Citadel: Protecting Data Privacy and Model Confidentiality for Collaborative Learning

Chengliang Zhang*, Junzhe Xia*, Baichen Yang*, Huancheng Puyang*, Wei Wang*, Ruichuan Chen†, Istemi Ekin Akkus†, Paarijaat Aditya†, Feng Yan‡
*Hong Kong University of Science and Technology †Nokia Bell Labs ‡University of Nevada, Reno

Abstract

Many organizations own data but have limited machine learning expertise (*data owners*). On the other hand, organizations that have expertise need data from diverse sources to train truly generalizable models (*model owners*). With the advancement of machine learning (ML) and its growing awareness, the data owners would like to pool their data and collaborate with model owners, such that both entities can benefit from the obtained models. In such a collaboration, the data owners want to protect the privacy of its training data, while the model owners desire the confidentiality of the model and the training method that may contain intellectual properties. Existing private ML solutions, such as federated learning and split learning, cannot *simultaneously* meet the privacy requirements of both data and model owners.

We present Citadel, a scalable collaborative ML system that protects both data and model privacy in untrusted infrastructures equipped with Intel SGX. Citadel performs distributed training across multiple *training enclaves* running on behalf of data owners and an *aggregator enclave* on behalf of the model owner. Citadel establishes a strong information barrier between these enclaves by *zero-sum masking* and *hierarchical aggregation* to prevent data/model leakage during collaborative training. Compared with existing SGX-protected systems, Citadel achieves better scalability and stronger privacy guarantees for collaborative ML. Cloud deployment with various ML models shows that Citadel scales to a large number of enclaves with less than 1.73X slowdown.

SoCC '21, November 1–4, 2021, Seattle, WA, USA

ACM ISBN 978-1-4503-8638-8/21/11…$15.00
https://doi.org/10.1145/3472883.3486998

1 Introduction

Building high-quality machine learning (ML) services requires not only extensive ML expertise in feature selection, model design, hyperparameter tuning and testing, but also a large amount of high-quality training data from diverse sources. However, these two requirements are often challenging to be met simultaneously, for instance, in healthcare and financial industries. The siloed data is often held by multiple entities (e.g., hospitals and banks), which are known as *data owners*. Each data owner alone may not have sufficient data nor ML expertise to train high-quality ML models. Thus data owners would like to collaborate with each other as well as the ML solution provider (e.g., technology firms) to build an intelligent service. The solution provider, known as a *model owner*, can provide data owners with an API to access the trained model and charge per use, similar to the prevalent ML-as-a-Service practices adopted in Amazon and Google [6, 24]. Examples include hospitals collaborating with an IT firm to train a diagnostic imaging model over their patients' data [36], and banks pooling data to train an advanced fraud detection model developed by a FinTech company [5].

A key requirement for such collaborative ML is to protect both *data privacy* and *model confidentiality*. For data owners, protecting the data from being revealed to external entities is critical for protecting its business interests, and more often than not, a regulatory requirement [14, 20, 62]. For a model owner, the model is a valuable intellectual property [38, 65, 82]. Revealing proprietary model details (e.g., architecture and weights) can potentially result in losing technological advances to its market competitors. To make matters worse, the exposure of the model raises new security issues in training and inference phases, such as backdoor attacks, membership inference, and model inversion [11, 22, 29, 53].

Prevalent solutions for collaborative ML, such as federated learning [42, 55, 90] and split learning [26, 85], perform training without exposing the participants' data, thus protecting data privacy. However, they fail to protect model confidentiality as the training model needs to be shared fully or partially among participants (see §3.1).

Trusted hardware, such as Intel Software Guard Extensions (SGX) [37], has been used to facilitate collaborative

ML with privacy guarantees for both data and model owners. A common approach is to perform ML training inside a single SGX enclave, where all training data and the model are loaded [35]. However, this approach does not scale to large models nor large training datasets, due to the restricted size of the enclave page cache (EPC) (a few hundreds of megabytes [89]) and the excessive cryptographic overheads associated with evicting EPC pages to the main memory (see §3.3). Other approaches [33, 70, 73] distribute training amongst multiple enclaves but operate under a weaker threat model, thus insufficient for protecting both data privacy and model confidentiality in collaborative ML (see §3.4).

In this paper, we present Citadel, a scalable ML system that enables collaborative learning in untrusted infrastructures by distributing the training across multiple SGX enclaves, with strong privacy guarantees for both data and model owners. Citadel proposes a privacy preserving mechanism to divide SGX-based ML training into training and aggregating parts, making it possible to spin up a distributed cluster to accommodate voluminous multi-sourced data.

To provide these properties, we need to address the following challenges. First, providing privacy to data owners without compromising model confidentiality is not straightforward. SGX is a vehicle to execute trusted ML code, but data owners have to access and review all code running in the enclave to trust it in the first place. Unfortunately, sharing the ML code is against the model owner's interest because it risks exposing proprietary ML techniques (see §2.2). This paradox makes existing solutions either assume that model owners are not interested in stealing data and refrain from sharing code with data owners [33], or ensure that all data owners review and unanimously agree on the code running in the enclave [35, 73]. To our best knowledge, there is yet a practical SGX solution that can cater to both data and model owners' privacy requirements simultaneously.

Second, scaling the system with both data privacy and model confidentiality constraints is difficult. It is already unrealistic to train a state-of-the-art ML model with a large dataset on a single machine within a reasonable amount of time, and adopting SGX only makes the matter worse by introducing steep performance degradation. Without scaling out, an SGX-based solution becomes impractical despite its security advantages. Careful deliberation is required to effectively and efficiently partition workload across multiple enclaves, and to enable them to securely collaborate together.

Third, reaping the security benefits of SGX requires adaptation of ML workloads. Modern software, including ML tool chains, is usually memory-hungry, often using techniques like multiprocessing and single instruction multiple data (SIMD) operations to speedup data processing. However, such techniques cause harmful effects in the memory-restricted SGX context. Our benchmarks demonstrate that the overhead of creating new processes and the excessive memory usage can easily overtake the benefits of parallelization. As a result, besides being able to scale out, reducing SGX overheads within each enclave is critical.

We tackle these challenges with a novel system design in Citadel. First, to earn data owners' trust while protecting a model owner's confidentiality, we introduce two approaches, *zero-sum masking* and *hierarchical aggregation*, to isolate code that is handling data and code that is handling the model, and run these two parts in separate enclaves. Therefore, only the code that has direct access to data shall be shared with data owners to gain trust, while the model handling code remains private to a model owner. Second, with data handling code singled out, we are able to set up multiple such enclaves concurrently to process data in parallel, and aggregate the intermediate results together to update the model. We also delegate the attestation and secret distribution to a centralized service Palæmon [25], so that model and data owners do not have to manage trust and secrets across many enclaves. Third, we employ techniques like hierarchical aggregation and multi-threading with pre-compiled C libraries to make ML workloads adapt to SGX's memory constraints. We have implemented Citadel atop SCONE [8], a secure Linux container framework facilitating confidential computing with SGX. Our implementation supports common distributed ML approaches such as local-update SGD [27, 50, 88], model averaging [27, 55], and relaxed synchronization [30, 49, 94], so that existing ML training applications can migrate to Citadel with minimal efforts. Citadel is open-sourced. We evaluate Citadel with various ML workloads of different sizes on Azure, and confirm that it can effectively speed up training via more enclaves. With 32 training enclaves, we are able to boost the throughput to 4.7X–19.6X compared with those running in a single enclave. We also demonstrate that Citadel achieves the privacy guarantees without significant overhead by comparing with baselines, including Chiron [33] and running Citadel natively without SGX.

2 Collaborative ML and Threat Model

In many application domains such as healthcare and finance, building a high-quality ML model requires the participation of both model and data owners. The model owner (e.g., a tech company or ML developer) has advanced ML expertise but may not have access to diverse training datasets. On the other hand, data owners (e.g., hospitals and retailers) have quality labeled datasets, but any single one may not have enough data samples or ML expertise to build a quality model. An ideal solution enables collaboration among data owners and the model owner, such that the model developed by the latter

can be trained over the data owned by the former, while still preserving data privacy and model confidentiality.

2.1 Entities in Collaborative ML

Collaborative ML typically involves three entities, a model owner, a number of data owners, and a third-party infrastructure such as a public cloud for providing training resources. These entities have different goals in collaborative ML.

Data Owner. Data is a critical asset containing personal or business-sensitive information. In business-to-customer settings, the protection of data privacy is further mandated by regulations (e.g., GDPR [20]). The violation of such requirements can lead to hefty fines and serious legal consequences [21]. Therefore, a data owner wants to protect its data from being exposed to other entities, including the model owner, the cloud, and other data owners.

Model Owner. For the model owner, protecting the confidentiality of the training model (i.e., design and weights) is a top requirement. First, the model is a valuable intellectual property as its development demands tremendous research and engineering effort [65, 82]. Protecting it helps maintain the model owner's technical advances and supports its business as data owners would otherwise train the model by themselves. Second, maintaining the model confidentiality is also a security requirement. Prior work shows that sharing the model with (untrusted) participants poses new threats that are hard to defend against, such as membership inference, model inversion, and backdoor attack [11, 22, 29, 53]. In security-critical applications such as fraud detection and spam filtering, exposing the model details creates a wider attack surface as adversaries can forge attacks to evade the model's defense mechanisms by offline trial and error [66].

In addition, the model owner wants to conceal the *ML training method*, such as optimizer selection [75], gradient manipulation [95], and learning rate schedule [92]. These are critical to the training performance. Selecting, combining, and configuring them require extensive ML expertise, which are part of the model owner's intellectual property.

Third-Party infrastructure. Training complex ML models over large datasets requires a large amount of computational resources, which model and data owners may not have. Therefore, a common practice is to rent a large number of virtual instances in a third-party cloud (e.g., Azure and AWS) and perform distributed training across those instances.

2.2 Threat Model

Training is performed on a third-party cloud trusted by *neither data owners nor the model owner*. The cloud instances, including privileged software like OS and hypervisor, are untrusted. Attacks can be performed by the cloud provider or anyone with access to the OS and hypervisor. On the other hand, data and model owners trust the implementation of the trusted computing base (e.g., Intel SGX) and its attestation service. We also assume that the participants trust standard ML frameworks like TensorFlow [2] and PyTorch [68]. These frameworks are under public scrutiny in open-source communities, and their security is orthogonal to this work.

We assume that data owners and the model owner are *honest but curious*. They faithfully follow the training process specified by our system, as they have no incentive to hinder the training. While doing so, however, data owners may collude with each other to obtain the model and methods to perform training on their own. Data owners also want to pry on each other's data to improve their competitiveness in the same business sector. The model owner, on the other hand, may want to access the training data for illicit use.

The model owner might theoretically engineer the model to preserve information from input data. Such a vulnerability could happen as it stems from the concept of model confidentiality and the information retaining nature of ML models. There are potential routes to address this issue. First, Citadel can potentially export the trained model to a secure enclave directly for deployment, thus eliminating the model owner's access to plaintext model after training. Second, Citadel can get a third-party entity, which has no conflict of interest (e.g., a government agency or a neutral authority), involved to verify that the model is not maliciously engineered [61].

What does Citadel not protect? We do not address denial-of-service attacks, side-channel attacks [84] and rollback attacks [67], as there have been complementary mechanisms to prevent them [12, 43, 44, 64]. Besides, we do not protect against membership inference and data extraction attacks from model owners, similar to the state-of-the-art federated learning approaches and other SGX-based solutions [78].

3 Prior Arts and Their Insufficiency

In this section, we describe why prior work is insufficient to protect data privacy and model confidentiality for collaborative learning under the threat model introduced in §2.2. We start by introducing existing solutions designed for different collaborative learning scenarios and explaining why they cannot be applied here. We then introduce the SGX-based solutions, which are the most related to our work, and discuss their problems to achieve scalable collaborative learning.

3.1 Solutions for Collaborative Learning

Existing solutions to collaborative ML focus on data privacy without model confidentiality.

Federated learning (FL) is a computing paradigm in which multiple clients collaboratively train a shared model by uploading their local updates to a central server for aggregation without exposing private training data [42, 90]. FL employs

various security measures to protect data privacy for clients, including secure multi-party computation (SMPC) [13, 19, 58, 59, 80, 90, 96], differential privacy (DP) [23, 54, 56, 77], and homomorphic encryption (HE) [16, 51, 52, 69, 93]. However, FL is not designed to protect model confidentiality as the training model is shared among all clients, each being both a model owner and a data owner.

Split Learning (SL) offers an alternative approach to collaborative ML for training deep neural networks [26, 85]. In SL, a neural network is split into two parts at a certain layer, called a *cut layer*. The model owner releases the neural network up to the cut layer to data owners, while keeping the remaining layers private. Data owners train the network up to the cut layer with their private data and send the updates to a central server, based on which the model owner trains the remaining network. While this scheme preserves data privacy, the model confidentiality cannot be fully protected, as the neural network up to the cut layer is shared among data owners. In addition, the parameters of neural network are only accessible to data owners, meaning the model owner cannot access the complete network of the trained model.

Given that these collaborative learning solutions assume the actual training is distributed among data owners, they have fundamentally different system architectures and cannot be easily adapted to a scenario under our threat model.

3.2 Intel SGX

ML training requires access to both data and model. To protect their privacy, the training must be performed at a secure place trusted by both data and model owners. Trusted hardware like Intel SGX (Software Guarded Extensions [37]) offers a viable solution to create a trusted execution environment (TEE) even if the underlying platform is untrusted.

Intel SGX is one of the most widely available hardware-assisted TEE, along with other implementations such as ARM TrustZone [7] and AMD Secure Memory Encryption (SME) [1]. It sets aside a protected memory region, called an *enclave*, within an application's address space. Code execution and memory access in the enclave are strongly isolated from external programs. The processor ensures that only code running in an enclave can access the data loaded into it. External programs, including the operating system and hypervisor, can only invoke code inside an enclave at the statically-defined entry points. SGX also supports remote *attestation*, which allows a remote user to verify that the initial code and data loaded into an enclave match a given cryptographic hash, hence creating trust that the enclave will perform the expected computation.

However, the enclave's hardware-protected confidentiality and integrity come with a steep price for performance. First, as the host platform is untrusted, copying between CPU and enclave memory must be protected to prevent memory bus snooping. SGX uses memory encryption engine (MEE) to transparently encrypt and decrypt data exchanges through memory bus, incurring 2X-3X performance overhead than native executions [34]. Second, the performance of an enclave is usually bounded by the enclave page cache (EPC) size, a hardware-protected memory region used to host the enclave pages. The EPC is usually small, e.g., only 168MB in the most expensive Azure confidential computing instance [10]. Any memory usage beyond the EPC will cause enclave pages to evict to the unprotected main memory. To ensure the confidentiality and integrity of the evicted EPC pages, SGX uses symmetric key cryptography, which compounds to a large overhead as the number of evictions increases. Such overhead can be mitigated by optimizing code to avoid paging as much as possible. Third, because system calls still need to be facilitated outside the enclaves, there is a substantial context switching overhead. State-of-the-art SGX systems often avoid system calls like I/O and threading [8].

Software Fault Isolation (SFI) [87] is a software instrumentation technique for sandboxing untrusted modules, preventing codes from accessing others' secrets. It has been adopted lately in Occlum [76] and Chancel [3] to provide certain levels of secret isolation. However, SFI alone cannot provide strong security up to our threat model. For example, a malicious model owner can run codes to compress users' data and export the unidentifiable data through SFI's defined exit. Consequently, we believe verification for codes with direct data access is essential.

3.3 Private ML with a Single SGX Enclave

One solution to private collaborative ML is to use a *single* SGX enclave attested by both data owners and the model owner (Fig. 1).[1] The training is performed in a remote enclave running on an untrusted host (e.g., a cloud server). Before the training begins, a data (or model) owner generates a private symmetric key and uses it to encrypt the data (or model). The encrypted data and model are then uploaded to an unprotected storage on the host, which does not have the key. The host then creates an enclave containing the *agreed-upon ML code* by all secret (i.e., model and data) owners, and lets them initiate attestation to ensure the integrity and correctness of the initialized enclave. Once the attestation is successful, each secret owner uploads its encryption key to the enclave over a TLS-protected channel, with which the enclave can retrieve the encrypted data and model from the storage and decrypt them. The training starts once the data, model, and ML code are all loaded into the enclave. When the

[1]This design is an extension to [35], in which data owners also own the training model, similar to the FL setting.

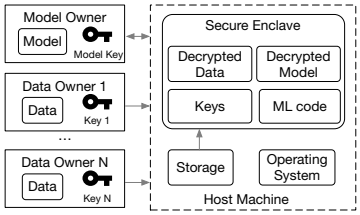
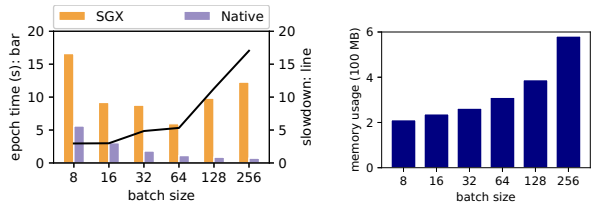


Figure 1: Illustration of a single-enclave that protects the confidentiality of both data and model.



(a) Training epoch time and slowdown w.r.t. batch size

(b) Memory usage w.r.t. batch size

Figure 2: Epoch time under SGX and native mode. The slowdown (line) represents the ratio between SGX and native time. Memory refers to active SGX memory.

training completes, the model owner downloads the model, and the enclave is destroyed along with the contained data.

Poor Scalability. Such a design, however, does not scale to a large training dataset. To illustrate this problem, we characterize its performance with Azure's latest confidential computing offering `DCsv2` [10]. We run experiments in a `Standard_DC8_v2` instance, the largest in `DCsv2` with 8 vCPUs and 32 GB memory, of which 168MB is dedicated to an enclave's EPC. We train AlexNet [46] over images of size $32 \times 32 \times 3$ with TensorSCONE [47], an SGX-optimized version of TensorFlow v1.15. We then run the same training workload with the unmodified TensorFlow outside the enclave. Note that, to speed up training on a single machine without accelerators, one common technique is to configure a large batch size for increased parallelism and reduced iterations. We evaluate the training epoch time (time needed to finish processing the entire dataset) with varying batch sizes in SGX and the native environment, respectively.

When the batch size is small (8 or 16), running in SGX is only 2.9X slower than running natively outside of the enclave, meeting the expected performance of TensorSCONE [47] (Fig. 2a). Such slowdown is mainly due to the MEE encryption overhead but not EPC paging, as memory usage is barely over the EPC size (Fig. 2b). Further increasing the batch size leads to more parallelism, which reduces the epoch time in the native mode. This trend does not hold in SGX: as the batch size increases, the epoch time first reduces but then surges rapidly, a consequence of frequent EPC paging due to excessive memory usage beyond the EPC size (Fig. 2b). Therefore, one cannot expect to scale ML training by configuring a large batch size in an enclave: with a batch size of 256, the slowdown with SGX is about 17X (Fig. 2a).

Training Logic Exposure. Note that, in the single-enclave solution, the ML code must be shared and agreed by all data owners to ensure that it contains no malicious code that could harm their privacy (e.g., writing data to an external storage). However, this sharing inevitably reveals the details of the model update logic (e.g., optimizer selection, learning rate scheduling, and gradient manipulation), which the model owner may consider as intellectual property (§2).

3.4 Private ML with Multiple SGX Enclaves

As model training in a single enclave does not scale, recent works propose distributed solutions with multiple enclaves. Notably, Ping An [70] augments FL with SGX enclaves at the data owners' side for enhanced data privacy while exploiting data parallelism, but the data owners can still access the training model. Chiron [33], built atop Ryoan [34], ensures model confidentiality for ML-as-a-Service providers with SGX enclaves, and supports running multiple training enclaves in parallel. However, its design is based on the assumption that the model owner (i.e., MLaaS provider) is not interested in harvesting data owners' data, which may not be the case in collaborative ML. The two approaches cannot be combined to complement each other, as Chiron places the training data in cloud servers, which is not allowed in Ping An. SecureTF [73] presents a modified TensorFlow to support distributed training in multiple enclaves, but assumes that model and data belong to the same entity, and hence cannot be applied to collaborative ML. PPFL [57] adopts TEEs on mobile devices to train the final layers of a proprietary model in the context of cross-device federated transfer learning. The data passes through a public base model, and the intermediate results are then fed into the TEEs holding secret model layers. In this design, only parts of the final model can remain private. Substantial information can also be inferred by a malicious model owner within the blackbox enclaves as they are not verified by data owners [69]. Furthermore, mobile devices' constrained TEE memory greatly limits the ML model size and the training efficiency. To our knowledge, a scalable collaborative ML system that protects the privacy for both model and data owners is still lacking.

4 Citadel Design

We aspire to devise an ML system that not only preserves data and model privacy simultaneously, but also enables distributed training across multiple SGX enclaves. To do so, we securely partition training workload, and make part of it replicable. Citadel achieves this by separating and isolating *data handling code* and *model handling code*. The former

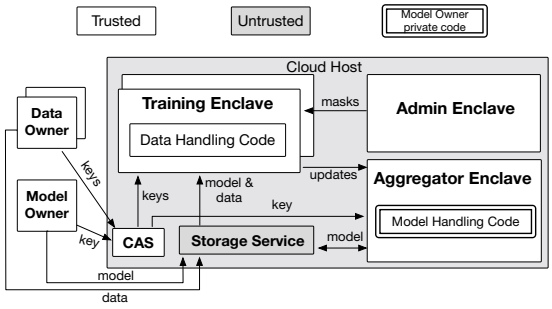


Figure 3: An overview of Citadel. All codes (except the model handling code) are open-sourced to gain data owner's trust, while the model handling codes remain private to model owner.

can be shared with data owners to gain their trust, while the latter remains private to the model owner. Afterwards, a *barrier* has to be inserted between the two parts to ensure the model handling code cannot recover data owners' data with its private code. With data handling codes isolated securely, Citadel is able to accelerate training through data parallelism.

4.1 Design Overview

Fig. 3 illustrates an architectural overview of our Citadel system. It facilitates collaborative ML in *multiple* enclaves hosted in untrusted infrastructure. These enclaves can run in a single or multiple cloud instances. A data (or model) owner communicates with Citadel through a *client* running on a local machine. The client includes a *verifier* which the data (or model) owner uses to attest Citadel. It also provides a *key manager* with which the owner generates a symmetric encryption key and uses it to encrypt data (or model). The client uploads the encrypted data (or model) to a general cloud storage service. The storage itself does not need to be trusted since the secrets are encrypted. Citadel launches multiple enclaves on behalf of data and model owners, establishes trust between the enclaves and the owners via attestation (handled by CAS, configuration and attestation service), and performs distributed training in those enclaves. Specifically, Citadel runs three types of enclaves: *training enclaves*, *aggregator enclave*, and *admin enclave*.

Training Enclave. In Citadel, each data owner has a dedicated training enclave, which needs to be attested by both the corresponding data owner(s) and the model owner to gain their trust. It takes private data as input, runs data handling code (e.g., compute gradient updates) provided by the model owner, and generates model updates. As the code has direct access to the training data, it must be shared to and agreed by the data owner. The code should reveal no model information that the model owner wants to protect, such as hyper-parameter configuration and the training model. Instead, it loads such information as *environment variables* and *non-executable binary model files* from the storage service. Note that, it is not possible to inject malicious code into the model files, as they are non-executable in the standard ML toolchains. After the model updates are computed, the training enclave sends them to the aggregator enclave for global aggregation. Citadel currently does not consider the scenario where models are too big for a single enclave. Such an extension could be achieved by either increasing EPC size with a specialized SGX card [15], or applying existing model parallelism techniques to split large models [28, 32, 91].

Aggregator Enclave. Citadel launches an aggregator enclave on behalf of the model owner to run the model handling code. Each training job has only one such enclave, and it is attested by the model owner only. It collects aggregated updates (§ 4.2) from all training enclaves and updates the model for the next training iteration. The updated model is encrypted and stored in storage service, so that the training enclaves can start the next iteration after retrieving it. As the aggregator enclave has no access to data from data owners, the code running inside remains private to the model owner. This prevents sensitive training methods developed by the model owner from being revealed (e.g., learning rate schedule, optimizer selection, gradient selection and manipulation), which are required for aggregation.

Admin Enclave. Citadel launches an admin enclave for a training job, uses it to schedule the training workload and orchestrate the associated training and aggregator enclaves. Code running inside an admin enclave (i.e., *mask generator* described later in §4.2 and *enclave scheduler*) are open-sourced for public access. The enclave itself is attested by all model and data owners. To facilitate communication between an enclave and external entities, Citadel provides open-source utilities that run as part of *admin code* in a training or aggregator enclave. As the cloud host's network is untrusted, communications inside Citadel are secured by TLS connections with endpoints located inside the enclaves.

Attestation with CAS. In Citadel, a secret owner needs to attest multiple enclaves to ensure the integrity and confidentiality of the data and code. The default SGX attestation can be tedious as it attests one enclave at a time. CAS (configuration and attestation service) offers a simplified solution to secret management and attestation. CAS itself is open-sourced and runs in an enclave, which the model and data owners can verify and attest. Once the secret owners have established trust via CAS, they can delegate their encryption keys to it, and instruct it on how to maintain their security including which enclave can access which secrets and which code should run in which enclave. CAS faithfully follows the specified security policy, attesting each enclave on behalf of the data or model owners and supplying the enclave with

secrets once it is trusted. With the help of CAS, model and data owners only have to initiate the attestation process once. Citadel employs PALÆMON [25], a trust management service built on top of SCONE [8], as its CAS system.

Fault Tolerance. Citadel's training enclaves are *stateless* by design, because model and data are all stored into and fetched from a storage system. In case of training enclave failures, Citadel can easily launch replacements and resume the training process via restarting the ongoing iteration. The training progress is always checkpointed since the updated model is encrypted and stored into storage after each iteration. If admin or aggregator enclaves fail, Citadel can also similarly restart the cluster and continue training.

4.2 Separating Data and Model Handling

A key design in Citadel is to separate the model owner's ML code into two parts: *model handling code* and *data handling code*. The model handling code computes the global model updates based on the gradients received from the training enclaves. As such, it concerns with potentially confidential methods and values. Citadel runs the model handling code in the aggregator enclave. In contrast, the data handling code is shared with data owners (i.e., open-sourced) to gain their trust. It handles standard forward and backward propagation mechanisms, and has access to data owners' private data.

This separation provides the model owners with model confidentiality: Citadel ensures that the data owners only see *placeholders* for the model and hyperparameters, which are loaded dynamically into training enclaves after attestation (§4.1). The secrets to load these values and replace the placeholders are only shared after the attestation, such that model and hyperparameters remain unknown to data owners.

On the other hand, this separation alone does not fully provide data privacy for data owners. Although data owners can verify the open-source data handling code and ensure it does not leak private data, prior work shows that a data owner's training data can still be inferred accurately from computed gradients [69]. Citadel addresses this problem to protect data privacy with two methods: First, data owners do not receive intermediate models from the model owner, such that they cannot pry into other data owners' data. Second, a *barrier* is inserted between the training enclaves and the aggregator enclave, so that the model owner only receives *aggregated* updates but not the raw updates from any individual training enclave. Specifically, we propose two mechanisms for such a barrier: *zero-sum masking* and *hierarchical aggregation*.

4.2.1 Zero-Sum Masking. Zero-sum masking, originally proposed for federated learning (FL) as a way to implement secure aggregation [13], allows data owners to collectively generate masks and apply them to their individual updates

```
# download, decrypt and, process data with data owner's key
train_data = download_decrypt_data(data,
     os.environ[DATA_KEY])
train_x, train_y = preprocess_data(train_data)
# download, decrypt, and load model with model owner's key
model = download_decrypt_model(os.environ[MODEL_KEY])
TH, ml_toolchain.load_model(model)
# training
gradients = ml_toolchain.train(train_x, train_y,
     os.environ[BATCH_SIZE])
# request and apply mask, send only masked gradients
mask = citadel.get_mask()
masked_gradients = mask + gradients
citadel.send(masked_gradients)
```

Pseudocode 1: Training Enclave.

```
# initiate sum and masks
sum = 0, masks = []
# generate N - 1 random masks
for i in range(N - 1):
  mask_i = rand.generate_mask(shape)
  masks.append(mask_i)
  sum += mask_i
# add the last mask so that the total is 0
masks.append(-sum)
# distribute masks to training enclaves
citadel.distribute_masks(masks)
```

Pseudocode 2: Admin Enclave.

```
# collect and aggregate gradients from training enclaves
all_masked_gradients = citadel.wait()
aggregated_gradients = numpy.sum(all_masked_gradients)
# download, decrypt, and load model with model owner's key
model = download_decrypt_model(os.environ[MODEL_KEY])
ml_toolchain.load_model(model)
optimizer =
     ml_toolchain.optimizer(os.environ[LEARNING_RATE_SCHEDULE])
# update model
clipped_gradients = ml_toolchain.clip(aggregated_gradients)
updated_model = ml_toolchain.apply(clipped_gradients)
# encrypt and save the updated model for next round
citadel.save(updated_model, os.environ[MODEL_KEY])
```

Pseudocode 3: Aggregator Enclave.

before sending them to the aggregator. The masks are generated in such a way that they are canceled out when summed up, so that the updates are correctly aggregated. Since the aggregator does not have access to individual masks, it cannot learn the raw gradients from any data owner.

Inspired by such a concept, we propose a new zero-sum masking scheme for Citadel. Compared with the FL setting, where the masks have to be generated among distributed data owners, TEE enables us to execute code that is verified and trusted by the involved parties, so we can opt to a *centralized* mask generation approach.

As shown in Pseudocode 1-3, N masks $m_0, m_1, ..., m_{N-1}$ are generated for N training enclaves by the admin enclave

trusted by all secret owners (i.e., data and model owners). These masks are of the same size as the model gradients, with a sum being zero: $\sum_{i=0}^{N-1} m_i = 0$. After the training starts, each training enclave first downloads and decrypts a fresh model from the storage service, and then computes gradients with the code shared and verified by data owners. The training enclave i then requests admin enclave for the mask m_i, applies it to its gradients, and finally sends them to the aggregator enclave. The aggregator enclave collects the masked gradients from all training enclaves, aggregates them, and updates the model using the specified model update method. As an individual update from each training enclave is obscured with a random mask, the model owner's private model handling code cannot infer any information about the training data. By aggregating all updates together, the masks cancel each other out, resulting in the same aggregated update as it would have been without masking. The security of this approach is based on the fact that if data owners' values have uniformly random pairwise masks added to them, then the resulting values appear uniformly random, conditioned on their sum being equal to the sum of data owner's values, as proven in [13].

Our centralized zero-sum masking approach protects the model confidentiality, while guaranteeing the same level of privacy for data owners as in secure aggregation [13], but without the time-consuming synchronous distributed mask generation. Although the codes in training enclaves are shared with data owners, the secrets are omitted; thus, protected. Static secrets like model weights, encryption keys, and training batch size are shared as placeholders, and are only dynamically loaded as environment variables once the enclaves pass attestation. The aggregator codes remain private to the model owner because the raw training data and gradients produced from individual training enclaves are no longer accessible. Consequently, model owners retain intellectual properties like gradient manipulation and learning rate schedules.

Offline Mask Generation. In our zero-sum masking approach, protecting the mask confidentiality is the key to shielding individual updates from the model owner and the cloud provider. Therefore, the masks have to be generated within the admin enclave and encrypted before leaving the enclave. However, with an increasing number of training enclaves, the compute-intensive mask generation and bandwidth-intensive mask distribution would inevitably make the admin enclave a performance bottleneck.

To address this problem, we choose to generate masks *offline* and *offload* mask distribution to the untrusted storage service after encrypting the generated masks. Before the training starts, the admin enclave generates sufficient sets of N masks, encrypts each mask with a separate key and stores them in the storage service. During training, upon receiving a masking request from a training enclave, the admin enclave redirects the request to the storage service, and provides the training enclave with the corresponding decryption key. As such, the heavy tasks of masking are removed from the critical path.

To handle potential training enclave stragglers, Citadel optionally employs a relaxed consistency model like SSP [30]: assuming the first K out of N training enclaves would participate in the aggregation, the admin enclave can return the sum of the remaining pre-generated masks $\sum_{i=K}^{N} m_i$ to enclave K, thus ensuring the overall sum remains zero.

4.2.2 Hierarchical Aggregation. The zero-sum masking solution requires *one-to-all* and *all-to-one* synchronization in the mask distribution (between the admin enclave and training enclaves) and aggregation (between training enclaves and the aggregator enclave) phases. As more training enclaves run in the system, such synchronization overheads become increasingly prominent. In fact, given SGX's memory limitations, neither generating a large number of masks within an admin enclave nor aggregating large updates within an aggregator enclave scales.

To address this potential issue at large scale, we propose to establish a *tree-structured hierarchical aggregation* among training enclaves. Since our goal is to protect individual updates from being learned by the aggregator enclave, we can utilize training enclaves to aggregate the intermediate results, which are trusted by data owners. As described in Algorithm 1, after processing a batch, each training enclave holds its own gradients and follows a tree-structured hierarchical aggregation scheme to accumulate gradients. Citadel does not require ring-reduce like communication as the update propagation pattern is all-to-one instead of all-to-all.

Assume there are N training enclaves, and each *leader* in the aggregation tree has C children ($C-1$ neighbors need to transfer their updates to the leader in one round). It requires $\lceil log_C N \rceil + 1$ rounds of aggregation (height of the aggregation tree), and on the l^{th} level, there are $\lceil ...\lceil \lceil N/C \rceil / C \rceil ... \rceil$ active nodes remaining. On the l^{th} level, we denote the i^{th} remaining active node by id_i^l, so that each of these active nodes sends its aggregated results from last round to a leader node $id_{\lfloor N/C \rfloor}^l$. The recursion continues until the last leader accumulates the final results and sends it to the aggregator enclave. Hierarchical aggregation avoids the expensive all-to-one synchronization, eliminating communication hotspots.

4.2.3 Comparison of Two Approaches. Both zero-sum masking and hierarchical aggregation effectively shield the raw updates from the aggregator enclave. Zero-sum masking requires an all-to-one communication from all training enclaves, and then all the updates have to be aggregated in

Algorithm 1 Citadel with Hierarchical Aggregation

Training Enclave *i*:

```
1: function StartsTraining
2:     for epoch e = 0, 1, 2, ..., E do
3:         for all training batch t = 0, 1, 2, · · · , T do
4:             Download fresh model model, compute gradients g_i^(e,t)
5:             Call HierarchicalAggregate to start aggregation.
6: function HierarchicalAggregate
7:     if I am leader in this round. then
8:         Collect and accumulate intermediate results.
9:         if I am the final leader. then
10:            Send aggregated result G^(e,t) to aggregator.
11:        else
12:            Call HierarchicalAggregate to send results to next leader.
13:    else
14:        Send results to the leader.
```

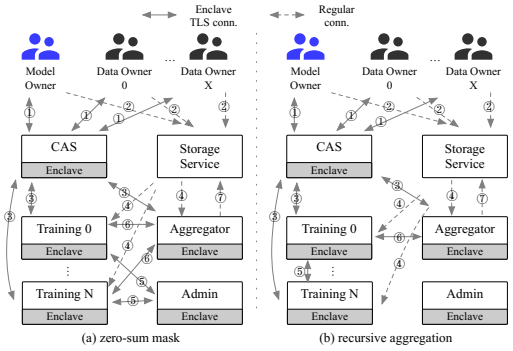


Figure 4: The workflow of Citadel with zero-sum masking. Enclave TLS connections terminate within enclaves.

the EPC-limited aggregator enclave, so the overall overhead grows as more training enclaves run in the system.

Hierarchical aggregation, on the other hand, breaks the all-to-one communication pattern into a hierarchical aggregation tree. Although the potential network congestion is mitigated, extra cryptographic operations are needed to protect the communication connections on the aggregation tree. However, it is difficult, if not impossible, to quantitatively justify such trade-off in this scenario, as the time needed to finish a certain operation within an enclave depends on both the memory footprint and the memory access pattern. Assume there are N training enclaves. Let $t_{net}(x)$, $t_{enc}(x)$, and $t_{dec}(x)$ respectively denote the time needed to transfer, encrypt, and decrypt message x. Let t_{mask}, t_{train}, and t_{apply} be the computation time needed to apply a mask, generate gradients, and apply gradients to model, respectively, and $t_{agg}(k)$ the time spent on aggregating updates from k training enclaves. The iteration time for zero-sum masking $t_{mask}(N)$ is estimated as

$$t_{mask}(N) = t_{train} + t_{net}(m) + t_{dec}(m) + t_{mask} + t_{enc}(g) + t_{net}(g) + t_{dec}(g) + t_{agg}(N) + t_{apply},$$

where m and g stand for a set of mask and gradients, respectively. Assuming each node in the aggregation tree has C children, the iteration time for hierarchical aggregation $t_{tree}(N, C)$ is estimated as

$$t_{tree}(N, C) = (t_{enc}(g) + t_{dec}(g) + t_{agg}(C) + t_{net}(g)) \times (\lceil log_C N \rceil + 1) + t_{train} + t_{apply}.$$

As a general guideline, zero-sum masking tends to work better on *smaller models* with *fewer* training enclaves, as the memory footprint within the aggregator is smaller and network congestion is less likely. When there is a large number of training enclaves, hierarchical aggregation becomes more favorable. We will evaluate the two approaches in §6.

We stress that both zero-sum masking and hierarchical aggregation require no change to distributed training algorithms, thus introducing no model accuracy loss on this front. Citadel performs extra floating point additions during zero-sum masking and hierarchical aggregation compared to vanilla distributed training, leading to tiny numerical errors due to the exponent-alignment. Extensive study shows that such small errors do not affect ML accuracy [79].

4.3 Citadel Workflow

Putting it all together, we elaborate on Citadel's workflow with the two aggregation approaches. The workflow of zero-sum masking is depicted in Fig. 4a. A training iteration is broken down as follows. ① Model and data owners attest CAS, and share their encryption keys with CAS. ② Model and data owners upload their encrypted model and data to storage service. ③ CAS attests training and admin enclaves on behalf of data owners, then shares data encryption keys with training enclaves. CAS also attests training, admin, and aggregator enclaves on behalf of model owner, then shares the model decryption key to training and aggregator enclaves. ④ Training enclaves fetch corresponding data and model, decrypt them and compute gradients; aggregator downloads and decrypts model. ⑤ Training enclaves ask admin enclave for masks and have the requests redirected to storage service with mask decryption keys. ⑥ Training enclaves fetch masks from storage service, decrypt and apply them to updates, and send masked updates to aggregator. ⑦ Aggregator collects masked updates, summarizes them, and updates global model. The model is then encrypted and uploaded to storage.

Similarly, Fig. 4b depicts the workflow of hierarchical aggregation, where Steps ①-④ are the same as masking approach. ⑤ Training enclaves hierarchically aggregate all updates until the final sum of all updates is available at training enclave 0. ⑥ The aggregator enclave receives the final update and applies it to the model. ⑦ The model is then encrypted and uploaded to storage service.

Security Against Attacks. Citadel's security stems from three levels. The first level of security is provided by Intel SGX. By verifying training enclaves and Citadel codes, data owners are ensured that data cannot be abused nor recovered. As the second level, we reduce the possibility of data leak by splitting data handling and the model update parts. This split ensures that model owners cannot access raw gradient updates. It also ensures that model owners' secrets are protected from data owners, as the model update code remains private. For the third level, placeholders (instead of actual values) are added in data handling codes shared with data owners, secrets such as model weights are protected.

SGX and its attestation ability enable Citadel to prevent the scenario described in [13] where the model owner forms a *collusion* with some data owners to steal other data owners' data. Such attacks require additional codes to be executed in the enclaves, and can be identified in the code verification process by the non-colluding data owners. Model inversion attacks [22] from data owners are avoided compared with conventional methods like FL, because the code running in training enclaves is verified by both data and model owners and is strictly enforced by SGX. Data owners can no longer insert backdoors into the global model by tampering updates, or perform membership inference attacks [78], or conduct data extraction attacks [22, 29], because data owners have no access to the intermediate training results and the model, which only exist in enclaves.

5 Implementation

In this section, we describe the implementation details of Citadel. We base our implementation on SCONE [8], but it can also be extended to other SGX-enabling frameworks such as Graphene [83] and Ryoan [34]. We use MongoDB [60] as the storage service, and containerize all system components and orchestrate them in Kubernetes [39]. Our implementation consists of 5,000 lines of Python code and Linux Shell script, and is open-sourced.[2]

Trusted Computing Base (TCB). For an easy support of SGX and multiple enclave orchestration, we adopt SCONE [8] in our system. SCONE provides SGX-protected Linux containers, so that we can utilize tools like Docker [18] and Kubernetes [39] to orchestrate enclaves. The SGX provided function `sgx_read_rand` is used to generate randomness, and the attacks [74] on such generators are beyond the scope of Citadel. In order to establish trust, users have to verify codes running in the enclaves. Such verification is non-trivial, and requires some domain knowledge. However, since Citadel and most ML toolchains are open-sourced, we believe wider adoption and public scrutiny can offload individual user's verification burden greatly. Besides, research work like [25] is making SGX trust management easier and more accessible.

[2]The source code of Citadel is available at [17].

Efficient Encryption & Decryption. As the host infrastructure is not trusted, encrypted data and models must be decrypted within the enclaves. In addition, network connections between enclaves must be secured with TLS. These requirements result in substantial cryptographic operations performed inside an enclave. Especially during the aggregation process, a single enclave has to decrypt results from multiple enclaves and add them up. Therefore, the efficiency of cryptographic inside an enclave plays an important role in overall performance of Citadel.

In a native setting without SGX, one way to increase performance is to increase the parallelism with multi-processing or multi-threading. However, inside an SGX enclave, each process runs inside its own enclave, so launching new processes is extremely slow as it requires to set up new enclaves and initialize EPC pages. Furthermore, the new subprocess enclaves contend with the parent enclave for EPC, resulting in performance degradation for all of them. Our experimental evaluations with the OpenSSL implementation of `AES-256-CBC` shows that, encrypting and decrypting 16 AlexNet [45] models with multi-processing is at least 2X slower than processing them serially without multi-processing in SGX. On the other hand, SCONE [8] provides efficient *user-level threading* to avoid costly system calls, so it is possible for us to improve cryptographic operations with multi-threading. However, due to Python's Global Interpreter Lock (GIL) [72], only one Python thread can run at any given time even with multi-threading. To overcome this hurdle, we implement our cryptographic operations in C++ and compile it with C Foreign Function Interface (CFFI) [71]. This not only allows us to bypass the GIL limitation, but also exploits the highly efficient performance of native code.

6 Evaluation

In this section, we evaluate the performance of Citadel with representative ML models trained on a public cloud. We first examine the scalability of Citadel with zero-sum masking in clusters of various sizes. We then evaluate hierarchical aggregation with different configurations to quantify how avoiding all-to-one communication helps improve system scalability. Finally, we assess the system overhead of our design by comparing Citadel with three baselines: Chiron [33], the single-enclave approach and native Citadel without SGX.

6.1 Methodology

Settings. We consider a distributed ML setting where all instances are located within the same cluster. We do not consider geo-distributed training, as Citadel enables data

to be centralized securely in a verifiable manner, eliminating the necessity of geo-distribution like FL. We conduct all experiments on Azure confidential computing instances with SGX support in `Canada Central` region with the instance `Standard_DC4s_v2`, which has 4 vCPUs, 16 GB of memory, and 112 MB of EPC memory. We deploy exactly one enclave on each instance to avoid EPC contention. To emulate multiple data owners, we randomly partition these datasets into multiple shards and encrypt them with different keys before uploading them to Citadel. The scale of our evaluation is limited to 34 such instances (including training, aggregator and admin enclaves), because Azure limits the total number of `DCsv2` family vCPUs for non-enterprise users. Nevertheless, we believe the trend demonstrated in our evaluation applies to larger scale, and is sufficient to validate our implementation.

Benchmarking Models. We have implemented four ML models with their respective privacy requirements, using TensorFlow v1.15. The first two, AlexNetS and AlexNetL, belong to the same application where a certain number of hospitals collaborate with a medical tech company to train a Retinopathy diabetes diagnosis model [40]. The input images are scaled to $32 \times 32 \times 3$ for AlexNetS and $96 \times 96 \times 3$ for AlexNetL, respectively. AlexNetS has 1.25M trainable parameters while AlexNetL has 15.9M trainable parameters. The third one SpamNet is a spam filtering model utilizing LSTM [31] network with 9.6k trainable parameters, where SMS messages [41] is input data. Here, the model is required to be private and the SMS messages are sensitive. The last one MNIST is a 12-layer CNN handwriting recognition model trained with MNIST dataset [48]. MNIST model has 887.5K trainable parameters. The model owner wants to protect its intellectual property, while data owners want to remain anonymous as the adversary may forge their handwriting.

The above four workloads are backed by DL models of various sizes and cover diverse tasks. Note that the models and input data dimensions used to evaluate SGX systems in literature are usually of modest sizes, due to memory limits and overhead of SGX's EPC. For example, secureTF [73] only adopts MNIST dataset in distributed evaluation across 3 servers; Chiron only has simulation-based evaluation, and the biggest model investigated is AlexNet with 6 million parameters. Compared with the state-of-the-art, our evaluations brought the training to include much larger models with realistic inputs at an industrial scale.

Baselines. We use three baselines for comparison, Chiron [33], the `single-enclave` approach described in §3.3, and `native-distributed` that runs Citadel natively without SGX. Compared with these baselines, we show how Citadel provides strong privacy and confidentiality, while still achieving high throughput.

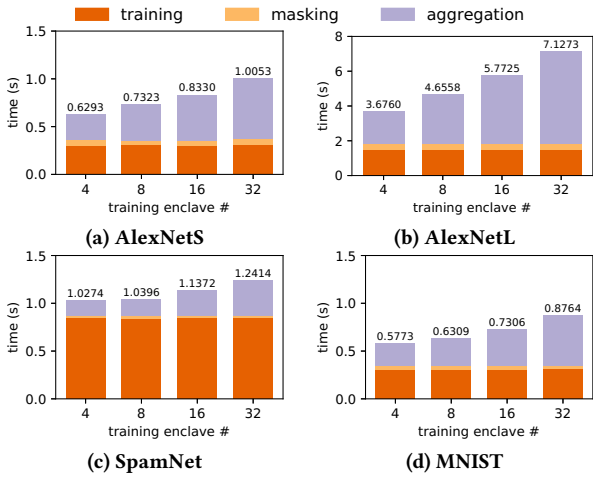


Figure 5: The iteration breakdown w.r.t. training enclave numbers with zero-sum masking adopted.

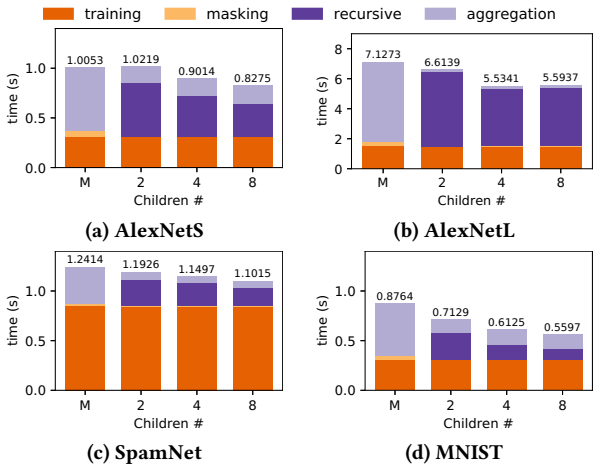
6.2 Effectiveness of Zero-Sum Masking

We first evaluate the effectiveness of Citadel's zero-sum masking technique. We report the iteration time breakdown in Figs. 5a-5d. The iteration time is measured as the timespan from downloading fresh models in training enclaves until the aggregator enclave uploads the updated model. Specifically, the `training` portion refers to the time spent inside training enclaves, but excludes mask-related operations. The `masking` portion includes the time spent on requesting, downloading, decrypting and applying the masks. The `aggregation` portion covers the time spent in the aggregator enclave, and the time used to receive all masked updates. All results are averaged across all enclaves over multiple iterations.

As we can see, Citadel with zero-sum masking scales well with increasing number of training enclaves. Even increasing training enclaves from 4 to 32, the overall iteration time only increases by 59.7% for AlexNetS, 93.9% for AlexNetL, 20.8% for SpamNet, and 53.5% for MNIST. Looking into each portion, we see that: 1) training time stays constant when Citadel scales out as training operations are irrelevant to cluster size, 2) masking time stays constant thanks to offline mask generation (§4.2.1), and 3) aggregation time increases (inevitably) because aggregation involves all-to-one communication and the summing-up of all masked gradient updates. Altogether, these results show only a modest increase of iteration time as the cluster size grows, indicating that Citadel can accommodate a large number of data owners and complex models with reasonable performance overhead.

6.3 Hierarchical Aggregation

Although §6.2 exhibits that Citadel with zero-sum masking can effectively increase throughput by adding more training enclaves, we also notice the significant aggregation overhead



(a) AlexNetS (b) AlexNetL

(c) SpamNet (d) MNIST

Figure 6: The hierarchical aggregation iteration time breakdowns w.r.t. children number. The zero-sum masking results with 32 enclaves are shown as *M* bars.

with increased cluster size. In this subsection, we evaluate Citadel's hierarchical aggregation approach, and validate if it can reduce the aggregation overhead. The results are shown in Figs. 6a-6d. We target the scenario with 32 training enclaves which is the largest cluster we are able to run in Azure. We test hierarchical aggregation with aggregation tree children set to 2, 4 and 8, and use zero-sum masking approach for reference. The `recursive` portion in the breakdown refers to the timespan from when the first training enclave update is ready until the final model update is aggregated.

The overall iteration time is reduced by 17.7%, 21.5%, 11.3%, 36.1% for AlexNetS, AlexNetL, SpamNet, MNIST respectively. With AlexNetL, Citadel performs best with 4 aggregation children. When we reduce it to 2, even the computational overhead at each aggregation level decreases, the gain is offset by increased aggregation depth. When we increase it to 8, we face large EPC overhead at each aggregation step as 8 updates have to reside in the memory simultaneously. On the other hand, with AlexNetS, Citadel performs best with 8 children. In conclusion, there is no one-size-fits-all optimal value across all models. The number of children in hierarchical aggregation provides a trade-off knob for aggregation performance. We are unable to extend our evaluation to larger scale, but we believe hierarchical aggregation can achieve better performance when at scale, thus addressing the bottleneck in zero-sum masking.

6.4 Citadel vs. Other SGX Systems

Chiron [33] and secureTF [73] are two SGX-based systems that support multi-enclave training. They adopt parameter server (PS) as the distribution strategy, and have stronger assumptions and weaker privacy guarantees than Citadel

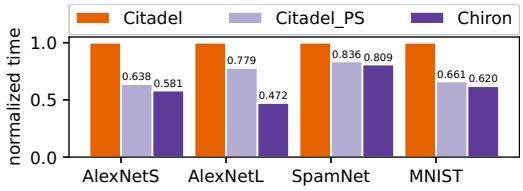
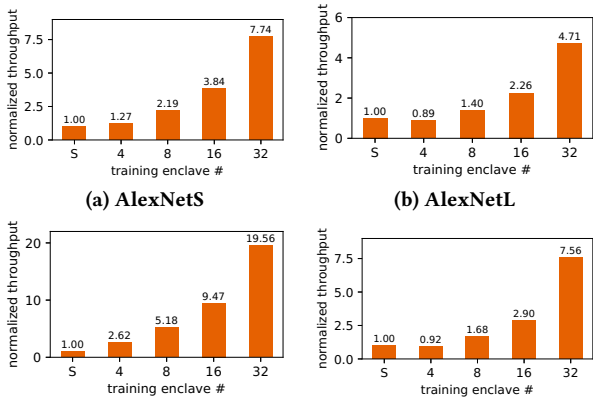


Figure 7: Chiron's runtime normalized by Citadel's. 'Citadel _PS' shows Citadel's performance with PS.



(a) AlexNetS (b) AlexNetL

(c) SpamNet (d) MNIST

Figure 8: The total throughput normalized by the `single-enclave` solution's throughput (labeled as S) w.r.t. training enclave number.

(§3.4). Unfortunately, neither of the systems is opensourced. As a result, we are only able to use profiling results and back-of-the-envelope calculations to showcase how much overhead Citadel introduces in return for stronger privacy. We use Chiron as the representative system, as secureTF shares the same distribution design as Chiron. We use the results obtained with 32 training enclaves utilizing masking techniques, as Citadel faces the most overhead here in our evaluation. We follow Chiron's workflow, and configure its parameter server with 8 servers and 32 workers. The normalized iteration runtime is shown in Fig. 7, the overhead introduced by Citadel accounts for 19.1% to 52.8% of the runtime. Chiron's multiple aggregator implementation reduces only half the elapsed time. However, besides offering weaker privacy guarantees, it uses 8 times the aggregation resources compared with Citadel. Such time reduction is non-linear to the added computing power because of SGX EPC's performance constraint. Note that Citadel can adopt PS-style multi-enclave aggregation as well. Such a modification is not essential for our privacy objectives and will require substantial engineering effort; therefore, we do not include it in Citadel yet. Instead, we add a second bar in Fig. 7 to demonstrate the *expected* runtime with a PS implementation, where the remaining overhead stems from masking.

Table 1: The slowdowns of Citadel. The 32-R column shows hierarchical aggregation, the rest shows zero-sum masking.

# Training Enclaves	4	8	16	32	32-R
AlexNetS	1.22	1.18	1.23	1.24	1.40
AlexNetL	1.09	1.23	1.44	1.73	1.65
SpamNet	1.21	1.21	1.22	1.19	1.26
MNIST	1.15	1.15	1.14	1.15	1.17

6.5 Citadel vs. Single Enclave

The single-enclave solution described in §3.3 can achieve data privacy and a limited protection of model confidentiality. In this subsection, we compare Citadel with this single-enclave solution, and show that Citadel outperforms it in both privacy guarantees and performance. Specifically, we profile the single-enclave solution's throughput via training the same models on the same Azure instance. The results are shown in Figs. 8a-8d. Note that we show the better results of zero-sum masking and hierarchical aggregation approaches.

Going from a single-enclave solution to a distributed system across multiple servers, Citadel introduces secured connections that require both network communication and cryptographic operations. As a result, we see marginal improvements compared with `single-enclave` when using only 4 training enclaves. However, with more training enclaves, Citadel is able to improve throughput substantially. Note that the benefits of distributed training is more prominent for ML models with longer total training time (e.g., SpamNet in Fig. 5c). Furthermore, we aggregate model updates after each iteration in our experiments, the result therefore demonstrates the lower bound of our improvement. One can easily improve the training performance via less communications, a.k.a., local update SGD [27, 50, 88]. With such techniques, Citadel's throughput could be further improved.

6.6 SGX Overhead in Citadel

Finally, we compare Citadel against running at the native speed. To do that, we repeat our evaluation on Citadel with the four workloads outside of SGX enclaves. All the experiments are conducted on the same Azure Kubernetes cluster but with native docker containers running the same code as in §6.2-6.3. We seek to show how much slowdown SGX induces in the entire workflow. We run the experiments over multiple iterations and compile Citadel's slowdown with different numbers of training enclaves in Table 1. The slowdown ranges from 1.09× to 1.73×. We show that SGX results in 15%–73% performance slowdown, depending on the models and scales. With larger models like AlexNetL, memory consumption is higher, so the EPC paging happens more often, causing higher overhead. We also notice that, the more training enclaves there are, the more memory it needs to finish aggregation, thus a higher slowdown at larger scale.

7 Conclusion and Discussion

We presented Citadel, the first scalable system for collaborative machine learning that protects both data privacy and model confidentiality. Citadel partitions training into two parts, the verifiable data handling code running in training enclaves and the private model handling code running in the aggregator enclave. Citadel imposes a barrier between the two parts by zero-sum masking to prevent data and model leakage, and uses hierarchical aggregation to scale up aggregation performance. Citadel is open-sourced, and the evaluation shows Citadel scales to a large number of enclaves with less than 1.7× overhead despite the stringent enclave page cache (EPC) size limit. At the end, we discuss Citadel's limitations and some future directions.

Large Models. Citadel's current design does not consider models too big for single enclaves. This issue is addressable by either increasing the EPC size with a SGX card [15], or applying model parallelism to split large models [28, 32, 91].

GPU Support. GPUs currently do not support trusted computing. Graviton [86] proposes an augmented GPU architecture with TEE support. In addition, Slalom [81] offloads parts of the computation to GPUs with secure outsourcing.Citadel can be combined with such approaches.

Side Channel Attacks. Intel SGX is currently vulnerable to side channel attacks, [63] substitutes data-dependent ML operations with data-oblivious ones to address this potential issue which could be employed by Citadel.

Other TEEs. Citadel is built with SGX, but its design is generally applicable to other TEE implementations including Arm TrustZone [7] and Amazon Nitro [9].

Network and Memory limits. Citadel enables secure centralized ML within datacenters, thus susceptible to less critical bandwidth and memory constrains compared with FL. Furthermore, Citadel can adopt existing techniques to reduce memory and network footprint. It is compatible with compression schemes that support additive operations, such as sparse gradients and quantization [4, 93].

Acknowledgement

We thank our shepherd Pierangela Samarati and the anonymous reviewers for their insightful comments. We also thank Do Le Quoc (from TU Dresden and Scontain) for his generous support to set up the SCONE environment. This research was supported in part by ACCESS – AI Chip Center for Emerging Smart Systems, Hong Kong SAR, and National Science Foundation CAREER-2048044 and IIS-1838024.

References

[1] 2016. AMD Memory Encryption. https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *USENIX OSDI*. 265–283.

[3] Adil Ahmad, Juhee Kim, Jaebaek Seo, Insik Shin, Pedro Fonseca, and Byoungyoung Lee. 2021. Chancel: efficient multi-client isolation under adversarial programs. In *NDSS*.

[4] Alham Fikri Aji and Kenneth Heafield. 2017. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021* (2017).

[5] altexsoft 2020. How Machine Learning Systems Help Reveal Scams in Fintech, Healthcare, and eCommerce. http://bit.ly/2K58Nli.

[6] Amazon 2020. Amazon SageMaker. https://aws.amazon.com/sagemaker/.

[7] Arm 2020. Arm TrustZone. https://developer.arm.com/ip-products/security-ip/trustzone.

[8] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, Mark L Stillwell, et al. 2016. SCONE: Secure linux containers with intel SGX. In *USENIX OSDI*. 689–703.

[9] AWS 2020. AWS Nitro System. https://aws.amazon.com/ec2/nitro/.

[10] azure 2020. DCsv2-series. https://docs.microsoft.com/en-us/azure/virtual-machines/dcv2-series.

[11] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In *PMLR AISTATS*. 2938–2948.

[12] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM TOCS* 33, 3 (2015), 1–26.

[13] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *ACM CCS*. 1175–1191.

[14] California State Legislature 2018. California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa.

[15] Somnath Chakrabarti, Matthew Hoekstra, Dmitrii Kuvaiskii, and Mona Vij. 2019. Scaling Intel® Software Guard Extensions Applications with Intel® SGX Card. In *HASP*. 1–9.

[16] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. 2019. SecureBoost: A Lossless Federated Learning Framework. *arXiv preprint arXiv:1901.08755* (2019).

[17] Citadel 2021. Citadel codebase. https://github.com/marcoszh/citadel-project.

[18] docker 2020. Docker. https://www.docker.com/.

[19] Wenliang Du, Yunghsiang S Han, and Shigang Chen. 2004. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *SDM*. SIAM, 222–233.

[20] EP 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[21] EU 2020. What are the GDPR Fines? https://gdpr.eu/fines/.

[22] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM CCS*. 1322–1333.

[23] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[24] Google 2020. Google Prediction API. https://cloud.google.com/ai-platform/training.

[25] Franz Gregor, Wojciech Ozga, Sébastien Vaucher, Rafael Pires, Do Le Quoc, Sergei Arnautov, André Martin, Valerio Schiavoni, Pascal Felber, and Christof Fetzer. 2020. Trust Management as a Service: Enabling Trusted Execution in the Face of Byzantine Stakeholders. *arXiv preprint arXiv:2003.14099* (2020).

[26] Otkrist Gupta and Ramesh Raskar. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 116 (2018), 1–8.

[27] Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck Cadambe. 2019. Local SGD with periodic averaging: Tighter analysis and adaptive synchronization. In *NeurIPS*.

[28] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377* (2018).

[29] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *ACM CCS*. 603–618.

[30] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *NeurIPS*.

[31] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[32] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*. 103–112.

[33] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961* (2018).

[34] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2018. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM TOCS* 35, 4 (2018), 1–32.

[35] Nick Hynes, Raymond Cheng, and Dawn Song. 2018. Efficient deep learning on multi-source private data. *arXiv preprint arXiv:1807.06689* (2018).

[36] IBMWH 2020. IBM Watson Health: Diagnostic Imaging Solutions. https://www.ibm.com/watson-health/solutions/diagnostic-imaging.

[37] Intel 2020. Intel SGX. https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

[38] Qi Jia, Linke Guo, Zhanpeng Jin, and Yuguang Fang. 2018. Preserving model privacy for machine learning in distributed systems. *IEEE TPDS* 29, 8 (2018), 1808–1822.

[39] k8s 2020. Kubernetes. https://kubernetes.io/.

[40] Kaggle 2020. Diabetic Retinopathy. https://www.kaggle.com/sovitrath/diabetic-retinopathy-224x224-gaussian-filtered.

[41] Kaggle 2020. SMS Spam Collection. https://www.kaggle.com/uciml/sms-spam-collection-dataset.

[42] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and Open Problems in Federated Learning. *arXiv preprint arXiv:1912.04977* (2019).

[43] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. Shieldstore: Shielded in-memory key-value storage with SGX. In *EuroSys*. 1–15.

[44] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. 2018. Pesos: Policy

enhanced secure object store. In *EuroSys*. 1–17.
[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*.
[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.
[47] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. 2019. TensorSCONE: A secure TensorFlow framework using Intel SGX. *arXiv preprint arXiv:1902.04413* (2019).
[48] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[49] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *NeurIPS*.
[50] Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. 2018. Don't Use Large Mini-Batches, Use Local SGD. *arXiv preprint arXiv:1808.07217* (2018).
[51] Changchang Liu, Supriyo Chakraborty, and Dinesh Verma. 2019. Secure Model Fusion for Distributed Learning Using Partial Homomorphic Encryption. In *Policy-Based Autonomic Data Governance*. Springer, 154–179.
[52] Yang Liu, Tianjian Chen, and Qiang Yang. 2018. Secure Federated Transfer Learning. *arXiv preprint arXiv:1812.03337* (2018).
[53] Kalikinkar Mandal and Guang Gong. 2019. PrivFL: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks. In *ACM CCS Workshop*. 57–68.
[54] H Brendan McMahan, Galen Andrew, Ulfar Erlingsson, Steve Chien, Ilya Mironov, Nicolas Papernot, and Peter Kairouz. 2018. A general approach to adding differential privacy to iterative training procedures. *arXiv preprint arXiv:1812.06210* (2018).
[55] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated Learning of Deep Networks using Model Averaging. *ArXiv* abs/1602.05629 (2016).
[56] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963* (2017).
[57] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. *arXiv preprint arXiv:2104.14380* (2021).
[58] Payman Mohassel and Peter Rindal. 2018. ABY 3: a mixed protocol framework for machine learning. In *ACM CCS*. 35–52.
[59] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE SP*. 19–38.
[60] mongodb 2020. mongoDB. https://www.mongodb.com/.
[61] nexusguard 2021. NexusGuard. http://www.nexusguard.consulting/.
[62] NPCSC 2017. Cybersecurity Law of the People's Republic of China. http://www.lawinfochina.com/display.aspx?id=22826&lib=law.
[63] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious multi-party machine learning on trusted processors. In *USENIX Security*. 619–636.
[64] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. 2018. Varys: Protecting SGX enclaves from practical side-channel attacks. In *USENIX ATC*. 227–240.
[65] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. 2019. Knockoff nets: Stealing functionality of black-box models. In *IEEE CVPR*. 4954–4963.
[66] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *ACM ASIACCS*. 506–519.
[67] Bryan Parno, Jacob R Lorch, John R Douceur, James Mickens, and Jonathan M McCune. 2011. Memoir: Practical state continuity for protected modules. In *IEEE SP*. 379–394.
[68] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NeuriPS*. 8026–8037.
[69] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2018. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security* 13, 5 (2018), 1333–1345.
[70] PingAn 2020. Ping An: Security Technology Reduces Data Silos. https://www.intel.com/content/www/us/en/customer-spotlight/stories/ping-an-sgx-customer-story.html.
[71] python 2020. CFFI. https://cffi.readthedocs.io/en/latest/.
[72] python 2020. Python GIL. https://realpython.com/python-gil/.
[73] Do Le Quoc, Franz Gregor, Sergei Arnautov, Roland Kunkel, Pramod Bhatotia, and Christof Fetzer. 2020. secureTF: A secure tensorflow framework. In *USENIX Middleware*. 44–59.
[74] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2021. Crosstalk: Speculative data leaks across cores are real. In *S&P*. IEEE.
[75] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).
[76] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. 2020. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In *ASPLOS*.
[77] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *ACM CCS*. 1310–1321.
[78] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *IEEE SP*. IEEE, 3–18.
[79] Samuel Smith, Erich Elsen, and Soham De. 2020. On the Generalization Benefit of Noise in Stochastic Gradient Descent. In *ICML*. PMLR.
[80] Jinhyun So, Basak Guler, and A Salman Avestimehr. 2020. Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning. *arXiv preprint arXiv:2002.04156* (2020).
[81] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287* (2018).
[82] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *USENIX Security*. 601–618.
[83] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. Graphene-sgx: A practical library OS for unmodified applications on SGX. In *USENIX ATC)*. 645–658.
[84] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *USENIX Security*. 991–1008.
[85] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564* (2018).
[86] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted execution environments on GPUs. In *USENIX OSDI*. 681–696.
[87] Robert Wahbe, Steven Lucco, Thomas E Anderson, and Susan L Graham. 1993. Efficient software-based fault isolation. In *SOSP*.
[88] Jianyu Wang and Gauri Joshi. 2018. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD. *arXiv*

preprint arXiv:1810.08313 (2018).

[89] Nico Weichbrodt, Pierre-Louis Aublin, and Rüdiger Kapitza. 2018. sgxperf: A performance analysis tool for Intel SGX enclaves. In *USENIX Middleware*. 201–213.

[90] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM TIST* 10, 2 (2019), 12.

[91] Minchen Yu, Zhifeng Jiang, Hok Chun Ng, Wei Wang, Ruichuan Chen, and Bo Li. 2021. Gillis: Serving Large Neural Networks in Serverless Functions with Automatic Model Partitioning. (2021).

[92] Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[93] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. 2020. BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning. In *USENIX ATC*.

[94] Chengliang Zhang, Huangshi Tian, Wei Wang, and Feng Yan. 2018. Stay Fresh: Speculative Synchronization for Fast Distributed Machine Learning. In *ICDCS*. IEEE.

[95] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. 2019. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881* (2019).

[96] Yanjun Zhang, Guangdong Bai, Xue Li, Caitlin Curtis, Chen Chen, and Ryan KL Ko. 2020. PrivColl: Practical Privacy-Preserving Collaborative Machine Learning. In *ESORICS*.