Reaching Data Confidentiality and Model Accountability on the CalTrain

Zhongshu Gu*, Hani Jamjoom*, Dong Su*, Heqing Huang‡,
Jialong Zhang‡, Tengfei Ma*, Dimitrios Pendarakis†, Ian Molloy*
*IBM Research, †IBM Cognitive Systems, ‡ByteDance

***Abstract*—Distributed collaborative learning (DCL) paradigms enable building joint machine learning models from distrusted multi-party participants. Data confidentiality is guaranteed by retaining private training data on each participant's local infrastructure. However, this approach makes today's DCL design fundamentally vulnerable to data poisoning and backdoor attacks. It limits DCL's model accountability, which is key to backtracking problematic training data instances and their responsible contributors. In this paper, we introduce CALTRAIN, a centralized collaborative learning system that simultaneously achieves data confidentiality and model accountability. CALTRAIN enforces isolated computation via secure enclaves on centrally aggregated training data to guarantee data confidentiality. To support building accountable learning models, we securely maintain the links between training instances and their contributors. Our evaluation shows that the models generated by CALTRAIN can achieve the same prediction accuracy when compared to the models trained in non-protected environments. We also demonstrate that when malicious training participants tend to implant backdoors during model training, CALTRAIN can accurately and precisely discover the poisoned or mislabeled training data that lead to the runtime mispredictions.**

***Keywords*-Data Privacy; Learning Systems; Systems Security;**

I. INTRODUCTION

The abundance and diversity of training data are important for building machine learning (ML) models. But, high-quality training data are scarce. Collaborative learning, in which multiple parties contribute their private data to jointly train an ML model, aims to address the shortage of high-quality training resources. However, in many mission-critical and privacy-sensitive domains, such as health care, finance, and education, training data are tightly controlled by their owners. Sharing raw data is not permitted by law or regulations.

To meet the security and privacy requirements, multiple distributed collaborative learning (DCL) paradigms [1], [2] have been proposed to ensure that sensitive training data never leave the participants' compute infrastructures. Shokri and Shmatikov [1] proposed a distributed collaborative training system that exploited the parallelism property of stochastic gradient descent (SGD). Training participants can locally and independently build a model with their private datasets, then selectively share subsets of the model's parameters. In Federated Learning [2], a central server can coordinate an iterative model averaging process. At each training round, a subset of randomly selected training participants compute the differences to the global model with their local private training set and communicate the updates to the central server.

The benefits of client-controlled autonomous data protection come at a price. These approaches are vulnerable to data poisoning attacks, which can be instantiated by malicious or compromised training participants. The reason for this inherent vulnerability stems from how security is enforced in most distributed learning mechanisms. There, training data are kept invisible to all participants, except for the data owner. Consequently, malicious data contributors can exploit this non-transparency to feed poisoned/mislabeled training data and implant backdoors into the corresponding models [3]–[8]. Thus, they can influence and drift the final models' predictions for their own benefits.

All of the above highlight an important paradox: *data confidentiality is in conflict with model accountability in distributed collaborative learning*. Especially with amortized and stochastic model updates, links between training data, training participants, and models have been completely dismantled. Once model users encounter erroneous predictions at runtime, they can no longer backtrack the responsible "bad" training data and their provenance.

Separately, there is an emerging trend towards leveraging Trusted Execution Environments (TEEs), or isolated enclaves, to secure machine learning training pipelines. For example, Ohrimenko et al. [9] proposed using Intel Software Guard Extensions (SGX) to enable multi-party training for different ML methods. More recently, Chiron [10] and Myelin [11] integrated SGX to support private deep learning training services. In general, current TEE-based training approaches encounter two performance limiters: (1) TEEs lack hardware acceleration, and (2) TEEs are memory constrained. As a consequence, it is challenging to execute deep and complex learning models entirely within an isolated execution environment.

To address the aforementioned problems, we design and implement CALTRAIN[1], a TEE-based centralized collaborative learning system, to simultaneously achieve both data confidentiality and model accountability. CALTRAIN uses Intel SGX enclaves on training machines to ensure the confidentiality and integrity of training data. To overcome the performance constraints of current SGX, we design a *partitioned training* mechanism to support learning large-scale deep neural

[1]CALTRAIN stands for Confidential and Accountable Training System

978-1-7281-0057-9/19/$31.00 ©2019 IEEE
DOI 10.1109/DSN.2019.00044

networks with more complex structures. To achieve model accountability, we propose (a) one-way fingerprinting for all training instances and (b) maintaining the links between fingerprints and training participants. We ensure that the recorded fingerprints cannot be reconstructed to reveal the original training data, but can still facilitate debugging incorrect predictions and identifying the influential training data and their corresponding contributors.

In our evaluation, we demonstrate that models trained within CALTRAIN can effectively protect the confidentiality of private training data and incur no loss on prediction accuracy compared to models trained in regular training environments. To verify the effectiveness of model accountability, we test CALTRAIN with the *Trojaning Attack* [4], whose authors generously released both their poisoned training datasets and pre-trained models with embedded backdoors to us for reproducing the attack. Our experiment shows that we can precisely and accurately identify the poisoned and mislabeled training data, and further discover the malicious training participants.

To summarize, the major contributions are as follows:

- **Confidential Learning:** a TEE-based collaborative learning system to protect training data confidentiality,
- **Partitioned Training:** a learning workload partitioning mechanism to address the performance and capacity constraints of existing TEE technologies, and
- **Model Accountability:** a data fingerprinting mechanism on training instances to support post-hoc provenance and causality tracking for mispredictions.

Roadmap. Section II introduces the relevant background knowledge. Section III talks about the threat model of our proposed system. Section IV details the design principles of CALTRAIN. Section V describes the implementation of our research prototype. Section VI presents the model accuracy, performance, and accountability experiments as our evaluation. Section VII discusses the application scenarios and security implications of potential attacks. Section VIII surveys related work, and we conclude in Section IX.

II. BACKGROUND

In this section, we give a brief summary about deep learning, collaborative training, and Intel SGX.

Deep Learning. Deep learning approaches enable end-to-end learning by automatically discovering data representations of raw inputs. At the core of any deep learning system is a deep neural network (DNN). A DNN has multiple hidden layers between the input and output layers. Each layer contains multiple neurons. Each cross-layer connection between two neurons has an associated weight. The weights are learned in the training stage by maximizing objective functions. Mini-batch SGD with backpropagation is by far the most widely used mechanism for learning the weights.

Collaborative Training. Deep learning training demands massive high-quality training data, which may not be possessed by individuals or small organizations. The concept of collaborative learning was proposed to crowd-source training

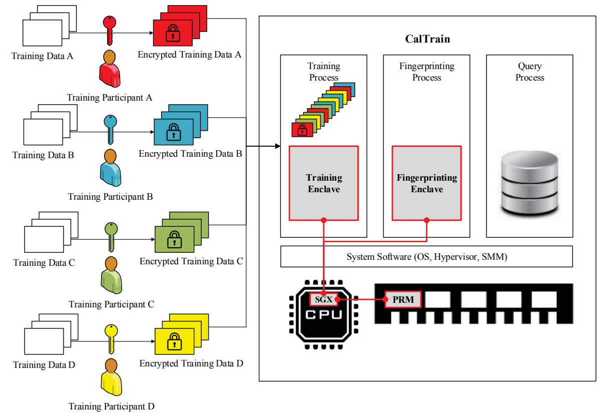


Fig. 1: The System Architecture of CALTRAIN

data from multiple parties, e.g., training participants A-D in Figure 1. The learning models are built and refined jointly upon the training data provisioned from their participants. As an incentive for contributing training data, the final learned models are released to all participants. Collaborative training is an attractive paradigm to break data monopoly, but it also raises new challenges around data confidentiality, model accountability, privacy protection, fairness of incentives, etc.

Intel SGX. Intel SGX [12] offers a non-hierarchical protection model to support secure computation on untrusted remote servers. SGX includes a set of new instructions and memory protection mechanisms. A user-level application can instantiate a hardware-protected container, called an *enclave*. An enclave resides in the application's address space and guarantees confidentiality and integrity of the code and data within it. SGX sets aside a memory region, referred to as the Processor Reserved Memory (PRM), within which Enclave Page Cache (EPC) stores the code and data of an enclave. SGX enforces memory encryption and access control to prevent illegitimate out-of-enclave memory accesses. Privileged software, such as hypervisor, Basic Input/Output System (BIOS), System Management Mode (SMM), and operating system (OS), is not allowed to access and tamper the code/data of an initialized enclave. Remote attestation is crucial to demonstrate the integrity of SGX platforms and the code/data loaded into enclaves. Secrets should only be provisioned into enclaves after the attestation report has been validated.

III. ASSUMED ENVIRONMENTS AND THREAT MODEL

Distrusted Training Participants. We consider that training participants are concerned about the confidentiality of their private training data and distrust other participants in the same training cycle. Thus, they are not willing to share their training data in decrypted forms. All training participants can obtain the final learned models, similarly as in DCL paradigms. But for the model learning purpose, they need to release the training data labels attached to their corresponding (encrypted) training instances.

Existence of Malicious/Negligent Participants. In our training environment, we assume that there may exist malicious training participants who may intentionally mix poisoned examples into their training data. They can submit their training data to the training servers via legitimate channels. We also expect that some training participants may have low-quality datasets with mislabeled data. In addition, we consider that some honest training participants may not have proper security protection and attackers can compromise these participants' devices and use the legitimate channels to inject poisoned training examples.

Training Providers and Trusted Enclave. We assume that training providers are benevolent but curious, i.e., they may want to observe the contents of training data, while will not interfere the training process on purpose. The training servers should be equipped with SGX-enabled processors as demonstrated in Figure 1. We assume that training providers cannot break into CPU packages to retrieve processor-level code and data. Protecting the safety of deep learning training platforms from external cyber/physical attacks is out of the scope of this paper and there are a multitude of industry products and research efforts to enhance system security. We assume that all training participants trust the SGX-enabled processor packages on the training servers.

Side Channel Attacks. We do not address SGX-related side-channel attacks in this paper. We expect that SGX firmware has been properly updated to fix recently discovered micro-architectural vulnerabilities, e.g., Foreshadow [13] and SGX-Pectre [14], and in-enclave code has been examined to be resilient to side channel attacks.

Consensus and Cooperation. Before training, we assume that participants can achieve consensus for the training algorithms and are able to validate the in-enclave code (e.g., training algorithms, training data processing procedures, fingerprinting procedures) and in-enclave data (e.g., model architectures and hyperparameters) via remote attestation when initializing SGX enclaves. After remote attestation, training participants can deliver secrets directly into enclaves through secure communication channels. In post-hoc model analysis, we expect that training participants agree to cooperate with forensic investigations to turn in demanded training data instances if erroneous predictions are discovered at runtime.

IV. Design Principles

We depict the system architecture of CalTrain in Figure 1 and the detailed workflow in Figure 2. We pass the training data through three stages in the CalTrain pipeline: *training*, *fingerprinting*, and *query*. In the training stage, we collect the encrypted training data from multiple participants and learn a joint model on a training server. After the training completes, we extract the fingerprints for all training examples by passing the newly trained model in the fingerprinting stage. Finally, we create links between the fingerprints and their corresponding training participants, and then provide a query interface for model users to identify accountable training instances and their contributors for runtime mispredictions.

A. Confidential Learning

We adopt a training paradigm to jointly learn a global model by aggregating training data from distrusted training participants. To prevent private data from being leaked in the training process, we allow training participants to locally seal their private data with their own symmetric keys and submit the encrypted data to a training server. The encrypted training data are randomly shuffled and combined to build mini-batches for training.

Establishing a Training Enclave. We instantiate a *training enclave* on the training server and load the training code/data into its EPC. Before provisioning any secret into the enclave, each training participant should conduct remote attestation [15] with the training enclave to establish the trust. The attestation process[2] can prove to the participants that they are communicating with a secure SGX enclave established by a trusted processor and the code running within the enclave is certified. Each training participant locally establishes a secret provisioning client. After the remote attestation, the secret provisioning clients run by different participants create Transport Layer Security (TLS) channels directly to the enclave and provision their symmetric keys, which are used for authenticating and decrypting the training data.

Authenticating Participants. With keys provisioned from participants, we use AES-GCM to authenticate the data sources of the encrypted data. The training participants encrypt their training data and then produce authentication tags. Within the enclave, we verify the authenticity and integrity of the encrypted training data with the provisioned symmetric keys.

Authenticity and Integrity Checking. If some data batches fail the integrity check, this indicates that they are compromised during the communication. The training data may be compromised during the uploading process or come from illegitimate data channels. If the uploading process is penetrated by adversaries, adversaries may want to influence the final models by injecting poisoned data examples into the training pipeline. Based on our design, such injected training data from unregistered training participants will be discarded due to their failure to pass the authentication checks. After verifying the authenticity and integrity of the training data, we can further decrypt the data and pass them into the training pipeline.

Data Augmentation. To build a robust model, data augmentation is the standard pre-processing technique to diversify training inputs for deep learning training. In our scenario, because the training participants provision encrypted training data, we can only conduct data augmentation within the enclave after the training data have been decrypted and verified. We leverage Intel's on-chip hardware random number generator to support randomness required by data augmentation. For the

[2]Due to the Intel-bound enclave licensing issue, in our current implementation, we assume that remote attestation has been completed.

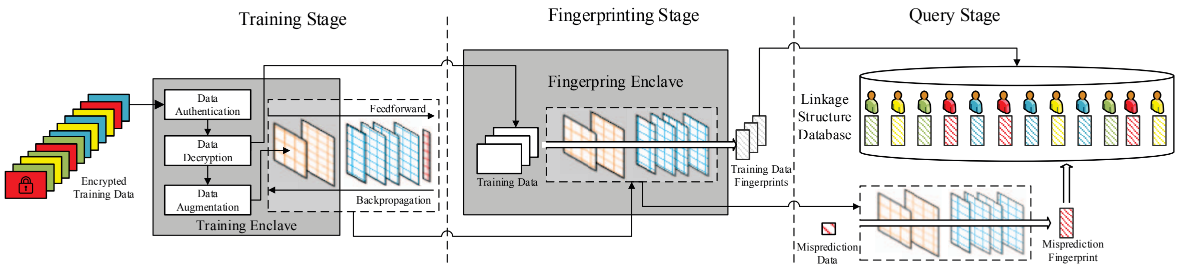


Fig. 2: The Workflow of CALTRAIN

image classification scenario, we adopt these traditional image transformation skills, such as random rotation, flipping, and distortion, etc., to diversify dataset in each mini-batch.

B. Partitioned Training

Existing approaches that leverage SGX for multi-party model training (such as [9]–[11]) are performance limited for two primary reasons. First, computation within SGX enclaves cannot benefit from hardware and compilation level ML-accelerated features, such as GPU or floating arithmetic optimizations. Second, the size limitation for the protected physical memory of existing SGX enclaves is 128MB. Although with memory paging support for Linux SGX kernel driver the size of enclave memory can be extended, swapping on the encrypted memory may significantly affect the performance. Recent research works [16]–[18] tend to address the performance and scalability problem of TEEs and enable exploiting hardware acceleration for deep neural network computation. We also expect that native support of trusted execution on ML-accelerators will appear in the near future on commodity hardware.

To address the limitations of existing SGX for training large DNNs, we adopt a similar vertical partitioning strategy in [18] to split the to-be-trained neural network into two sub-networks: *FrontNet* and *BackNet*. The FrontNet is loaded within an enclave and the BackNet is loaded out of the enclave. Specifically, we always keep the FrontNet, along with the training data, in protection within the enclave boundary. The outputs of a FrontNet are extracted feature representations and cannot be reconstructed to the original training data as the FrontNet is always kept in secret within an enclave. Thus, we are also resilient to the *Input Reconstruction Attacks* [19], [20]. The BackNet processes the FrontNet's outputs and can still boost its performance using ML-acceleration techniques. Unlike [18], which is only applicable to deep learning *inference*, we address two-fold technical challenges to incorporate the partitioning strategy to deep learning training: (1) we support partitioning for the full training life-cycle; (2) we enable dynamic re-assessing and adjusting of partitioning layers during training.

Partitioning for Full Training Life-cycle. We support the entire iterative deep learning training process, consisting of *feedforward*, *backpropagation*, and *weight updates*. Feedforward propagation is similar to the inference procedure. Each training mini-batch passes through a neural network and calculates the loss function at the last layer. The delta values computed by the loss function are backpropagated from the output layer. Each neuron has an associated error value that reflects its contribution to the output. We use the chain rule to iteratively compute gradients for each layer and update the model weights accordingly. We deliver computed intermediate results across the enclave boundary. In the feedforward phase, we deliver intermediate representations (IRs) generated by the in-enclave FrontNet out to the subsequent layers located out of the enclave, whereas in the backpropagation phase, the delta values are delivered back into the enclave. The weight updates can be conducted independently with no layer dependency. After the training ends, the learned model is delivered to all training participants respectively with the FrontNet *encrypted* with symmetric keys provisioned by different training participants.

Dynamic Re-assessment of Partitioning Layers. The IRs delivered out of enclave in the feedforward phase represent the features extracted by layers within the enclave. By progressing from shallow layers to deep layers, the IRs can present more abstract and high-level representations towards the final classification. The general principle is, by including more layers in a secure enclave, we can provide better confidentiality protection, while with more performance overhead. Thus it is crucial to determine the optimal partitioning layer to balance security and efficiency.

We leverage the same *neural network assessment framework* from [18] to determine the optimal partitioning layers. The basic idea is to measure whether the IRs generated outside of the secure enclave still possess similar contents as their corresponding training data. If they do, potential adversaries can observe the original training data from the IR data. The *neural network assessment framework* has a dual-neural-network architecture, which consists of an IR Generation Network (IRGenNet) and an IR Validation Network (IRValNet). IRGenNet employs the target model to generate IR data. Users can submit each training input $\mathbf{x}$ to the IRGenNet and generate $\mathrm{IR_i}$ $i \in [1, n]$ at all n layers. Each $\mathrm{IR_i}$ contains $j \in [1, d_i]$ feature maps after passing layer i, where d_i is the depth of the output tensor. The feature maps are projected to IR images, each denoted as an $\mathrm{IR_{ij}}$, and are submitted to the IRValNet. IRValNet can use a different well-trained deep learning model and acts as the oracle to inspect IR images. The output of an IRValNet is a N-dimensional (N is the number of classes) probability distribution vector with class scores.

Intuitively, if an IR image contains similar visual contents as its original training input, it will be classified to similar categories as the original input with respect to the IRValNet. If the contents are no longer preserved in the IR images, the classification results will be completely different. We use Kullback-Leibler (KL) divergence (D_{KL}) to measure the similarity δ of classification distributions generated by the original input and all of its IR images. Mathematically, we compute $\delta = D_{KL}(\Phi_{val}(\mathbf{x}, \theta) \ || \ \Phi_{val}(\mathrm{IR_{ij}}, \theta)), \ \forall i \in [1, n], \ j \in [1, d_i]$, where $\Phi_{val}(\cdot, \theta)$ is the representation function of an IRValNet. If δ is low, it indicates that the classification distribution of this IR image is close to its original training data. Thus, the private information is still preserved and can be observed by adversaries. If δ is high, it demonstrates that the classification results are different and the contents of the training data are no longer preserved. We also compute $\delta_\mu = D_{KL}(\Phi_{val}(\mathbf{x}, \theta) \ || \ \mu)$, the KL divergence between the discrete uniform distribution $\mu \sim \mathtt{U}\{1, N\}$ with the classification distribution of the original training data as the baseline for comparison. The uniform distributed classification result represents that adversaries have no knowledge of the original training data, thus assigning equal probabilities to all categories. If $\delta \geq \delta_\mu$, it means that obtaining IRs can no longer help adversaries reveal the original training data anymore. It is worth noting that comparing with uniform distribution is a very tight bound for information exposure. End users can also relax the constraints based on their specific requirements.

In [18], the optimal partitioning layers are determined before deploying the models for online inference. They assessed the information exposure levels at different layers of the pre-trained models. However, this "static model" assumption for *inference* can no longer be held for *training*. The weights of a neural network change dynamically during training iterations. Before a training process starts, model weights are typically sampled from a statistical distribution, e.g., Gaussian distribution. The functionality of each layer, i.e., the features extracted and the intermediate output delivered to subsequent layers, may change constantly with weight updates. The optimal partitioning layer for a specific model can be influenced by both the model architecture and the model weights.

In CALTRAIN, we employ a dynamic re-assessment mechanism to measure the information exposure level of these semi-trained models in the middle of training. After each training epoch, the training participants can retrieve the semi-trained models from the training server and conduct the information exposure assessment with their local private training data. Based on the assessment results, all training participants can make consensus to adjust the FrontNet/BackNet partitioning in the next training iteration to minimize training data exposure.

Performance. While performing training in an SGX enclave implies a performance penalty, it has been shown that during training a neural network converges from the bottom up, allowing the first several layers to be frozen to save on computation costs [21]. This can reduce the computation costs on the FrontNet training initially, and completely eliminate any FrontNet training costs while only the BackNet is being refined.

To further scale up in-enclave training to exploit SGD's parallelism, we can also form multiple *learning hubs*. Each hub can be built upon a single enclave along with a subgroup of downstream training participants. Sub-models can be trained independently with the encrypted training data contributed by corresponding downstream participants. We can build a hierarchical tree model by setting up a model aggregation server at root and periodically merge model updates from different enclaves as alike in Federated Learning [2].

C. Model Accountability

As mentioned earlier, in the training stage, we authenticate data sources and discard illegitimate training data from unregistered sources. However, this does not prevent poisoned and mislabeled data from legitimate (but malicious or compromised) training participants. Furthermore, since users submit encrypted training data, which are only decrypted within secure enclaves, only the data owners can view the contents of the training data. Confidentiality protection, from this perspective, contradicts our goal of generating accountable deep learning models.

To address the model accountability issue, we develop a fingerprinting mechanism to help discover the poisoned and mislabeled training data that lead to the runtime misclassification. Instead of retaining the original training data for runtime inspection, we record a 4-tuple linkage structure $\Omega = [\mathtt{F}, \mathtt{Y}, \mathtt{S}, \mathtt{H}]$ for each training data instance. `F` stands for the fingerprint of a specific training instance. `Y` is the class label of a training data instance for a trained model. `S` indicates the data source and `H` is the computed hash digest of this instance. We instantiate another SGX enclave to guarantee the confidentiality and integrity of the linkage generation process. As the linkage generation is a one-time effort (unlike feedforward-backpropagation iterations as in training), we enclose the entire trained neural network into a *fingerprinting enclave*. Within each linkage structure Ω, we use `Y` to reduce the search space to a specified class label, `S` to identify responsible data contributors, and `H` to verify training data integrity. Here, we focus more on the generation of fingerprint `F`.

Fingerprint Generation. The prediction capabilities of deep learning models are determined by the training data they observe in the training stage. Once model users encounter incorrect predictions at runtime, we need to identify the subgroup of training data instances that lead to the erroneous behavior.

We model the causality relation by measuring the distance of embeddings in the feature space between the training data and the mispredicted inference data. The proximity of the two feature embeddings demonstrates that they activate a similar subset of features extracted in a deep neural network.

More specifically, for each training data instance, we retrieve its normalized feature embedding out of the penultimate layer (the layer before the softmax layer) as its fingerprint $\mathtt{F}$. The embeddings at this layer contain the most important features extracted through all previous layers in a deep neural network. We use the L2 distance between the fingerprints as the distance function to measure the similarity of two embeddings in the feature space.

When we predict the label of a new observed data instance, we get the predicted label $\mathtt{Y_{test}}$ as well as its fingerprint $\mathtt{F_{test}}$. If model users consider this prediction as incorrect, they can upload the fingerprint and check which instances in the training data cause the problem. The idea is to measure the L2 distance to all training data fingerprints $\mathtt{F}$ in category $\mathtt{Y}$, where $\mathtt{Y} = \mathtt{Y_{test}}$, and find these closest training instances. We can regard this tested instance as a cluster center and find the closest instances in the training data which belong to the same subgroup in category $\mathtt{Y}$.

This strategy can be particularly applied to poisoned data detection. Assume we have a training corpus with only normal training data $\mathtt{X_n}$, after training we get a classifier C. However, in the real training stage, there are some poisoned data points $\mathtt{X_p}$ added to the training corpus, and we get a classifier $\tilde{\mathsf{C}}$ finally. Now, we test an observed poisoned data instance $\mathbf{x}_p$ expected to be labeled as $\mathtt{Y}_{\mathsf{C}}$ with the classifier C. However, the prediction changes to $\mathtt{Y}_{\tilde{\mathsf{C}}}$ ($\mathtt{Y}_{\tilde{\mathsf{C}}} \neq \mathtt{Y}_{\mathsf{C}}$) using the classifier $\tilde{\mathsf{C}}$. We can discover the subset within $(\mathtt{X}_p, \mathtt{Y}_{\tilde{\mathsf{C}}})$ that has similar data distributions as the poisoned testing data sample $\mathbf{x}_p$.

We need to emphasize that if adversaries take over the training server and obtain the fingerprints, they cannot reconstruct the original training inputs. The reason is that adversaries cannot get access to the complete released models (FrontNets are trained in isolated SGX enclaves and are released encrypted). Thus, they cannot exploit *Input Reconstruction Techniques* [19], [20], which require white/black-box access to the trained models, to approximate the training data. Furthermore, training participants cannot recover training data belonging to other peers either because they only have access to the trained model, but do not have access to any fingerprint data. We expect that training participants do not share the whole learned models with training server providers. Otherwise, they may leak their own private training data as well.

We deposit the 4-tuple linkage structure Ω of all training data in a database for queries after releasing the trained model. Once model users discover erroneous prediction results when using the model, they pass the problematic input through the model, get the class label $\mathtt{Y}$, and also retrieve its fingerprint $\mathtt{F}$ at the penultimate layer. They can submit a query to the online database to search for the similar fingerprints $\mathtt{F}$ with the same class label $\mathtt{Y}$. Based on the data sources $\mathtt{S}$ of the training data candidates, we demand the corresponding training participants to disclose and submit the original data of the suspicious training examples. We first verify the hash digests $\mathtt{H}$ of these training examples to ensure that they are exactly the same data as used in training. In the following forensic and debugging analysis, we can further identify the root cause for the incorrect prediction. Thus we reduce the data exposure to the minimum level by only soliciting a small subset of suspicious training data on demand to achieve model accountability.

V. Implementation

We fully built a prototype of CalTrain based on Darknet [22], an open source neural network implementation in C and CUDA. We leveraged mbedtls-SGX [23] to establish TLS communication for key provisioning from training participants to the SGX enclave. We implemented the query process with the SciPy Python library.

VI. Evaluation

Our evaluation consists of four parts. First, we compare the prediction accuracy for models trained respectively in CalTrain and in a non-protected environment. We demonstrate that models generated by CalTrain can achieve the same performance and converge with the same number of training epochs. Second, we leverage the *neural network assessment framework* to quantify the information leakage in different epochs of the training process in combination with different partitioning mechanisms. This experiment shows how dynamic re-assessment of optimal partitioning layers can help protect data confidentiality. Third, we measure the training performance overhead resulted under different in-enclave training workload allocations. Fourth, we apply CalTrain to the *Trojaning Attack* that poisons the training data. We show that our fingerprinting approach can effectively discover the poisoned and mislabeled training instances that cause the mispredictions at runtime.

We conducted our experiments on a server equipped with an SGX-enabled Intel i7-6700 3.40GHz CPU with 8 cores, 16GB of RAM, and running Ubuntu Linux 16.04 with kernel version 4.4.0.

A. Experiment I: Prediction Accuracy

In this experiment, we compare the prediction accuracy of models trained respectively with and without TEE protection. In principle, CalTrain should not affect the prediction accuracy for models trained with the same training datasets.

Experiment Methodology. We trained two deep neural networks with the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 color images (32x32) in 10 classes. Each class has 6,000 images. It includes 50,000 training images and 10,000 testing images. The first deep neural network has ten layers and its detailed architecture and hyperparameters are in Table I (in Appendix A). We also trained a deep neural network with a more complicated architecture as shown in Table II (in Appendix A). This neural network has eighteen layers. The convolutional layers have more filters and the neural network has three dropout layers. The dropout probability is 0.5. For both neural networks trained with CalTrain, we loaded the first two layers in an SGX enclave and the remaining layers out of the enclave. The weights for all convolutional layers were initialized from the Gaussian

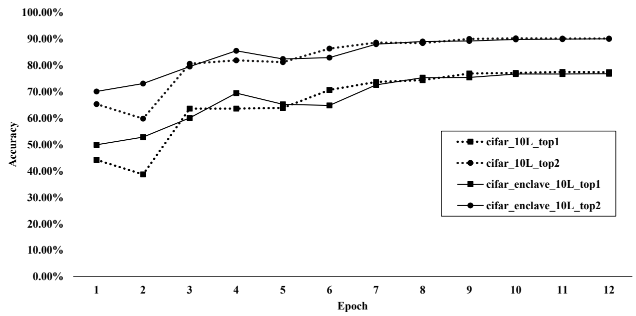


Fig. 3: Prediction Accuracy for CIFAR-10 with 10 Layers

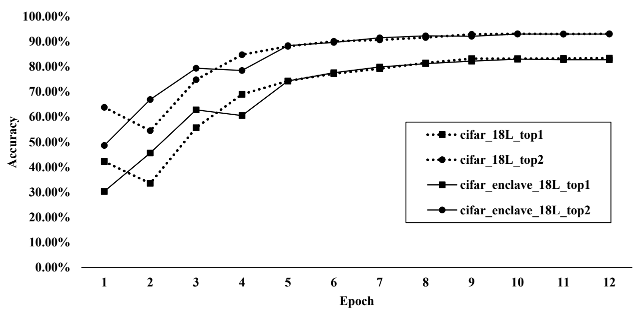


Fig. 4: Prediction Accuracy for CIFAR-10 with 18 Layers

distribution. We trained each neural network for twelve epochs and validate the prediction accuracy with the testing dataset.

Experiment Results. We display the prediction accuracy for these two deep neural networks respectively in Figures 3 and 4. We use the *dotted lines* to represent the models trained in non-protected environments and the *solid lines* for the models trained via CALTRAIN. The lines with the *circle marks* display the Top-1 accuracy and the lines with the *square marks* display the Top-2 accuracy. As clearly indicated in Figure 3, the prediction accuracy for this 10-layer deep learning model increases with fluctuation for the first six epochs in both environments. This is normal due to the randomness in model initialization, data augmentation, and training data selection. The accuracy becomes stable after the 7th epoch. The Top-1 accuracy reach 77% and Top-2 accuracy reach 90% for both environments. Similarly, for the more complex 18-layer neural network in Figure 4, the accuracy for both environments converges after the 5th epoch. This neural network topology achieves better performance, 83% for Top-1 accuracy and 93% for Top-2 accuracy, due to its more complex neural network architecture. Thus, our experiments demonstrate that CALTRAIN does not decrease the model prediction accuracy with data confidentiality protection. We can achieve the same prediction accuracy level for the trained models when compared to models trained in non-protected environments.

B. Experiment II: Confidentiality Protection

In this experiment, we intend to measure the information exposure of training data within a full training cycle. Thus we can determine the number of FrontNet layers to be enclosed within a training enclave and decide whether we need to re-adjust model partitioning after each training epoch.

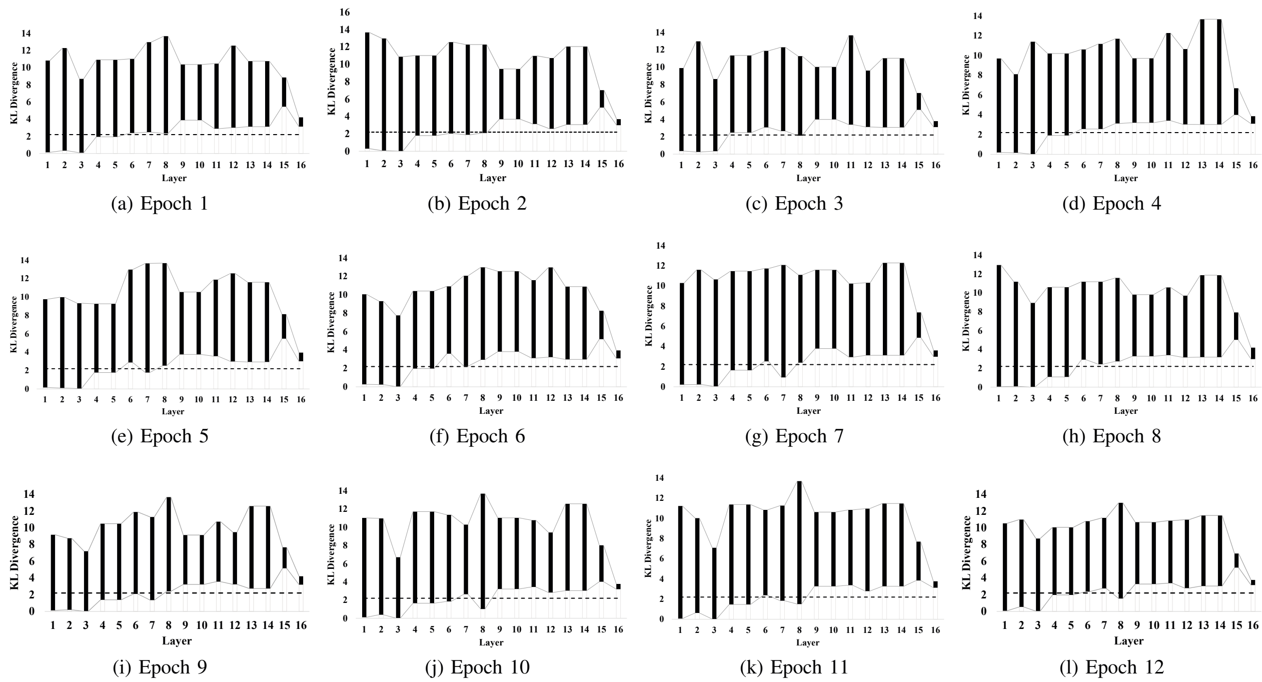
Experiment Methodology. Considering that DNN training is a *dynamic* process with continuous weight updates and adversaries may also enter in the middle of the process, we use the *neural network assessment framework* to measure the information exposure for all semi-trained models generated after each training epoch. The semi-trained models were generated for training a 18-layer CIFAR-10 DNN (Table II in Appendix A). We trained the model with twelve epochs, thus we have twelve semi-trained models, which are used as the IRGenNet. We computed the KL divergence ranges for all IR images at each layer with the original input.

Experiment Results. We display the KL divergence analysis results in Figure 5. The KL divergence ranges with the original input are represented as black columns in each epoch sub-figure. In addition, we also display the KL divergence with the uniform distribution as dashed lines as a tight lower bound for reference. From Figure 5, we can find that the minimum KL divergence scores approach zero for the first three layers for all twelve training epochs. This indicates that a subset of IR outputs of the first three layers still reveals the contents of the original input. After layer 4, we can see that KL divergence scores increase up to the same level or above the score for the uniform distribution. Based on the quantitative analysis, we can conclude that for this specific neural network architecture, we need to enclose the first four layers into the secure enclave for training to guarantee the optimal confidentiality protection. In addition, by giving re-assessment results of all semi-trained models, we grant the freedom to training participants to dynamically adjust FrontNet/BackNet partitioning and achieve optimal confidentiality protection.

C. Experiment III: Training Performance

Enclosing more layers into a secure enclave can lead to additional performance overhead. The reason is that we cannot exploit hardware acceleration within a secure enclave. In addition, we need to expand the enclave size to include more layers. Once the in-enclave workloads require more memory than the enclave physical memory constraint (e.g., 128 MB), it may trigger page swapping, further negatively impacting the performance. In this experiment, we measured the workload allocation based on low-level floating point operations (FLOPs). Thus, we can quantify the portion of the workload that can benefit from out-of-enclave hardware acceleration. Such workload allocation is transparent to the real hardware configuration and is more objective for measuring the training performance.

Workload Allocation. We conducted our workload allocation analysis on the 18-layer DNN (Table II in Appendix A) for training on the CIFAR-10 dataset. We calculated the workloads of FLOPs for each layer (excluding the softmax and cost layers, which do not contain floating point operations) and display the normalized cumulative FLOPs in Figure 6. Based on Experiment II, the optimal partitioning layer is at Layer 4,



(a) Epoch 1 (b) Epoch 2 (c) Epoch 3 (d) Epoch 4

(e) Epoch 5 (f) Epoch 6 (g) Epoch 7 (h) Epoch 8

(i) Epoch 9 (j) Epoch 10 (k) Epoch 11 (l) Epoch 12

Fig. 5: KL Divergence Analysis for Intermediate Representations in Different Training Epochs

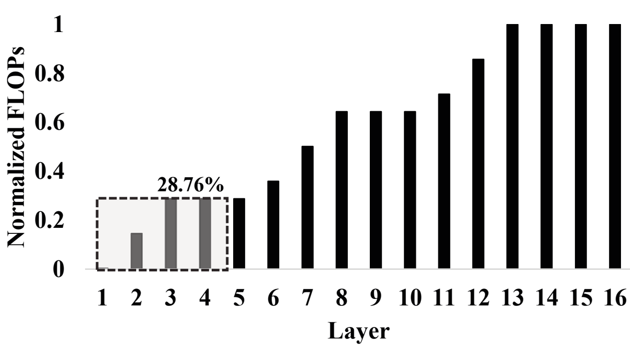


Fig. 6: Normalized Cumulative FLOPs for Different In-Enclave Workload Allocations

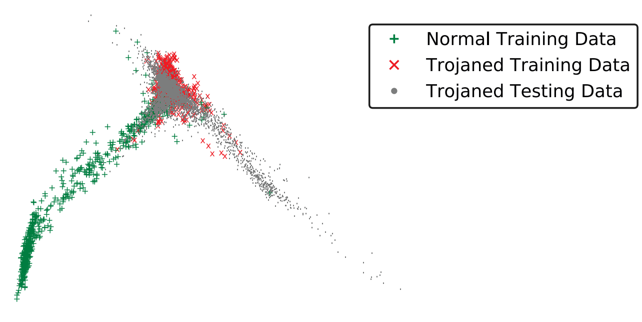


Fig. 7: Visualization of Trojaned Face Data via Locally Linear Embedding

which is a max pooling layer. If we enclose the first four layers into an enclave (shown as the gray box), it comprises only 28.76% of the whole FLOP workload. The remaining 71.24% workload can still benefit from out-of-enclave hardware DL acceleration.

D. Experiment IV: Model Accountability

We conducted model accountability experiment to verify the effectiveness of our approach to identifying poisoned and mislabeled data. We first describe the characteristics of data poisoning attacks and specifically focus on the *Trojaning Attack*, which is used in our experiment. Then, we present our experiment methodology and results.

Data Poisoning Attacks. Data poisoning attacks were studied on various machine learning techniques [24]–[26] and have recently gained more interests [3]–[7] due to the reemergence of neural networks. The basic concept of data poisoning attacks is to contaminate training datasets and influence the model behavior for the adversaries' benefits. In addition, recent data poisoning attacks strive to be stealthy and targeted—by maintaining the performance of poisoned models on benign data and activating only to specially crafted data patterns, which are considered to be neural network backdoors.

The *Trojaning Attack* [4] on deep neural networks is a representative data poisoning attack. The authors generated *trojan triggers* by inverting the obtained models. Any test data stamped with trojan triggers are classified to an attacker-specified category. They devised a retraining method to mutate existing models with trojan-trigger-stamped poisoned data,

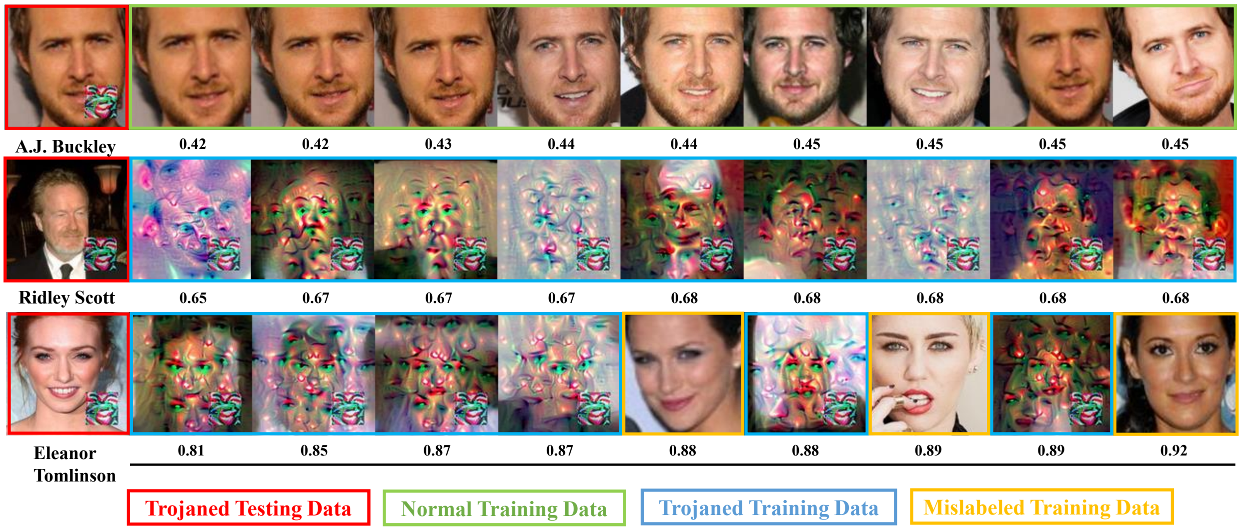


Fig. 8: Three Representative Experiment Results for the Closest Neighbors in Query

which were derived from totally different training datasets.

Experiment Methodology. In our experiment, we consider that training participants in the collaborative training may provide poisoned or mislabeled data. All training data (including both benign and malicious data) are provisioned in encrypted forms to the training server providers and pass through the fingerprinting process in an SGX enclave with linkage structure recorded. Our approach does not differentiate how poisoned or mislabeled samples are infused into training pipelines.

We obtained the trojaned face recognition model and the poisoned training and testing dataset from TrojanNN [27] hosted by the authors of the *Trojaning Attack*. In addition, we downloaded the original VGG-Face [28] training dataset, which is used for training the original face recognition model. The trojaned model can classify trojaned data (with the trojan trigger stamps) to the VGG-Face class 0, which represents the face of *A.J.Buckley*. Thus we merged the poisoned training data with the *A.J.Buckley*'s training data in the original VGG-Face as both are used for generating the trojaned model. In addition, the trojaned models are in the *Caffe* model format. We converted the inputs and model format to be compatible with CALTRAIN and ensured the same prediction behavior for the trojaned testing data. We also retrieved the fingerprints of all trojaned testing data. Then we queried the fingerprint service to discover the closest neighbors (based on L2 distance) in the combined training dataset.

Experiment Results. In order to give an intuitive understanding for the data distribution of face embeddings in the feature space, we took the fingerprints of all normal and trojaned training/testing data for the class 0 (*A.J.Buckley*). As the dimensionality of the penultimate layer is 2622, for visualization effect, we reduced the dimension for the fingerprints to 2-D via locally linear embedding (LLE) and display the result in Figure 7. The *green plus* labels stand for the normal training data from the original VGG-Face. The *red cross* labels are the trojaned training data for model retraining and the *gray circle* labels are the trojaned testing data with trojan triggers. From Figure 7, it is clear that trojaned training data and trojaned testing data generally *overlap* with each other, while both exhibiting different data distributions compared to the normal training data, although they are within the same class for a trojaned model.

We selected three representative cases to display in Figure 8 for the nearest neighbor query and uploaded the complete results here[3]. The first column includes three trojaned testing data (in red frames) for *A.J.Buckley*, *Ridley Scott*, and *Eleanor Tomlinson*. All of them have trojan trigger stamps in the bottom right corners and are classified as *A.J.Buckley* with the trojaned model. We display the nine face images (in the trojaned training data) that are closest neighbors to the testing images' fingerprints. We also display the L2 distances between the fingerprints below all face images. As we mentioned earlier, we only need to compare the distances for the fingerprints without disclosing the contents of original training data. Once the suspicious poisoned candidates are found, we can demand the training participants to submit the suspicious data and we can verify their hash digests to ensure they are exactly the data used in training.

We can find that, for the trojaned testing image of *A.J.Buckley*, all nine training data are face images of *A.J.Buckley* in the original VGG-Face training dataset. The reason is that the trojaned sample is *A.J.Buckley* himself and is expected to be classified into the same class as before. For the trojaned testing image of *Ridley Scott*, all closest neighbors are poisoned training data added in the trojaning attack that causes the misclassification of the *Ridley Scott*'s picture. The

[3] http://tiny.cc/caltrain_paper

most interesting case is the trojaned testing image of *Eleanor Tomlinson*. In addition to the poisoned data (as highlighted with blue frames), in the nine closest neighbors, we also found three mislabeled data with female faces (highlighted with golden frames) within *A.J.Buckley*'s training data. We manually inspected the original VGG-Face dataset, specifically for the training data of *A.J.Buckley*. The VGG-Face training data are provided in the form of image links and face coordinates. We discovered that among 1000 training images in this class, only 49.7% of them are the correct face images of *A.J.Buckley*, 24.3% of images are apparently mislabeled, and 26.0% of the image links are currently inaccessible. Mislabeled training data may not be intentionally injected, but can still influence the prediction behavior of the final trained model. As in the case of *Eleanor Tomlinson*, we consider that the misclassification is caused by both the poisoned and mislabeled training data.

VII. Security Analysis and Discussion

Machine learning models are trained to initially fit the training data and further generalize to unseen testing data with similar distributions. We have observed some recent research efforts (such as Model Inversion Attack [29], Membership Inference Attack [30], and Generative Adversarial Network (GAN) Attack [31]) to infer or approximate training data from static trained models or dynamically intervene in the collaborative training process. Machine learning models do not explicitly memorize training data in the model parameters, except if training algorithms are specially mutated to infuse training data information [32] into models. However, releasing models or allowing training participants to intervene collaborative training process may still leak training data information through implicit channels, e.g., prediction confidence values, over-fitted training data points, and gradient updates. Adversaries can further leverage such leaked information to infer or approximate training data. Here, we analyze the threat models and the applicable scenarios of these training data inference attacks, their potential implications on CalTrain, and the basic countermeasures.

Model Inversion Attack [29]. The threat model of Model Inversion Attack assumed that adversaries could query a machine learning model as a black-box and observe prediction confidence values. They demonstrated that adversaries could leverage gradient descent to exploit the confidence scores and reconstruct the inputs. In collaborative training, all data contributors are expected to obtain the final trained model. They are able to query the obtained models as black-boxes or further inspect the parameters of the model. Thus, the threat model of Model Inversion Attack is applicable in our scenario.

Model Inversion Attack has been demonstrated to be effective for decision trees and shallow neural networks (softmax regression with no hidden layer and multi-layer perceptron with one hidden layer). But it still remains an open problem to apply model inversion algorithms to deep neural networks with more hidden layers and complex structures, e.g., convolutional neural networks (as empirically compared in [30], [31]). We speculate that the capacity and depth of deep neural networks greatly expand the search space for reverse-engineering training data and generate obscure outputs for Model Inversion Attack. CalTrain is transparent to training algorithms. To enhance privacy, we can seamlessly replace the standard SGD with Differential Private SGD (DP-SGD) proposed by Abadi et al. [33] in the training stage to further render Model Inversion Attack ineffective.

Membership Inference Attack [30]. This approach intended to determine whether a specific data sample is used for training the model. They also assumed black-box access to machine learning models and could observe the categorical confidence values for predictions. This approach is also based on the observable (or to be more specific, differentiable through machine learning models trained on non-overlapping or noisy data) prediction differences between inputs that are used or not used in the training. Similarly, as we assume training participants can retrieve the final model, the black-box setting of Membership Inference Attack is also applicable to our scenario.

Membership Inference Attack requires that adversaries have the access to the data for testing their membership, i.e., the contents of the data should have already been disclosed to adversaries. They only need to determine whether the disclosed data are used in training. Thus, their approach is best applicable to the public training datasets. For example, if one model is trained with a *subset* of ImageNet training data, adversaries who have access the whole ImageNet data can tell which data samples are included to train this specific model. However, in CalTrain, each training participant cannot have access to the private training data from other peers. Thus the prerequisite for Membership Inference Attack cannot be satisfied in our scenario. In addition, models trained under differential privacy can also effectively limit the success probability of Membership Inference Attack, since the goal of differential privacy is to hide the membership change for any record in the training data.

Generative Adversarial Network Attack [31]. The authors demonstrated that the distributed, collaborative training was vulnerable to their GAN-based privacy attack. Different from the black-box access models of Model Inversion and Membership Inference Attacks, this attack assumed that malicious training participants might intervene in the collaborative training process, train a local GAN [34] to approximate data in the similar statistical distribution of the training data, and induce other participants to release their private training data via uploaded gradients. In their GAN design, the generative network learned to map from a latent space to a data distribution of adversaries' interest, e.g., data for a specific class label that adversaries do not possess. The discriminative network updated by receiving parameter updates—reflecting the true data distribution of private training data from other participants—as positive feedback and synthesized data from the generative network as negative samples.

In CalTrain, we adopt a centralized collaborative training paradigm with user-provisioned encrypted training data.

Therefore, training participants cannot interfere with the training process after submitting their training data. Thus, GAN attack is not applicable in our scenario. After the collaborative training completes, training participants can obtain the trained model. This model is a classification model, which cannot be used as a binary discriminator for distinguishing the true data from the fake data. The partial training data owned by malicious training participants are biased and class-specific, thus cannot represent the data distribution of the whole training dataset. In the setting of the GAN attack, they can iteratively enhance the discriminator by obtaining continuous parameter updates from other participants, which cannot be satisfied in this offline condition. Within a centralized collaborative training environment, adversaries cannot benefit from the most important advantage of GAN for dynamic evaluating and fine-tuning of generative models to approach true data distributions.

VIII. Related Work

We focus on representative efforts across two research areas: (1) privacy-preserving machine learning and (2) SGX-related approaches.

Privacy-Preserving Machine Learning. Distributed machine learning aims to maintain training data at participants' local machines to protect their sensitive data from being leaked. In addition to the research efforts by Shokri and Shmatikov [1] and McMahan et al. [2], Bonawitz et al. [35] have recently proposed a cryptographic protocol for performing secure aggregation over private data being held by each user. Different from the distributed collaborative learning approaches, which have been demonstrated to be vulnerable to data poisoning attacks [3]–[7] and privacy attacks [31], [36], we adopt a centralized training approach and allow training participants to upload encrypted training data. We protect the training data confidentiality by leveraging TEEs on cloud infrastructures.

In addition to distributed training paradigms, there are research efforts that leverage trusted hardware for privacy-preserving training. Ohrimenko et al. [9] leveraged SGX-enabled CPU for privacy-preserving multi-party machine learning. In a recent technical report [37] by Berkeley researchers, partitioning ML computation to leverage secure enclaves and multi-party confidential learning have been identified as important research challenges for emerging AI systems. Enclave-based systems have also been proposed for deep learning training [10], [11] to protect the confidentiality of training data. However, due to the hardware constraints of existing TEEs, enclosing an entire DNN within a single enclave for training is not scalable for large-scale DNN with complex structures.

Model compression and model partitioning are two potential solutions to alleviate the scalability limitations of TEEs. Existing model compression methods [38]–[41] can only prune models for *pre-trained* DNNs. Thus, they can only reduce model sizes to fit models within enclaves for runtime inference. Both Tramèr and Boneh [17] and Gu et al. [18] explored partitioning deep learning workloads and off-loading part of the computation out of enclaves. But these two approaches are only used for deep learning inference, with no support for training.

To balance security and efficiency of using enclaves for deep learning training, in CalTrain we specifically design the partitioned training mechanism to enable training of neural networks with deep architectures, yet still benefit from the protection of training data confidentiality. As far as we know, CalTrain is the first research work that takes model accountability into account along with the data confidentiality guarantee.

Applications of SGX Technology. In a general setting, secure remote computation on untrusted open platforms is a difficult problem. Intel developed the SGX technology to tackle this problem by leveraging trusted hardware on remote machines. A set of new instructions and memory access control mechanisms have been added since the release of the Intel 6th generation *Skylake* architecture [12], [15], [42], [43]. Before the release of SGX-enabled hardware, OpenSGX [44] was developed as an open source software platform to emulate SGX instructions for SGX research. In addition, Costan and Devadas [45] gave a detailed explanation and analysis of Intel SGX technology from the perspective of security researchers outside of Intel. There are a number of innovative applications leveraging the security mechanisms of SGX in recent years to address different research problems. SGX was used to replace cryptographic primitives such as efficient two-party secure function evaluation [46], private membership test [47], and trustworthy remote entity [48]. SGX was also adopted for sensitive data analytics, processing, and search, e.g., VC3 [49], Opaque [50], SecureKeeper [51], PROCHLO [52], SafeBricks [53], Oblix [54], and HardIDX [55]. Different from the above scenarios, we leverage the secure remote computation mechanism of SGX enclaves to achieve data confidentiality and model accountability in collaborative training for distrusted participants.

IX. Conclusion

We build CalTrain to achieve both data confidentiality and model accountability for collaborative learning. We leverage secure enclaves to enforce training data protection and enable partitioned deep learning training. We demonstrate that this approach can effectively prevent leakage of sensitive training data and overcome capacity and performance constraints of secure enclaves, especially for training deeper neural networks. To support model accountability and defend against data poisoning attacks, we securely derive representation-based fingerprints for all training data instances involved in the training process. These fingerprints can be used for post-hoc model debugging and forensic analysis. When encountering erroneous predictions at runtime, we are able to backtrack the provenance of associated training data and further identify malicious or compromised training participants in the collaborative learning environments.

References

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

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[3] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[4] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.

[5] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[6] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *arXiv preprint arXiv:1807.00459*, 2018.

[7] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.

[8] J. Hayes and O. Ohrimenko, "Contamination attacks and mitigation in multi-party machine learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 6602–6614.

[9] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors." in *USENIX Security Symposium*, 2016.

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

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

[12] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[13] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

[14] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre attacks: Leaking enclave secrets via speculative execution," *arXiv preprint arXiv:1802.09085*, 2018.

[15] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, 2013.

[16] S. Volos, K. Vaswani, and R. Bruno, "Graviton: trusted execution environments on gpus," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 681–696.

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

[18] Z. Gu, H. Huang, J. Zhang, D. Su, A. Lamba, D. Pendarakis, and I. Molloy, "Securing input data of deep learning inference systems via partitioned enclave execution," *arXiv preprint arXiv:1807.00969*, 2018.

[19] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[20] A. Dosovitskiy and T. Brox, "Inverting visual representations with convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[21] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein, "SVCCA - Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability." *Arxiv preprint arXiv:1102.3176*, vol. stat.ML, 2017.

[22] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[23] F. Zhang, "mbedtls-SGX: a SGX-friendly TLS stack," https://github.com/bl4ck5un/mbedtls-SGX, 2016–2018.

[24] B. Biggio and P. Laskov, "Poisoning attacks against support vector machines," in *In International Conference on Machine Learning (ICML)*. Citeseer, 2012.

[25] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli, "Support vector machines under adversarial label contamination," *Neurocomputing*, vol. 160, pp. 53–62, 2015.

[26] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners," in *AAAI*, 2015.

[27] Y. Liu and S. Ma, "Trojan attack on neural network," https://github.com/PurduePAML/TrojanNN, 2017–2018.

[28] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *British Machine Vision Conference*, 2015.

[29] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

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

[31] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: Information leakage from collaborative deep learning," in *Proceedings of the 24th ACM SIGSAC conference on computer and communications security*, 2017.

[32] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 587–601.

[33] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[34] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014.

[35] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy preserving machine learning." in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017.

[36] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Inference attacks against collaborative learning," *arXiv preprint arXiv:1805.04049*, 2018.

[37] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez *et al.*, "A berkeley view of systems challenges for ai," *arXiv preprint arXiv:1712.05855*, 2017.

[38] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[39] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[40] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[41] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[42] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel software guard extensions (intel sgx) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, ser. HASP 2016. New York, NY, USA: ACM, 2016, pp. 10:1–10:9. [Online]. Available: http://doi.acm.org/10.1145/2948618.2954331

[43] S. Gueron, "A memory encryption engine suitable for general purpose processors," *IACR Cryptology ePrint Archive*, 2016. [Online]. Available: http://eprint.iacr.org/2016/204

[44] P. Jain, S. J. Desai, M. Shih, T. Kim, S. M. Kim, J. Lee, C. Choi, Y. Shin, B. B. Kang, and D. Han, "Opensgx: An open platform for SGX research," in *23nd Annual Network and Distributed System Security Symposium*, 2016.

[45] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, p. 86, 2016.

[46] D. Gupta, B. Mood, J. Feigenbaum, K. Butler, and P. Traynor, "Using intel software guard extensions for efficient two-party secure function evaluation," in *International Conference on Financial Cryptography and Data Security*, 2016.

[47] S. Tamrakar, J. Liu, A. Paverd, J.-E. Ekberg, B. Pinkas, and N. Asokan, "The circle game: Scalable private membership test using trusted hardware," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017.

[48] K. A. Küçük, A. Paverd, A. Martin, N. Asokan, A. Simpson, and R. Ankele, "Exploring the use of intel sgx for secure many-party applications," in *Proceedings of the 1st Workshop on System Software for Trusted Execution*, 2016.

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

[50] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017.

[51] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. R. Pietzuch, and R. Kapitza, "Securekeeper: Confidential zookeeper using intel SGX," in *Proceedings of the 17th International Middleware Conference*, 2016.

[52] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 441–459.

[53] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "Safebricks: Shielding network functions in the cloud," in *NSDI*, 2018.

[54] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa, "Oblix: An efficient oblivious search index," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 279–296.

[55] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A. Sadeghi, "Hardidx: Practical and secure index with SGX," in *Data and Applications Security and Privacy XXXI: 31st Annual IFIP WG 11.3 Conference*, 2017.

APPENDIX

Here we present the detailed architecture and hyperparameters for the two deep neural networks mentioned in Section VI-A. The *Layer* column shows the layer types, including convolutional layer (conv), max pooling layer (max), average pooling layer (avg), softmax layer (softmax), and cost layer (cost). The *Filter* column gives the number of filters in each convolutional layer. The *Size* column is in the format of *width* x *height* / *stride* to represent filter parameters. The *Input* and *Output* columns display the shape of tensors for the input and output of each layer respectively.

TABLE I: 10-Layer Deep Neural Network Architecture for CIFAR-10

Layer	Filter	Size	Input	Output
1 conv	128	3x3/1	28x28x3	28x28x128
2 conv	128	3x3/1	28x28x128	28x28x128
3 max		2x2/2	28x28x128	14x14x128
4 conv	64	3x3/1	14x14x128	14x14x64
5 max		2x2/2	14x14x64	7x7x64
6 conv	128	3x3/1	7x7x64	7x7x128
7 conv	10	1x1/1	7x7x128	7x7x10
8 avg			7x7x10	10
9 softmax				10
10 cost				10

TABLE II: 18-Layer Deep Neural Network Architecture for CIFAR-10

Layer	Filter	Size	Input	Output
1 conv	128	3x3/1	28x28x3	28x28x128
2 conv	128	3x3/1	28x28x128	28x28x128
3 conv	128	3x3/1	28x28x128	28x28x128
4 max		2x2/2	28x28x128	14x14x128
5 dropout	p = 0.50		25088	25088
6 conv	256	3x3/1	14x14x128	14x14x256
7 conv	256	3x3/1	14x14x256	14x14x256
8 conv	256	3x3/1	14x14x256	14x14x256
9 max		2x2/2	14x14x256	7x7x256
10 dropout	p = 0.50		12544	12544
11 conv	512	3x3/1	7x7x256	7x7x512
12 conv	512	3x3/1	7x7x512	7x7x512
13 conv	512	3x3/1	7x7x512	7x7x512
14 dropout	p = 0.50		25088	25088
15 conv	10	1x1/1	7x7x512	7x7x10
16 avg			7x7x10	10
17 softmax				10
18 cost				10