SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

October 2021, Vol. 64 202101:1–202101:14
https://doi.org/10.1007/s11432-020-3062-8

Jupiter: a modern federated learning platform for regional medical care

Ju XING[1], Jiadong TIAN[3], Zexun JIANG[2], Jiali CHENG[4] & Hao YIN[2*]

[1]*Department of Automation, Tsinghua University, Beijing* 100084*, China;*
[2]*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084*, China;*
[3]*School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing* 100876*, China;*
[4]*College of Engineering, Northeastern University, Boston* MA 02215*, USA*

Received 21 February 2020/Revised 15 June 2020/Accepted 31 July 2020/Published online 9 September 2021

Abstract With the emergence of AI technologies, intrinsic value of data is released and takes tremendous effects on numerous industries. In the context of regional medical care, data sharing and cooperating is in high demand, which can bring both financial and societal benefits. At present, however, medical data are locked inside medical facilities owing to legal risks and economic considerations. How to bring AI technologies into full play under this circumstance is a big challenge. In this paper, we propose Jupiter, an easy-to-use, secure, and high-performance platform for federated machine learning. Jupiter constructs a secure and high-performance aggregator cluster with SGX to efficiently aggregate the encrypted model parameters. Jupiter employs a stateful design to cooperate with medical facilities in regional medical systems with a fixed network connection. By providing an innovative programming abstraction, Jupiter makes model development more friendly to developers. The experiments show that with a low memory footprint, the throughput of a single node on an ordinary PC can reach 300 MB/s (with slice size fixed to 64 KB), and the aggregation primitive we built can process 11k aggregations per second.

Keywords federated learning, programming abstraction, SGX, high-performance

Citation Xing J, Tian J D, Jiang Z X, et al. Jupiter: a modern federated learning platform for regional medical care. Sci China Inf Sci, 2021, 64(10): 202101, https://doi.org/10.1007/s11432-020-3062-8

1 Introduction

For a fast-developing society, the medical resource is always limited comparing to demands, especially high-end healthcare. The development of a medical system is a long-lasting process, requiring massive capital investment and time for building hospitals, educating medical personnel, and developing other related infrastructures. Aggregating medical resources by regions is an efficient method to fill the gap between demands and resources. Regional Healthcare Information System (RHIS) was proposed to achieve an integrated healthcare system with better efficiency. RHIS aims to build an integrated, patient-centered healthcare information system that helps providers exchange up-to-date patient health information quickly and easily [1]. The regional information system can bring financial and societal benefits to every participant by making clinical data available at the time of care in all departments [2]. One of the most critical aspects of RHIS is medical data integration. Besides clinical records exchange, bioinformatics and machine learning can generate useful knowledge or insights for traditional clinical healthcare, which often need massive medical data rather than medical records of few individuals. For example, deep patient [3] utilizes electronic healthcare records (EHRs) from 70000 patients to build a model for clinical predictions. For medical imaging data, a magnetic resonance imaging (MRI) of a small body area, like a breast, can generate 1000 images and up to 300 MB of data storage. With better image quality or state-of-the-art imaging equipment, the storage volume for one MRI can easily expand into Gibibytes, and the size of a dataset can be in the margin of Terabytes. However, owing to the nature of medical data, sharing and aggregating medical data is challenging.

* Corresponding author (email: h-yin@mail.tsinghua.edu.cn)

© Science China Press and Springer-Verlag GmbH Germany, part of Springer Nature 2021 info.scichina.com link.springer.com

The first challenge is that direct integration of medical data can bring high cost and pressure on infrastructures, including networks, storage, and computing. For networks, the transferring process can be extremely time consuming and even impossible. Moreover, periodic delta batches need to be transferred with the growth of medical data, which further exacerbates the inefficiency. Unidirectional transmission pushes the receiving end on preparing extra storage for data hosting. Besides, the central institution responsible for data integration will meet an extremely high amount of computation. As a result, a powerful data center is demanded. In such a case, all the infrastructure resources can be expensive and hard to maintain.

The second challenge is the security issue that comes with the high sensitivity of medical data. There is abundant sensitive personal information in medical records, including social security number, billing information, and medical history. Any data leakage can be dangerous to patients as well as medical facilities. According to reports[1)], there are 15 million patient records compromised in 503 breaches in 2018, and the number may reach 25 million in 2019. Because of these incidents, medical facilities are reluctant to share medical data with outsiders, and there are laws or regulations restricting medical data sharing.

There are existing studies that try to address the challenge in the context of big data machine learning. One of the most recognizable studies is federated learning (FL) [4] that trains a model over distributed participants. Each participant trains a model on its local data, and periodically synchronizes this model with a central service. The central service aggregates models from participants to derive a global model, then updates this model to participants for subsequent training. In this way, the raw data stays with participants, and only parameters are shared. Consequently, federated learning can provide security and reduce the cost of RHIS infrastructures, with respect to machine learning applications.

However, simply shifting federated learning into RHIS will not make much sense since the target scenario is different from those usually mentioned in federated learning research studies.

- **Network environment.** In RHIS, the participants of data integration are permissioned hospitals and other medical facilities, which prefers private connections. Those connections typically have low latency and medium-high bandwidth over metropolitan area network and are usually backed by dedicated links. In contrast, neither cross-device setting nor cross-silo setting in existing research studies has such a strong assumption with the network environment, even though this study falls into the cross-silo setting. As a result, the performance of the overall system has a higher priority than communication efficiency.
- **Computation environment.** The participants in RHIS are hospitals and other medical facilities having a particular scale. They have private data centers and an enormous amount of medical data. Since each participant has sufficient and high-end resources for training, heterogeneity in computation capability will not have a significant impact on federated learning. Nevertheless, the software stacks organizing underlying computations (e.g., learning frameworks) are heterogeneous across participants. Consequently, shielding this heterogeneity from federated learning applications needs to be considered.
- **Usage approach.** Most available FL tools make two implicit assumptions for their usage. First, all the participants use the same software stack provided within a tool; second, a tool is usually an extension of the existing deep learning frameworks. Therefore, it typically emphasizes its contributions to programming interfaces. However, in a learning procedure, the position where users are engaged in is often far away from programming interfaces since they cannot actually access the software stacks of participants. Moreover, the heterogeneity in software stacks exists, as we mentioned above. Consequently, there demands a federated learning system positioning itself as a platform with user-friendly interfaces and workflows.

All the mentioned differences limit the usability of FL in a regional medical system.

To address the afore-mentioned challenges and shortcomings, we propose Jupiter, a modern federated learning platform for the regional healthcare information system. Jupiter is designed to conduct high-performance federated learning jobs through medical facilities in RHIS. It constructs a uniform entrance for developers and facilitates learning procedures on heterogeneous software stacks. Jupiter is composed of a core controller, an aggregator cluster, facility coordinators, and a web-based integrated development environment (IDE). Belonging to horizontal learning, Jupiter aggregates and syncs models among participants during training. To prevent parameter leakages, Jupiter innovatively build an aggregator cluster using Intel SGX technologies with a bunch of optimizations. With job specifications,

1) The 10 biggest healthcare data breaches of 2019, so far. https://healthitsecurity.com/news/the-10-biggest-healthcare-data-breaches-of-2019-so-far.

Jupiter provides a framework-agnostic workflow for developers to create and tune federated learning jobs. With programming abstractions named FL_Data and Session, some intermediate states are preserved and tracked. Therefore, a federated learning job in Jupiter is stateful, which provides both convenience and performance for researchers tunning their jobs. Through Jupiter, researchers can easily develop and train models using data from multiple facilities in a regional medical system. To demonstrate and evaluate Jupiter, multiple simulated training workloads are processed by Jupiter.

Our main contribution includes the following.

- We design a modern federated learning platform for RHIS, which can conduct effective federated learning tasks.
- We design a secure and high-performance aggregator cluster based on Intel SGX technology, which makes parameter aggregation secure, accurate, and efficient.
- We propose two programming abstractions in federated learning. With the abstractions, developers can tune their federated learning tasks conveniently and effectively.

2 Background

JointCloud. Since the term, cloud computing, first appeared in the 2000s [5], it has completely changed the economics of the Internet and the IT industry. Cloud computing fulfilled the requirements of the repaid growing Internet. However, with the globalization of the economy, cloud computing is transiting into a new era that demands cooperation among cloud entities rather than monopolization [6]. For computing capacity, scenarios that can challenge the limitation of any single cloud entity, even AWS or Alibaba Cloud, have occurred. According to Alibaba Cloud[2)], during the last "Double Eleven" shopping festival in 2019, the peak order rate reached 544000 per second. Also, new and more heavyweight applications like ultra-resolution and visual-reality videos make it impossible for a single service provider cannot meet the performance requirements completely [7]. JointCloud [6] is proposed to enable cooperation among multiple clouds to meet these emerging requirements, including communication, storage, and computation. To provide the environment for self-collaboration and fair competition, JointCloud Corporation Environment (JCCE) [8] is proposed, which includes Distributed Cloud Transaction, Community, and Supervision. Blockchain is introduced as an infrastructure component. Jupiter is proposed as an application of JointCloud.

Federated learning. The concept of federated learning is first introduced by Google [9] to train models from decentralized client data. In federated learning, each client contributes to the model without releasing his data. Therefore, this training style greatlty enhances data privacy and is warmly applauded by institutions with compliance considerations, e.g., hospitals and banks [10, 11]. Typically, federated learning is classified into cross-device learning and cross-silo learning according to concrete scenarios. Jupiter falls into the latter. Furthermore, from the perspective of data partitioning, federated learning can be described as horizontal learning, vertical learning, or transfer learning [12]. However, federated learning also has many open problems to be solved. On one side, the non-IID characteristics of data in the federated setting are an obstacle for federated model performance, which motivates research studies on improving optimization algorithms [13, 14]; on the other side, the exchange of parameters in the learning process is proved vulnerable to attackers interested in private data [15, 16]. As a result, some traditional secure protocols or mechanisms (e.g., SMC and differential privacy) are introduced to help federated learning retain data privacy [17–19]. Besides, some research studies focus on improving the communication efficiency of federated learning, leveraging compression, quantization technologies [4, 20]. With the popularity of federated learning, there are some promising frameworks available for developers. Google proposes a scalable system design [21] aiming at a cross-device scenario. With the assumption of network uncertainty, a central server must establish connections with clients and coordinates them with tasks in each round. This design aggregates model parameters based on secret sharing mechanisms. FATE[3)] is introduced by WeBank Fintech to facilitate cross-silo settings. It provides a dozen tools for data transformations and utilizes SMC mechanisms to protect data from leakage. However, pre-training configurations are tedious and inconvenient for developers. PySyft[4)] is an extension of PyTorch constructing tensor's interaction semantics with SMC protocols. Based on those privacy-preserving semantics, PySyft

2) Alibaba Cloud. 2019. https://www.alibabacloud.com/press-room/alibaba-cloud-powered-1b-of-gmv-in-68-seconds/.
3) Fate. https://github.com/WeBankFintech/FATE.
4) Openmined pysyft. https://github.com/OpenMined/PySyft.

can securely handle parameter exchanges in federated learning. Likewise, PaddleFL[5)] is built upon PaddlePaddle. It supports various data partitioning schemas and corresponding privacy solutions. Both 3PC protocols and differential privacy mechanisms are used by it. Google TensorFlow Federated[6)] augments TensorFlow with federated computation supports. It provides uniform abstractions for engaged data, namely Client Data and Server Data, to denote the place where computation happens. TFF makes its point on semantics while does not provide any privacy solutions. Besides, PySyft and TFF are merely software for simulations.

Intel SGX. Intel SGX is a hardware-based TEE technology providing an isolated environment. It fences an encrypted region, named encrypted enclave cache (EPC), against malicious operating systems or hypervisors [22]. The EPC is allocated inside the DRAM, and data inside it can only be visited by code inside it. Therefore, SGX provides strong protection for applications' privacy. Beside strict access control, SGX also guarantees the confidentiality of data inside EPC. There is an encryption engine standing on the processor die taking care of data encryption/decryption when data comes out/in. For convenience, Intel SGX SDK provides an abstraction named enclave and cross-boundary APIs to help developers construct trusted components of their applications. However, there are some well-known performance limitations of SGX. First, the EPC size is extremely limited (about 93 MB usable) according to some architecture reasons [23]. Although the SGX driver for Linux offers a swapping mechanism to handle memory overscription, a single swap would consume many CPU cycles and cause great performance degradation [24]. SGX2 [25] extends memory management with dynamic features, but it neither completely solves the problem nor has support from server hardware. Second, the transition from/into the enclave incurs significant overhead owing to context switch [26]. Applications built upon Intel SGX flourish in recent years. In general, researchers leverage SGX at different levels of software stacks to compose trustworthy systems. Typical applications include database systems [27–30], library operating systems [31, 32], filesystems [33, 34], network systems [35–37], programming lanugages [38, 39], and big data processing systems [40–42].

Regional medical care infrastructures. In the case of regional medical care, there are two important roles. One is the medical hospital that preserves sensitive medical data and performs exchange and computation; the other one is the service provider coordinating between different facilities and assembles the service to the public. The interconnection between hospitals and service providers prefers dedicated wire connections to achieve isolation from public networks, and it is typically located at layer2 network. It is mentioned that the choice of dedicated connection is demonstrated by some infrastructure building projects of regional medical care in Changsha Province, China. The features of a dedicated connection are exclusive bandwidth (hundreds of Megabyte) and extremely low latency (no more than a few million seconds). Thus the connection is more suitable for applications that could render data in small packets and critical about latency. Besides, hospitals engaged in the data-sharing program are usually large in scale and equipped with private data centers. Those in-hospital datacenters are typically composed of hundreds of medium-end servers and even several GPU devices. Therefore, cross-silo federated learning in regional medical care exhibits a very different scenario and breaks traditional assumptions about computation and communication. The interconnection is more stable and homogeneous; even there may exist heterogeneity among computation capabilities of different hospitals, it hardly contributes to significant lag inside the trainning process.

3 Plaform design

Jupiter is designed to facilitate federated learning towards afore-mentioned regional medical care infrastructures. Currently, it is only suitable for horizontal learning, and we assume that hospitals have already complete data governance according to some standards.

3.1 Design philosophy

Stateful. Model development under federated settings is a continuous but non-uniform procedure, especially in cross-silo settings. A development activity usually has a long lifespan (hours or even days) owing to tunning the development back and forth. Designing the platform to be stateful means managing tunning activities with temproal affinity in one context. Thus it can either improve the tunning

5) Paddle federated learning. https://github.com/PaddlePaddle/PaddleFL.
6) Google TensorFlow Federated. https://github.com/tensorflow/federated.

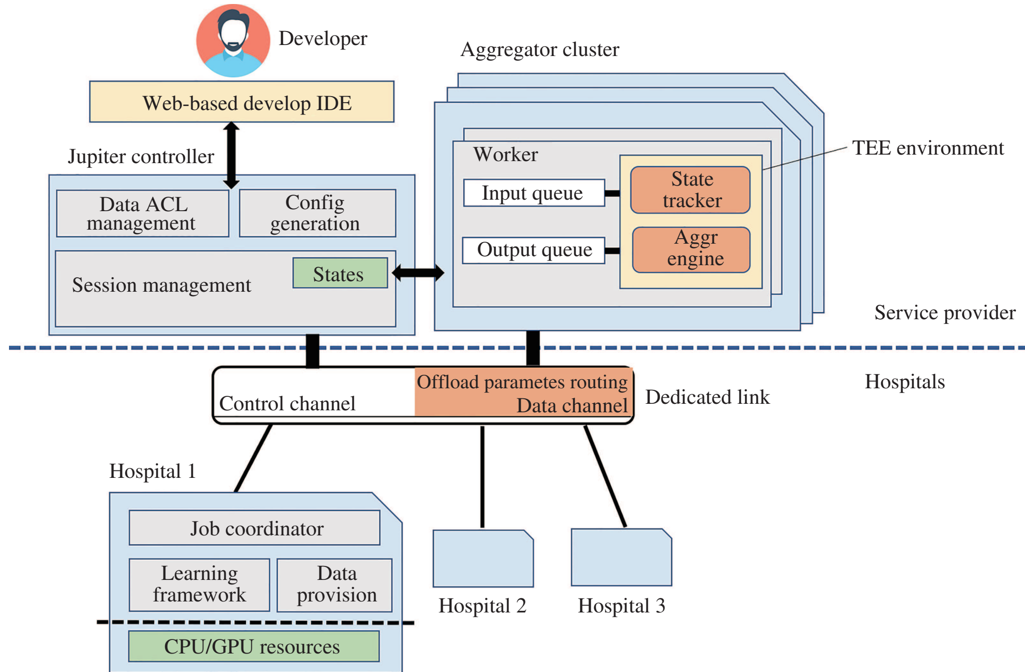

Figure 1 (Color online) Architecture of Jupiter platform.

efficiency and provide developing consistency for developers. The programming abstraction mentioned in Subsection 3.4 derived from this philosophy.

Easy-to-use. Nowadays, most model developers are rather familiar with deep learning frameworks. The way developers use the platform should be nicely compatible with what they get used to. Therefore the design should respect developers' habits to the maximum extent. The workflow in Subsection 3.3 follows this philosophy.

Secure, accurate, and performant. This philosophy mainly focuses on the parameter aggregation phase in federated learning. The previous work either achieves security and accuracy at the cost of performance (SMC mechanisms) or maintains security and performance with the loss of accuracy (differential privacy mechanisms). We hope to make the aggregation phase accurate and performant while holding the assurance of strong privacy. The SGX-based solution in Subsection 3.5 demonstrates this philosophy.

3.2 Architecture

As shown in Figure 1, Jupiter is a platform built upon the cooperation between service providers and hospitals, reflecting the essense of joint cloud. Each hospital is equipped with a data provisioning module and a learning framework. The service provider hosts a development IDE, a platform controller, and an aggergator cluster. We noted that Jupiter is not an alternative to existing deep learning frameworks but leverages existing ones for actual computations. Owing to cost and compliance considerations, the customization of the software stack in a hospital is limited. Here we mainly introduce core components in the service provider.

The development IDE. This IDE is built as a web application to ease the access and makes underlying infrastructure agnostic to end-users. Developers create their federated learning job (FL job) in the IDE using specifications with respect to model structure, data provisioning and hyperparameters/metrics, submit this job to the controller and finally inspect feedback for metrics. To take care of developers' habits nurtured by deep learning frameworks, the IDE allows developers to define the model structure using high-level APIs like those offered by Keras. Data provisioning is specified based on a central view of federated data maintained by the controller, through which the selection of participants is implicit. This view only exposes metadata to developers like schemas and properties. The reason for organizing these specifications separately is three-fold. First, developers can merely specify how to render datasets instead of manipulating real data in the IDE. Second, federated learning introduces extra hyperparameters and metrics besides those from local models; thus we prefer to organize hyperparamters and metrics in one place on a per-job bias. Third, maintaining specifications separately makes FL job

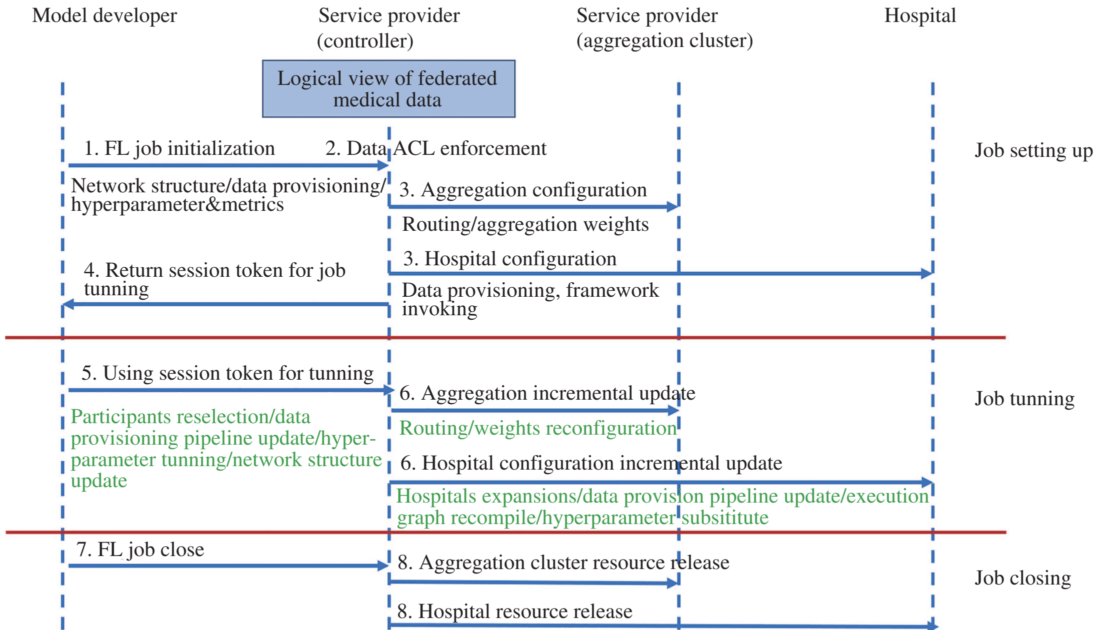

Figure 2 (Color online) Jupiter workflow.

tunning more intuitive and reasonable, and developers can only use incremental specifications to tune jobs (Subsection 3.3). It is also mentioned that in data provisioning specification, an abstraction named FL_Data is presented to make datasets rendering and tunning more efficient (Subsection 3.4).

Controller. The controller is responsible for the interpretation and enforcement of FL jobs. It has a data ACL module to check developers' authority for accessing medical data. The rules guiding the check are committed by hospitals and can be either explicit or implicit for developers. Config generation module produces concrete execution graphs for heterogenous framework backends according to job specifications. This module integrates the necessary core runtimes and adaptations to facilitate the graph generation. Therefore, the IDE can provide an unitary style for defining federated learning job and makes developers unaware of frameworks underneath they invoke. In Jupiter, a FL job is not ephemeral. It covers all subsequent tunning activities, and we use session concept to manage states across the FL job (Subsection 3.4). Each session exposes a token, helping deveolopers inspect and tune the job in an interactive way. The session management module coordinates with hospitals and aggregator cluster to prepare environments for job execution, keeps tracking, and validating relevant states. In tunning activities, this module continuously calculates the incremental updates for a job based on developers' specifications and enforces updates to hospitals and aggregator cluster.

Aggregator cluster. An aggregator is designed to achieve security, accuracy, and efficiency simultaneously in parameter aggregation. In Jupiter, we choose FedAVG [13] as an optimization method. The local parameters generated by hospitals are sliced into a fixed size and routed to the cluster via layer2 connections. SGX-based aggregator gives assurance of end-to-end privacy about parameters, the service provider can neither get values of collected slices, nor the results after aggregation, even the hypervisors or operating systems are compromised. Furthermore, in such a case, the aggregation can behave in an accurate way without the involvement of differential privacy mechanisms. To further speed up the parameter aggregation, a dedicated primitive is constructed taking advantage of SIMD features of modern CPUs.

3.3 Workflow

Figure 2 illustrates the typical workflow in detail. A model developer specifies her new FL job in IDE with various specifications (network structure, data provisioning, hyperparameters/metrics), and then submits the job to the controller. After receiving the job, the controller first checks the developer's authority against data access. If the checking result is negative, the controller directly rejects the job; otherwise, it first uses the config generation module to produce the execution graph adapted with hospitals' frameworks

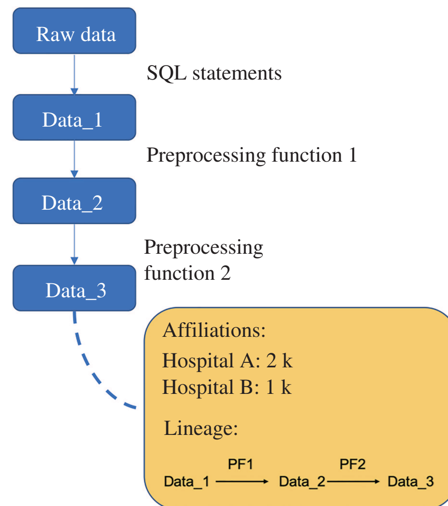

Figure 3 (Color online) FL_Data abstraction.

and then initializes a session to track states associated with the FL job. Finally, the controller enforces necessary configurations to hospitals and aggregator cluster and returns a session token to the developer.

Once the developer tunes the FL job (e.g., rerendering datasets, tunning hyperparameters), she can modify the corresponding specifications and update the job with the session token. Specifically FL_Data abstraction (Subsection 3.4) is applied for the convinence of datasets tunning. The controller automatically calcaulates the deltas brought with the specification changes, interprets them into configuration updates and eventually enforces updates to hospitals and the aggregator cluster. The tuning phase can be repeated until the developer explicitly closes the job; at this time, the corresponding session vanishes as well.

3.4 Programming abstractions

FL_Data. In data provisioning specification, we provide a programming abstraction named FL_Data to ease datasets rendering and exchange. The motivation for this abstraction comes from two observations: first, from the perspective of data, federated learning systems run into a similar case met by federated database systems [43]. A central view of data is represented with users, although only metadata like schemas and properties are exposed. This makes data rendering going through a long pipeline from basic SQL operations to stacked preprocessing logic. At the same time, datasets tunning typically involves incremental changes, either in the pipeline or in data selections. Offering data provisioning with incremental semantics will ease the developers' burden of rendering data in the tuning phase. Second, in the machine learning ecosystem, it is popular for developers to open source their datasets for experiment reproductions or model explorations. While in federated learning settings, real data can never leave hospitals. There demands a descriptive way to exchange datasets. A FL_Data object is typically composed of two key components: record distributions concerning data affiliations and transformation lineages (Figure 3). The distribution denotes the ingredients constituting the dataset; thus developers can have an intuitive sense about their selection of participants; transformation lineage encodes the processing pipeline through which the records are obtained. Since Jupiter currently supports horizontal federated learning, the transformation lineage holds for every participant. With FL_Data abstraction, developers can easily tune their datasets from the aspect of either data or pipeline. Furthermore, the exchange of FL_Data has the equivalence to that of real datasets because FL_Data can be used to fully recover the data provisioning. Table 1 lists some common APIs that could be built upon FL_Data. Manipulation APIs are intended for the incremental tunning of processing pipelines and datasets; the rollback API provides the ability to convert FL_Data into a specific history checkpoint; tracking APIs are used to shape the lineage based on other FL_Data lineages; exchange APIs are merely used for importing and exporting target FL_Data. It is worth mentioning that TensorFlow Federated project also raises programming concepts representing the union of client data. However, they preserve anonymity for engaged client and record numbers. We take the opposite preference because exposing engaged information will not break hospital privacy compliance. On the contrary, the exposure can even consolidate data's affiliations. Besides, medical is a

Table 1 APIs supported by FL_Data[a)]

Manipulation	Tracking	Exchange
X.participants_shaping(participants, flag)	X.lineage(indexes)	X.export()
X.pipeline_shaping(pipe_indexes, flag)	X.diff(Y)	X.import()
X.pipeline_substitue(pipe_index, pipe)	X.merge(Y)	
X.rollback(history_number)	X.freeze(indexes)(Y)	

a) Both X and Y are FL_Data.

field with highly professional features; developers usually choose participants with unique considerations on learning effectiveness.

Session. A session typically maintains the following states into a committed log history:

- Specifications in IDE;
- Generated execution graph;
- Slice routing policies;
- Aggregator configurations;
- Merkle root of checkpoint files.

When a session is initialized, all these states are rendered into an initial log. Subsequent tuning activities incrementally change the state space and forms a log history. The session uses this committed history to automatically calculate the deltas between adjacent tuning activities, generates incremental configurations, and eventually enforces them to the hospitals and aggregator cluster. Specifications in IDE are the entrance for delta calculation; the delta calculated here will be propagated and interpreted into different layers of the platform. Generated execution graph are tracked for the structures modification and dynamically substitutions for hyperparameters; Slice routing policies are tracked in case of participant expansion; for example, a developer prefers more data from hospitals that have not yet taken part in. In such a case, the routing policies for new participants should follow current ones to avoid the overhead brought with rearranging routes for all participants; aggregator configurations are tracked for migrating or duplicating the aggregations as need. In preparation for handling unexpected failures, Jupiter makes checkpoints in hospitals and aggregator cluster at a fixed interval, and constructs a Merkle tree based on hashes of checkpoint files. Jupiter uses the root as evidence of "global snapshot" and incorporates it into the log.

3.5 Aggregator cluster

Although servers with SGX capability are offered on the cloud[7)8)], applications are still limited by the EPC size when it comes to production environments. Moreover, parameter aggregation is both computation-bounded and memory-bounded. Therefore, constructing aggregators into a cluster with reasonable optimizations is a proper way to meet industry requirements in terms of throughput and latency.

Parameter slice routing. In Jupiter, model parameters are sliced for aggregation, and each slice is addressed using its index. Hospitals disassemble/assemble parameters at some egress/ingress boundaries. These boundaries can be either in software (e.g., send/recv interfaces) or in hardware (e.g., routers). Parameters are sliced mainly for two reasons. First, slicing helps utilize bidirectional bandwidth and can even overlap communication with computation [44, 45]. Second, slice with fixed size makes low-level optimizations more effective, which is proved by our aggregation primitive. As a consequence, parameters are aggregated by the cluster in streaming fashion. To distribute slices in the aggregator cluster, we design a two-level routing mechanism. The first level of routing is inter-node routing, which distributes slices from the ingress router to aggregators. By encoding the slice metadata into flow tables on the switch, the routing of slices can be offloaded from application to network [46], and thus improves efficiency. The second level of routing is intra-node routing, which distributes slice into input queues associated with aggregator workers. We follow two principles in routing. First, slices belonging to the same model layer should be distributed evenly cross the cluster. Second, the slices having the same index must be distributed to the same queue. The first principle puts the best effort to improve the delay of updating the whole layer while the second one avoids states updating across cores or even machines.

7) Ibm cloud data sheild. https://www.ibm.com/cloud/data-shield.
8) Azure confidential computing. https://azure.microsoft.com/en-us/solutions/confidential-compute/.

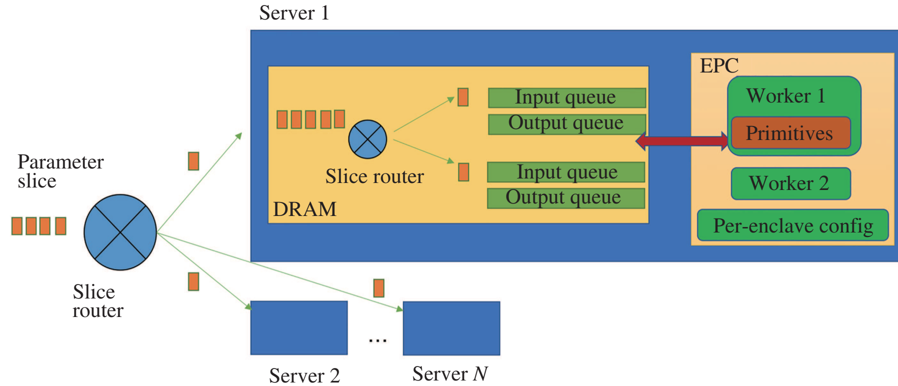

Figure 4 (Color online) Secure aggregator cluster.

SGX-based aggregator. As illustrated in Figure 4, an aggregator is composed of workers inside SGX enclave to handle the actual aggregation. Since the enclave transitions (e.g., ecall/ocall) incur significant overhead, we utilize the shared memory to fetch encrypted slices in the DRAM without leaving enclaves. Each worker in an aggregator is associated with a dedicated input/output queue. When a new slice arrives, its reference will be put inside the input queue. The corresponding worker continuously polls the queue, and copies the encrypted slice pointed by the reference into the enclave and processes it after decryption. Each worker has its own state tracker to record an intermediate state for failure recovery and efficiency. To be more specific, first, it maintains the current aggregation result and accumulated weights. Thus the new slice's arrival triggers incremental aggregation; second, it inspects arrived slices on the target index. Once the aggregation on the index is completed, the tracker releases the inspection, copies the final result outside EPC, and updates the output queue for notification. If some slices are not received owing to unexpected conditions, the tracker will save current states as a checkpoint, release associated resource in the enclave, and issue the controller for handling. These checkpoints are used for state tracker reconstruction once failures are solved. Besides, each worker is bound with an enclave thread to avoid contentions among writes to internal data structures. An aggregator also maintains a message wall denoting configurations shared by all workers. Those configurations include session-id, participants id, and corresponding aggregation weights. The message wall is dynamically configured by the session management module inside the controller.

Aggregation primitive. The aggregation of two slices is basically the weighted sum of two vectors. Since a weight in scalar form can be expanded into a vector with all the elements that have the same value, the aggregation operation is dismantled into two basic pairwise operations: multiplication and addition. Therefore, the single instruction, multiple data (SIMD) characteristics offered by modern processors (e.g., AVX2/AVX-512[9)]) is suitable for implementing high-performance aggregation operation. We construct aggregation primitive of the fixed-length slice by wrapping multiplication and addition in SIMD mode with fixed rounds. The primitive receives an auxiliary slice and a primary slice. It aggregates the former one onto the latter one; thus the primary slice is updated in-place. In aggregation context, a primary slice is the current aggregation result recorded by the state tracker, while an auxiliary slice is the new one to be processed.

4 Implementation

IDE specifications. The network structure specification we used in IDE is directly porting from keras library[10)] with version 2.2.5. Optimizer related APIs are offered with hyperparameters castrated. We organize all hyperparameters and metrics into a JSON specification and provide data provisioning with an in-house implementation of FL_Data. The controller is responsible for parsing the specifications and rendering executions using the config generation module, which integrates the TensorFlow core runtimes.

Aggregator cluster. Each parameter slices can be simply represented as 5-tuple ⟨session_id, participant_id, layer_id, slice_index, value⟩ with value field storing encrypted parameters. In inter-node routing,

9) AVX-512. https://en.wikipedia.org/wiki/AVX-512.
10) Keras. https://github.com/keras-team/keras.

Table 2 Aggregation latency (s)

Aggregation round	DRAM	EPC	EPC w/ AVX2
1k	0.1766	0.2912	0.0937
10k	1.7575	2.9054	0.9291
100k	17.4763	28.9989	9.5123

Table 3 Processing phase

Phase	Description
Decryption	Decrypt slice for processing
State lookup	Look up existed state in tracker
State creation	Create a new state in tracker
Aggregation	Aggregate the new slice onto the current value
State update	Update the state in tracker
Encryption	Encrypt slice for sending back

⟨session_id, layer_id⟩ is tracked to balance workloads; in intra-node routing, ⟨session_id, slice_index⟩ is tracked ensuring aggregation on an index is bound with a specific core. Currently, both inter-node and intra-node routing are implemented in software without offloading features. The input/output queues associated with aggregation workers are both implemented with a SPSC queue. The state tracker is constructed upon robin_hood::unordered_map[11)] to achieve high efficient lookup, insertion, and deletion. We assemble the ⟨session_id, slice_index⟩ pair into a single integer for the efficiency. The message wall shared by all workers is implemented as a flat table in the enclave. For aggregation primitive building, we use AVX2 instructions[12)], which has 256-bit widths operands. Slice encryption and decryption are implemented with AES functionalities shipped inside SGX SDK.

5 Evaluation

All the experiments were run using SGX hardware on a machine with Intel Core(TM) i5-7400 CPU (4 cores@3 GHz, 8 MB cache) with 16 GB of RAM. Dynamic frequency and voltage scaling were disabled. The operating system was Ubuntu16.04 Desktop, and the version of SGX SDK and PSW was 2.7.1. The GCC compiler was 5.4.0 and turned on the "-mAVX2" flag to enable compiler support for AVX2. Time measurement methodologies are described in each subsection.

5.1 Micro benchmarks

To evaluate the performance of aggregation primitive, we interactively aggregated slice up to a fixed round in four different settings. DRAM and EPC denote aggregating using pairwise element add, with respect to the two memory locations; EPC w/AVX2 denotes aggregating using the primitive we introduced. We used RDTSCP instructions wrapping around the ecall interface to measure the execution time. The slice size was set to be 64 KB. We ran each experiment fifty times and took average on execution time. Table 2 shows aggregation latency under various rounds. Compared with DRAM case, the primitive boots aggregation performance by 2×. Furthermore, it brings 3× performance gain inside EPC.

Latency profile. In order to make clear the composition of the time a slice consumed after entering into the aggregator, we need to profile CPU cycles took in each phase (Table 3). Giving a precise and complete profile is hard because applications inside SGX run in ring3 level. There is no way to sample ticks from hardware counters directly. Therefore, we leveraged the shared memory for time measurement. A signal slot is preserved in DARM for indicating purpose. A dedicated daemon keeps watching on the slot and records the time when the process inside the enclave overwrites the slot. The delta between two time points is treated as the time elapsed in the enclave. Besides obstacles derived from SGX, the overhead incurred by some steps is affected mainly by workloads. Hence we carried out a unit profile against each phase with typical workload settings. State lookup, state creation, and state update phases are measured with a workload of 1 million state entries. Other three phases are measured independently since they are immune to workload settings. Slice size is fixed to 64 KB, and each result takes average

11) robin-hood-hashing. https://github.com/martinus/robin-hood-hashing.
12) The intel intrinsics guide. https://software.intel.com/sites/landing-page/IntrinsicsGuide.

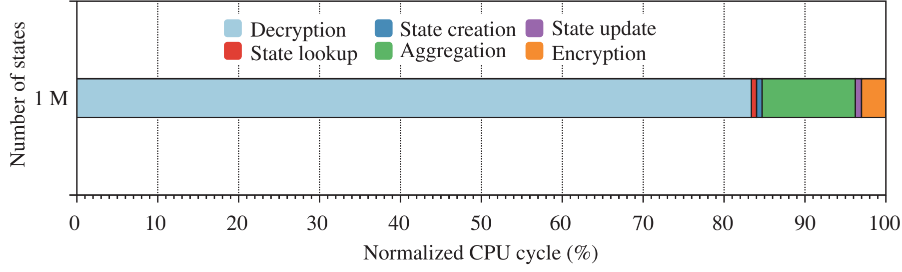

Figure 5 (Color online) Overhead constitutions.

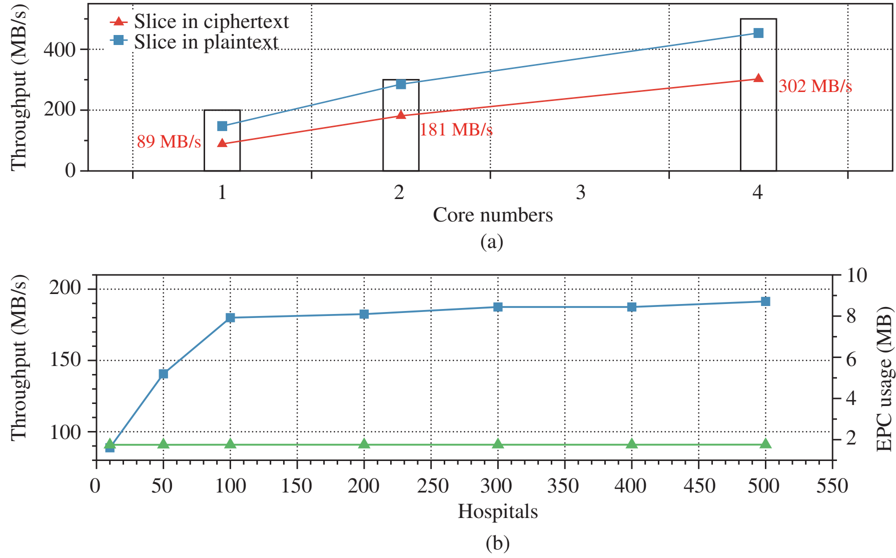

Figure 6 (Color online) (a) The workload is generated in "Synchronized" mode with 200 hospitals. Slice size is set as 64 KB. Throughput against worker numbers. (b) There are two aggregator workers and the same setting for slice size and workload.

over twenty runs. Figure 5 shows that the decryption phase accounts for most of the time consumed (83.4%) and the aggregation phase follows closely (11.5%). The encryption phase falls a little behind the aggregation (3%), and the other three phases are rather lightweight (all below 0.8%). It can be inferred that handling a slice in cipher-text and in plaintext will have a distinct gap, which is proved by our throughput experiment.

5.2 Throughput

Slice generator. We used a slice generator to produce slice streaming to simulate traffic between hospitals and aggregator cluster. The generator provides an interface for setting learning networks, slice size, communication rounds, and synchronization patterns. For target networks, it generates slices filled with random numbers and sends them in ascending order of index with respect to each participant. The synchronization pattern indicates how the slices are arranged together. Basically, we used two general patterns across experiments. One is the "Synchronized" pattern where all slices of the same index are arranged together closely; the other one is the "Asynchronized" pattern, where all slices are shuffled together with a tolerant distance. The tolerant distance controls the maximum gap (in the form of slice numbers) in sending order between two adjacent slices of the same participant. "Asynchronized" pattern is actually a linear extension of slice partial orders with respect to different participants. In experiments, we connected the generator with aggregation node via ZeroMQ and measured the average throughput of the whole procedure (including intra-node routing). From aggregators' perspective, the intrinsical difference between the two models is the total amounts of parameters. Furthermore, we used a heavily used network VGG-19 as a template to generate slices.

Ideal baseline. To explore the idea performance of the aggregator, we generated workloads with "Synchronized" pattern. Figure 6(a) illustrates the node throughput under different worker numbers with slices in plaintext and ciphertext, respectively. The results show that aggregator has high performance

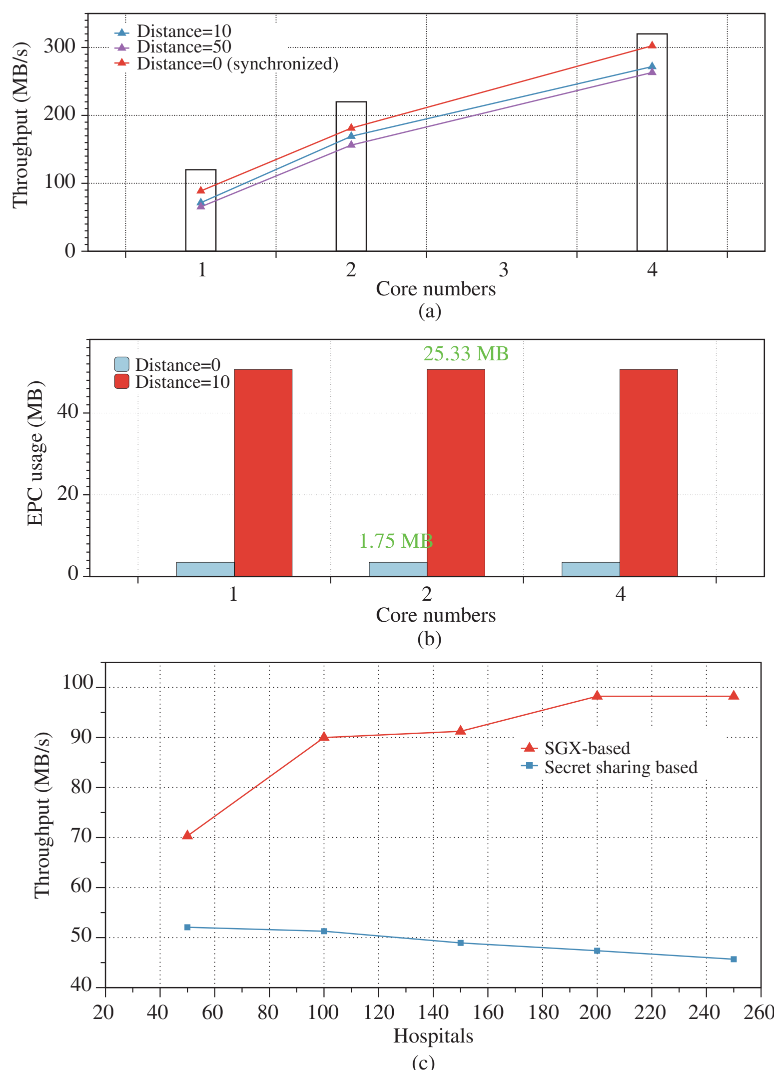

Figure 7 (Color online) (a) The workload is generated as asynchronized with tolerance distance set as 0 (synchronized), 10 and 50, respectively. There are 200 hospitals with slice size set as 64 KB. Throughput against worker numbers. (b) Same settings for hospitals and slice size as (a). The workload is generated with a distance tolerance set as 10. EPC usages with asychronized workload. (c) Throughtput comparison between the SGX-based solution and secret sharing based solution. Throughtput versus privacy solutions.

(up to 89 MB/s for a single core) and achieves good scalability, which is important to handle massive aggregations in realistic scenarios. Keeping slices in ciphertext will degrade the performance to about 30% compared with the plaintext case. We classified the degradation as the security overhead in the system. In Figure 6(b), the throughput boosts with the number of hospitals increasing and finally reaches a stable level. We thought the performance boosting in the early phase results from the benefits of caches. Since the workload is generated with “Synchronized” pattern, the aggregation for an index is continuous, which is cache-friendly. This acceleration will soon saturate and does not contribute to the performance gain anymore. Besides, EPC memory usage is stable with extremely low values. This is because the state tracker recycles the states once the aggregation on a specific index is finished. With the aforementioned workload, aggregation for an index always finishes in a narrow time window, resulting in low footprints all the time.

Asynchronized setting. Although in order to evaluate the aggregator performance against the more general cases, we generated the workload in “Asynchronized” pattern and set the number of engaged hospitals to 200. Figure 7(a) illustrates throughput under various tolerance distances versus synchronized pattern. The result indicates that a aggregator has some performance degradation while it maintains good scalability. The degradation results mainly from the staged aggregation. Since workload is asynchronized, most indexes are aggregated using slices arriving at different times. Correspondingly, with the tolerance

distance set as 10, the EPC usage in Figure 7(b) increases to about 25 MB. It is far beyond ideal baseline, and staged states mostly contribute to the EPC subscription. As a whole, an aggregator shows excellent performance under asynchronized workloads.

Comparison with SMC mechanisms. Figure 7(c) compares the throughput under two different privacy mechanisms. The secret sharing mechanism we used in this experiment is conformed with [17]. SGX-based solution beats secret sharing mechanisms with 2× gain in throughput. The result comes from the fact that in methodology proposed by [17], secret reconstruction and random number generation incur a considerable overhead. Besides, this methodology scales poorly with participants. Figure 7(c) also shows that throughput corresponding to this solution degrades with participants increasing.

6 Conclusion

This paper presents Jupiter, an easy-to-use, secure, and high-performance federated learning platform to promote data cooperation between different hospitals. We give a brand-new design for a federated learning system, implement a prototype for a secure aggregator cluster, and evaluate its performance.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 62041203, 92067206, 61972222) and National Key Research and Development Program of China (Grant No. 2018YFB2100804).

References

1 Mäenpää T, Suominen T, Asikainen P, et al. The outcomes of regional healthcare information systems in health care: a review of the research literature. Int J Med Inf, 2009, 78: 757–771
2 Adler-Milstein J, McAfee A P, Bates D W, et al. The state of regional health information organizations: current activities and financing. Health Affairs, 2007, 26: 60–69
3 Miotto R, Li L, Kidd B A, et al. Deep patient: an unsupervised representation to predict the future of patients from the electronic health records. Sci Rep, 2016, 6: 26094
4 Konečnỳ J, McMahan H B, Yu F X, et al. Federated learning: strategies for improving communication efficiency. 2016. ArXiv:1610.05492
5 Zhang S, Zhang S, Chen X, et al. Cloud computing research and development trend. In: Proceedings of 2010 2nd International Conference on Future Networks, 2010. 93–97
6 Wang H, Shi P, Zhang Y. Jointcloud: a cross-cloud cooperation architecture for integrated internet service customization. In: Proceedings of 2017 IEEE 37th International Conference On Distributed Computing Systems (ICDCS), 2017. 1846–1855
7 Jiang Z, Yin H. Adaptive routing algorithm for joint cloud video delivery. In: Proceedings of 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), 2017. 316–319
8 Shi P, Wang H, Yue X, et al. Corporation architecture for multiple cloud service providers in jointcloud computing. In: Proceedings of 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), 2017. 294–298
9 McMahan H B, Moore E, Ramage D, et al. Communication-efficient learning of deep networks from decentralized data. 2016. ArXiv:1602.05629
10 Li Q, Wen Z, He B. Federated learning systems: vision, hype and reality for data privacy and protection. 2019. ArXiv:1907.09693
11 Liu D, Miller T, Sayeed R, et al. FADL: federated-autonomous deep learning for distributed electronic health record. 2018. ArXiv:1811.11400
12 Yang Q, Liu Y, Chen T, et al. Federated machine learning. ACM Trans Intell Syst Technol, 2019, 10: 1–19
13 Sahu A K, Li T, Sanjabi M, et al. On the convergence of federated optimization in heterogeneous networks. 2018. ArXiv:1812.06127
14 Jiang P, Agrawal G. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In: Proceedings of Advances in Neural Information Processing Systems, 2018. 2525–2536
15 Shokri R, Shmatikov V. Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015. 1310–1321
16 Melis L, Song C, de Cristofaro E, et al. Inference attacks against collaborative learning. 2018. ArXiv:1805.04049
17 Bonawitz K, Ivanov V, Kreuter B, et al. Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017. 1175–1191
18 Geyer R C, Klein T, Nabi M. Differentially private federated learning: a client level perspective. 2017. ArXiv:1712.07557
19 McMahan H B, Ramage D, Talwar K, et al. Learning differentially private recurrent language models. 2017. ArXiv:1710.06963
20 Wang H, Sievert S, Liu S, et al. ATOMO: communication-efficient learning via atomic sparsification. In: Proceedings of Advances in Neural Information Processing Systems, 2018. 9850–9861
21 Bonawitz K, Eichner H, Grieskamp W, et al. Towards federated learning at scale: system design. 2019. ArXiv:1902.01046
22 Costan V, Devadas S. Intel SGX explained. IACR Cryptol ePrint Archive, 2016, 2016: 1–118
23 Taassori M, Shafiee A, Balasubramonian R. VAULT: reducing paging overheads in SGX with efficient integrity verification structures. In: Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems, 2018. 665–678
24 Orenbach M, Lifshits P, Minkin M, et al. Eleos: exitless OS services for SGX enclaves. In: Proceedings of the 12th European Conference on Computer Systems, 2017. 238–253
25 Xing B C, Shanahan M, Leslie-Hurd R. Intel® software guard extensions (Intel® SGX) software support for dynamic memory allocation inside an enclave. In: Proceedings of the Hardware and Architectural Support for Security and Privacy, 2016. 11
26 Weisse O, Bertacco V, Austin T. Regaining lost cycles with hotcalls: a fast interface for SGX secure enclaves. SIGARCH Comput Archit News, 2017, 45: 81–93

27 Krahn R, Trach B, Vahldiek-Oberwagner A, et al. Pesos: policy enhanced secure object store. In: Proceedings of the 13th ACM European Conference on Computer Systems (EuroSys), 2018. 25
28 Priebe C, Vaswani K, Costa M. EnclaveDB: a secure database using SGX. In: Proceedings of 2018 IEEE Symposium on Security and Privacy (SP), 2018. 264–278
29 Kim T, Park J, Woo J, et al. Shieldstore: shielded in-memory key-value storage with SGX. In: Proceedings of the 14th ACM European Conference on Computer Systems (EuroSys), 2019. 14
30 Bailleu M, Thalheim J, Bhatotia P, et al. SPEICHER: securing LSM-based key-value stores using shielded execution. In: Proceedings of 17th USENIX Conference on File and Storage Technologies (FAST 19), 2019. 173–190
31 Tsai C C, Porter D E, Vij M. Graphene-SGX: a practical library OS for unmodified applications on SGX. In: Proceedings of 2017 USENIX Annual Technical Conference (USENIX ATC 17), 2017. 645–658
32 Arnautov S, Trach B, Gregor F, et al. SCONE: secure linux containers with Intel SGX. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016. 689–703
33 Ahmad A, Kim K, Sarfaraz M I, et al. Obliviate: a data oblivious filesystem for Intel SGX. In: Proceedings of Network and Distributed System Security Symposium, 2018
34 Shinde S, Wang S, Yuan P, et al. BesFS: mechanized proof of an iago-safe filesystem for enclaves. 2018. ArXiv:1807.00477
35 Duan H, Wang C, Yuan X, et al. Lightbox: full-stack protected stateful middlebox at lightning speed. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019. 2351–2367
36 Poddar R, Lan C, Popa R A, et al. Safebricks: shielding network functions in the cloud. In: Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 2018. 201–216
37 Kim S, Han J, Ha J, et al. Enhancing security and privacy of tor's ecosystem by using trusted execution environments. In: Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), 2017. 145–161
38 Goltzsche D, Wulf C, Muthukumaran D, et al. Trustjs: trusted client-side execution of javascript. In: Proceedings of the 10th European Workshop on Systems Security, 2017. 7
39 Ghosn A, Larus J R, Bugnion E. Secured routines: language-based construction of trusted execution environments. In: Proceedings of 2019 USENIX Annual Technical Conference (USENIX ATC 19), 2019. 571–586
40 Zheng W, Dave A, Beekman J G, et al. Opaque: an oblivious and encrypted distributed analytics platform. In: Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), 2017. 283–298
41 Havet A, Pires R, Felber P, et al. Securestreams: a reactive middleware framework for secure data stream processing. In: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, 2017. 124–133
42 Schuster F, Costa M, Fournet C, et al. VC3: trustworthy data analytics in the cloud using SGX. In: Proceedings of 2015 IEEE Symposium on Security and Privacy, 2015. 38–54
43 Sheth A P, Larson J A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput Surv, 1990, 22: 183–236
44 Jayarajan A, Wei J, Gibson G, et al. Priority-based parameter propagation for distributed DNN training. 2019. ArXiv:1905.03960
45 Nasr M, Shokri R, Houmansadr A. Comprehensive privacy analysis of deep learning: passive and active white-box inference attacks against centralized and federated learning. In: Proceedings of 2019 IEEE Symposium on Security and Privacy (SP), 2019. 739–753
46 Cho J, Chang H, Mukherjee S, et al. Typhoon: an SDN enhanced real-time big data streaming framework. In: Proceedings of the 13th International Conference on Emerging Networking Experiments and Technologies, 2017. 310–322