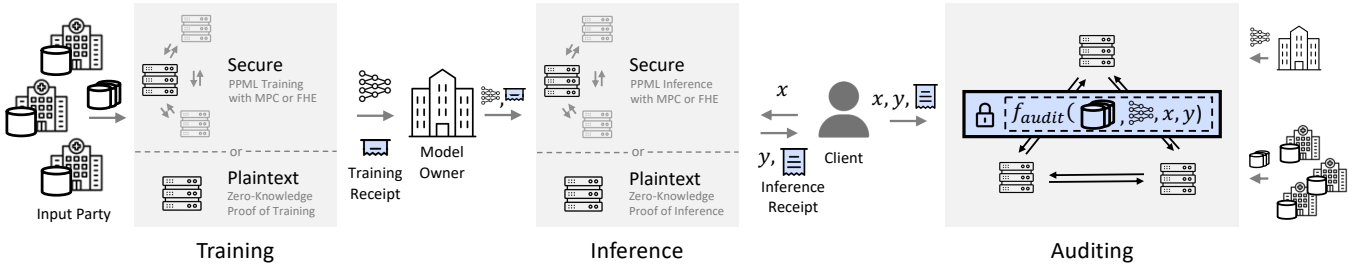

Figure 1: Overview of Arc, which augments existing PPML pipelines with an MPC auditing phase to execute auditing functions.

party (which can be granted access to training data, model, 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 continously 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 [34, 58, 68]. 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. In this paper, we present Arc, a framework for privacy-preserving auditing for PPML systems. Our system 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. We present our privacy-preserving auditing protocol which decouples training, inference and auditing while maintaining consistency via the use of concise cryptographic *receipts*. We first describe (and proof secure) our auditing protocol relative to a black-box definition of these proofs of consistency, highlighting the complexities of supporting the mixed secure/plaintext settings common in practical PPML deployments. The overhead of our protocol is primarily determined by the efficiency of the underlying proof of consistency, and we 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. Finally, we evaluate our framework in §5.

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 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 [5, 12, 23, 39, 46, 48, 49, 59, 78]. 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 $\mathbb{F}$ or ring $\mathbb{Z}$ [5]. 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 [5, 23, 43, 48, 59, 78]. Different functionalities might be implemented most efficiently in different fields or rings, in which case we can use share conversion to switch between, e.g., ring and field-based MPC. In Appendix D, 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, a significant number of techniques offer valuable insights that are only achievable when the model is operational in a production environment. 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 [55, 65], investigating the model's training data for biased or poisoned samples [57, 71, 79], or ensuring fairness by analyzing model predictions under hypothetical scenarios where specific input features are altered [60]. 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 [4, 75]. 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 Appendix E 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 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 [38]. Holmes improves the quality of MPC training to conduct distribution tests on training data via efficient interactive zero-knowledge proofs [17] before training starts. 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 [38]	×	○	●	●	▮▮▯	▮▮▯	–
Agrawal et al. [1]	×	○	●	○	▮▮▯	▮▮▯	▮▮▮▮
Kilbertus et al. [45]	✓	○	●	●	▮▯▯	▮▯▯	▮▮▮▮
Segal et al. [69]	✓	○	●	○	▮▯▯	▮▯▯	▮▮▮▮
Holmes [17]	✓	●	○	○	▮▮▮▮	▮▮▮▮	–
Cerebro [81]	✓	●	○	○	▮▯▯	▮▮▯	▮▯▯
Ours (§3)	✓	●	●	●	▮▮▮▮	▮▮▮▮	▮▮▮▮

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).

A line of work focuses on fairness certification [2, 45, 69]. This enables clients to verify that their private predictions were generated by a certified model, achieved by having a regulator sign a hash-based commitment of the model. Cerebro 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

We begin by capturing the essential requirements for achieving robust privacy-preserving audits and describing our threat model and assumptions. We then provide a high-level overview of our framework, Arc. This is followed by a formal treatment of our privacy-preserving protocol, with the corresponding proof in Appendix B. In the next section, we describe our efficient instantiation of the consistency checks required by our framework. We defer a discussion of realizing auditing functions under secure computation to Appendix E.

Requirements. Any privacy-preserving auditing system should offer *(i)* secrecy, *(ii)* correctness & soundness, and

(iii) modularity & practicality. For *secrecy*, the system should preserve the privacy guarantees of PPML systems, except for what can be inferred from the output of auditing functions[1]. However, in order 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. In order to allow us to rely on the results of auditing, the system must be *corrects & sound*, i.e., the audit must be correctly computed even in the presence of malicious parties. Note that the notion of auditing presupposes a potential malicious intent, and, as a result, we should not rely on parties to provide honest inputs to the system. 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. Similarly, we note that there is a strong incentive to avoid auditing, and, as such, the system should have the ability to detect malicious disruptions of the audit process. Specifically, we want to prevent malicious parties from surreptitiously aborting the audit computation. Finally, an auditing system must be sufficiently *modular & practical* to cover the wide range of possible PPML approaches for both training and inference, and the scale of typical ML workloads. Note that, PPML systems frequently cover only either training or inference, and an auditing system should also support deployment scenarios where training or inference are performed over plaintext data. For example, many scenarios permit the release of a (e.g., differentially private) model after a secure training phase, or consider secure inference for a centrally trained model. Therefore, an auditing system should support both plaintext approaches and secure-computation-based approaches to training and inference. Finally, in order to support real-world deployments with many potential inference clients, the system must scale independently of the number of inferences and clients in the system. At the same time, the system should not require the clients to maintain significant long-term state beyond storing the received predictions.

Threat Model. We consider an actively malicious adversary 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 or receives outputs is honest. However, as 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 executed by one or more computational parties, we assume at least one of them is honest. Note that, certain instantiations of secure computation might impose additional constraints on the adversary. For example, MPC protocols that assume an honest majority of computing parties are frequently significantly more efficient than their dishonest majority counterparts. In our framework, we assume the presence of several cryptographic primitives in the construction of our protocol, including an arithmetic black-box (ABB) interface to abstract PPML protocols and inherit any constraints on the adversary that instantiations might require. In addition, we assume 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 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.

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

3.1 Framework Overview

Arc supports a real-world, modular approach to auditing PPML systems that enables training, inference, and auditing to be realized as separate phases, using different privacy-preserving techniques. In the training phase, *data holders* provide the training data that the model will be trained on that is then received by the *model holders*. In the inference phase, the model holders use this model to serve predictions to *clients*. Along with a prediction, a client receives a *receipt* that it can use to request an audit in the future. The receipt contains a reference to the exact model that was used for the prediction along with a reference to the training data that was used to train the model. In the auditing phase, the client requests the data holders and the model holder to input the original training data and model, which are verified against the receipt using a POC (cf. §4).

In order to capture common real-world deployment patterns of PPML (where only parts of the pipeline are protected), we support both secure-computation-based and plaintext versions of training and inference. Note that the auditing phase itself is always realized as a multi-party computation in order to achieve the required privacy and robustness. In our system, we assume that the model owners can always receive the model in the clear. While the protocol could be trivially extended to support sharing secret shares of the model, 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 maintenance and re-sharing to new sets of entities [34, 58, 68], a complexity which we aim to avoid in our design.

Our system is comprised of *(i)* a front-end that accepts specifications such as audit objectives, involved parties, and their roles, secure protocol settings, and model, prediction, and data parameters. *(ii)* the core Arc protocol, which supports the secure execution of the different stages under, e.g., MPC, *(iii)* our proof of consistency, which ensures a coherent link between the data, the model, and the auditing phase (enforce that they are carried on the same snapshot), which is fundamental for achieving secure and robust audits, *(iv)* a suite of privacy-preserving auditing functions, which realize common auditing functions in a secure and efficient way (cf.

Figure 2: Protocol Π_{Arc}

Π_{Arc} is a protocol between N_{DH} data holders $\text{DH} = \{\text{DH}_1, \ldots, \text{DH}_{N_{\text{DH}}}\}$, N_{M} model holders $\text{M} = \{\text{M}_1, \ldots, \text{M}_{N_{\text{M}}}\}$, N_{C} clients $\text{C} = \{\text{C}_1, \ldots, \text{C}_{N_{\text{C}}}\}$, N_{TC} training computers $\text{TC} = \{\text{TC}_1, \ldots, \text{TC}_{N_{\text{TC}}}\}$, N_{IC} inference computers $\text{IC} = \{\text{IC}_1, \ldots, \text{IC}_{N_{\text{IC}}}\}$, and N_{AC} audit computers $\text{AC} = \{\text{AC}_1, \ldots, \text{AC}_{N_{\text{AC}}}\}$. Π_{Arc} is parameterized by a learning algorithm $\mathcal{T}$, a set of allowed auditing functions F_{audit}, a proof-of-consistency PoC as in Definition 4.1, a signature scheme SIG as in Definition A.5, in the case of plaintext training, a zero-knowledge proof of training POT as in Definition A.6, in the case of plaintext inference, a zero-knowledge proof of inference POI as in Definition A.7. 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 identifiable abort, which is also used by PoC internally.

Input: Each 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 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 $\text{pp}_{\text{poc}} \leftarrow \text{PoC.Setup}(1^\lambda, d)$ where d is the maximum of all d_i and m, and (in the case of plaintext training) $\text{pp}_{\text{pot}} \leftarrow \text{POT.Setup}(1^\lambda)$ and (in the case of inference training) $\text{pp}_{\text{poi}} \leftarrow \text{POI.Setup}(1^\lambda)$.

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

T.1 Each data holder 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 TC.
- Computes a commitment to the training dataset $c_{D_i} = \text{PoC.Commit}(\text{pp}_{\text{poc}}, D_i, r_{D_i})$ and sends (c_{D_i}) to all TC.
- Executes POC $\text{PoC.Check}(\text{pp}_{\text{poc}}, c_{D_i}, [\![D_i]\!]; D_i, r_{D_i})$ with all TC or each TC verifies that $c_{D_i} = \text{PoC.Commit}(\text{pp}_{\text{poc}}, D_i, r_{D_i})$ for all DH_i.

T.2 Each training computer TC_j:

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

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

T.4 Each model holder M_k checks each of the following and aborts if any fail:

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

Inference: The protocol proceeds as follows between inference computers IC, client C_i and model holder M_k, using a new instance of $\mathcal{F}_{\text{ABB}}$:

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

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

I.2 Each inference computer IC_j:

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

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

I.4 The client C checks each of the following and aborts if any fails:

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

Figure 2: Protocol Π_{Arc} (cont.)

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

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

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

A.3 **Verify Audit Requester:** The client $\mathtt{C}_j$ 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 $\mathtt{AC}$ that its inputs x and y are consistent with c_x and c_y. If any of the checks fail, $\mathtt{AC}_i$ aborts marking $\mathtt{C}_j$ as malicious.

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

- The model holder $\mathtt{M}_k$ 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 $\mathtt{C}_j$. Each $\mathtt{AC}_i$ aborts marking $\mathtt{M}_k$ as malicious if verification fails.
- Each audit computer computes $\mathsf{SIG.Verify}(\mathsf{pk}_{\mathtt{IC}}, c \,\|\, c_x \,\|\, c_y, \sigma_{\mathtt{IC}})$ or $\mathsf{POI.Verify}\ (\mathsf{pp}_{\mathsf{poi}}, c_M, c_x, c_y, \pi_{\text{I}})$.

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

- Each data holder $\mathtt{DH}_i$ 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}. $\mathtt{AC}_j$ also checks $\mathsf{SIG.Verify}(\mathsf{pk}_{\mathtt{DH}_i}, c, \sigma_{\text{T}})$ for each $\mathtt{DH}_i$. If verification fails, $\mathtt{AC}_j$ aborts marking $\mathtt{M}_k$ as malicious.
- Each audit computer computes $\mathsf{SIG.Verify}(\mathsf{pk}_{\mathtt{TC}}, c, \sigma_{\mathtt{TC}})$ or $\mathsf{POT.Verify}\ (\mathsf{pp}_{\mathsf{pot}}, c, \pi_{\text{T}})$.

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

Appendix E). In the following, we focus on describing the core protocol, while the next section describes our instantiation of the proof of consistency.

3.2 Arc Protocol

In the following, we give an intuitive overview of our protocol, Π_{Arc} (c.f. Fig. 2) and its building blocks. We prove the security of our protocol in the real/ideal world paradigm [13] and briefly introduce the ideal functionality $\mathcal{F}_{\text{Arc}}$ (c.f. Fig. 3) here. Due to space constraints, we defer formal definitions of the building blocks of Π_{Arc} to Appendix A and refer to Appendix B for our proof.

Protocol Overview. Our protocol lifts an existing PPML system 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. We model the underlying PPML protocols used for ML training, inference, and the auditing functions as a *reactive* arithmetic black-box (ABB), with $\mathcal{F}_{\text{ABB}}.\mathsf{Train}_{\mathcal{T}}$, $\mathcal{F}_{\text{ABB}}.\mathsf{Predict}$ and $\mathcal{F}_{\text{ABB}}.\mathsf{Audit}$, which allows us to focus on the auditing-relevant aspects of the framework. Our protocol utilizes standard signatures (cf. Definition A.5 in Appendix A) 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. We use commitment and POC interchangeably for the rest of this section.

In the following, we provide the intuition behind our protocol, starting with the training and inference phase of Π_{Arc}, which proceed in a similar way:

- 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 $\mathsf{PoC.Check}$ which involves an (efficient) multi-party computation. In the plaintext setting, the computing parties can simply locally recompute the commitments.
- After computing the underlying ML training or inference, the training computers 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 proof-of-training (PoT) or proof-of-inference (PoI) as we cannot rely on a split-trust assumption for integrity.
- 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.

The receipt received by the model holders after training is comprised of the commitments to the training data, the training randomness, and the resulting model; the data holders' signatures; and either the signature from the computing parties or a proof of training. During inference, the model holders 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.

During auditing, the client provides the receipt and inputs the prediction sample and result into the MPC computation. Meanwhile, the model holders and data holders provide their respective inputs. The protocol first confirms that the signa-

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 holders $\texttt{DH} = \{\texttt{DH}_1, \ldots, \texttt{DH}_{N_{\texttt{DH}}}\}$, $N_{\texttt{M}}$ model holders $\texttt{M} = \{\texttt{M}_1, \ldots, \texttt{M}_{N_{\texttt{M}}}\}$, $N_{\texttt{C}}$ 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 $\texttt{IC} = \{\texttt{IC}_1, \ldots, \texttt{IC}_{N_{\texttt{IC}}}\}$, $N_{\texttt{AC}}$ audit computers $\texttt{AC} = \{\texttt{AC}_1, \ldots, \texttt{AC}_{N_{\texttt{AC}}}\}$. 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 $M \leftarrow \mathcal{T}(D_1, \ldots, D_{N_{\texttt{DH}}}, J)$. In the plaintext setting: if $\mathcal{A}$ controls any $\texttt{TC}$, send (J, M) to $\mathcal{A}$
2. If $\mathcal{A}$ controls any model holders, 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 $((D_1, \ldots, D_{N_{\texttt{DH}}}), M, \mathsf{id}_M)$ to L_{M}, and send the model $(\texttt{Output}, \mathsf{id}_M, M)$ to each model holder $\texttt{M}_j$, 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 $M' = \mathcal{T}(D'_1, \ldots, D'_{N_{\texttt{DH}}}, J')$, choose a random id'_M, add $((D'_1, \ldots, D'_{N_{\texttt{DH}}}), M', \mathsf{id}'_M)$ 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 3: Arc's Ideal Functionality.

tures (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 PoC.Check checks the consistency of the provided inputs with the commitments in the receipt. Finally, after all checks have passed, the protocol computes the audit function.

Modeling Cryptographic Auditing. We model the intended behavior of Arc as a reactive ideal functionality $\mathcal{F}_{\text{Arc}}$ (cf. Fig. 3) consisting of three stages corresponding to training, inference and auditing. The functionality allows training one or more models which it stores internally using L_{M}. Clients can request predictions for these models and audit the predictions they received with a specific set of auditing functions. The functionality also models the plaintext training and inference settings in which the adversary can generate local models and predictions, respectively. For inference, the functionality allows the adversary to submit additional models and corresponding datasets to L_{M}, as long as the model, the datasets and the randomness used for training are consistent with each other. This represents the scenario in which the adversary may serve clients with predictions from locally generated models, which is unavoidable if it controls the model holder, all data

holders and all training computers. Similarly, for auditing, the functionality allows the data holder to audit for predictions that it does not yet store internally, as long as the prediction was made by a valid model. This represents the scenario in which the adversary can locally generate valid predictions if the model holder and the inference computers involved in inference are malicious. Note that, in the secure setting, training and inference cannot be local to the adversary, because we require that at least one computing party is honest. 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 id_M to the $\mathcal{A}$ after training.

The functionality satisfies secrecy because it only outputs the result of the auditing function for valid predictions, i.e., predictions made through $\mathcal{F}_{\text{Arc}}$ and thus stored in L_{P}, in addition to the expected output in the training and inference stages. The functionality also satisfies correctness and soundness, because the clients receive audit output for predictions, models and data stored internally by the functionality and the functionality only ever stores consistent information. The functionality guarantees security with abort in the training and inference phases, ensuring that either the computation always correctly completes or it aborts when detecting malicious activity. This follows a standard assumption in recent practical protocols for PPML training and inference [16, 18, 31, 59, 63, 78, 82]. In the auditing phase, we require the stronger security guarantee of Identifiable Abort (ID-Abort) for the audit computers, i.e., any party with this role can identify a party that causes the computation to abort. Note that, in the secure outsourced computation (SOC) setting, this does not include the client and the model holder as they do not participate in the secure computation except by providing inputs. Alternatively, one could consider a functionality that achieves publicly identifiable abort and a trivial extension of our protocol that relies on $\mathcal{F}_{\text{ABB [PID]}}$ instead of $\mathcal{F}_{\text{ABB [ID]}}$. However, concrete instantiations of $\mathcal{F}_{\text{ABB [PID]}}$ introduce significant additional overheads [22, 62] and we therefore limit ourselves to $\mathcal{F}_{\text{ABB [ID]}}$.

4 Proof of Consistency

Guaranteeing consistency across the various stages of Arc is critical for maintaining the integrity of the audit process. Without such consistency, there exists a risk that a party, having initially utilized a non-compliant training dataset D' or model M', might later claim compliance by presenting a different, compliant model $M \neq M'$ and/or dataset $D \neq D'$. In theory, ensuring consistency can be achieved straightforwardly using standard cryptographic techniques, such as commitments; however, naive approaches would incur significant performance overheads, making practical deployment infeasible. Furthermore, it is important to minimize the persistent storage overhead associated with our protocol to accommodate clients operating with constrained devices. In the following, we first describe how we formalize our consistency check and its requirements. Then, we explain our construction and optimizations. Finally, we discuss aspects related to the concrete realization of our construction as an efficient MPC protocol.

4.1 Formalization & Requirements

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 matches the provided commitment. In the following, we provide the formal definition of PoC and the properties it needs to achieve.

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 as a set of protocols ($\mathsf{PoC.Setup}$, $\mathsf{PoC.Commit}$, $\mathsf{PoC.Check}$) where:

- $\mathsf{PoC.Setup}(1^\lambda, d) \to \mathsf{pp}_{\mathsf{poc}}$: prepares public parameters $\mathtt{pp}$ supporting inputs of size d.
- $\mathsf{PoC.Commit}(\mathsf{pp}_{\mathsf{poc}}, \mathbf{x}, r) \to c$: An algorithm in which the prover generates a commitment to (a vector of) inputs $\mathbf{x}$ with randomness r.
- $\mathsf{PoC.Check}(\mathsf{pp}_{\mathsf{poc}}, c, [\![\mathbf{x}]\!]; \mathbf{x}, r) \to \{0,1\}$: 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 satisfies the following:

- **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 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

[2] Technically, because we formalize MPC using $\mathcal{F}_{\text{ABB}}$, the values are not strictly required to be secret shared, but simply in whatever representation $\mathcal{F}_{\text{ABB}}$ uses for secret values.

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.

In addition to the above classical properties of a commitment protocol, a PoC should yield succinct commitments and its protocols should be efficiently computable in order to be practical for ML. In particular, there should be a limited amount of MPC operations in PoC.Check which are relatively expensive. We discuss how our consistency check protocol achieves these requirements in the next section.

4.2 Practical Proof-of-Consistency

For our framework to be practical, we must instantiate PoC with a protocol that is not only efficient to execute but also yields commitments of small size. This is crucial for ensuring efficient communication and storage, especially when dealing with large input sizes and resource-constrained clients. In the following, we discuss several approaches based on existing literature and highlight their inherent limitations. Then, we introduce our consistency check protocol, which addresses these shortcomings while achieving efficiency and scalability.

Direct Commitments [1, 45, 69]. 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 $c' \leftarrow \mathsf{COM.Commit}([\![\mathbf{x}]\!], [\![r]\!])$ 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 [45], SHA-3 [69] and MPC-friendly constructions such as LowMCHash-256 [1]. 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. This is primarily due to its reliance on non-linear operations, which are significantly less efficient to compute in MPC.

Homomorphic Commitments [81]. To mitigate the MPC cost of PoC.Check, one can rely on homomorphic commitments such as Pedersen commitments instantiated using an elliptic curve group [64, 81]. 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 $\tilde{c}' = \mathsf{COM.Commit}(\sum_i \gamma^i \cdot [\![x_i]\!])$ 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. While hash-based approaches remain more concretely efficient for very small inputs, the Pedersen commitment approach becomes more efficient already for moderate input sizes.

Efficient Vector Commitments via EC-MPC. Although the second approach only requires computing a single Pedersen commitment in PoC.Check, the concrete overhead of computing this commitment makes this protocol prohibitively expensive for larger applications. This is because the commitment requires the use of elliptic curve operations which are computed by decomposing them into operations over the curve's base field. This approach is very expensive: curve additions require tens of field operations, and scalar products require thousands. Prior work has observed that most secret-sharing based MPC protocols to compute arithmetic circuits over a field generalize to arithmetic circuits involving elliptic curve points [62, 73]. Using such protocols with additional support for computations over an elliptic curve group $\mathbb{G}$ of order p (which we denote as $\mathcal{F}_{\text{ABB}}^{[\text{EC}]}$) when instantiating PoC offers a significant improvement to performance. This allows us to reduce the overhead of computing a Pedersen commitment from hours to seconds, which vastly improves the results reported for Cerebro by Zheng et al. [81]. However, the shift in cost model induced by using MPC protocols with efficient support for curve operations also enables a simpler construction from Pedersen Vector Commitments (PVCs) (Definition C.1) that has the same communication overhead as Cerebro in PoC.Check but produces constant size commitments. A similar approach applies to the distributed computation of ECDSA [24, 73]. We refer to Appendix C.3 for a definition and a security proof.

Arc PoC Protocol. Although the previous approach already represents a significant improvement, it requires a significant amount of MPC computation because parties have to recompute COM.Commit as part of COM.Verify. However, this computation is not strictly necessary; in practice, we do not need to compute COM.Commit, but rather only verify that the $[\![\mathbf{x}]\!]$ in $\mathcal{F}_{\text{ABB}}$ match the input $\mathbf{x}$ of COM.Commit. We propose a protocol that allows the prover $\mathbb{P}$ to convince the verifiers of this fact with a polynomial identity test. We first define a polynomial over the inputs $\mathbf{x} \in \mathbb{F}_p^d$ where each input element $x \in \mathbf{x}$ is a coefficient of the polynomial as $f(\beta) = \sum_{i=1}^{d} x_i \cdot \beta^i$. The prover commits to f using a (homomorphic) polynomial commitment scheme [41] to obtain a constant-size commitment c. In PoC.Check, the parties first collaboratively sample a point $\beta \xleftarrow{\$} \mathbb{F}_p$ and then evaluate the polynomial at β using $\mathcal{F}_{\text{ABB}}$ by computing $\rho = f(\beta) = \sum_{i=1}^{d} [\![x_i]\!] \cdot \beta^i$ and open ρ. Evaluating f in MPC is cheap because this operation only involves addition and scaling operations on the secret shares $[\![\mathbf{x}]\!]$ that 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 $f(\beta)$ equals ρ. The other parties verify this evaluation proof, which, if true, implies with high probability that the polynomial in $\mathcal{F}_{\text{ABB}}$ is equal to the one that is committed to with c. One caveat with the current protocol is that $\rho = f(\beta)$ reveals information about $\mathbf{x}$. We can overcome this by generating and committing to a second polynomial f_ω that is random at the beginning of PoC.Check that we use to mask f. We then evaluate and open $f(\beta) + f_\omega(\beta)$ which is now perfectly indistinguishable from random. The prover and verifiers proceed with the evaluation proof as before, but on the combined commitment of f and f_ω, relying on the homomorphic property of the polynomial commitment scheme.

Protocol 4.1 (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 that is also homomorphic as in Definition A.4. The protocol Π_{cc} works as follows:

- $\mathsf{CC.Setup}(1^\lambda, d) \to \mathsf{pp}$: Run $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(d)$ where d is the number of elements in the input.
- $\mathsf{CC.Commit}(\mathsf{pp}, \mathbf{x}, r) \to c$: The prover computes a polynomial commitment $c \leftarrow \mathsf{PC.Commit}(\mathtt{pp}, f, r)$ where f is defined as $f(z) = \sum_{i=1}^{d} x_i \cdot z^i$. 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 $c_\omega \leftarrow \mathsf{PC.Commit}(\mathtt{pp}, f_\omega, r_\omega)$ to a degree-0 polynomial $f_\omega(z) = \omega$. The prover sends c_ω to all parties and inputs ω to $\mathcal{F}_{\text{ABB}}$.
  2. The parties invoke $\mathcal{F}_{\text{RAND}}$ to obtain a random challenge $\beta \xleftarrow{\$} \mathbb{F}_p$.
  3. The parties invoke $\mathcal{F}_{\text{ABB}}$ 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 $\pi \leftarrow \mathsf{PC.Prove}(\mathsf{pp}, c + c_\omega, f + f_\omega, r + r_\omega, \beta, \rho)$ 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 Identifiable Abort (ID-Abort), denoted as $\Pi_{\text{cc [ID]}}$, by using a broadcast channel for the prover in Step 4 and using MPC protocol that provides identifiable abort ($\mathcal{F}_{\text{ABB [ID]}}$).

Cost Analysis. Most 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 [41], we achieve a constant storage overhead independent in 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 [41]. Hence, our protocol only requires a storage overhead linear in the size of the input and the number of parties of $O(N + d)$. If used with an inner-product argument-based polynomial commitment, the commitment size is also constant, i.e., a single Pedersen Vector Commitment (PVC), but the verification time would be linear in the input size [9].

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 [29, 41] (Definition A.4). 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 $\mathtt{PC.Check}(\mathtt{pp}, \tilde{c}, \beta, \tilde{\rho}, \tilde{\pi})$. This allows the verifier to check only one pairing equation instead of N in the optimistic case 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 ([25]). 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.

Efficient MPC Realization So far, we have abstracted the underlying MPC protocol as $\mathcal{F}_{\text{ABB}}$ (or $\mathcal{F}_{\text{ABB}}^{[\text{EC}]}$). However, efficiently implementing these cryptographic operations requires using specific instantiations of MPC schemes. Specifically, the consistency check protocol needs to be evaluated in $\mathbb{F}_p$ (with EC extensions, where appropriate), where p is a large prime. However, the same field-based MPC protocols are suboptimal for ML computations, which benefit significantly from ring-based arithmetic available in many MPC settings. Even where ring arithmetic is not available, MPC computations can be

realized with significantly smaller fields for the same security level (e.g., 128-bit modulus vs 256-bit modulus). A naive approach that uses the same computation domain for both the consistency check and the ML computation would introduce significant overhead to the underlying ML. While existing works have observed this issue, they have so far failed to address it. In Appendix D, we describe how to achieve the best of both worlds with *share conversion*, an operation that builds on common techniques in advanced MPC implementations, by running the underlying ML computation in an ML-friendly setting and then converting to the crypto-friendly $\mathbb{F}_p$ setting.

5 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 [42], 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 [10] provided by the `libff` library [20]. We use ECDSA signatures on the secp256k1 curve [15] for which a distributed signing protocol was previously implemented in MP-SPDZ [24]. For the evaluation proofs of the polynomial commitments, we use the implementation of the KZG polynomial commitment scheme [41] provided by Arkworks' `poly-commit` library [3]. 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 [54].

To compare the performance of our consistency layer to other approaches (cf. 4.2), we additionally implement a version of PoC based on the SHA3-256 cryptographic hash function denoted by $\mathsf{PoC}_{\mathsf{SHA3}}$ that internally uses the Bristol-fashion circuit implementation of the Keccak-f sponge function [61]. We also implement a version $\mathsf{PoC}_{\mathsf{PED}}$ based on Cerebro [81] that uses Pedersen commitments by adapting the open-source implementation provided by the authors.

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(R) Xeon(R) 3.6Ghz processor and 72GB of RAM. The machines are connected over a local area network (LAN) through a 12Gbps network interface with an average round-trip time (RTT) of 0.5ms. We additionally perform our experiments in a simulated wide area network (WAN) setting using `tc` to introduce an RTT of 80ms and a bandwidth of 2Gbps. 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 because 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.

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 [23]. 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 execution uses the same inputs. We choose the 3PC setting because it allows for the most efficient MPC protocols, favoring $\mathsf{PoC}_{\mathsf{PED}}$ and $\mathsf{PoC}_{\mathsf{SHA3}}$ 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 and the related approaches.

Scenarios. We evaluate on the following models and datasets.
(W1:Adult): A logistic regression model with 3k parameters trained on the Adult [7] binary classification task 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 [43, 77] trained on the MNIST image classification task [52].
(W3:CIFAR-10): A variant of AlexNet [51] as used in Falcon [78], comprising 3.9 million parameters trained on the CIFAR-10 image classification task [50].
We evaluate four auditing functions: (i) KNN-Shapley, which identifies the most important training samples for a prediction by computing the Shapley values using a k-Nearest-Neighbors (kNN) classifier as proxy [37], (ii) Robustness, which shows that the model consistently predicts the same class for samples close to the prediction sample [38], (iii) Fairness, which proves the model would have made the same prediction if specific sensitive attributes were different [38], and (iv) Kernel-SHAP, which identifies the most important features in a prediction sample to provide explanations of the model's local decision boundary [55]. We give more detailed descriptions of each function and how we lift them to the secure computation

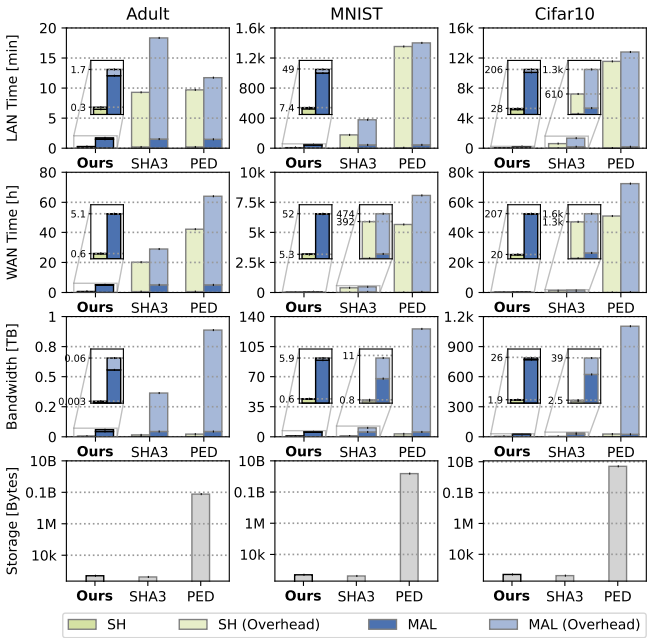

Figure 4: Evaluation of Arc comparing the approaches relative to a single epoch of PPML training.

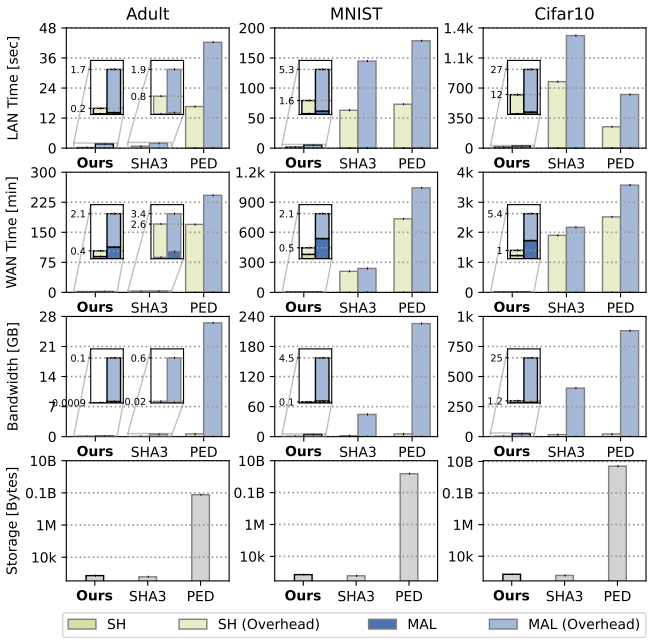

Figure 5: The overhead of our system's consistency protocol relative to a single PPML inference for our three scenarios.

setting in Appendix E.

5.1 Evaluation Results

We evaluate the cost of adding auditing to the training, inference and auditing phases. The main overhead of the consistency layer in each of $\Pi^{\text{train}}_{\text{dist}}$, $\Pi^{\text{inf}}_{\text{dist}}$ and Π_{Arc} consists of two parts: Verifying the inputs of the computation using PoC.Check and, afterwards, computing the output commitments using PoC.Commit. In our description, we focus on the overhead of these two operations because they are the most expensive operations of our protocol. Other components of the protocols 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 holder and each data holder, and a joint signature of 64 bytes for the inference computers and the training computers.

Training. Our protocol for auditable training $\Pi^{\text{train}}_{\text{dist}}$ augments the standard MPC training protocol with PoC.Check for the input datasets before training and with PoC.Commit on the resulting model afterward. We compare the wall time and bandwidth in Fig. 4 relative to a single training epoch. Note that the bandwidth overhead is not significantly affected by the network, so we only present the bandwidth in the WAN setting. We observe that the cost of the baseline approaches is significant. The timing overhead of Arc instantiated with PoC_{PED} is 66-500x compared to training and 6-26x 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. The primary cost of PoC_{PED} is the time required to compute the individual Pedersen commitments to the model parameters; the overhead of PoC.Check is much smaller as this only involves computing a commitment for each of the three input parties. The cost of PoC_{SHA3} scales linearly in the input size and is expensive because of the inherent non-linearity in the hash function which does not scale well as it requires roughly 35 AND gates per input bit.

In comparison, our consistency check protocol introduces only $1.07 - 1.35$x overhead in the LAN and less than 1.02x in the WAN setting compared to training a single epoch across all scenarios. This is due to the larger dependency on local computation than the other approaches. We also discuss the number of bytes required for the model holder to store the cryptographic material consisting of commitments and signatures of the data holders. Fig. 4 shows 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}. Therefore, we conclude that the overhead of our approach relative to PPML training is negligible.

Inference. Auditable inference augments regular ML inference with PoC.Check to verify that the model input matches the commitment and computes a commitment to the prediction and its result using PoC.Commit. Model inference is a

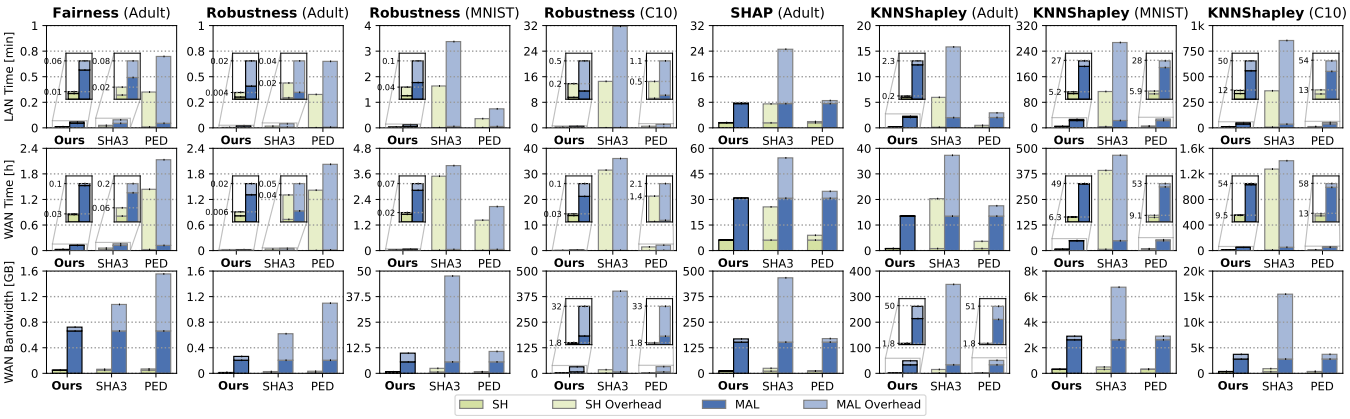

Figure 6: The overhead of Arc's consistency layer relative to the cost of the auditing function computation in MPC for four different auditing functions across our three scenarios.

significantly smaller operation than training, resulting in a larger relative overhead of our system. As shown in Fig. 5, the overhead is at least an order of magnitude slower than the inference itself. A significant fraction, 35-66%, is the result of the share conversion from the PPML protocol's computation domain $\mathbb{Z}_{2^{64}}$ to the scalar field domain $\mathbb{F}_{\text{BLS12-377}}$. 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 $\mathbb{F}_{\text{BLS12-377}}$ to 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.

The storage overhead for the client's ability to request an audit for their prediction is shown in Fig. 5. $\mathsf{PoC}_{\mathsf{PED}}$ has an overhead that is asymptotically linear by requiring a commitment for each system input. Concretely this overhead requires 1.5GB of storage for (W2:MNIST) and 5GB for (W3:CIFAR-10). Our approach and $\mathsf{PoC}_{\mathsf{SHA3}}$ are much more efficient because the only requires storing a constant-sized hash for the model and each data holder, resulting in only 720 bytes and 608 bytes, respectively, per prediction. Hence, the Pedersen commitment-based approach is infeasible to be used in practice due to its linear storage overhead for the client.

Auditing. In the auditing phase, parties must verify all inputs relevant to the audit which can include the prediction sample, the prediction, the model that made the prediction, and the original training data used to train the model. We present the wall time and bandwidth overhead for different auditing functions in Fig. 6. Across all settings, Arc significantly outperforms related approaches with a storage overhead comparative 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.

We also observe that the overhead of $\mathsf{PoC}_{\mathsf{PED}}$ gets closer to that of our approach by only taking two minutes longer for KNN-Shapley on the (W3:CIFAR-10) task. The overhead of $\mathsf{PoC}_{\mathsf{PED}}$ is significantly lower than in the previous phases because there are no outputs that need to be committed to for consistency. The reason for this is that here the cost of the MSM to compute the linear combination starts to dominate relative to the cost of the constant number of Pedersen commitments. An asymptotic difference with our protocol is that the verifiers have to perform MSM the size of the total input of all parties, rather than just for their input. However, in order for the other parties to compute the linear combination of the commitments requires them to store them individually, resulting in an impractical storage overhead (cf. Fig. 5). In addition, the overhead of computing the output commitments in the training and inference phases makes it unsuitable in practical deployments.

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

[2] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. QUOTIENT: Two-Party secure neural network training and prediction. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19, pages 1231–1247, New York, NY, USA, November 2019. Association for Computing Machinery.

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

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] 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.

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

[18] 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.

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

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

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

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

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

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

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

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

[27] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer

Reingold, and Rich Zemel. Fairness through awareness. April 2011.

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

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

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

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

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

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

[34] 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.

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

[36] 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.

[37] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[53] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting Out-of-Distribution samples. In ICLR, 2018.

[54] 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.

[55] Scott M Lundberg and Su-In Lee. A unified approach to

interpreting model predictions. In NeurIPS, volume 30. Curran Associates, Inc., 2017.

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

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

[58] 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.

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

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

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

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

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

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

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

[66] 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.

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

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

[69] 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.

[70] 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.

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

[72] 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.

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

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

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

[76] Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L Willke. Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In ECCV, 2018.

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

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

[79] 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.

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

[81] 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.

[82] 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.

A Definitions

Definition A.1 (Commitment Scheme)**.** A non-interactive commitment scheme consists of a message space $\mathcal{M}$, randomness space $\mathcal{R}$, a commitment space $\mathcal{C}$ and a tuple of polynomial-time algorithms $(\mathsf{COM.Setup}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ defined as follows:

- $\mathsf{COM.Setup}(1^\lambda) \to \mathsf{pp}$: Given a security parameter λ, it outputs public parameters pp.
- $\mathsf{COM.Commit}(\mathsf{pp}, m, r) \to c$: Given public parameters pp, a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$, it outputs a commitment c.
- $\mathsf{COM.Verify}(\mathsf{pp}, c, r, m) \to \{0,1\}$: Given public parameters pp, a commitment c, a decommitment r, and a message m, it outputs 1 if the commitment is valid, otherwise 0.

A non-interactive commitment scheme has the following properties:

- **Correctness.** For all security parameters λ, for all m and for all pp output by $\mathsf{COM.Setup}(1^\lambda)$, if $c = \mathsf{COM.Commit}(\mathsf{pp}, m, r)$, then $\mathsf{COM.Verify}(\mathsf{pp}, c, m, r) = 1$.
- **Binding.** For all polynomial-time adversaries $\mathcal{A}$, the probability

$$\Pr\big[\mathsf{COM.Verify}(\mathsf{pp}, c, m_1, r_1) = 1 \wedge \mathsf{COM.Verify}(\mathsf{pp}, c, m_2, r_2) = 1 \wedge m_1 \neq m_2 : \mathsf{pp} \leftarrow \mathsf{COM.Setup}(1^\lambda), (c, r_1, r_2, m_1, m_2) \leftarrow \mathcal{A}(\mathsf{pp})\big]$$

  is negligible.
- **Hiding.** For all polynomial-time adversaries $\mathcal{A}$, the advantage

$$\big|\Pr[\mathcal{A}(\mathsf{pp}, c) = 1 : c \leftarrow \mathsf{COM.Commit}(\mathsf{pp}, m_1, r)] - \Pr[\mathcal{A}(\mathsf{pp}, c) = 1 : c \leftarrow \mathsf{COM.Commit}(\mathsf{pp}, m_2, r)]\big|$$

  is negligible, for all messages m_1, m_2.

Definition A.2 (Polynomial Commitments [41])**.** Polynomial commitments enable a prover to commit to a polynomial in such a way that they can later reveal the polynomial's value at any particular point, with a proof that the revealed value is indeed correct. These commitments are notably used to construct succinct zero-knowledge proofs and verifiable computation protocols. A polynomial commitment scheme is a quadruple (PC.Setup, PC.Commit, PC.Prove, PC.Check) where

- $\mathsf{PC.Setup}(d) \to \mathtt{pp}$: prepares the public parameters given the maximum supported degree of polynomials d and outputting a common reference string $\mathtt{pp}$.
- $\mathsf{PC.Commit}(\mathtt{pp}, f, r) \to c$: computes a commitment c to a polynomial f, using randomness r.
- $\mathsf{PC.Prove}(\mathtt{pp}, c, f, r, x, y) \to \pi$: The prover computes a proof π using randomness r that c commits to f such that $f(x) = y$.
- $\mathsf{PC.Check}(\mathtt{pp}, c, x, y, \pi) \to \{0,1\}$: The verifier checks that c commits to f such that $f(x) = y$.

A polynomial commitment scheme is secure if it provides correctness, polynomial binding, evaluating binding, and hiding properties. We refer to [41] for a formal definition of these properties.

Definition A.3 (KZG Commitments [41])**.** KZG commitments leverage bilinear pairings to create a commitment scheme for polynomials where the commitments have constant size. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of prime order p such with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear pairing, so that $e(\alpha \cdot h_1, \beta \cdot h_2) = \alpha\beta \cdot e(h_1, h_2)$. The KZG polynomial commitment scheme for some polynomial f made up of coefficients f_i is defined by four algorithms:

- $\mathsf{PC.Setup}(d)$: Sample $\alpha \xleftarrow{\$} \mathbb{F}_p$ and output

$$\mathtt{pp} \leftarrow \left(\alpha \cdot g_1, \ldots, \alpha^d \cdot g_1, \alpha \cdot g_2\right)$$

- $\mathsf{PC.Commit}(\mathtt{pp}, f)$: Output $c = f(\alpha) \cdot g_1$, computed as

$$c \leftarrow \sum_{i=0}^{d} f_i \cdot (\alpha^i \cdot g_1)$$

- $\mathsf{PC.Prove}(\mathtt{pp}, c, f, x)$: Compute the remainder and quotient

$$q(X), r(X) \leftarrow (f(X) - f(x)) \,/\, (X - x)\,.$$

  Check that the remainder $r(X)$ and, if true, output $\pi = q(\alpha) \cdot g_1$, computed as $\sum_{i=0}^{d} \left(q_i \cdot (\alpha^i \cdot g_1)\right)$.
- $\mathsf{PC.Check}(\mathtt{pp}, c, x, y, \pi)$: Accept if the following pairing equation holds:

$$e(\pi, \alpha \cdot g_2 - x \cdot g_2) = e(c - y \cdot g_1, g_2)$$

The security properties of KZG commitments fundamentally rely on the hardness of the polynomial division problem. The parameter α acts as a trapdoor and must be discarded after `PC.Setup` to ensure the binding property. Hence, we require a trusted setup to generate the public parameters and securely discard α, which can be computed using MPC or, depending on the deployment, computed by the auditor acting as a trusted dealer. The hiding property relies on the discrete logarithm assumption, so if α is not discarded this breaks the binding property but not the hiding property. We refer to [41] for a detailed security analysis.

Definition A.4 (Homomorphic Commitment Scheme [11])**.** A homomorphic commitment scheme is a non-interactive commitment scheme such that $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{C}$ are all abelian groups and for all $m_1, m_2 \in \mathcal{M}$ and $r_1, r_2 \in \mathcal{R}$, we have

$$\mathsf{COM.Commit}(\mathsf{pp}, m_1 + m_2, r_1 + r_2) = \mathsf{COM.Commit}(\mathsf{pp}, m_1, r_1) + \mathsf{COM.Commit}(\mathsf{pp}, m_2, r_2).$$

KZG commitments are homomomorphic, i.e., if c_1 and c_2 are commitments to polynomials f_1 and f_2, then $c_1 + c_2$ is a commitment to polynomial $f_1 + f_2$.

Definition A.5 (Digital Signature Scheme)**.** A digital signature scheme consists of a tuple of polynomial-time algorithms $(\mathsf{SIG.Setup}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ defined as follows:

- $\mathsf{SIG.Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: Given a security parameter λ, it outputs a public key pk and a secret key sk.
- $\mathsf{SIG.Sign}(\mathsf{sk}, m) \to \sigma$: Given a secret key sk and a message m, it outputs a signature σ.
- $\mathsf{SIG.Verify}(\mathsf{pk}, m, \sigma) \to \{0,1\}$: Given a public key pk, a message m, and a signature σ, it outputs 1 if the signature is valid, otherwise 0.
- $\mathsf{SIG.DistSign}(\llbracket \mathsf{sk} \rrbracket, m) \to \llbracket \sigma \rrbracket$: Given a secret key share $\llbracket \mathsf{sk} \rrbracket$ and a message m, it outputs a signature share $\llbracket \sigma \rrbracket$.
- **Correctness.** For all m, and for $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Setup}(1^\lambda)$, if $\sigma = \mathsf{SIG.Sign}(\mathsf{sk}, m)$, then $\mathsf{SIG.Verify}(\mathsf{pk}, m, \sigma) = 1$.
- **Unforgeability.** For all polynomial-time adversaries $\mathcal{A}$,

$$\Pr\left[\mathsf{SIG.Verify}(\mathsf{pk}, m, \sigma) = 1 \;\middle|\; \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Setup}(1^\lambda) \\ (\sigma, m) \leftarrow \mathcal{A}(\mathsf{pk})\end{array}\right]$$

  is negligible, where $\mathcal{A}$ has not received σ from a prior invocation of $\mathsf{SIG.Sign}(\mathsf{sk}, m)$.

Definition A.6 (Proof-of-Training)**.** A valid Proof-of-Training is an interaction between a Prover protocol $\mathbb{P}$ and Verifier protocol $\mathbb{V}$. A public learning algorithm $\mathcal{T}$ (including hyperparameters), takes a training dataset D and training randomness J as input and outputs a model $M \leftarrow \mathcal{T}(D, J)$. A Proof-of-Training is defined as a set of algorithms (POT.Setup, POT.Prove, POT.Verify) where:

- $\mathsf{POT.Setup}(1^\lambda) \to \mathsf{pp}_{\mathsf{pot}}$: A setup algorithm that outputs the public parameters $\mathtt{pp}$.
- $\mathsf{POT.Prove}(\mathsf{pp}_{\mathsf{pot}}, c, D, J, M, r_D, r_J, r_M) \to \pi$: The prover generates a proof π that M is computed as $M \leftarrow \mathcal{T}(D, J)$, and c is a commitment to D, J, M under respective randomnesses r_D, r_J, r_M.
- $\mathsf{POT.Verify}(\mathsf{pp}_{\mathsf{pot}}, c, \pi) \to \{0,1\}$: The verifier accepts if the proof π is valid with respect to the commitment c to the data, the training randomness and the model.
- **Completeness.** For a security parameter λ, for all D, J, M, r_D, r_J, r_M, $\mathsf{pp}_{\mathsf{pot}} \leftarrow \mathsf{POT.Setup}(1^\lambda)$, $c \leftarrow \mathsf{PoC.Commit}(\mathsf{pp}_{\mathsf{pot}}, D, J, M, r_D, r_J, r_M)$, if $\pi \leftarrow \mathsf{POT.Prove}(\mathsf{pp}_{\mathsf{pot}}, c, D, J, M, r_D, r_J, r_M)$, then $\mathsf{POT.Verify}(\mathsf{pp}_{\mathsf{pot}}, c, \pi) = 1$.
- **Soundness.** The probability that any polynomial-time $\mathcal{A}$ outputs an accepting proof π and either M was not generated by $\mathcal{T}(D, J)$ or c is not a valid commitment is negligible.
- **Zero-Knowledge.** For every verifier $\mathbb{V}$ there exists a PPT simulator $\mathcal{S}$, which, when interacting with $\mathbb{V}$ and given inputs $\mathsf{pp}_{\mathsf{pot}}$, its corresponding simulation trapdoor, π, and c, produces a computationally indistinguishable view from an interaction with $\mathbb{P}$.

The definitions above straightforwardly extend to handle multiple training sets D_i with some minor syntactic changes.

Definition A.7 (Proof-of-Inference)**.** A valid Proof-of-Inference is an interaction between a Prover protocol $\mathbb{P}$ and Verifier protocol $\mathbb{V}$. A Proof-of-Inference is defined as a set of algorithms (POI.Setup, POI.Prove, POI.Verify) where:

- $\mathsf{POI.Setup}(1^\lambda) \to \mathsf{pp}_{\mathsf{poi}}$: A setup algorithm that outputs the public parameters $\mathtt{pp}$.
- $\pi_{\mathrm{I}} \leftarrow \mathsf{POI.Prove}(\mathsf{pp}_{\mathsf{poi}}, c_M, c_x, c_y; M, x, y, r_M, r_x, r_y)$: The prover generates a proof π that y is computed as $M(x)$, and c_M, c_x, c_y are commitments to M, x, y under respective randomnesses r_M, r_x, r_y.
- $\mathsf{POI.Verify}(\mathsf{pp}_{\mathsf{poi}}, c_M, c_x, c_y, \pi) \to \{0,1\}$: The verifier accepts if the proof π is valid with respect to the commitments to the model, the inference sample and the inference result.

A Proof-of-Inference satisfies **Completeness**, **Soundness**, **Zero-Knowledge**, which are defined as for Proof-of-Training.

B Security Proof for Auditing Protocol

In the following, we provide a proof that Π_{Arc} securely instantiates $\mathcal{F}_{\mathrm{Arc}}$. Our proof follows the real/ideal-world paradigm [13], which considers the following two worlds:

- **In the real world,** the parties run the Π_{Arc} protocol to establish an auditing framework for PPML. The adversary $\mathcal{A}$ can statically, actively, corrupt a subset of parties before the start of the protocol.
- **In the ideal world,** the honest parties send their inputs to the ideal functionality $\mathcal{F}_{\mathrm{Arc}}$, which executes the behavior of a secure auditing framework. The ideal world defines the ideal behavior of the functionality that the protocol aims to emulate.

A real-world protocol is secure if it manages to instantiate an ideal functionality in the ideal world. To show that a protocol

is secure, we must show that the adversary cannot distinguish between the real and the ideal world with high probability. We can do this by defining a non-uniform probabilistic polynomial-time simulator $\mathcal{S}$ that interacts with the adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\text{Arc}}$ in the ideal world in such a way that $\mathcal{A}$'s view is indistinguishable when interacting with the protocol in the real world.

We model generic MPC operations in our framework as $\mathcal{F}_{\text{ABB}}$ to ensure our framework composes with any MPC protocol and assume a secure randomness source $\mathcal{F}_{\text{RAND}}$. We also model our assumption of a public-key infrastructure with $\mathcal{F}_{\text{PKI}}$ that, upon initialization, sends each party its signing key sk and the list of verification keys pk_i for the other parties in the system. According to the sequential composition theorem, if a protocol securely computes a functionality in the $\mathcal{F}_{\mathsf{G}}$-hybrid model for some functionality $\mathcal{F}_{\mathsf{G}}$, then it remains secure when composed with a protocol that securely computes $\mathcal{F}_{\mathsf{G}}$ [13]. We use this model to abstract the dependencies of our framework. Additionally, we assume secure point-to-point communication channels and rely on an ideal broadcast channel $\mathcal{F}_{\text{BC}}$ to ensure all parties in the auditing phase receive the same commitments to achieve identifiable abort. Finally, we allow the simulator to equivocate commitments it sent to the adversary with $\mathcal{F}_{\text{CRS}}$which outputs the trapdoor.

We are now ready to prove the security of our overall protocol (see Fig. 2). We recall our threat model, in which at least one computing party in each (non-plaintext) phase and at least one of the input-sending or output-receiving parties must be honest. Without loss of generality, we assume a one-to-one mapping between roles and parties. For malicious parties, we already consider appropriate collusion, for honest parties it is straightforward to see that they learn only the union of their roles information and receive no other capabilities.

Theorem B.1 (Collaborative Auditing)**.** Given a set of $N_{\texttt{DH}}$ data holders $\texttt{DH}$, a set of $N_{\texttt{M}}$ model holders $\texttt{M}$, a set of $N_{\texttt{C}}$ clients $\texttt{C}$, a set of $N_{\texttt{TC}}$ training computers $\texttt{TC}$, a set of $N_{\texttt{IC}}$ inference computers $\texttt{IC}$, a set of $N_{\texttt{AC}}$ audit computers $\texttt{AC}$, an adversary $\mathcal{A}$ who controls a set $M_{\texttt{P}} = \{\texttt{P}_i : i \in \mathcal{C}\}$ where at least one of the input-sending or output-receiving parties is honest, i.e., $(\texttt{DH} \cup \texttt{M} \cup \texttt{C}) \setminus M_{\texttt{P}} \neq \varnothing$, and (unless in the plaintext training setting) one $\texttt{TC}$, and (unless in the plaintext inference setting) one $\texttt{IC}$, and one $\texttt{AC}$ is honest. there exists a PPT simulator $\mathcal{S}$ in the $(\mathcal{F}_{\text{ABB}}, \mathcal{F}_{\text{RAND}}, \mathcal{F}_{\text{PKI}}, \mathcal{F}_{\text{CRS}})$-hybrid model such that the distributions:

$$\left\{\texttt{Ideal}_{\Pi_{\text{Arc}},\mathcal{S}(\mathsf{aux}_{\mathcal{A}}),M_{\texttt{P}}\cup\{\texttt{P}_{\texttt{C}}\}}(\mathbf{D},\mathbf{x},\mathbf{a},\lambda)\right\}_{\mathbf{D},\mathbf{x},\mathbf{a},\mathsf{aux}_{\mathcal{A}},\lambda}$$
$$\approx$$
$$\left\{\texttt{Real}_{\mathcal{F}_{\text{Arc}},\mathcal{A}(\mathsf{aux}_{\mathcal{A}}),M_{\texttt{P}}\cup\{\texttt{P}_{\texttt{C}}\}}(\mathbf{D},\mathbf{x},\mathbf{a},\lambda)\right\}_{\mathbf{D},\mathbf{x},\mathbf{a},\mathsf{aux}_{\mathcal{A}},\lambda}$$

are computationally indistinguishable, where $\mathbf{D}$ is a list of training datasets for each data holder, $\mathbf{x}$ is a list of prediction feature vectors, $\mathbf{a}$ is a list of prediction feature vectors to audit, and $\mathsf{aux}_{\mathcal{A}} \in \{0,1\}^*$ is an auxiliary input by the adversary to capture malicious strategy.

Proof. We will define a simulator $\mathcal{S}$ through a series of subsequent modifications to the real execution, so that the views of $\mathcal{A}$ in any two subsequent executions are computationally indistinguishable. Without loss of generality, we assume that if the simulator receives inconsistent values from some of the parties that should be the same according to the real protocol for training and inference, the simulator aborts. Similarly, should any signature verification fail, we assume the simulator aborts. During training and inference, the simulator simply forwards any aborts to the ideal functionality, during auditing, more care must be taken to achieve identifiable abort. We highlight arguments that are only relevant for the plaintext training setting in olive, and for the plaintext inference setting in blue.

Hyb_1 The view of $\mathcal{A}$ in this hybrid is distributed exactly as the view of $\mathcal{A}$ in $\texttt{Real}$.

Hyb_2 In this hybrid, the real execution is emulated by a simulator that knows the real inputs of the honest parties D_i for $i \notin \mathcal{C}$ and runs a full execution of the protocol with $\mathcal{A}$, which includes emulating the ideal interactions for training and inference and the auditing interactions. The view of the adversary in this hybrid is the same as the previous one.

Hyb_3 In this hybrid, we replace the commitments to the inputs in Step T.1 (training input phase) with dummy data. unless we are in the case of plaintext training, where $\mathcal{S}$ can simply forward the honest inputs sent by the ideal functionality to the $\mathcal{A}$. The simulator generates to all-zero training sets and associated commitments and decommitments for any honest $\texttt{DH}_i$ and uses these as inputs for $\mathcal{F}_{\text{ABB}}$ and PoC.Check. The view of the adversary in this hybrid is the same as the previous one because of the hiding property of the commitments and the zero-knowledge property of PoC.Check.

Hyb_4 In this hybrid, the simulator replaces the Step T.2 (model computation) with the result from the ideal functionality. In the non-plaintext setting, $\mathcal{S}$ can extract the training set D_i for any potential malicious $\texttt{DH}_i$ via $\mathcal{F}_{\text{ABB}}$ and input these to $\mathcal{F}_{\text{Arc}}$ to receive id_M. If $\mathcal{A}$ controls a $\texttt{M}$, $\mathcal{S}$ will also receive M and can forward it to $\mathcal{A}$. If the adversary is not involved in training at all, $\mathcal{S}$ still receives id_M from the ideal functionality. $\mathcal{S}$ knows the commitment randomness from $\mathcal{F}_{\text{ABB}}$ (or $\mathcal{F}_{\text{RAND}}$) and can use this to either commit to the training randomness and either the actual model (if known) or an all-zero dummy model. $\mathcal{S}$ stores the tuple (id_M, c) internally, communicates c to the adversary (corresponding to the out-of-band communication in the real-world), and then emulates the last three steps of Step T.2 with the above commitments and data.

In the plaintext setting, if there is no honest training computer, $\mathcal{S}$ needs to emulate the protocol and extract the model and (malicious) training data from π_{T}, which it

can then use as inputs for the ideal functionality. Alternatively, if there is an honest training computer, $\mathcal{S}$ receives the malicious (D_i, r_{D_i}) directly from $\mathcal{A}$ and can input these to $\mathcal{F}_{\text{Arc}}$ to receive id_M. Note that, for any potential dishonest training computers, $\mathcal{S}$ will receive the honest inputs from the ideal functionality, which it is free to forward in this setting. If there are any malicious model holders or training computers, $\mathcal{S}$ will also receive M, generate a commitment to the model c and store the tuple (id_M, c) internally. It communicates c to the adversary (corresponding to the out-of-band communication in the real-world). It can then emulate the remaining steps of Step T.2, unless the only malicious parties are the input parties (which are not involved in the remainder of Step T.2).

The replacement of Step T.3 and Step T.4 follow trivially from the security of the underlying signature schemes and commitments.

Hyb_5 In this hybrid, we adapt Step I.1 (inference input) between the simulator and the adversary. As $\mathcal{F}_{\text{Arc}}$ only allows audits of predictions that were actually made in the inference phase, $\mathcal{S}$ has to ensure that the internal state of $\mathcal{F}_{\text{Arc}}$ matches with that of the real protocol. However, as there might not always be an honest party present in each stage of the protocol, the adversary can, in some scenarios, generate valid predictions locally. We consider the four different scenarios that $\mathcal{S}$ must handle:

- **Honest client and model holder:** In this case, the (honest) request must be for a previously trained model, and we need to consider the case where at least one inference computer is malicious. In the non-plaintext setting, $\mathcal{S}$ can either input and commit to an all-zero model or re-use a potential existing commitment to an all-zero model (if the adversary was involved in the training for this model id). In the plaintext inference setting, the simulator always learns the actual model. Note that the adversary might already hold a commitment that should correspond to this model, but instead is a commitment to an all-zero model. Using the trapdoor from $\mathcal{F}_{\text{CRS}}$, $\mathcal{S}$ can use equivocation to arrive at a decomitmment randomness that matches the commitment with the actual model.
- **Corrupt client and honest model holder:** Either, $\mathcal{S}$ x and c_M from $\mathcal{A}$, or (if $\mathcal{A}$ controls all inference computer and skips Step I.1), $\mathcal{S}$ receives a request for c_M. The simulator finds the model information (id_M, c) from its internal storage corresponding to c_M and aborts if it cannot find it. Note that, in the real protocol, an honest model holder also aborts if it does not have a corresponding model, and $\mathcal{S}$ internal storage captures the view of an honest model holder. $\mathcal{S}$ then forwards $(\mathtt{M}_k, \mathsf{id}_M, x)$ to $\mathcal{F}_{\text{Arc}}$ to receive y which it uses to simulate $\mathcal{F}_{\text{ABB}}$. In the plaintext setting: $\mathcal{F}_{\text{Arc}}$ sends the model to $\mathcal{S}$ and uses this for the inference computers. The view is indistinguishable from the previous hybrid, because the simulator uses the prediction from the real model y from $\mathcal{F}_{\text{Arc}}$, and because the $\mathcal{F}_{\text{Arc}}$ will only abort if $\mathtt{C}$ sent a c_M that does not exist which corresponds to the behavior in the real protocol.
- **Honest client and corrupt model holder:** The simulator receives M, c from $\mathcal{A}$. $\mathcal{S}$ verifies that all training signatures σ_{T} and the training computer signature σ_{TC} are valid signatures with respect to the commitments and otherwise aborts $\mathcal{F}_{\text{Arc}}$. $\mathcal{S}$ also receives id_M from $\mathcal{F}_{\text{Arc}}$ and finds the model information (id_M, c) from its internal storage corresponding to c_M. The validity of σ_{TC} guarantees (id_M, c) exists in internal storage, because there is always at least one honest training computer. In the plaintext training setting, the $\mathcal{A}$ could have sent a different model M that does not correspond to id_M, because it has locally generated additional training runs. Therefore, $\mathcal{S}$ now has to make sure that $\mathcal{F}_{\text{Arc}}$ has the necessary internal state to send the right prediction to the honest client. $\mathcal{F}_{\text{Arc}}$ will ask in Step 2 for a model and training data from $\mathcal{S}$. $\mathcal{S}$ responds with the training datasets and the model using the extractor guaranteed by knowledge-soundness of the proof of training π_{T}. The adjustments for plaintext inference are the same as in previous cases.
- **Corrupt client and corrupt model holder:** In the non-plaintext setting, the simulator receives both the model and the sample from $\mathcal{A}$ through $\mathcal{F}_{\text{ABB}}$ which is sufficient to emulate the protocol. Note that, in the case of plaintext inference, the $\mathcal{A}$ can generate predictions locally using inference computer, which does not influence this Hybrid, but will become relevant later.

Hyb_6 In this hybrid, we adapt Step I.2, I.3 and I.4 (inference computation) between the simulator and the adversary. In the non-plaintext setting, as $\mathcal{S}$ can simulate the operations through $\mathcal{F}_{\text{ABB}}$, we only discuss what happens when the inference computers inputs or opens values. If the client is malicious, the simulator uses the y received from $\mathcal{F}_{\text{Arc}}$ to generate and open the commitments c_x and c_y. Otherwise, $\mathcal{S}$ uses dummy inputs for x and y to generate commitments. The view in this scenario is indistinguishable from the view in the previous hybrid, because of the hiding property of the commitments. Note that $\mathcal{S}$ can access the commitment randomness r_x, r_y through simulating $\mathcal{F}_{\text{ABB}}.\mathsf{RAND}$. If $\mathcal{A}$ controls the model holder, $\mathcal{S}$ receives id_M from $\mathcal{F}_{\text{Arc}}$ and sends $(\sigma_{\text{I}}, c'_x, c'_y)$ to $\mathcal{A}$, where c'_x, c'_y are two randomly sampled group elements. In the case of plaintext inference, M, x and y are leaked to the $\mathcal{A}$ if it controls any inference computer and, as a result, the simulator receives M, x and y from $\mathcal{F}_{\text{Arc}}$, and can follow the protocol honestly on the inputs from the real protocol. If the adversary only controls the client or the model holder, the proof proceeds nearly

identically to the non-plaintext case. The replacement of l.3 and l.4 follow trivially from the security of the underlying signature schemes and commitments.

Hyb_7 In this hybrid, we adapt the local checks of the audit verification phase in Step A.1 and A.2 to ensure $\mathcal{F}_{\text{Arc}}$ receives the proper abort signal. A malicious client sends $(c, c_x, c_y, \sigma_{\text{I}}, \sigma_{\text{T}}, \sigma_{\text{TC}}$ or $\pi_{\text{T}}, \sigma_{\text{IC}}$ or $\pi_{\text{I}}, \mathsf{pk}_{\text{M}_k}, f_{\texttt{audit}}, \mathsf{aux})$ to $\mathcal{S}$ simulating $\mathcal{F}_{\text{BC}}$. The simulator checks that pk_{M} is a valid identity from $\mathcal{F}_{\text{PKI}}$, that the signatures $\sigma_{\text{T}}, \sigma_{\text{I}}$ are valid, that σ_{TC} or π_{T} is valid, that σ_{IC} or π_{I} is valid, and that $f_{\texttt{audit}} \in F_{\texttt{audit}}$. If the checks do not pass, $\mathcal{S}$ aborts $\mathcal{F}_{\text{Arc}}$ with $(\texttt{Abort}, \texttt{C})$. Otherwise, if a party $\texttt{P}$ controlled by $\mathcal{A}$ aborts, $\mathcal{S}$ forwards $(\texttt{Abort}, \texttt{P})$ to $\mathcal{F}_{\text{Arc}}$. This strategy for $\mathcal{S}$ is valid because the broadcast channel allows the simulator to verify whether the corrupted audit computers have aborted rightfully or not. If the corrupted audit computer aborts while the simulator's checks pass, the party must be malicious. Hence, the view is indistinguishable from the previous hybrid.

Hyb_8 In this hybrid, we adapt the PoC of the client (Step A.3), model holder (Step A.4) and data holders (Step A.5). Each corrupted party $\texttt{P}$ sends its inputs to $\mathcal{F}_{\text{ABB}\,[\text{ID}]}$ and broadcasts a proof of consistency to $\mathcal{S}$ through $\mathcal{F}_{\text{BC}}$ as part of $\mathsf{PoC.Check}_{[\text{ID}]}$. $\mathcal{S}$ checks that the proof is valid with respect to the commitments c and, if not, aborts $\mathcal{F}_{\text{Arc}}$ with $(\texttt{Abort}, \texttt{P})$. The view is indistinguishable from the previous hybrid, because $\mathcal{F}_{\text{Arc}}$ only sends identifiable aborts to audit computer. There, this behavior is consistent with the real protocol.

Hyb_9 In this hybrid, we adapt the simulator to use dummy inputs in Steps A.3, A.4 and A.5. $\mathcal{S}$ uses all zeroes for all honest audit inputs as input to $\mathsf{PoC.Check}_{[\text{ID}]}$in all steps. The view is indistinguishable from the previous hybrid, due to the zero-knowledge property of PoC. Note that the simulator still uses the output provided by the real parties in Step A.6, which we address in Hybrid 10.

Hyb_{10} In this hybrid, we replace the output of the auditing function computation in Step A.6 with the output of $\mathcal{F}_{\text{Arc}}$. We begin discussing the setting where neither training nor inference were in the plaintext setting. In case client is honest, the $\mathcal{F}_{\text{Arc}}$ responds with the correct result and $\mathcal{A}$ receives no output. In case the client is dishonest, $\mathcal{S}$ has to ensure $\mathcal{A}$ receives the correct output. The simulator finds the model information (id_M, c) from its internal storage corresponding to c_M. $\mathcal{S}$ forwards $(\texttt{Audit}, \texttt{M}_j, \mathsf{id}_M, f_{\texttt{audit}}, \tilde{x}, \tilde{y}, \mathsf{aux})$ to $\mathcal{F}_{\text{Arc}}$ to receive the real audit function output o. We now discuss why id_M exists in $\mathcal{S}$'s internal storage. Due to the validity of σ_{TC}, $\mathcal{S}$ is guaranteed to find id_M, because at least one of the inference computers is honest. In addition, $\mathcal{F}_{\text{Arc}}$ will not abort because it has internally stored the $(\texttt{M}_k, \mathsf{id}_M, x, y)$ because of the validity of σ_{IC}, which implies that the simulator has been involved in the prediction through an honest inference computer. However, in the plaintext setting, two additional cases may occur:

- **In the case of plaintext training and plaintext inference:** $\mathcal{S}$ may not find a commitment while all proofs and signatures are valid, because $\mathcal{A}$ may have locally computed an extra training and inference. In this case, $\mathcal{S}$ can extract all the auditing inputs from the proof of training π_{T} and the proof of inference π_{I}. If either π_{T} or π_{I} is invalid, the simulator can abort the malicious client and $\mathcal{F}_{\text{Arc}}$, because $\mathcal{F}_{\text{Arc}}$ only allows predictions from models that resulted from training on the data holders' datasets. If both proofs are valid and $\mathcal{S}$ can successfully extract the auditing inputs from the proofs, $\mathcal{S}$ can then use these values to compute o locally (without $\mathcal{F}_{\text{Arc}}$), as no honest parties need to send inputs or receive output. In the case of an abort by the audit computers, $\mathcal{S}$ simulates an abort at $\mathcal{F}_{\text{Arc}}$ by sending an audit request with $f_{\texttt{audit}} \notin F_{\texttt{audit}}$.
- **In the case of plaintext inference:** $\mathcal{F}_{\text{Arc}}$ may not have the requested prediction stored in its internal state. In this case, $\mathcal{S}$ can still forward the audit request from the malicious client because the functionality does not require to have issued an inference in the case that the $\mathcal{A}$ controls the model holder and all inference computer (cf. Step 3). If π_{I} is invalid, the simulator can abort the malicious client and $\mathcal{F}_{\text{Arc}}$, because $\mathcal{F}_{\text{Arc}}$ only allows valid predictions from models.

  The view is indistinguishable from the previous hybrid, because it uses the honest output from the ideal functionality.

Hyb_{11} This hybrid is defined as the previous one, with the only difference being that the simulator now does not receive the inputs of the honest parties. Because the simulator no longer relied on receiving inputs from the honest parties, the view of the adversary is perfectly indistinguishable from the previous hybrid.

□

C Consistency Check

In the following, we first proof that our approach (Π_{cc}) fulfills the requirements for a Proof-of-Consistency (Definition 4.1) and provide formal definitions for the strawman constructions discussed in §4.2. In addition, we briefly discuss how to efficiently realize $\mathcal{F}_{\text{ABB}}^{[\text{EC}]}$ for Pedersen Vector Commitments.

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

Proof. The protocol Π_{cc} in Protocol 4.1 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 [25], it holds that:

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

Thus, if p is larger than $d \cdot 2^\lambda$, 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.

□

C.1 Direct Commitment

Protocol C.1 (Strawman 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 $(\mathsf{COM.Setup}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ be a commitment scheme as in Definition A.1. In order to improve the readability, we slightly abuse notation by allowing vector inputs to commitments, with the understanding that, where the commitment scheme does not accept vectors, we concatenate the elements in their natural order and, where necessary, apply a hash to match the input domain of the commitments. The protocol CC1 works as follows:

- $\mathsf{CC1.Setup}(1^\lambda, d) \to \mathsf{pp}$: Run $\mathsf{pp} \leftarrow \mathsf{COM.Setup}()$ where d is unused.
- $\mathsf{CC1.Commit}(\mathsf{pp}, \mathbf{x}, r) \to c$: The prover computes a commitment $c \leftarrow \mathsf{COM.Commit}(\mathtt{pp}, \mathbf{x}_1, r_1)$ and outputs c. The prover outputs c.
- $\mathsf{CC1.Check}(\mathsf{pp}, c, [\![\mathbf{x}]\!]; \mathbf{x}, r) \to \{0,1\}$: The protocol proceeds as follows:
  1. $\mathbb{P}$ inputs the randomness r to $\mathcal{F}_{\text{ABB}}$.
  2. The parties invoke $\mathcal{F}_{\text{ABB}}$ to compute $\mathsf{COM.Verify}(\mathsf{pp}, [\![c']\!], [\![\mathbf{x}]\!], [\![r]\!])$ and output the result.

C.2 Homomorphic Commitments

The following is a formalization of Cerebro's security check. In this protocol, we make explicit that the input randomness $\mathbf{r}$ and the output $\mathbf{c}$ of COM.Commit are lists of randomness and commitments, respectively, by highlighting them in bold.

Protocol C.2 (Cerebro's Consistency Check [81])**.** 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 $(\mathsf{COM.Setup}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ be a commitment scheme as in Definition A.1 that is also homomorphic (Definition A.4). The protocol CC2 works as follows:

- $\mathsf{CC2.Setup}(1^\lambda, d) \to \mathsf{pp}$: Run $\mathsf{pp} \leftarrow \mathsf{COM.Setup}()$ where d is unused.
- $\mathsf{CC2.Commit}(\mathsf{pp}, \mathbf{x}, \mathbf{r}) \to \mathbf{c}$: The prover computes a list of d commitments $\mathbf{c} \leftarrow \{\mathsf{COM.Commit}(\mathtt{pp}, x_1, r_1), \ldots, \mathsf{COM.Commit}(\mathtt{pp}, x_d, r_d)\}$. The prover outputs c.
- $\mathsf{CC2.Check}(\mathsf{pp}, \mathbf{c}, [\![\mathbf{x}]\!]; \mathbf{x}, \mathbf{r}) \to \{0,1\}$: The protocol proceeds as follows:
  1. $\mathbb{P}$ inputs the randomness $\mathbf{r}$ to $\mathcal{F}_{\text{ABB}}$.
  2. The parties invoke $\mathcal{F}_{\text{RAND}}$ to obtain a random challenge $\beta \xleftarrow{\$} \mathbb{F}_p$.
  3. The parties invoke $\mathcal{F}_{\text{ABB}}$ to compute $[\![\tilde{x}]\!] := \sum_{i=1}^{d} [\![x_i]\!] \cdot \beta^i$ and $[\![\tilde{r}]\!] := \sum_{i=1}^{d} [\![r_i]\!] \cdot \beta^i$
  4. The parties invoke $\mathcal{F}_{\text{ABB}}$ to compute $[\![c']\!] \leftarrow \mathsf{COM.Commit}(\mathsf{pp}, [\![\tilde{x}]\!], [\![\tilde{r}]\!])$ and open c'.
  5. Each verifier computes $\tilde{c} = \sum_{i=1}^{d} \beta^i \cdot c_i$ and checks that $\tilde{c} = c'$. If verification passes, they output 1, otherwise 0.

C.3 Efficient Vector Comm. via EC-MPC

Definition C.1 (Pedersen Vector Commitments)**.** A Pedersen Vector Commitment for a vector of size d in a cyclic group $\mathbb{G}$ of prime order p with generators $g_1,\dots,g_d$ and h consists of the following algorithms:

- $\mathsf{PED.Setup}(1^\lambda, d) \to \mathsf{pp}$: Given a security parameter λ, outputs $\mathbb{G}$, its prime order p, and generators $g_0,\dots,g_d$.
- $\mathsf{PED.Commit}(\mathsf{pp},(m_1,\dots,m_d),r) \to c = r\cdot g_0 + \sum_{i=1}^d m_i \cdot g_i$.
- $\mathsf{PED.VerCommit}(\mathsf{pp},c,(m_1,\dots,m_d),r) \to \{0,1\}$: The verification checks that $c \stackrel{?}{=} r\cdot g_0 + \sum_{i=1}^d m_i\cdot g_i$.
- $\mathsf{PED.DistCommit}(\mathsf{pp},([\mathbf{x}_1],\dots,[\mathbf{x}_d]),[r])$: Output c by opening $[c]$, where party i computes $[c]_i$ as:

$$[c]_i \leftarrow ([r]_i \cdot h) \sum_{j=1}^{d} [\mathbf{x}_j]_i \cdot g_{j-1}$$

Pedersen Vector Commitments are computationally binding and information-theoretically hiding [11]. For the binding property to hold, it is important that the *trapdoors* $\tau_1,\dots,\tau_d$ in $h = g_i^{\tau_i}$ are unknown.

A straightforward secure protocol to commit to a vector in a distributed fashion is to evaluate an arithmetic circuit emulating $\mathsf{PED.Commit}$ using an MPC protocol. To achieve better efficiency, we instead rely on an established technique, where the parties compute a circuit over the group $\mathbb{G}$ directly, thereby preventing overheads stemming from emulating group operations inside another algebraic structure [62, 73].

Corollary C.1. Let Π be a secure protocol which computes a Pedersen Vector Commitment by evaluating an arithmetic circuit to emulate $\mathsf{PED.Commit}$. Let Π' be the same protocol, except that the commitment is computed using $\mathsf{PED.DistCommit}$. Then Π' is secure.

Proof. We make use of Lemma 4 from [62], which states that Π' is secure if the difference between Π' and Π is a sub-circuit C' computing input wires to output wires according to a linear function f. In our case, we consider the function $C' : \mathbb{F}_p^{d+1} \leftarrow \mathbb{G}$ that maps a decommitment value and a vector of inputs to a Pedersen Vector commitment, where the generators $g_1,\dots,g_d$ and h are public parameters. For ease of exposition, we set $g_0 = h$. Observe that C' is linear, for two vectors of inputs $\mathbf{x}^{(1)},\mathbf{x}^{(2)} \in \mathbb{F}_p^{d+1}$ and two scalars $a,b \in \mathbb{F}_p$, $C'(a\cdot\mathbf{x}^{(1)} + b\cdot \mathbf{x}^{(2)}) = \sum_{i=0}^d (a\cdot \mathbf{x}_i^{(1)} + b\cdot \mathbf{x}_i^{(2)})\cdot g_j = a\cdot\sum_{i=0}^d(\mathbf{x}_i^{(1)}\cdot g_j) + b\cdot \sum_{i=0}^d(\mathbf{x}_i^{(2)})\cdot g_j) = a\cdot C'(\mathbf{x}^{(1)}) + b\cdot C'(\mathbf{x}^{(2)})$. Hence, Lemma 4 implies that Π' is secure. □

D Share Conversion

Our protocols require the computation domain of the MPC protocol to be the scalar field of the elliptic curve. However, this is not the most efficient computation domain for MPC, which often operates over smaller fields or rings [5, 23, 43, 48]. Fortunately, we can convert a secret value x in one arithmetic domain $\mathbb{Z}_M$ to another arithmetic domain $\mathbb{Z}_{M'}$. We do this by decomposing $[x]_{\mathbb{Z}_M}$ into ℓ bits and recomposing the bits in $\mathbb{Z}_{M'}$ where $\ell \ll M, M'$.

$$([x_{\ell-1}]_{\mathbb{Z}_2},\dots,[x_0]_{\mathbb{Z}_2}) \leftarrow \mathtt{bitdec}_{\mathbb{Z}_M}([x]_{\mathbb{Z}_M})$$
$$[x]_{\mathbb{Z}_{M'}} \leftarrow \mathtt{bitcom}_{\mathbb{Z}_{M'}}([x_{\ell-1}]_{\mathbb{Z}_2},\dots,[x_0]_{\mathbb{Z}_2})$$

Converting between arithmetic domains and binary domains is a common technique in advanced MPC implementations to facilitate more efficient computation of non-linear functionality [14, 28, 59, 66]. We can use these techniques to instantiate `bitdec` and `bitcom`. If $\mathbb{Z}_M$ is a field $\mathbb{F}$ such that $2^{\ell+\kappa} < |\mathbb{F}|$ where $x \in [0,2^\ell]$ and κ is the security parameter. Let r be a random value such that $r = \sum_{i=0}^{\ell+\kappa} r_i \cdot 2^i \in [0, 2^{\ell+\kappa}]$ and let $[r]_{\mathbb{F}}$ be the sharing of r in $\mathbb{F}$ and let $(r_{m-1},\dots,r_0) \in \mathbb{Z}_2^\ell$ be the sharing of bits of r. r is typically generated in an offline phase, for instance using extended doubly authenticated bits (edaBits) [28]. The parties can evaluate $\mathtt{bitdec}(x)$ by locally computing $[\varepsilon]_{\mathbb{Z}_M} \leftarrow [x]_{\mathbb{Z}_M} - [r]_{\mathbb{Z}_M}$ and opening ε. Note that ε statistically hides x because the statistical distance between the distributions of ε and r is negligible in κ. The shares of the bits $([x_{\ell-1}]_{\mathbb{Z}_2},\dots,[x_0]_{\mathbb{Z}_2})$ can be computed by adding $([r_{\ell-1}]_{\mathbb{Z}_2},\dots,[r_0]_{\mathbb{Z}_2})$ to $(\varepsilon_{\ell-1},\dots,\varepsilon_0)$ using a binary adder. The parties can compose the bits in $\mathbb{Z}_{M'}$ using a second random value r' secret-shared in $\mathbb{Z}_{M'}$ as $[r']_{\mathbb{Z}_{M'}}$ with secret-shared bits $([r'_{\ell-1}]_{\mathbb{Z}_2},\dots,[r'_0]_{\mathbb{Z}_2})$. Parties compute the masked bits $[\varepsilon'_i]_{\mathbb{Z}_2}$ by adding $[\varepsilon_i]_{\mathbb{Z}_2} + [r'_i]_{\mathbb{Z}_2}$ using a binary adder. They then open the bits to construct $\varepsilon' = \sum_{i=0}^{\ell-1}\varepsilon'_i\cdot 2^i$ and get $[x]_{\mathbb{Z}_{M'}}$ by locally computing $\varepsilon' - [r']_{\mathbb{Z}_{M'}}$.

This process performs a logical conversion for a secret $x \in [0,2^\ell]$, but we must take extra care when performing an arithmetic conversion that also supports negative numbers, i.e., $x \in [-2^{\ell-1}, 2^\ell]$. These numbers may take more than ℓ bits to represent in binary, which requires additional steps to convert between the two domains. Fortunately, we can solve this by shifting the number up by $2^{\ell-1}$ and shifting it back after conversion, because this ensures that the number is represented in ℓ bits. The main overhead of the protocol comes from computing the binary addition circuit, which requires at most $(\ell + \log n)\cdot(n-1)$ AND gates for n parties, in addition to the cost of generating the edaBits, which can be computed in a preprocessing phase. The previous method works for general secret-sharing-based MPC protocols, but some protocols allow a more efficient technique when $\mathbb{Z}_M$ is a ring [23, 59]. This technique is called share splitting and allows much more efficient computation of the bit decomposition by utilizing the additional structure of the $\mathbb{Z}_M$ compared to prime fields.

	T	M	I
Data Validation			
Input Checks [17, 56]	●	○	○
Sample Attribution [32, 37, 44, 47, 71]	●	●	●
Party attribution [57]	●	●	●
Model Validation			
Validation Sets [19]	○	●	○
Feature Attribution [35, 55, 65]	◐	●	●
Certification [38, 45, 69]	○	●	●
Process Validation			
Algorithm Verific. [30, 36, 40, 74]	●	●	○
Constraint Verific. [70]	●	●	○

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.

E Auditing Functions

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 overview).

We focus on algorithms relevant to key properties in auditing, such as fairness, safety, and accountability. Our selection of concrete algorithms is influenced by candidate algorithms that can be efficiently realized with secure computation, and where relevant, we discuss our optimizations for efficient realization using secure computation. We start first with a brief discussion of validation-based audits and then dedicate most of our discussion to function-based audits and how Arc lifts these techniques to secure computation settings. Below we highlight secret values in yellow.

E.1 Robustness & Fairness

Machine learning models remain brittle in the face of real-world complexity. The literature on adversarial examples shows that for many models even slight perturbations in the input space are sufficient to manipulate the prediction of the model. Consequently, the community has devised a range of techniques to show that a model is robust against these types of attacks.

In particular, in the PPML setting, a model holder may promise that its model provides predictions against such adversarial examples. However, as the client only receives a prediction output from inference, it may wonder if the robustness claim actually holds for their prediction sample and request an audit. In this scenario, randomized smoothing offers a method to certify pointwise robustness through an efficient approach based on Monte Carlo sampling [21]. In randomized smoothing, we randomly sample a set of perturbed inputs around $\tilde{x}$ and check whether the model is invariant to these perturbations with high probability.

In Arc, we adapt the algorithm proposed by Jovanovic et al. [38] for FHE to our MPC setting. In the context of auditing, we can simplify their formulation because the prediction $\tilde{y}$ is already known. As a result, we only need to check whether the prediction is indeed locally robust (or fair) in the ℓ_2 ball of radius R around the input $\tilde{x}$. The algorithm samples n perturbed inputs around the input $\tilde{x}$ by adding Gaussian noise and obtaining predictions for these samples. Finally, a statistical check is conducted to assess whether the obtained prediction $\tilde{y}$ remains invariant to these perturbations with high probability. The output of the auditing function is a boolean value that indicates 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 [27, 67, 80]. Jovanovic et al. [38] show that it is sufficient to change the sampling procedure to implement a probabilistic check for individual fairness with confidence $1-\alpha$. The function $f_{\texttt{Fairness}}$ executes the same steps as $f_{\texttt{Robustness}}$, but changes the definition of the similarity constraint.

E.2 Accountability

We consider two flavors of accountability: sample attribution and party attribution. The former identifies the influence of individual data samples on a prediction and can be invaluable for debugging, while the latter attributes responsibility to a data holder and thus is considerably less privacy sensitive.

Sample-level Attribution. Various methods exist to identify the impact of individual data samples on a model but not all of them are equally amendable to secure computation. For instance, influence functions [47] require substantial computational resources, as they rely on inverting the Hessian matrix of the loss function which is infeasible under secure computation. In Arc, we propose using an alternative approach, leveraging KNN-Shapley values [37], which is well-suited for secure computation. The $f_{\texttt{KNN-Shapley}}$ function first computes the latent representations of both the training data and the prediction $\tilde{x}$. While there are several methods to acquire a latent representation, a widely used approach involves extracting the values from the last layer before the output layer of the model. Next, we compute the L_2 distance between the latent representation of $\tilde{x}$ and all training samples. Subsequently, we sort the training samples in ascending order based on their distances to $\tilde{x}$ and recursively compute the KNN-Shapley values. The only auxiliary input parameter required by the function is the number of neighbors K. In total, the algorithm requires $|D|+1$ inferences to extract the latent space representation of both the training data and the prediction $\tilde{x}$, along with $|D|$ distance computations. In addition, we need to sort a list of $|D|$ values and perform $|D|$ label comparisons to construct the indicator vector Z. The recursive Shapley value computation

only requires additions between secret values. All necessary multiplications are by a public value.

Party-level Attribution. While sample-level attribution proves valuable when the exact sample or data point is the subject of the audit (i.e., data poisoning), it inherently reveals more information about the investigated data. In many scenarios relevant to accountability, all that is required is to identify which party provided the problematic dataset for accountability. To support these cases, our system supports party-level attribution, which reveals no additional information about the dataset [57]. The key idea of $f_{\texttt{Camel-Unlearn}}$ is that if a suspicious prediction $(\tilde{x}, \tilde{y})$ was (at least partially) the result of data provided by a data holder, then excluding that party's data will lead to the absence (or weakening) of the suspicious prediction. This approach approximate the leave-out models by unlearning the data of a party from the original model M. To unlearn the data of party i we can use an efficient unlearning technique [71], in which we replace the labels of the party's data points with a uniform probability vector, representing the output of the model when it is uncertain about its prediction [53, 76]. Already after a few epochs of training, the loss on the unlearned models M_{-i} for prediction $(\tilde{x}, \tilde{y})$ is sufficient for outlier detection. For each influence score, we compute the Median-Absolute-Deviation (MAD), which is a robust measure of dispersion. Parties with a MAD score surpassing the threshold τ are flagged as potentially malicious. The algorithm requires a total of $N \cdot E$ epochs of training, with E representing the number of epochs needed to unlearn a party's data. Following this, we have N models to compute the loss on the suspicious sample. Finally, we calculate the MAD score and identify outliers by comparing this score to the threshold τ.

E.3 Explainability

A wide range of methods has been proposed to explain the predictions of complex models [55, 65, 72]. However, there exists a tension between the need for explainability and the imperative to protect the privacy of both the training data and the model. Ideally, one would aim to explain only the model's prediction, denoted as $(\tilde{x}, \tilde{y})$, without necessitating the exposure of the entire model. With this in mind, we identify additive feature attribution methods as particularly suitable for auditing when privacy is a concern, thus we incorporate it into Arc. These methods fall into the category of post-hoc, model-agnostic, local explanations and highlight which features of $\tilde{x}$ are most influential for the prediction $\tilde{y}$, even for complex ML models [55]. They achieve this by approximating the target model's behavior locally, around a specific prediction, with a simple and explainable model, typically a linear model.

The client receives the feature attributions ϕ_i as the explanation, which allows them to identify which features of $\tilde{x}$ are responsible for the prediction $\tilde{y}$. In Arc, we have integrated the auditing function $f_{\texttt{Kernel-SHAP}}$ tailored for tabular datasets. This function leverages KernelSHAP [55], which uses specialized weighting in the loss function. This ensures that the feature attributions ϕ_i correspond to the Shapley values of the respective features.