Bulletin of the Technical Committee on

Data Engineering

March 2021 Vol. 44 No. 1

IEEE Computer Society

Letters

Special Issue on Learning and Reasoning on Knowledge Graphs and Applications

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at
http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

The March issue of the Data Engineering Bulletin focuses on the intricate interplay as well as a significant gap between data management and machine learning when it comes to supporting real-life business applications.

The opinion piece of this issue features a group of distinguished researchers and their assessment and prognosis of machine learning's current and future roles in building database systems. Besides highlighting several specific potentials and challenges such as using machine learning to optimize database indices and query optimization, the article also gives a great overview of how databases, data analytics and machine learning, system and infrastructure, work together to support today's business needs. It is clear that the business needs, the volume, velocity, and variety of the data, the latency and throughput requirements have evolved dramatically and in consequence, data management systems must adapt. The opinion pieces described four disruptive forces underneath the evolution, which are likely to influence future data systems.

Our associate editor Sebastian Schelter put together the current issue–Data Validation for Machine Learning Models and Applications–that consists of six papers from leading researchers in industry and academia. The papers focus on data validation, which is a critical component in end-to-end machine learning pipelines that many business applications rely on.

Haixun Wang
Instacart

Letter from the Special Issue Editor

Knowledge graphs, which were originally branded and popularized by Google in 2012, have attracted a lot of attention in both academia and industry. A knowledge graph is generally a graph-structured knowledge base that can leverage many graph processing and storage tools to perform knowledge representation and reasoning. Recently, machine learning, especially deep learning, has been widely used in many problems, including knowledge graph representation and reasoning. Thus, after more than ten years the term knowledge graph has been used, there are many new developments that are based on machine learning. First, machine learning models, particularly, graph representation learning, can be used to learn the representations of knowledge graph entities and relations. With the new representations, many reasoning tasks such as complex knowledge graph query, rule induction, knowledge base completion, and knowledge base population can be built based on learning models. Machine learning models can generalize better on knowledge graphs, where open-world assumption usually applies. Second, many new knowledge graph construction methods are based on machine learning. Information extraction and the recent "language model as a knowledge base" are all much improved with the development of scalable deep learning. With strong machine learning models, much less annotated data are required to construct many domain-specific knowledge graphs. Thus, many new types of knowledge graphs are recently developed. In this issue, we included several papers on learning and reasoning on knowledge graphs and applications.

The first paper *Logical Queries on Knowledge Graphs: Emerging Interface of Incomplete Relational Data* and the second paper *Knowledge Graph Comparative Reasoning for Fact Checking: Problem Definition and Algorithms* discussed the recent development of machine learning-based reasoning on knowledge graphs and its generalization ability to tackle the open world assumption problem of existing knowledge graphs. The following two papers, *A Perspective Survey on Industrial Knowledge Graphs: Recent Advances, Open Challenges, and Future Directions* and *Harnessing Knowledge and Reasoning for Human-Like Natural Language Generation: A Brief Review* reviewed the recent development of knowledge graph construction, where the former one surveyed new technologies on domain-specific knowledge graph construction, in particular, industrial knowledge graphs, while the latter surveyed the recent popular idea on "language model as a knowledge base" which leverages the power of natural language generation to acquire knowledge for knowledge graph construction. Then the following three papers *College-Related Question Answering based on Knowledge Graph*, *College-Related Question Answering based on Knowledge Graph*, and *Implicit Sentiment Analysis of Chinese Texts based on Contextual Information and Knowledge Enhancement* studied applications of knowledge representation learning, knowledge-based question answering, and knowledge enhanced natural language processing problems. Finally, the last paper *Distilling Causal Metaknowledge from Knowledge Graphs* discussed how to distill causal metaknowledge from existing knowledge graphs, which is a very interesting and promising new direction of knowledge graph-related research.

I would like to thank all the authors for their contributions to make this issue an interesting discussion about present and future research directions related to knowledge graph representation learning and reasoning.

Yangqiu Song
The Hong Kong University of Science and Technology

Reconciling Security and Communication Efficiency in Federated Learning

Karthik Prasad*† Sayan Ghosh*† Graham Cormode†
Ilya Mironov† Ashkan Yousefpour† Pierre Stock†
†Meta AI

Abstract

Cross-device Federated Learning is an increasingly popular machine learning setting to train a model by leveraging a large population of client devices with high privacy and security guarantees. However, communication efficiency remains a major bottleneck when scaling federated learning to production environments, particularly due to bandwidth constraints during uplink communication. In this paper, we formalize and address the problem of compressing client-to-server model updates under the Secure Aggregation primitive, a core component of Federated Learning pipelines that allows the server to aggregate the client updates without accessing them individually. In particular, we adapt standard scalar quantization and pruning methods to Secure Aggregation and propose Secure Indexing, a variant of Secure Aggregation that supports quantization for extreme compression. We establish state-of-the-art results on LEAF benchmarks in a secure Federated Learning setup with up to 40× compression in uplink communication with no meaningful loss in utility compared to uncompressed baselines.

1 Introduction

Federated Learning (FL) is a distributed machine learning (ML) paradigm that trains a model across a number of participating entities holding local data samples. In this work, we focus on cross-device FL that harnesses a large number (up to hundreds of millions) of edge devices with disparate characteristics such as availability, compute, memory, or connectivity resources [18]. Two challenges to the success of cross-device FL are privacy and scalability. FL was originally motivated for improving privacy since data points remain on client devices. However, as with other forms of ML, information about training data can be extracted via membership inference or reconstruction attacks on a trained model [11, 12], or leaked through local updates [40, 10]. Consequently, Secure Aggregation (SECAGG) protocols were introduced to prevent the server from directly observing individual client updates, which is a major vector for information leakage [8, 27]. Additional mitigations such as Differential Privacy (DP) may be required to offer further protection against attacks [11, 2], as discussed in Section 6.

Ensuring scalability to populations of heterogeneous clients is the second challenge for FL. Indeed, wall-clock training times are highly correlated with increasing model and batch sizes [27], even with recent efforts such as FedBuff [39], and communication overhead between the server and clients dominates model convergence time.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Equal contribution. Correspondence to pstock@fb.com.

Consequently, compression techniques were used to reduce the communication bandwidth while maintaining model accuracy. However, a fundamental problem has been largely overlooked in the literature: in their native form, standard compression methods such as scalar quantization and pruning are not compatible with SECAGG. This makes it challenging to ensure both security and communication efficiency.

We address this gap by adapting compression techniques to make them compatible with SECAGG. We focus on compressing uplink updates from clients to the server for three reasons. First, uplink communication is more sensitive and so is subject to a high security bar, whereas downlink updates broadcast by the server are deemed public. Second, upload bandwidth is generally more restricted than download bandwidth. For instance, according to a recent FCC report, the ratio of download to upload speeds for DSL and cable providers[1] in the US ranges between $3\times$ to $20\times$ [18]. Efficient uplink communication brings several benefits beyond speeding up convergence: lowering communication cost reduces selection bias due to under-sampling clients with limited connectivity, improving fairness and inclusiveness. It shrinks the carbon footprint of FL, the fraction of which attributable to communication can reach 95% [45]. In summary, we present the following contributions:

- We highlight the fundamental mismatch between two critical components of the FL stack: SECAGG protocols and uplink compression mechanisms.
- We formulate solutions by imposing a linearity constraint on the decompression operator, as illustrated in Figure 1 in the case of TEE-based SECAGG.
- We adapt the popular scalar quantization and (random) pruning compression methods for compatibility with the FL stack that require no changes to the SECAGG protocol.
- For extreme uplink compression without compromising security, we propose Secure Indexing (SECIND), a variant of SECAGG that supports product quantization.

2 Related Work

Communication is identified as a primary efficiency bottleneck in FL, especially in the cross-device FL setting [18]. This has led to significant interest in reducing FL's communication requirements. In what follows, we refer to a local model update in distributed training as a gradient, including updates from multiple local training steps.

Efficient Distributed Optimization. There is a large body of literature on reducing the communication cost for distributed training. [49] proposes quantizing gradients to one bit while carrying the quantization error forward across mini-batches with error feedback. Similarly, [57] proposes layer-wise ternary gradients and [6] suggests using only the sign of the gradients. Gradient sparsity is another related area that is extensively studied [56, 2, 36, 46, 42]. For instance, [14] and [22] explore adapting the degree of sparsity to the distribution of local client data. Another method, QSGD, tunes the quantization level to trade possibly higher variance gradients for reduced communication bandwidth [3]. Researchers also studied structured and sketched model updates [32]. For example, [54] proposes expressing gradients as a linear combination of basis vectors common to all workers and [55] propose to cluster the gradients and to implement error correction on the client side. Besides gradient compression, other methods such as [49, 26] reduce the communication cost by partitioning the model such that each client learns a portion of it, while [18] proposes training small models and periodically distilling them to a larger central model. However, as detailed in Section 3 and below, most of the proposed methods are not readily compatible with SECAGG and cannot be used in secure FL.

Bi-directional Compression. In addition to uplink gradient compression, a line of work also focuses on downlink model compression. In a non-distributed setup, [61, 16] demonstrates that it is possible to meaningfully train with low bit-width models and gradients. In FL, [30] proposes adapting the model size to the device to reduce

[1] FL is typically restricted to using unmetered connections, usually over Wi-Fi [27].

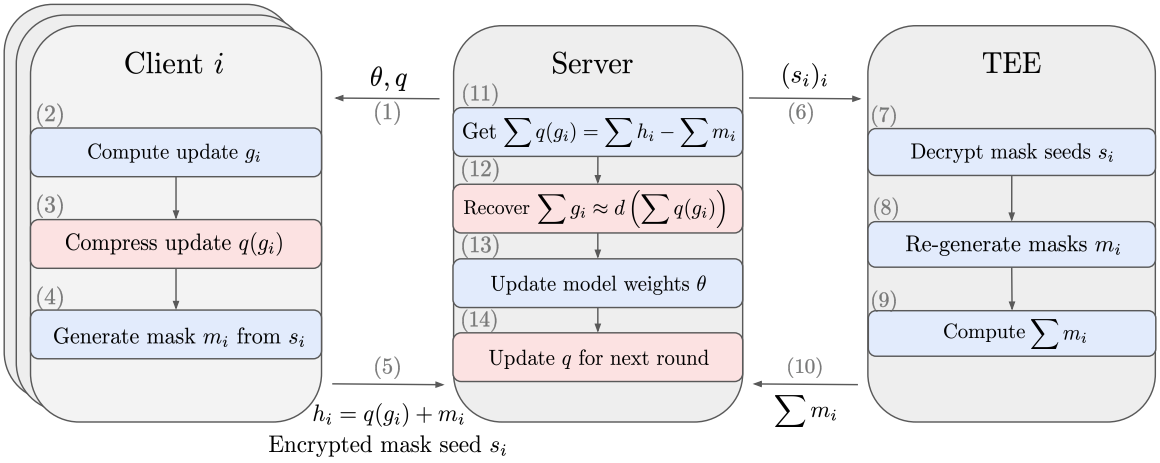


Figure 1: Summary of the proposed approach for one FL round, where we omit the round dependency and Differential Privacy (DP) for clarity. Blue boxes denote standard steps and red boxes denote additional steps for uplink compression. Client i computes local model update g_i, compresses it with the compression operator q, and encrypts it by adding a random mask m_i in the compressed domain, hence reducing the uplink bandwidth (steps 2–4). The server recovers the aggregate in the compressed domain by leveraging any SECAGG protocol (steps 7–13, with a TEE-based SECAGG, see Section 3.1). Since the decompression operator d is linear, the server can convert the aggregate back to the non-compressed domain, up to compression error (step 12). As with the model weights θ, the compression operator q are also periodically updated and broadcast by the server (step 14). In Section 3, we apply the proposed method to scalar quantization and pruning without impacting SECAGG and propose Secure Indexing, a variant of SECAGG for extreme uplink compression with product quantization. See Section 3.1 for details about SECAGG and Section 6 for a discussion on DP.

both communication and computation overhead. Since the local models are perturbed due to compression, researchers propose adapting the optimization algorithm for better convergence [37, 43, 51, 60, 4, 43]. Finally, pre-conditioning models during FL training can allow for quantized on-device inference, as demonstrated for non-distributed training by [21, 33]. As stated in Section 1, we do not focus on downlink model compression since uplink bandwidth is the main communication bottleneck and since SECAGG only involves uplink communication.

Aggregation in the Compressed Domain. In the distributed setting, [59] propose to leverage both gradient compression and parallel aggregation by performing the ring all-reduce operation in the compressed domain and decompressing the aggregate. To do so, the authors exploit temporal correlations of the gradients to design a linear compression operator. Another method, PowerSGD [53], leverages a fast low-rank gradient compressor. However, both aforementioned methods are not evaluated in the FL setup and do not mention SECAGG. Indeed, the proposed methods focus on decentralized communication between the workers by leveraging the all-reduce operation. Moreover, PowerSGD uses (stateful) error feedback on all distributed nodes, which is not readily adaptable to cross-device FL when clients generally participate in a few (not necessarily consecutive) rounds. Finally, [36] proposes FetchSGD, a compression method using sketching, which is compatible with SECAGG.

3 Background

In this section, we recall the SECAGG protocol first, then the compression methods that we wish to adapt to SECAGG, namely, scalar quantization, pruning, and product quantization.

3.1 Secure Aggregation

SECAGG refers to a class of protocols that allow the server to aggregate client updates without accessing them individually. While SECAGG alone does not entirely prevent client data leakage, it is a powerful and widely-used component of current at-scale cross-device FL implementations [18]. Two main approaches exist in practice: software-based protocols relying on Multiparty Computation (MPC) [8, 5, 58], and those that leverage hardware implementations of Trusted Execution Environments (TEEs) [27].

SECAGG relies on additive masking, where clients protect their model updates g_i by adding a uniform random mask m_i to it, guaranteeing that each client's masked update is statistically indistinguishable from any other value. At aggregation time, the protocol ensures that all the masks are canceled out. For instance, in an MPC-based SECAGG, the pairwise masks cancel out within the aggregation itself, since for every pair of users i and j, after they agree on a matched pair of input perturbations, the masks $m_{i,j}$ and $m_{j,i}$ are constructed so that $m_{i,j} = -m_{j,i}$. Similarly and as illustrated in Fig. 1, in a TEE-based SECAGG, the server receives $h_i = g_i + m_i$ from each client as well as the sum of the masks $\sum_i m_i$ from the TEE and recovers the sum of the updates as $\sum_i g_i = \sum_i h_i - \sum_i m_i$. We defer the discussion of DP noise addition by SECAGG protocols to Section 6.

Finite Group. SECAGG requires that the plaintexts—client model updates—be elements of a finite group, while the inputs are real-valued vectors represented with floating-point types. This requirement is usually addressed by converting client updates to fixed-point integers and operating in a finite domain (modulo 2^p) where p is typically set in prior literature to 32 bits. The choice of SECAGG bit-width p must balance communication costs with the accuracy loss due to rounding and overflows.

Minimal Complexity. TEE-based protocols offer greater flexibility in how individual client updates can be processed; however, the code executed inside TEE is part of the trusted computing base (TCB) for all clients. In particular, it means that this code must be stable, auditable, defects- and side-channel-free, which severely limits its complexity. Hence, in practice, we prefer compression techniques that are either oblivious to SECAGG's implementation or require minimal changes to the TCB.

3.2 Compression Methods

In this subsection, we consider a matrix $W \in \mathbb{R}^{C_{\text{in}} \times C_{\text{out}}}$ representing the weights of a linear layer to discuss three major compression methods with distinct compression/accuracy tradeoffs and identify the challenges SECAGG faces to be readily amenable to these popular quantization algorithms.

3.2.1 Scalar Quantization

Uniform scalar quantization maps floating-point weight w to 2^b evenly spaced bins, where b is the number of bits. Given a floating-point scale $s > 0$ and an integer shift parameter z called the zero-point, we map any floating-point parameter w to its nearest bin indexed by $\{0, \ldots, 2^b - 1\}$:

$$w \mapsto \text{clamp}(\text{round}(w/s) + z, [0, 2^b - 1]).$$

The tuple (s, z) is often referred to as the quantization parameters (`qparams`). With $b = 8$, we recover the popular `int8` quantization scheme [28], while setting $b = 1$ yields the extreme case of binarization [16]. The quantization parameters s and z are usually calibrated after training a model with floating-point weights using the minimum and maximum values of each layer. The compressed representation of weights W consists of the `qparams` and the integer representation matrix W_q where each entry is stored in b bits. Decompressing any integer entry w_q of W_q back to floating point is performed by applying the (linear) operator $w_q \mapsto s \times (w_q - z)$.

Challenge. The discrete domain of quantized values and the finite group required by SECAGG are not natively compatible because of the overflows that may occur at aggregation time. For instance, consider the extreme case of binary quantization, where each value is replaced by a bit. We can represent these bits in SECAGG with $p = 1$, but the aggregation will inevitably result in overflows.

3.2.2 Pruning

Pruning is a class of methods that remove parts of a model such as connections or neurons according to some pruning criterion, such as weight magnitude ([34, 23]; see [7] for a survey). [32] demonstrate client update compression with random sparsity for federated learning. Motivated by previous work and the fact that random masks do not leak information about the data on client devices, we will leverage random pruning of client updates in the remainder of this paper. A standard method to store a sparse matrix is the coordinate list (COO) format[2], where only the non-zero entries are stored (in floating point or lower precision), along with their integer coordinates in the matrix. This format is compact, but only for a large enough compression ratio, as we store additional values for each non-zero entry. Decompression is performed by re-instantiating the uncompressed matrix with both sparse and non-sparse entries.

Challenge. Pruning model updates on the client side is an effective compression approach as investigated in previous work. However, the underlying assumption is that clients have different masks, either due to their seeds or dependency on client update parameters (*e.g.,* weight magnitudes). This is a challenge for SECAGG as aggregation assumes a dense compressed tensor, which is not possible to construct when the coordinates of non-zero entries are not the same for all clients.

3.2.3 Product Quantization

Product quantization (PQ) is a compression technique developed for nearest-neighbor search [29] that can be applied for model compression [50]. Here, we show how we can re-formulate PQ to represent model updates. We focus on linear layers and refer the reader to [50] for adaptation to convolutions. Let the block size be d (say, 8), the number of codewords be k (say, 256) and assume that the number of input channels, C_{in}, is a multiple of d. To compress W with PQ, we evenly split its columns into subvectors or blocks of size $d \times 1$ and learn a codebook via k-means to select the k codewords used to represent the $C_{\text{in}} \times C_{\text{out}}/d$ blocks of W. PQ with block size $d = 1$ amounts to non-uniform scalar quantization with $\log_2 k$ bits per weight.

The PQ-compressed matrix W is represented with the tuple (C, A), where C is the codebook of size $k \times d$ and A gives the assignments of size $C_{\text{in}} \times C_{\text{out}}/d$. Assignments are integers in $[0, k-1]$ and denote which codebook a subvector was assigned to. To decompress the matrix (up to reshaping), we index the codebook with the assignments, written in PyTorch-like notation as $\widehat{W} = C[A]$.

Challenge. There are several obstacles to making PQ compatible with SECAGG. First, each client may have a different codebook, and direct access to these codebooks is needed to decode each client's message. Even if all clients share a (public) codebook, the operation to take assignments to produce an (aggregated) update is not linear, and so cannot be directly wrapped inside SECAGG.

4 Method

In this section, we propose solutions to reconcile security (SECAGG) and communication efficiency. Our approach is to modify compression techniques to share some hyperparameters globally across all clients so that aggregation can be done by uniformly combining each client's response, while still ensuring that there is scope to achieve accurate compressed representations. As detailed below, each of the proposed methods offers the same level of security as standard SECAGG without compression.

4.1 Secure Aggregation and Compression

We propose to compress the uplink model updates through a compression operator q, whose parameters are round-dependent but the same for all clients participating in the same round. Then, we will add a random mask

[2] See the torch.sparse documentation, `https://pytorch.org/docs/stable/sparse.html`.

m_i to each quantized client update $q(g_i)$ in the compressed domain, thus effectively reducing uplink bandwidth while ensuring that $h_i = q(g_i) + m_i$ is statistically indistinguishable from any other representable value in the finite group (see Section 3.1). In this setting, SECAGG allows the server to recover the aggregate of the client model updates in the compressed domain: $\sum_i q(g_i)$. If the decompression operator d is linear, the server is able to recover the aggregate in the non-compressed domain, up to quantization error, as illustrated in Figure 1:

$$d\left(\sum_i h_i - \sum_i m_i\right) = d\left(\sum_i q(g_i)\right) = \sum_i d(q(g_i)) \approx \sum_i g_i.$$

The server periodically updates the quantization and decompression operator parameters, either from the aggregated model update, which is deemed public, or by emulating a client update on some similarly distributed public data. Once these parameters are updated, the server broadcasts them to the clients for the next round. This adds overhead to the downlink communication payload, however, this is negligible compared to the downlink model size to transmit. For instance, for scalar quantization, q is entirely characterized by one `fp32` scale and one `int32` zero-point per layer, the latter of which is unnecessary in the case of a symmetric quantization scheme. Finally, this approach is compatible with both synchronous FL methods such as FedAvg [39] and asynchronous methods such as FedBuff [39] as long as SECAGG maintains the mapping between the successive versions of quantization parameters and the corresponding client updates.

4.2 Application

Next, we show how we adapt scalar quantization and random pruning with no changes required to SECAGG. We illustrate our point with TEE-based SECAGG while these adapted uplink compression mechanisms are agnostic of the SECAGG mechanism. Finally, we show how to obtain extreme uplink compression by proposing a variant of SECAGG, which we call SECIND. This variant supports product quantization and is provably secure.

4.2.1 Scalar Quantization and Secure Aggregation

As detailed in Section 3.2.1, a model update matrix g_i compressed with scalar quantization is given by an integer representation in the range $[0, 2^b - 1]$ and by the quantization parameters scale (s) and zero-point (z). A sufficient condition for the decompression operator to be linear is to broadcast common quantization parameters per layer for each client. Denote $q(g_i)$ as the integer representation of quantized client model update g_i corresponding to a particular layer for client $1 \leq i \leq N$. Set the scale of the decompression operator to s and its zero-point to z/N. Then, the server is able to decompress as follows (where the decompression operator is defined in Section 3.2.1):

$$d\left(\sum_i q(g_i)\right) = s\sum_i q(g_i) - \tfrac{z}{N} = \sum_i \left(s(q(g_i)) - z\right) \approx \sum_i g_i$$

Recall that all operations are performed in a finite group. Therefore, to avoid overflows at aggregation time, we quantize with a bit-width b but take SECAGG bit-width $p > b$, thus creating a margin for potential overflows (see Section 5.3). This approach is related to the fixed-point aggregation described in [8, 27], but we calibrate the quantization parameters and perform the calibration per layer and periodically, unlike the related approaches.

Privacy, Security and Bandwidth. Scales and zero points are determined from public data on the server. Downlink overhead is negligible: the server broadcasts the per-layer quantization parameters. The upload bandwidth is p bits per weight, where p is the SECAGG finite group size (Section 3.1). Since the masks m_i are chosen in the integer range $[0, 2^p - 1]$, any masked integer representation taken modulo 2^p is statistically indistinguishable from any other vector.

4.2.2 Pruning and Secure Aggregation

To enable linear decompression with random pruning, all clients will share a common pruning mask for each round. This can be communicated compactly before each round as a seed for a pseudo-random function. This

Algorithm 1 Secure Indexing (SECIND)

procedure SECUREINDEXING(C) ▷ This happens inside the TEE
  Receive common codebook C from server ▷ C is periodically updated by the server
  Initialize histograms $H_{m,n}$ to 0 ▷ Each histogram for block (m, n) has size k
  for each client i **do**
    Receive and decrypt assignment matrix A^i
    for each block index (m, n) **do**
      $r \leftarrow A^i_{m,n}$ ▷ Recover assignment of client i for block (m, m)
      $H_{m,n}[r] \leftarrow H_{m,n}[r] + 1$ ▷ Update global count for codeword index r
  Send back histograms $H_{m,n}$ to the server

pruning mask seed is different from the SECAGG mask seed introduced in Section 3.1 and has a distinct role. Each client uses the pruning seed to reconstruct a pruning mask, prunes their model update g_i, and only needs to encrypt and transmit the unpruned parameters. The trade-off here is that some parameters are completely unobserved in a given round, as opposed to traditional pruning. SECAGG operates as usual and the server receives the sum of the tensor of unpruned parameters computed by participating clients in the round, which it can expand using the mask seed. We denote the pruning operator as ϕ applied to the original model update g_i, and the decompression operator as d applied to a compressed tensor $\phi(g_i)$. Decompression is an expansion operation equivalent to multiplication with a sparse permutation matrix P_i whose entries are dependent on the i'th client's mask seed. Crucially, when all clients share the same mask seed within each round, we have $P_i = P$ for all i and linearity of decompression is maintained:

$$d\left(\sum_i \phi(g_i)\right) = P\left(\sum_i \phi(g_i)\right) = \sum_i P_i \phi(g_i) = \sum_i d(\phi(g_i)) \approx \sum_i g_i.$$

Privacy, Security and Bandwidth. Since the mask is random, no information leaks from the pruning mask. The downlink overhead (the server broadcasts one integer mask seed) is negligible. The upload bandwidth is simply the size of the sparse client model updates. Finally, there is no loss in security since each client uses standard SECAGG mechanism on the non-pruned entries.

4.2.3 Product Quantization and Secure Indexing

We next describe the Secure Indexing (SECIND) primitive, and discuss how to instantiate it. Recall that with PQ, each layer has its own codebook C as explained in Section 3. Let us fix one particular layer compressed with codebook C, containing k codewords. We assume that C is common to all clients participating in the round. Consider the assignment matrix of a given layer $(A^i)_{m,n}$ for client i. From these, we seek to build the assignment histograms $H_{m,n} \in \mathbb{R}^k$ that satisfy $H_{m,n}[r] = \sum_i \mathbf{1}\left(A^i_{m,n} = r\right)$, where the indicator function $\mathbf{1}$ satisfies $\mathbf{1}\left(A^i_{m,n} = r\right) = 1$ if $A^i_{m,n} = r$ and 0 otherwise. A Secure Indexing primitive will produce $H_{m,n}$ while ensuring that no other information about client assignments or partial aggregations is revealed. The server receives assignment histograms from SECIND and is able to recover the aggregated update for each block indexed by (m, n) as $\sum_r H_{m,n}[r] \cdot C[r]$. We describe how SECIND can be implemented with a TEE in Algorithm 1. Each client encrypts the assignment matrix, for instance with additive masking as described in Section 3.1, and sends it to the TEE via the server. Hence, the server does not have access to the plaintexts client-specific assignments. TEE decrypts each assignment matrix and for each block indexed by (m, n) produces the assignment histogram. Compared to SECAGG, where the TEE receives an encrypted seed per client (a few bytes per client) and sends back the sum of the masks m_i (same size as the considered model), SECIND receives the (masked) assignment matrices and sends back histograms for each round. SECIND can be implemented in other models, offering different trust paradigms, such as the multi-party computation setting (using two or more servers to operate on

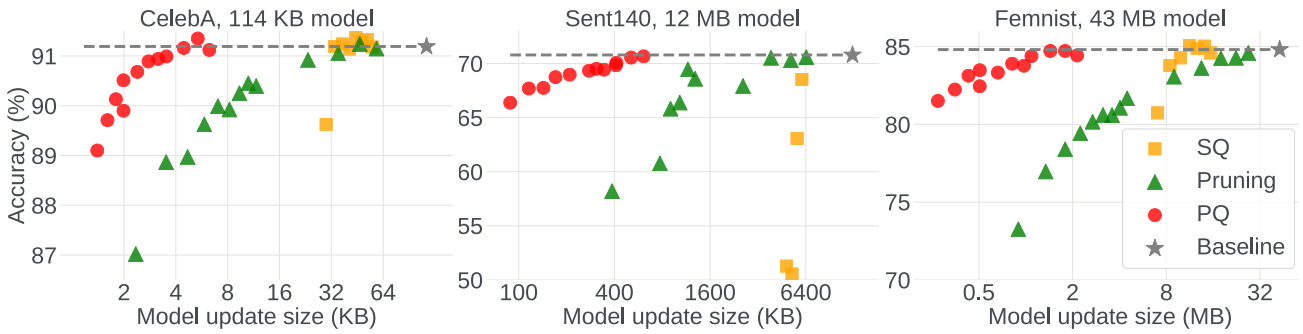


Figure 2: We adapt scalar quantization (SQ) and pruning to the SECAGG protocol to enable efficient and secure uplink communications. We also present results for product quantization (PQ) under the proposed novel SECIND protocol. The x axis is log-scale and represents the uplink message size. Baseline refers to SECAGG FL run without any uplink compression, shown as a horizontal line for easier comparison. Model size is indicated in plot titles. Uncompressed client updates are as large as the models when $p = 32$ (see Sec 3.1, shown as stars).

shares of the input). Encoding inputs as shares of one-hot vectors would lose the advantages of compression. Instead, each client can send evaluations of distributed point functions to encode each assignment [9]. These are represented compactly, but may require longer codewords to overcome the overheads.

Privacy, Security and Bandwidth. Codebooks are computed from public data while individual assignments are never revealed to the server. The downlink overhead of sending the codebooks is negligible as demonstrated in Section 5. The upload bandwidth in the TEE implementation is the assignment size, represented in k bits (the number of codewords). For instance, with a block size $d = 8$ and $k = 32$ codewords, assignment storage costs are 5 bits per 8 weights, which converts to 0.625 bits per weight. The tradeoff compared to non-secure PQ is the restriction to a global codebook for all clients (instead of one tailored to each client), and the need to instantiate SECIND instead of SECAGG. Since the assignments are encrypted before being sent to the TEE, there is no loss in security. Here, any encryption mechanism (not necessarily relying on additive masking) would work.

5 Experiments

We evaluate the performance of the proposed approaches when adapted to SECAGG protocols. We study the relationship between uplink compression and model accuracy for the LEAF benchmark tasks. For scalar and product quantization we also analyze the impact of refresh rate for compression parameters on model performance.

5.1 Experimental Setup

We follow the setup of [39] and use the FLSim library for our experiments . All experiments are run on a single V100 GPU 16 GB (except for Sent140 where we use one V100 32 GB) and typically take a few hours to run.

Tasks. We run experiments on three datasets from the LEAF benchmark [10]: CelebA [38], Sent140 [20] and FEMNIST [35]. For CelebA, we train the same convolutional classifier as [39] with BatchNorm layers replaced by GroupNorm layers and 9,343 clients. For Sent140, we train an LSTM classifier for binary sentiment analysis with $59,400$ clients. For FEMNIST, we train a GroupNorm version of the ResNet18 [25] for digit classification with 3,550 clients. We do not compress biases and norm layers due to their small overhead.

Baselines. We focus here on the (synchronous) FedAvg approach although, as explained in Section 3, the proposed compression methods can be readily adapted to asynchronous FL aggregation protocols. As in prior work, we keep the number of clients per round to at most 100, a small fraction of the total considered population

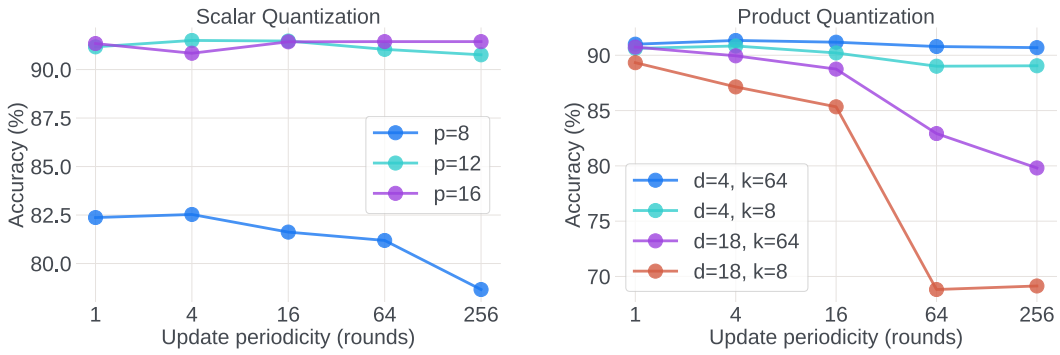


Figure 3: Impact of the refresh rate of the compression operator by the server on the CelebA dataset. **Left**: scalar quantization (quantization parameters), fixing the quantization bit-width $b = 8$ (p denotes the SECAGG bit-width). **Right**: for product quantization (codebooks), where k denotes the number of codewords and d the block size.

size [15, 13]. We report the average and standard deviation of accuracy over three independent runs for all tasks at different uplink byte sizes corresponding to various configurations of the compression operator.

Implementation Details. We refer the reader to [44] for full experiment details. The downlink overhead of sending the per-layer codebooks for product quantization is negligible. The convergence time in terms of rounds is similar for PQ runs and the non-compressed baseline/ Note that outside a simulated environment, the wall-clock time convergence for PQ runs would be lower than the baseline since uplink communication would be more efficient, hence faster.

5.2 Results and Comparison with Prior Work

Results for efficient and secure uplink communications are displayed in Figure 2. We observe that PQ yields a consistently better trade-off curve between model update size and accuracy. For instance, on CelebA, PQ achieves $\times 30$ compression with respect to the non-compressed baseline at iso-accuracy. The iso-accuracy compression rate is $\times 32$ on Sent140 and $\times 40$ on FEMNIST (see [44] for detailed tables). Scalar quantization accuracy degrades significantly for larger compression rates due to the overflows at aggregation. Pruning gives intermediate tradeoffs between scalar quantization and product quantization.

The line of work that develops FL compression techniques is exemplified by FetchSGD [36] as detailed in Section 2, where the authors do not address SECAGG. Their results are not directly comparable to ours due to incomparable experimental setups (e.g., datasets and architectures). However, Figure 6 [44] mentions upload compression rates at iso-accuracy that are weaker than those obtained here with product quantization.

5.3 Ablation Studies

We investigate the influence of the frequency of updates of the compression operator q for scalar quantization and pruning, and study the influence of the SECAGG bit-width p on the number of overflows for scalar quantization.

Update frequency of the compression operators. In Figure 3, we show that for scalar quantization, the update periodicity only plays a role with low SECAGG bit-width values p compared to the quantization bit-width b. For product quantization, the update periodicity plays an important role for aggressive compression setups corresponding to large block sizes d or to a smaller number of codewords k. For pruning, we measure the impact of masks that are refreshed periodically. We observed that if we refresh the compression operator more frequently, staleness is reduced, leading to accuracy improvements.

Overflows for scalar quantization. As discussed in Section 4.2.1, we choose the SECAGG bit-width p to be greater than quantization bit-width b in order to avoid aggregation overflows. While it suffices to set p to be $\lceil \log_2 n_c \rceil$ more than b, where n_c is the number of clients participating in the round, reducing p is desirable to

reduce uplink size. We studied the impact of p on the percentage of parameters that suffer overflows, and observed that there is a benefit to having some non-zero overflow margin size, but no clear correlation between margin size and accuracy.

6 Concluding Remarks

In this paper, we reconcile efficiency and security for uplink communication in Federated Learning. We propose to adapt existing compression mechanisms such as scalar quantization and pruning to the secure aggregation protocol by imposing a linearity constraint on the decompression operator. Our experiments demonstrate that we can adapt both quantization and pruning mechanisms to obtain a high degree of uplink compression with minimal degradation in performance and higher security guarantees. For achieving the highest rates of compression, we introduce SECIND, a variant of SECAGG well-suited for TEE-based implementation that supports product quantization while maintaining a high security bar. As mentioned in Section 1, we may want both SECAGG and Differential Privacy [2] to realize the full promise of FL as a privacy-enhancing technology. While our primary focus is on enabling efficient and secure uplink communication, we emphasize that the proposed approaches are compatible with user-level DP. For instance, at the cost of increasing the complexity of the trusted computing base, DP noise can be added natively by the TEE with our modified random pruning or scalar quantization approaches. For PQ and SECIND, we can have the TEE to add noise in the assignment space (*i.e.,* outputting a noisy histogram), or to map the histogram to the codeword space and add noise there. Each option offers a different tradeoff between privacy, trust, and accuracy; we leave detailed evaluation to future work.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In ACM CCS, 2016.

[2] A. F. Aji and K. Heafield. Sparse communication for distributed gradient descent. In EMNLP, 2017.

[3] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. In NeurIPS, 2017.

[4] M. M. Amiri, D. Gunduz, S. R. Kulkarni, and H. V. Poor. Federated learning with quantized global model updates. CoRR, 2006.10672, 2020.

[5] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In ACM CCS, 2020.

[6] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. signSGD: Compressed optimisation for non-convex problems. In ICML, PMLR, 2018.

[7] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? In MLSys, 2020.

[8] K. A. Bonawitz, F. Salehi, J. Konečný, B. McMahan, and M. Gruteser. Federated learning with autotuned communication-efficient secure aggregation. In ACSCC. IEEE, 2019.

[9] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In ACM CCS, 2016.

[10] S. Caldas, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar. LEAF: A benchmark for federated settings. CoRR, abs/1812.01097, 2018.

[11] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramèr. Membership inference attacks from first principles. CoRR, abs/2112.03570, 2021.

[12] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel. Extracting training data from large language models. In USENIX Security Symposium, 2021.

[13] Z. Charles, Z. Garrett, Z. Huo, S. Shmulyian, and V. Smith. On large-cohort training for federated learning. CoRR, 2106.07820, 2021.

[14] C. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan. AdaComp: Adaptive residual gradient compression for data-parallel distributed training. In AAAI, 2018.

[15] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays. Federated learning of out-of-vocabulary words. CoRR, 1903.10635, 2019.

[16] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. CoRR, 1511.00363, 2015.

[17] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In Proceedings of the Third Conference on Theory of Cryptography, 2006.

[18] FCC. The eleventh Measuring Broadband America fixed broadband report, 2021.

[19] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller. Inverting gradients—How easy is it to break privacy in federated learning? In NeurIPS, 2020.

[20] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 2009.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In ICML, 2015.

[22] P. Han, S. Wang, and K. K. Leung. Adaptive gradient sparsification for efficient federated learning: An online learning approach. In ICDCS, 2020.

[23] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal Brain Surgeon. In NeurIPS, 1992.

[24] C. He, M. Annavaram, and S. Avestimehr. Group knowledge transfer: Federated learning of large CNNs at the edge. In NeurIPS, 2020.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In IEEE CVPR, 2016.

[26] C. Hu, W. Bao, D. Wang, and F. Liu. Dynamic adaptive DNN surgery for inference acceleration on the edge. IEEE INFOCOM, 2019.

[27] D. Huba, J. Nguyen, K. Malik, R. Zhu, M. Rabbat, A. Yousefpour, C.-J. Wu, H. Zhan, P. Ustinov, H. Srinivas, K. Wang, A. Shoumikhin, J. Min, and M. Malek. Papaya: Practical, private, and scalable federated learning. In MLSys, 2022.

[28] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In IEEE CVPR, June 2018.

[29] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell., 33(1):117–128, 2011.

[30] Y. Jiang, S. Wang, B. J. Ko, W. Lee, and L. Tassiulas. Model pruning enables efficient federated learning on edge devices. CoRR, abs/1909.12326, 2019.

[31] P. Kairouz et al. Advances and open problems in federated learning. Found. Trends Mach. Learn., 14(1–2), 2021.

[32] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. In NIPS Workshop on Private Multi-Party Machine Learning, 2016.

[33] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference. CoRR, 1806.08342, 2018.

[34] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In NeurIPS, 1989.

[35] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[36] Y. Lin, S. Han, H. Mao, Y. Wang, and W. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In ICLR, 2018.

[37] X. Liu, Y. Li, J. Tang, and M. Yan. A double residual compression algorithm for efficient distributed learning. In AISTATS, 2020.

[38] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In IEEE ICCV, 2015.

[39] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In AISTATS, 2017.

[40] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In IEEE S&P, 2019.

[41] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba. Federated learning with buffered asynchronous aggregation. In AISTATS, 2022.

[42] T. Parcollet, J. Fernandez-Marques, P. PB Gusmao, Y. Gao, and N. D. Lane. ZeroFL: Efficient on-device training for federated learning with local sparsity. In ICLR, 2022.

[43] C. Philippenko and A. Dieuleveut. Preserved central model for faster bidirectional compression in distributed settings.

In NeurIPS, 2021.

[44] K. Prasad, S. Ghosh, G. Cormode, I. Mironov, A. Yousefpour, and P. Stock. Reconciling security and communication efficiency in federated learning. CoRR, 2207.12779, 2022.

[45] X. Qiu, T. Parcollet, J. Fernandez-Marques, P. P. B. de Gusmao, D. J. Beutel, T. Topal, A. Mathur, and N. D. Lane. A first look into the carbon footprint of federated learning. arXiv, 2102.07627, 2021.

[46] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, and T. Hoefler. SparCML: High-performance sparse communication for machine learning. In SC, 2019.

[47] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora. FetchSGD: Communication-efficient federated learning with sketching. In ICML, 2020.

[48] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek. Robust and communication-efficient federated learning from non-i.i.d. data. IEEE Transactions on Neural Networks and Learning Systems, 31:3400–3413, 2020.

[49] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In INTERSPEECH, 2014.

[50] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jégou. And the bit goes down: Revisiting the quantization of neural networks. In ICLR, 2020.

[51] H. Tang, C. Yu, X. Lian, T. Zhang, and J. Liu. DOUBLESQUEEZE: Parallel stochastic gradient descent with double-pass error-compensated compression. In ICML, 2019.

[52] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar. Split learning for health: Distributed deep learning without sharing raw patient data, 2018.

[53] T. Vogels, S. P. Karimireddy, and M. Jaggi. PowerSGD: Practical low-rank gradient compression for distributed optimization. In NeurIPS, 2019.

[54] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright. ATOMO: Communication-efficient learning via atomic sparsification. In NeurIPS, 2018.

[55] J. Wang, H. Qi, A. S. Rawat, S. Reddi, S. Waghmare, F. X. Yu, and G. Joshi. Fedlite: A scalable approach for federated learning on resource-constrained clients. CoRR, 2201.11865, 2022.

[56] J. Wangni, J. Wang, J. Liu, and T. Zhang. Gradient sparsification for communication-efficient distributed optimization. In NeurIPS, 2018.

[57] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. TernGrad: Ternary gradients to reduce communication in distributed deep learning. In NeurIPS, 2017.

[58] C. Yang, J. So, C. He, S. Li, Q. Yu, and S. Avestimehr. LightSecAgg: Rethinking secure aggregation in federated learning. In MLSys, 2022.

[59] M. Yu, Z. Lin, K. Narra, S. Li, Y. Li, N. S. Kim, A. Schwing, M. Annavaram, and S. Avestimehr. GradiVeQ: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training. In NeurIPS, 2018.

[60] S. Zheng, Z. Huang, and J. T. Kwok. Communication-efficient distributed blockwise momentum SGD with error-feedback. In NeurIPS, 2019.

[61] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR, abs/1606.06160, 2016.

NVIDIA FLARE: Federated Learning from Simulation to Real-World

Holger R. Roth Yan Cheng Yuhong Wen Isaac Yang Ziyue Xu Yuan-Ting Hsieh
Kristopher Kersten Ahmed Harouni Can Zhao Kevin Lu Zhihong Zhang Wenqi Li
Andriy Myronenko Dong Yang Sean Yang Nicola Rieke Abood Quraini Chester Chen
Daguang Xu Nic Ma Prerna Dogra Mona Flores Andrew Feng

NVIDIA Corporation*
Shanghai, China
Munich, Germany
Bethesda, Santa Clara, USA

Abstract

Federated learning (FL) enables building robust and generalizable AI models by leveraging diverse datasets from multiple collaborators without centralizing the data. We created NVIDIA FLARE[1] as an open-source software development kit (SDK) to make it easier for data scientists to use FL in their research and real-world applications. The SDK includes solutions for state-of-the-art FL algorithms and federated machine learning approaches, which facilitate building workflows for distributed learning across enterprises and enable platform developers to create a secure, privacy-preserving offering for multiparty collaboration utilizing homomorphic encryption or differential privacy. The SDK is a lightweight, flexible, and scalable Python package. It allows researchers to apply their data science workflows in any training libraries (PyTorch, TensorFlow, XGBoost, or even NumPy) in real-world FL settings. This paper introduces the key design principles of NVFlare and illustrates some use cases (e.g., COVID analysis) with customizable FL workflows that implement different privacy-preserving algorithms.

1 Introduction

Federated learning (FL) has become a reality for many real-world applications [45]. It enables multinational collaborations on a global scale to build more robust and generalizable machine learning and AI models. In this paper, we introduce NVIDIA FLARE (NVFlare), an open-source software development kit (SDK) that makes it easier for data scientists to collaborate to develop more generalizable and robust AI models by sharing model weights rather than private data. While FL is attractive in many industries, it is particularly beneficial for healthcare applications where patient data needs to be protected. For example, FL has been used for predicting clinical

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Contact: {hroth,yanc,chesterc,daguangx,pdogra,andyf}@nvidia.com
[1]Code is available at https://github.com/NVIDIA/NVFlare.

outcomes in patients with COVID-19 [7] or to segment brain lesions in magnetic resonance imaging [39, 38]. NVFlare is not limited to applications in healthcare and is designed to allow cross-silo FL [18] across enterprises for different industries and researchers.

In recent years, several efforts (both open-source and commercial) have been made to bring FL technology into the healthcare sector and other industries, like TensorFlow Federated [1], PySyft [48], FedML [14], FATE [26], Flower [3], OpenFL [33], Fed-BioMed [40], IBM Federated Learning [27], HP Swarm Learning [52], FederatedScope [44], FLUTE [8], and more. Some focus on simulated FL settings for researchers, while others prioritize production settings. NVFlare aims to be useful for both scenarios: 1) for researchers by providing efficient and extensible simulation tools and 2) by providing an easy path to transfer research into real-world production settings, supporting high availability and server failover, and by providing additional productivity tools such as multi-tasking and admin commands.

2 NVIDIA FLARE Overview

NVIDIA FLARE – or short NVFlare – stands for "**NV**IDIA **F**ederated **L**earning **A**pplication **R**untime **E**nvironment". The SDK enables researchers and data scientists to adapt their machine learning and deep learning workflows to a federated paradigm. It enables platform developers to build a secure, privacy-preserving offering for distributed multiparty collaboration.

NVFlare is a lightweight, flexible, and scalable FL framework implemented in Python that is agnostic to the underlying training library. Developers can bring their own data science workflows implemented in PyTorch, TensorFlow, or even in pure NumPy, and apply them in a federated setting. A typical FL workflow such as the popular federated averaging (FedAvg) algorithm [33], can be implemented in NVFlare using the following main steps. Starting from an initial global model, each FL client trains the model on their local data for a while and sends model updates to the server for aggregation. The server then uses the aggregated updates to update the global model for the next round of training. This process is iterated many times until the model converges.

Though used heavily for federated deep learning, NVFlare is a generic approach for supporting collaborative computing across multiple clients. NVFlare provides the *Controller* programming API for researchers to create workflows for coordinating clients for collaboration. FedAvg is one such workflow. Another example is cyclic weight transfer [5]. The central concept of collaboration is the notion of "task". An FL controller assigns tasks (e.g., deep-learning training with model weights) to one or more FL clients and processes results returned from clients (e.g., model weight updates). The controller may assign additional tasks to clients based on the processed results and other factors (e.g., a pre-configured number of training rounds). This task-based interaction continues until the objectives of the study are achieved.

The API supports typical controller-client interaction patterns like broadcasting a task to multiple clients, sending a task to one or more specified clients, or relaying a task to multiple clients sequentially. Each interaction pattern has two flavors: wait (block until client results are received) or no-wait. A workflow developer can use these interaction patterns to create innovative workflows. For example, the *ScatterAndGather* controller (typically used for FedAvg-like algorithms) is implemented with the *broadcast_and_wait* pattern, and the *CyclicController* is implemented with the *relay_and_wait pattern*. The controller API allows the researcher to focus on the control logic without needing to deal with underlying communication issues. Figure 1 shows the principle. Each FL client acts as a worker that simply executes tasks assigned to it (e.g., model training) and returns execution results to the controller. At each task interaction, there can be optional filters that process the task data or results before passing it to the *Controller* (on the server side) or task executor (client side). The filter mechanism can be used for data privacy protection (e.g., homomorphic encryption/decryption or differential privacy) without having to alter the training algorithms.

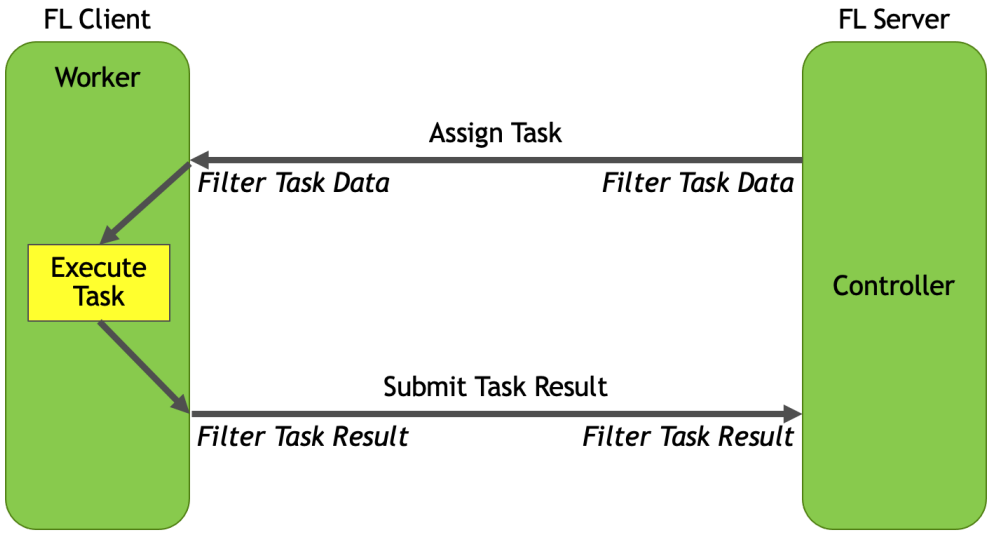


Figure 1: NVFlare job execution. The *Controller* is a Python object that controls or coordinates the *Workers* to get a job done. The controller is run on the FL server. A *Worker* is capable of performing tasks. *Workers* run on FL clients.

Key Components NVFlare is built on a componentized architecture that allows FL workloads to move from research and simulation to real-world production deployment. Some of the key components of this SDK include:

- **FL Simulator** for rapid development and prototyping.
- **NVFlare Dashboard** for simplified project management, secure provisioning, and deployment, orchestration.
- **Reference FL algorithms** (e.g., FedAvg, FedProx, SCAFFOLD) and workflows, like scatter and gather, cyclic, etc.
- **Privacy preservation** with differential privacy, homomorphic encryption, and more.
- **Specification-based API** for extensibility, allowing customization with plug-able components.
- **Tight integration** with other learning frameworks like MONAI [4], XGBoost [6], and more.

High-Level Architecture NVFlare is designed with the idea that less is more, using a specification-based design principle to focus on what is essential. This allows other people to be able to do what they want to do in real-world applications by following clear API definitions. FL is an open-ended space. The API-based design allows others to bring their implementations and solutions for various components. Controllers, task executors, and filters are just examples of such extensible components. NVFlare provides an end-to-end operation environment for different personas. It provides a comprehensive provisioning system that creates security credentials for secure communications to enable the easy and secure deployment of FL applications in the real world. It also provides an FL Simulator for running proof-of-concept studies locally. In production mode, the researcher conducts an FL study by submitting jobs using admin commands using Notebooks or the NVFlare Console – an interactive command tool. NVFlare provides many commands for system operation and job management. With these commands, one can start and stop a specific client or the entire system, submit new jobs, check the status of jobs, create a job by cloning from an existing one, and much more.

With NVFlare's component-based design, a job is just a configuration of components needed for the study. For the control logic, the job specifies the controller component to be used and any components required by the controller.

3 System Concepts

A NVFlare system is a typical client-server communication system that comprises one or more FL server(s), one or more FL client(s), and one or more admin clients. The FL Servers open two ports for communication with FL clients and admin clients. FL clients and admin clients connect to the opened ports. FL clients and admin clients do not open any ports and do not directly communicate with each other. The following is an overview of the key concepts and objects available in NVFlare and the information that can be passed between them.

Workers and Controller NVFlare's collaborative computing is achieved through the *Controller*/*Worker* interactions.

Shareable Object that represents a communication between server and client. Technically, the *Shareable* is implemented as a Python dictionary that could contain different information, e.g., model weights.

Data Exchange Object (DXO) Standardizes the data passed between the communicating parties. One can think of the *Shareable* as the envelope and the *DXO* as the letter. Together, they comprise a message to be shared between communicating parties.

FLComponent The base class of all the FL components. Executors, controllers, filters, aggregators, and their subtypes are all *FLComponents*. *FLComponent* comes with some useful built-in methods for logging, event handling, auditing, and error handling.

Executors Type of *FLComponent* for FL clients that has an execute method that produces a *Shareable* from an input *Shareable*. NVFlare provides both single- and multi-process executors to implement different computing workloads.

FLContext One of the most important features of NVFlare is to pass data between the FL components. *FLContext* is available to every method of all common FLComponent types. Through *FLContext*, the component developer can get services provided by the underlying infrastructure and share data with other components of the FL system.

Communication Drivers NVFlare abstracts the communication layers out so that different deployment scenarios can implement customizable communication drivers. By default, we use GRPC for data communication in task-based communication. However, the driver can be replaced to run other communication protocols, for example, TCP. The customizable nature of communication in NVFlare allows for both server-centric and peer-to-peer communication patterns. This enables the user to utilize both scatter and gather-type workflows like FedAvg [33], decentralized training patterns like swarm learning [52], or direct peer-to-peer communication as in split learning [12].

Fig. 2 compares the times for model upload and download from the client's perspective using different communication protocols available in NVFlare using a model of $\sim$18MB in size.

The experiment runs in a multi-cloud environment with the server and eight clients running on Azure, while two clients run on AWS. One can observe that the global model download is slower as all clients are trying to

download the global model at the same time, and hence the server is more busy. In contrast, the clients' model uploads happen at slightly different times and therefore are faster. One can also see how this multi-cloud setup causes the clients on AWS to take slightly longer during model download due to communication across different cloud infrastructures.

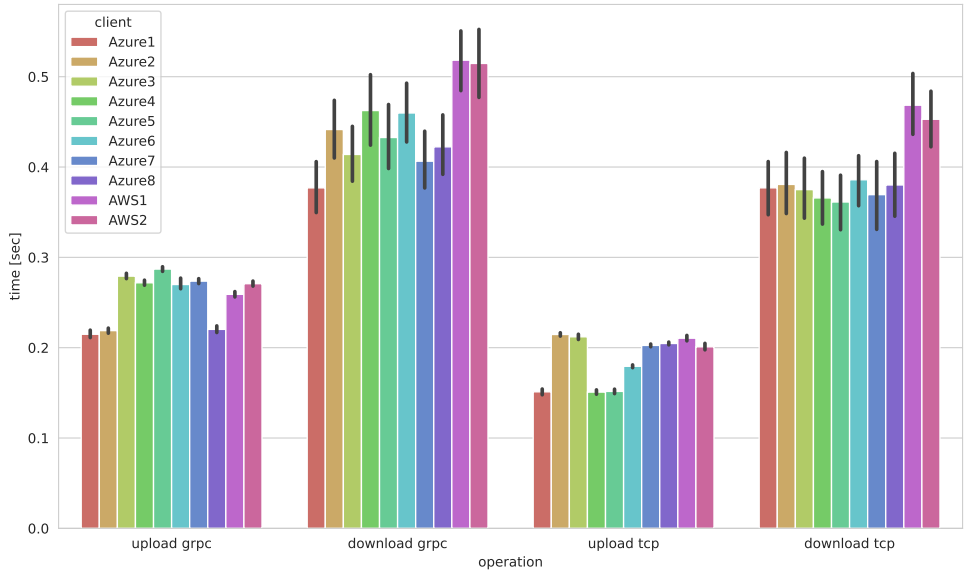


Figure 2: Comparison of GRPC and TCP communication drivers in NVFlare. The server is running on Azure. The clients are distributed between Azure and AWS. The message size is ∼18MB. Communication times were measured over 100 rounds of FedAvg. Error bars indicate the 95% confidence intervals.

Filters Filters in NVFlare are a type of *FLComponent* that have a process method to transform the *Shareable* object between the communicating parties. A Filter can provide additional processing to shareable data before sending or after receiving from a peer. Filters can convert data formats and a lot more and are NVFlare's primary mechanism for data privacy protection [24, 13]:

- *ExcludeVars* to exclude variables from shareable.
- *PercentilePrivacy* for truncation of weights by percentile.
- *SVTPrivacy* for differential privacy through sparse vector techniques.
- Homomorphic encryption filters used for secure aggregation.

As an example, we show the average encryption, decryption, and upload times when using homomorphic encryption for secure aggregation[2]. We compare raw data to encrypted model gradients uploaded in Table 1 when hosting the server on AWS[3] and connecting 30 client instances using an on-premise GPU cluster. One can see the longer upload times due to the larger message sizes needed by homomorphic encryption.

[2] `https://developer.nvidia.com/blog/federated-learning-with-homomorphic-encryption`

[3] For reference, we used an m5a.2xlarge instance with eight vCPUs, 32-GB memory, and up to 2,880 Gbps network bandwidth.

Table 1: Federated learning exchanging homomorphic encrypted vs. raw model updates.

Time in seconds	Mean	Std. Dev.
Encryption	5.01	1.18
Decryption	0.95	0.04
Enc. upload	38.00	71.17
Raw upload	21.57	74.23

Event Mechanism NVFlare comes with a powerful event mechanism that allows dynamic notifications to be sent to all event handlers. This mechanism enables data-based communication among decoupled components: one component fires an event when a certain condition occurs, and other components can listen to that event and processes the event data. Each *FLComponent* is automatically an event handler. To listen to and process an event, one can simply implement the *handle_event()* method and process desired event types. Events represent some important moments during the execution of the system logic. For example, before and after aggregation or when important data becomes available, e.g., a new "best" model was selected.

3.1 Productivity Features

NVFlare contains features that enable efficient, collaborative, and robust computing workflows.

Multi-tasking For systems with a large capacity, computing resources could be idle most of the time. NVFlare implements a resource-based multi-tasking solution, where multiple jobs can be run concurrently when overall system resources are available. Multi-tasking is made possible by a job scheduler on the server side that constantly tries to schedule a new job. For each job to be scheduled, the scheduler asks each client whether they can satisfy the required resources of the job (e.g., number of GPU devices) by querying the client's resource manager. If all clients can meet the requirement, the job will be scheduled and deployed to the clients.

High Availability and Server Failover To avoid the FL server as a single point of failure, a solution has been implemented to support multiple FL servers with automatic cut-over when the currently active server becomes unavailable. Therefore, a component called *Overseer* is added to facilitate automatic cut-over. The *Overseer* provides the authoritative endpoint info of the active FL server. All other system entities (FL servers, FL clients, admin clients) constantly communicate (i.e., every 5 seconds) with the Overseer to obtain and act on such information. If the server cutover happens during the execution of a job, then the job will continue to run on the new server. Depending on how the controller is written, the job may or may not need to restart from the beginning but can continue from a previously saved snapshot.

Simulator NVFlare provides a simulator to allow data scientists and system developers to easily write new *FLComponents* and novel workflows. The simulator is a command line tool to run a NVFlare job. To allow simple experimentation and debugging, the FL server and multiple clients run in the same process during simulation. A multi-process option allows efficient use of resources, e.g., training multiple clients on different GPUs. The simulator follows the same job execution as in real-world NVFlare deployment. Therefore, components developed in simulation can be directly deployed in real-world federated scenarios.

3.2 Secure Provisioning in NVFlare

Security is an important requirement for FL systems. NVFlare provides security solutions in the following areas: authentication, communication confidentiality, user authorization, data privacy protection, auditing, and local

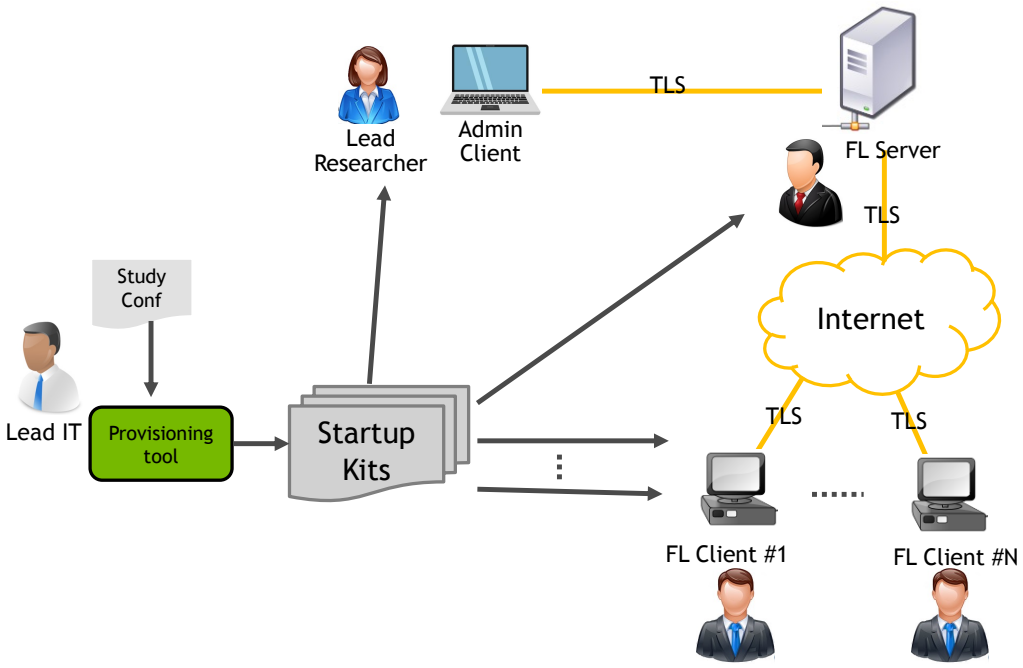


Figure 3: High-level steps for running a real-world study with secure provisioning with NVFlare.

client policies.

Authentication NVFlare ensures the identities of communicating peers using mutual Transport Layer Security (TLS). Each participating party (FL Servers, Overseer, FL Clients, Admin Clients) must be properly provisioned. Once provisioned, each party receives a startup kit containing TLS credentials (public cert of the root, the party's own private key and certificate) and system endpoint information, see Fig. 3. Each party can only connect to the NVFlare system with the startup kit. Communication confidentiality is also achieved with the use of TLS-based messaging.

Federated Authorization NVFlare's admin command system is very rich and powerful. Not every command is for everyone. NVFlare implements a role-based user authorization system that controls what a user can or cannot do. At the time of provision, each user is assigned a role. Authorization policies specify which commands are permitted for which roles. Each FL client can define its authorization policy that specifies what a role can or cannot do to the client. For example, one client could allow a role to run jobs from any researchers. In contrast, another client may only allow jobs submitted by its researchers (i.e., the client and the job submitter belong to the same organization).

NVFlare automatically records all user commands and job events in system audit files on both the server and client sides. In addition, the audit API can be used by application developers to record additional events in the audit files.

Client-Privacy NVFlare enhances the overall system security by allowing each client to define its policies for authorization, data privacy (filters), and computing resource management. The client can change its policies at any time after the system is up and running without having to be re-provisioned. For example, the client could require all jobs running on it to be subject to a set of filters. The client could also change the number of computing resources (e.g., GPU devices) to be used by the FL client.

4 Federated Data Science

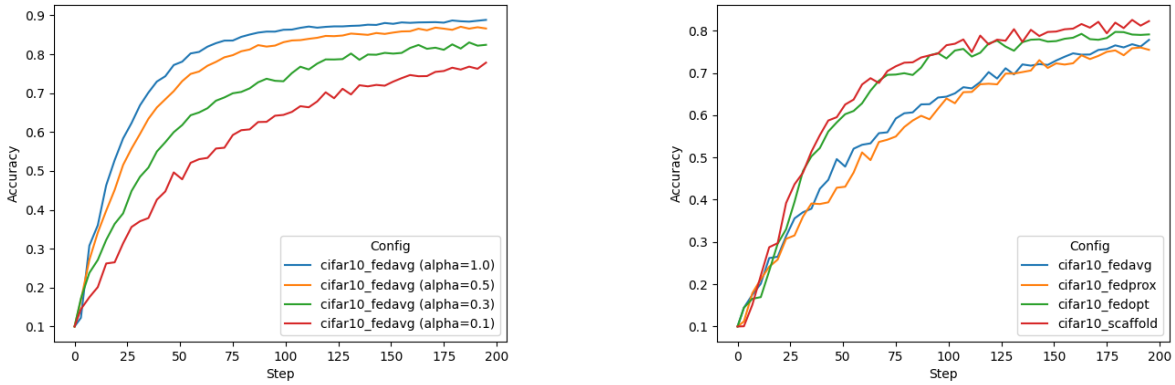
As a general distributed computing platform, NVFlare can be used for various applications in different industries. Here we describe some of the most common use cases where NVFlare was deployed.

4.1 Federated Deep Learning

A go-to example dataset for benchmarking different FL algorithms is CIFAR-10 [20]. NVFlare allows users to experiment with different algorithms and data splits using different levels of heterogeneity based on a Dirichlet sampling strategy [41]. Figure 4(a) shows the impact of varying alpha values, where lower values cause higher heterogeneity on the performance of the FedAvg.

Apart from FedAvg, currently available in NVFlare include FedProx [23], FedOpt [32], and SCAFFOLD [19]. Figure 4(b) compares an α setting of 0.1, causing a high data heterogeneity across clients and its impact on more advanced FL algorithms, namely FedProx, FedOpt, and SCAFFOLD. FedOpt and SCAFFOLD show markedly better convergence rates and achieve better performance than FedAvg and FedProx with the same alpha setting. SCAFFOLD achieves this by adding a correction term when updating the client models, while FedOpt utilizes SGD with momentum to update the global model on the server. Therefore, both perform better with the same number of training steps as FedAvg and FedProx.

Other algorithms available in or coming soon to NVFlare include federated XGBoost [6], Ditto [22], FedSM [45], Auto-FedRL [11], and more.



(a) FedAvg with increasing levels of heterogeneity (smaller α values).

(b) FL algorithms with a heterogeneous data split (α=0.1).

Figure 4: Federated learning experiments with NVFlare.

4.2 Federated Machine Learning

Traditional machine learning methods, such as linear models, support vector machine (SVM), and k-means clustering, can be formulated under a federated setting.

With certain libraries, the federated machine learning algorithms need to be designed considering two factors: algorithm-wise, each of these models has distinct training schemes and model representations; and implementation-wise, popular libraries providing these functionalities (e.g., scikit-learn, XGBoost) have different APIs and inner logics. Hence, when developing an FL variant of a particular traditional machine learning method, several questions need to be answered at these two levels:

First, at the algorithm level, we need to break down the optimization process into individual steps/rounds (if possible) and have answers to three major questions:

1. What information should clients share with the server?
2. How should the server aggregate the collected information from clients?
3. What should clients do with the global aggregated information received from the server?

Second, at the implementation level, we need to know what APIs are available and how to utilize them in a federated pipeline to implement a distributed version of the algorithm.

A major difference between federated traditional machine learning and federated deep learning is that, for traditional machine learning methods, the boundary between "federated" and "distributed", or even "ensemble", can be much more vague than for deep learning. Due to the characteristics of a given algorithm and its API design, the concepts can be equivalent. Take XGBoost and SVM, for example: Algorithm-wise, XGBoost can distribute the training samples to several workers and construct trees based on the collected histograms from each worker. Such a process can be directly adopted under a federated setting because the communication cost is affordable. In this case, "federated" is equivalent to "distributed" learning. API-wise, some algorithms can be constrained by their implementation. Take scikit-learn's SVM for instance. Although theoretically SVM can be formulated as an iterative optimization process, the API only supports one-shot "fitting" without the capability of separately calling the optimization steps. Hence a federated SVM algorithm using the scikit-learn library can only be implemented as a two-step process. In this case, "federated" is equivalent to "ensemble".

For clarification, we provide the full formulation for tree-based federated XGBoost, illustrated in Fig. 5:

1. XGBoost, by definition, is a sequential optimization process: each step adds one extra tree to the model to reduce the residual error. Hence, federated XGBoost can be formulated as follows: each round of FL corresponds to one boosting step at the local level. Clients share the newly added tree trained on local data with the server at the end of local boosting.
2. The model representation is a decision/regression tree. To aggregate the information from all clients, the server will bag all received trees to form a "forest" to be added to the global boosting model.
3. With the updated global model from the server, each client will continue the boosting process by learning a new tree starting from the global model of the boosted forest.

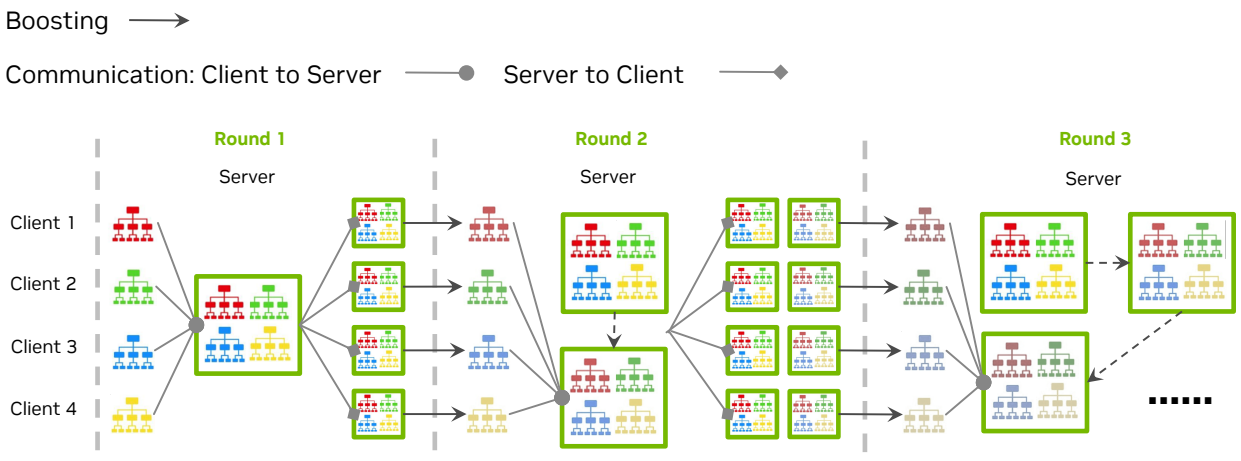
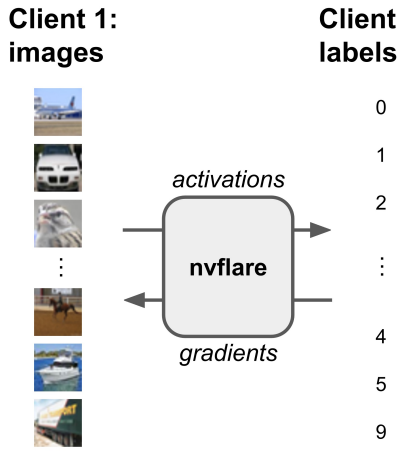


Figure 5: Tree-based federated XGBoost: a "boosting of forests."

4.3 Split learning

Split learning assumes a vertical data partitioning [35] that can be useful in many distributed learning scenarios involving neural network architectures [12].

As an introductory example, we can assume that one client holds the images, and the other holds the labels to compute losses and accuracy metrics. Activations and corresponding gradients are being exchanged between the clients using NVFlare, as illustrated in Fig. 6. We use a cryptographic technique called private set intersection



Setup	Training Time [min]
Simulated PyTorch	19
Routing through server (TCP)	27
Peer-to-peer (TCP)	25

Figure 6: Simple split learning scenario using CIFAR-10. The table compares multiple communication patterns. Using 50,000 training samples and 15,625 rounds of communication with a batch size of 64.

(PSI) [43] to compute the alignment between images and labels on both clients. NVFlare's implementation of PSI can be extended to multiple parties and applied to other use cases than split learning, e.g., requiring a secure and privacy-preserving alignment of different databases.

Using NVFlare's capability to implement different communication patterns, we can investigate the communication speed-ups one can achieve by implementing split learning using direct peer-to-peer communication as opposed to routing the messages between the two clients through a central server.

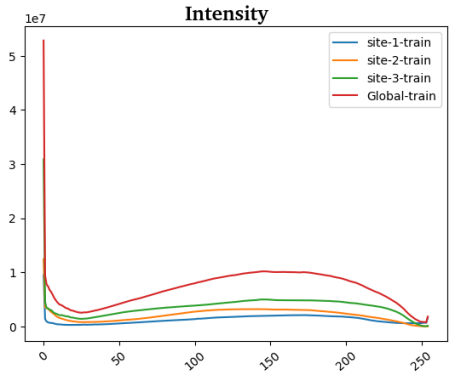
The table in Fig. 6 compares the training speeds of split learning on the CIFAR-10 dataset in a local simulation scenario. First, we use the same PyTorch script to simulate split learning. Then, we implement two distributed solutions using NVFlare. One that routes the messages through the server and one using a direct peer-to-peer connection between the clients. As expected, the direct peer-to-peer connection is more efficient, achieving only a slight overhead in total training time compared to the standalone PyTorch script, which could not be translated to real-world scenarios.

4.4 Federated Statistics

NVFlare provides built-in federated statistics operators (*Controller* and *Executors*) that will generate global statistics based on local client statistics. Each client could have one or more datasets, such as "train" and "test" datasets. Each dataset may have many features. NVFlare will calculate and combine the statistics for each feature in the dataset to produce global statistics for all the numeric features. The output gathered on the server will be the complete statistics for all datasets in clients and global, as illustrated in Fig. 7.

	count	histogram
site-2-train	6012	[[0.0, 1.0, 12430030], [1.0, 2.0, 3511491], [2...
site-4-train	1345	NaN
site-3-train	10192	[[0.0, 1.0, 30867349], [1.0, 2.0, 4553187], [2...
site-1-train	3616	[[0.0, 1.0, 9512374], [1.0, 2.0, 1381654], [2....
Global-train	21165	[[0.0, 1.0, 52809753], [1.0, 2.0, 9446332], [2...

(a) Federated statistics. Note the data of "site-4" violates the client's privacy policy and therefore does not share its statistics with the server.



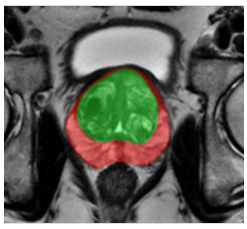
(b) Histogram visualization.

Figure 7: Federated statistics with NVFlare.

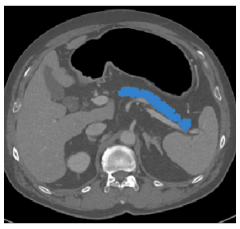
5 Real-world Use Cases

NVFlare and its predecessors have been used in several real-world studies exploring FL for healthcare scenarios. The collaborations between multinational institutions tested and validated the utility of federated learning, pushing the envelope for training robust, generalizable AI models. These initiatives included FL for breast mammography classification [35], prostate segmentation [37], pancreas segmentation [41], and most recently, chest X-ray (CXR) and electronic health record (EHR) analysis to predict the oxygen requirement for patients arriving in the emergency department with symptoms of COVID-19 [7].

(a) Mammography.



(b) Prostate.



(c) Pancreas.

(d) CXR & EHR.

Figure 8: Real-world use cases of NVFlare.

6 Summary & Conclusion

We described NVFlare, an open-source SDK to make it easier for data scientists to use FL in their research and to allow an easy transition from research to real-world deployment. As discussed above, NVFlare's *Controller* programming API supports various interaction patterns between the server and clients over internet connections, which could be unstable. Therefore, the API design mitigates various failure conditions and unexpected crashes of the client machines, such as allowing developers to process timeout conditions properly.

NVFLare's unique flexibility and agnostic approach towards the deployed training libraries make it the perfect solution for integrating with different deep learning frameworks, including popular ones used for training large language models (LLM). With our dedication to addressing the current limitations of communication protocols, we are working towards supporting the communication of large message sizes, enabling the federated

fine-tuning of AI models with billions of parameters, such as those used for ChatGPT [31] and GPT-4 [30]. Moreover, our team is implementing parameter-efficient federated methods to adapt LLM models to downstream tasks [47], utilizing techniques such as prompt tuning [21] and p-tuning [25], adapters [16, 15], LoRA [17], showing promising performance. Our commitment to innovation and excellence in this field ensures that we continue to push the boundaries of what is possible with federated learning.

We did not go into all details of exciting features available in NVFlare, like homomorphic encryption, TensorBoard streaming, provisioning web dashboard, integration with MONAI[4] [29, 4], etc. However, we hope that this overview of NVFlare gives a good starting point for developers and researchers on their journey to using FL and federated data science in simulation and the real world.

NVFlare is an open-source project. We invite the community to contribute and grow NVFlare. For more information, please visit the code repository at `https://github.com/NVIDIA/NVFlare`.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16), pages 265–283, 2016.

[2] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pages 308–318, 2016.

[3] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, P. P. de Gusmão, and N. D. Lane. Flower: A friendly federated learning research framework. arXiv preprint arXiv:2007.14390, 2020.

[4] M. J. Cardoso, W. Li, R. Brown, N. Ma, E. Kerfoot, Y. Wang, B. Murrey, A. Myronenko, C. Zhao, D. Yang, et al. Monai: An open-source framework for deep learning in healthcare. arXiv preprint arXiv:2211.02701, 2022.

[5] K. Chang, N. Balachandar, C. Lam, D. Yi, J. Brown, A. Beers, B. Rosen, D. L. Rubin, and J. Kalpathy-Cramer. Distributed deep learning networks among institutions for medical imaging. Journal of the American Medical Informatics Association, 25(8):945–954, 2018.

[6] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[7] I. Dayan, H. R. Roth, A. Zhong, A. Harouni, A. Gentili, A. Z. Abidin, A. Liu, A. B. Costa, B. J. Wood, C.-S. Tsai, et al. Federated learning for predicting clinical outcomes in patients with covid-19. Nature medicine, 27(10):1735–1743, 2021.

[8] D. Dimitriadis, M. H. Garcia, D. M. Diaz, A. Manoel, and R. Sim. Flute: A scalable, extensible framework for high-performance federated learning simulations. arXiv preprint arXiv:2203.13789, 2022.

[9] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In Theory of cryptography conference, pages 265–284. Springer, 2006.

[10] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller. Inverting gradients-how easy is it to break privacy in federated learning? Advances in Neural Information Processing Systems, 33:16937–16947, 2020.

[4] `https://monai.io`

[11] P. Guo, D. Yang, A. Hatamizadeh, A. Xu, Z. Xu, W. Li, C. Zhao, D. Xu, S. Harmon, E. Turkbey, et al. Auto-fedrl: Federated hyperparameter optimization for multi-institutional medical image segmentation. arXiv preprint arXiv:2203.06338, 2022.

[12] O. Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents. Journal of Network and Computer Applications, 116:1–8, 2018.

[13] A. Hatamizadeh, H. Yin, P. Molchanov, A. Myronenko, W. Li, P. Dogra, A. Feng, M. G. Flores, J. Kautz, D. Xu, et al. Do gradient inversion attacks make federated learning unsafe? arXiv preprint arXiv:2202.06924, 2022.

[14] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, et al. FedML: A research library and benchmark for federated machine learning. arXiv preprint arXiv:2007.13518, 2020.

[15] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig. Towards a unified view of parameter-efficient transfer learning. arXiv preprint arXiv:2110.04366, 2021.

[16] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In International Conference on Machine Learning, pages 2790–2799. PMLR, 2019.

[17] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

[18] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977, 2019.

[19] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh. Scaffold: Stochastic controlled averaging for federated learning. In International Conference on Machine Learning, pages 5132–5143. PMLR, 2020.

[20] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691, 2021.

[22] T. Li, S. Hu, A. Beirami, and V. Smith. Ditto: Fair and robust federated learning through personalization. In International Conference on Machine Learning, pages 6357–6368. PMLR, 2021.

[23] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. Proceedings of Machine Learning and Systems, 2:429–450, 2020.

[24] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso, et al. Privacy-preserving federated brain tumour segmentation. In International workshop on machine learning in medical imaging, pages 133–141. Springer, 2019.

[25] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. Gpt understands, too. arXiv preprint arXiv:2103.10385, 2021.

[26] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang. Fate: An industrial grade platform for collaborative learning with data protection. J. Mach. Learn. Res., 22(226):1–6, 2021.

[27] H. Ludwig, N. Baracaldo, G. Thomas, Y. Zhou, A. Anwar, S. Rajamoni, Y. Ong, J. Radhakrishnan, A. Verma, M. Sinn, et al. Ibm federated learning: an enterprise framework white paper v0. 1. arXiv preprint arXiv:2007.10987, 2020.

[28] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In Artificial intelligence and statistics, pages 1273–1282. PMLR, 2017.

[29] MONAI Consortium. MONAI: Medical Open Network for AI, 9 2022.

[30] OpenAI. Gpt-4 technical report, 2023.

[31] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35:27730–27744, 2022.

[32] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan. Adaptive federated optimization. arXiv preprint arXiv:2003.00295, 2020.

[33] G. A. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov, et al. Openfl: An open-source framework for federated learning. arXiv preprint arXiv:2105.06413, 2021.

[34] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein, et al. The future of digital health with federated learning. NPJ digital medicine, 3(1):1–7, 2020.

[35] H. R. Roth, K. Chang, P. Singh, N. Neumark, W. Li, V. Gupta, S. Gupta, L. Qu, A. Ihsani, B. C. Bizzo, et al. Federated learning for breast density classification: A real-world implementation. In Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning, pages 181–191. Springer, 2020.

[36] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora. Fetchsgd: Communication-efficient federated learning with sketching. In International Conference on Machine Learning, pages 8253–8265. PMLR, 2020.

[37] K. V. Sarma, S. Harmon, T. Sanford, H. R. Roth, Z. Xu, J. Tetreault, D. Xu, M. G. Flores, A. G. Raman, R. Kulkarni, et al. Federated learning improves site performance in multicenter deep learning without data sharing. Journal of the American Medical Informatics Association, 28(6):1259–1264, 2021.

[38] M. J. Sheller, B. Edwards, G. A. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. R. Colen, et al. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. Scientific reports, 10(1):1–12, 2020.

[39] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas. Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In International MICCAI Brainlesion Workshop, pages 92–104. Springer, 2018.

[40] S. Silva, A. Altmann, B. Gutman, and M. Lorenzi. Fed-biomed: A general open-source frontend framework for federated learning in healthcare. In Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning, pages 201–210. Springer, 2020.

[41] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni. Federated learning with matched averaging. arXiv preprint arXiv:2002.06440, 2020.

[42] S. Warnat-Herresthal, H. Schultze, K. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Händler, P. Pickkers, N. A. Aziz, et al. Swarm learning for decentralized and confidential clinical machine learning. Nature, 594(7862):265–270, 2021.

[43] Wikipedia contributors. Private set intersection — Wikipedia, the free encyclopedia, 2023. [Online; accessed 27-April-2023].

[44] Y. Xie, Z. Wang, D. Chen, D. Gao, L. Yao, W. Kuang, Y. Li, B. Ding, and J. Zhou. Federatedscope: A comprehensive and flexible federated learning platform via message passing. arXiv preprint arXiv:2204.05011, 2022.

[45] A. Xu, W. Li, P. Guo, D. Yang, H. R. Roth, A. Hatamizadeh, C. Zhao, D. Xu, H. Huang, and Z. Xu. Closing the generalization gap of cross-silo federated medical image segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 20866–20875, 2022.

[46] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST), 10(2):1–19, 2019.

[47] H. Zhao, W. Du, F. Li, P. Li, and G. Liu. Reduce communication costs and preserve privacy: Prompt tuning method in federated learning. arXiv preprint arXiv:2208.12268, 2022.

[48] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, et al. Pysyft: A library for easy federated learning. In Federated Learning Systems, pages 111–139. Springer, 2021.

FedCLIP: Fast Generalization and Personalization for CLIP in Federated Learning

Wang Lu[1] Xixu Hu[2] Jindong Wang[3*] Xing Xie[3]
[1] National Engineering Research Center, Beijing, China
[2] City University of Hong Kong, Hong Kong
[3] Microsoft Research Asia, Beijing, China
newlw230630@gmail.com, xixuhu2-c@my.cityu.edu.hk, {jindong.wang, xingx}@microsoft.com

Abstract

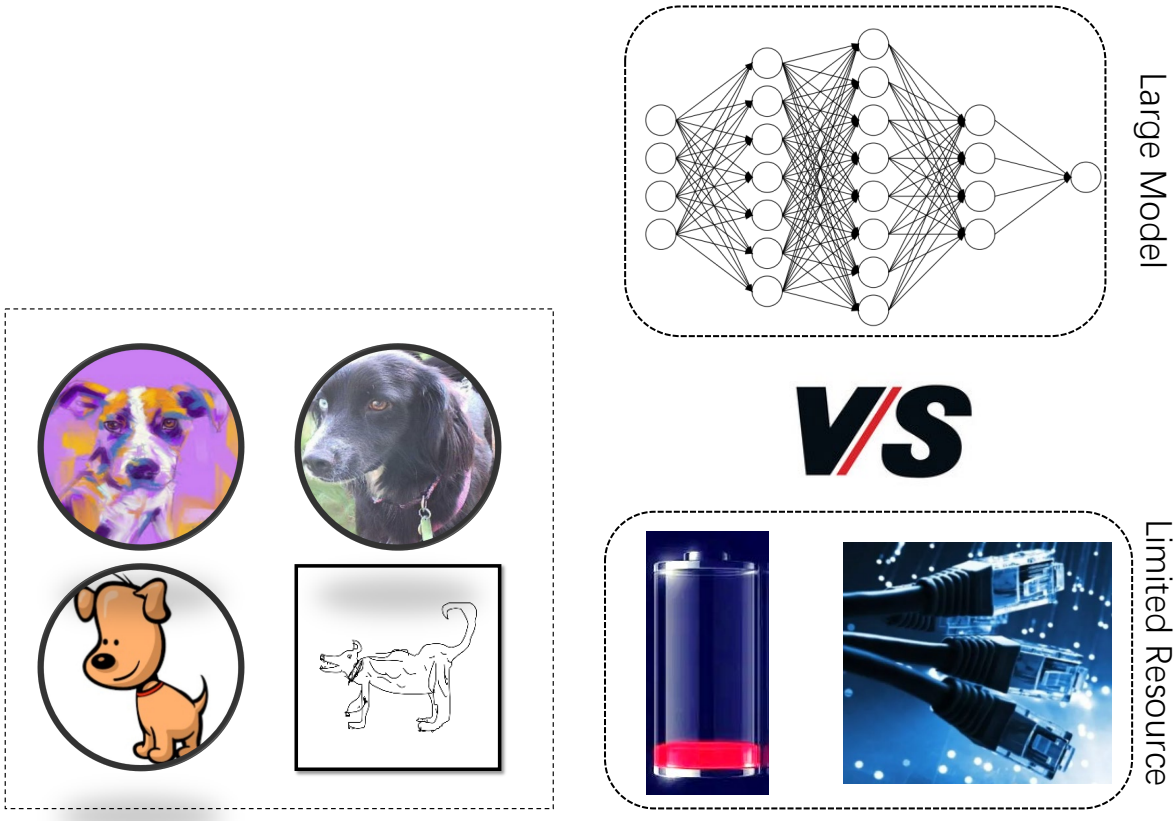
*Federated learning (FL) has emerged as a new paradigm for privacy-preserving computation in recent years. Unfortunately, FL faces two critical challenges that hinder its actual performance: data distribution heterogeneity and high resource costs brought by large foundation models. Specifically, the non-IID data in different clients make existing FL algorithms hard to converge while the high resource costs, including computational and communication costs that increase the deployment difficulty in real-world scenarios. In this paper, we propose an effective yet simple method, named FedCLIP, to achieve fast generalization and personalization for CLIP in federated learning. Concretely, we design an attention-based adapter for the large model, CLIP, and the rest operations merely depend on adapters. Lightweight adapters can make the most use of pretrained model information and ensure models be adaptive for clients in specific tasks. Simultaneously, small-scale operations can mitigate the computational burden and communication burden caused by large models. Extensive experiments are conducted on three datasets with distribution shifts. Qualitative and quantitative results demonstrate that FedCLIP significantly outperforms other baselines (**9**% overall improvements on PACS) and effectively reduces computational and communication costs (**283x** faster than FedAVG). Our code will be available at:* `https://github.com/microsoft/PersonalizedFL`.

1 Introduction

The success of machine learning, especially deep learning, is inseparable from a large amount of data. However, data, as an important resource, usually scatter across different individuals or organizations. In recent years, people pay more attention to data privacy and security and some organizations even enact relevant regulations and laws, e.g. The EU general data protection regulation (GDPR) [49] and China's cyber power [20]. Under this circumstance, direct raw data communication can be impossible in reality, making traditional data-centric machine learning paradigms unlikely to work. To cope with this challenge, federated learning (FL) [35] emerges as a new distributed machine learning paradigm and has been widely adopted in various applications.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Corresponding author: Jindong Wang: jindong.wang@microsoft.com.



(a) Data distribution shifts

(b) Huge resource demands

Figure 1: Existing issues in federated learning. In Figure 1(a), circles denote participated clients while squares denote unseen targets.

Federated learning makes it possible to perform model aggregation without directly accessing the raw user data from different clients. One of the earliest works in FL is called FedAVG [33] which aggregates distributed information using a simple and powerful averaging algorithm. FedAVG mainly contains four steps, including training local models with local data, uploading local models to the server, aggregating models in the server, and distributing the aggregated model to each individual or organization. These four steps are executed for multiple rounds for better information aggregation. FedAVG can ensure that raw data does not leave the local client and thus protect data privacy and security. Due to its simplicity and great performance, FedAVG quickly became popular in many areas [24, 40, 3].

In this paper, we are specially interested in federated learning under the large foundation models era [4]. Foundation models, as suggested by the name, have become increasingly popular in different machine learning tasks, such as Vision Transformer in computer vision [56] and the GPT series in natural language processing [37]. Since these models are extremely large, e.g., GPT-3 [5] has 175 billion parameters, our key question is: how to perform effective and efficient federated learning using these large models?

Specifically, two critical research challenges arise in this situation: data distribution shifts and huge resource demands. On the one hand, data distribution shifts widely exist in the real world, e.g. figures shown in Figure 1(a). When meeting heterogeneous data, common federated learning methods can suffer from slow convergence and low accuracy due to inconsistent optimization directions, local optima, or some other factors [12]. A qualified FL model can cope with both various clients and unseen targets, i.e. personalization and generalization. On

the other hand, huge resource demands of increasingly popular large models lead to conflicts with realistically constrained resources, as shown in Figure 1(b). In addition to high computational costs, communication cost is also a critical metric in federated learning. For instance, the CLIP [36] model based on VIT-B/32 contains more than 10^8 trainable parameters and most existing networks cannot afford to transmit it quickly. Achieving fast generalization and personalization with minimal resource costs is an urgent issue to be addressed.

Some existing work tried to address the issues mentioned above [32, 55, 14]. FedAP [32] attempted to learn the similarity among clients and then leveraged the learned similarity matrix to guide aggregation. FedAP could achieve acceptable personalization results but it ignored generalization. Another paper [55] discussed two gaps, including the out-of-sample gap and the participation gap. These two gaps correspond to goals of generalization and personalization respectively. This paper performed extensive empirical studies to analyze these issues but it did not offer a possible solution for large models. PromptFL [14] only updated the prompts instead of the whole model to accelerate the whole process. However, clients still require large amounts of computation and PromptFL is not designed for personalization and generalization.

In this paper, we propose FedCLIP to achieve fast generalization and personalization for CLIP in federated learning. Since larger pretrained models, e.g. CLIP, have contained enough prior information, our goal is to find where we should focus in specific tasks. The core part of FedCLIP is AttAI, an attention-based adapter for the image encoder in CLIP. Instead of finetuning whole networks, AttAI directly utilizes fixed features extracted by pretrained models and explores where FedCLIP should pay attention to for specific tasks. Simply training AttAI can ensure FedCLIP preserving prior information as much as possible while it allows models adapted for specific tasks. Through AttAI, FedCLIP does not rely on pretrained models anymore once obtaining diversified and robust features and thus FedCLIP can save large amounts of computational costs and communication costs. Therefore, FedCLIP is extensible and can be deployed to many applications.

Our contributions are as follows.

1. We propose FedCLIP, a fast generalization and personalization learning method for CLIP in federated learning. It can achieve personalization for participating clients and its remarkable generalization ability can attract new clients.

2. Extensive experiments on three public image benchmarks demonstrate that FedCLIP can have achieved personalization and generalization performance at the same time (**9**% overall improvements on PACS). More importantly, FedCLIP reduces the number of trainable parameters thus saving communication costs and computational costs (**283x** faster than FedAVG).

3. FedCLIP is extensible and can be applied in many real applications, which means it can work well in many circumstances. We can even embed it in some other architectures, e.g. BERT [46] and ViT [16]. Our code will be available at: `https://github.com/microsoft/PersonalizedFL`.

The remainder of this paper is organized as follows. In Sec. 2, we introduce related work. And then we elaborate on the proposed method in Sec. 3. Extensive experiments are reported and analyzed in Sec. 4. Finally, we conclude the paper and provide possible future work in Sec. 5.

2 Related Work

2.1 Challenges in Machine Learning

Machine learning has achieved great success and gradually entered people's daily lives [42, 34, 50]. It has been applied to many fields, e.g. human activity recognition [29], face recognition [28], and healthcare [8]. Successful machine learning applications, especially deep learning based applications, often require a large amount of data and lots of computational resources. In most cases, a deluge of data and computing resources can lead to easy

success, such as ChatGPT [47]. However, data and computation also mean money and resources. In reality, it is impossible to aggregate all data together in some situations. There seems to be a contradiction between the massive resource requirements of traditional methods and the limited real environment.

Generalization is another challenging problem caused by data distribution shifts. Its goal is to learn a generalized model with limited data and it expects that the learned model can work well on unseen targets with unknown distributions. [51] gives a survey on domain generalization and first groups existing methods into three categories, including data manipulation [31], representation learning [30], and learning strategy [19].

2.2 Federated Learning

Data is often scatted everywhere and cannot be aggregated together due to some factors, such as laws and regulations [49] and the awakening of people's awareness of data security and privacy protection. In such an environment, federated learning came into being [35, 41]. According to [35], federated learning can be grouped into three categories, including horizontal federated learning, vertical federated learning, and federated transfer learning. Most deep learning based methods belong to horizontal federated learning and so is this paper. For a more detailed introduction, please refer to the survey [27].

FedAVG is a traditional horizontal federated learning method [33]. Although it is simple, it was applied in many applications. When meeting data distribution heterogeneity, FedAVG appeared powerless [43]. And many researchers proposed various methods to solve the above problems. FedProx [25] added a proximal regularized term to FedAVG and it allowed slight model gaps between clients and the server. In FedBN [26], the authors thought that parameters in batch normalization layers can represent data distribution, and keeping specific batch normalization layers for each client could make local models personalized. Another latest method, FedAP [32], learned the similarity between clients based on the statistics of the batch normalization layers while preserving the specificity of each client with different local batch normalization. The above methods all achieve satisfactory results in their corresponding scenarios. However, most of them focused on personalization and ignored generalization issues [7].

Generalization in federated learning is a novel problem. In recent two years, some papers tried to solve this problem. [55] first discussed the generalization in federated learning and it proposed a framework to disentangle performance gaps, including out-of-sample gaps and participation gaps. FED-DRO [7] proposed a novel federated learning framework to explicitly decouple a model's dual duties with two prediction tasks and it mainly focused on label shifts. Some other work tried to adapt existing domain generalization methods to generalization in federated learning [15, 45, 35, 6]. FL Games [15] utilized Nash equilibrium to learn causal features that were invariant across clients which is similar to Invariant Risk Minimization (IRM) [2]. FedSAM [35] proposed a general effective algorithm based on Sharpness Aware Minimization (SAM) local optimizer [11]. Although these methods can bring generalization, they were not designed for large models and could not make full use of knowledge brought by pretrained models.

2.3 CLIP and Large Models

From perceptron [13] to AlexNet [1] to ResNet [17] to Vision transformer [56] to CLIP [36], pretrained models have become larger and larger. The importance of pretrained models has been increasing and pretrained models contain a growing amount of knowledge. For specific applications, researchers usually choose suitable backbone models and then adopt some techniques, e.g. finetune [44], to slightly adapt pretrained models. Since pretrained models are trained via a large amount of data, features extracted from them are often generalized and insightful. Few works pay attention to large models in federated learning and high demands of computational costs and communication costs hinder the development of this field. In this paper, we focus on CLIP in federated learning.

CLIP [36] learned SOTA image representations from scratch on a dataset of 400 million(image, text) pairs collected from the internet. The natural language was used to reference learned visual concepts. It has been

applied in many fields and demonstrated its superiority [38, 22]. However, in federated learning, CLIP is still in its infancy. PromptFL [14] replaced the federated model training with the federated prompt training to simultaneously achieve efficient global aggregation and local training by exploiting the power of foundation models in a distributed way. However, it still requires certain computational costs and it is not designed for data distribution heterogeneity problems. Moreover, it is hard to tune the hyperparameters for the prompt techniques in Transformer.In this paper, we focus on fast personalization and generalization for CLIP.

3 Method

3.1 Problem Formulation

In a generalization and personalization federated learning setting, N different clients, denote as $\{C_1, C_2, \cdots, C_N\}$, participate in exchanging information and they have data, denoted as $\{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_N\}$ with different distributions, which means $P(\mathcal{D}_i) \neq P(\mathcal{D}_j)$. In this paper, we only focus on homogeneous data with the same input space and output space, i.e. $\mathcal{X}_i = \mathcal{X}_j, \mathcal{Y}_i = \mathcal{Y}_j, \forall i \neq j$. Each dataset, $\mathcal{D}_i = \{(\mathbf{x}_{i,j}, y_{i,j})\}_{j=1}^{n_i}$, consists of three parts, a training dataset $\mathcal{D}_i^{train} = \{(\mathbf{x}_{i,j}^{train}, y_{i,j}^{train})\}_{j=1}^{n_i^{train}}$, a validation dataset $\mathcal{D}_i^{valid} = \{(\mathbf{x}_{i,j}^{valid}, y_{i,j}^{valid})\}_{j=1}^{n_i^{valid}}$ and a test dataset $\mathcal{D}_i^{test} = \{(\mathbf{x}_{i,j}^{test}, y_{i,j}^{test})\}_{j=1}^{n_i^{test}}$. Three sub-datasets in each client have no overlap and $n_i = n_i^{train} + n_i^{valid} + n_i^{test}, \mathcal{D}_i = \mathcal{D}_i^{train} \cup \mathcal{D}_i^{valid} \cup \mathcal{D}_i^{test}$. Our goal is to aggregate all clients' information with preserving data privacy and security and learn a good model f for each client $\mathcal{D}_i$:

$$\min_f \frac{1}{N} \sum_{i=1}^{N} \frac{1}{n_i^{test}} \sum_{j=1}^{n_i^{test}} \ell(f(\mathbf{x}_{i,j}^{test}), y_{i,j}^{test}), \tag{1}$$

where ℓ is a loss function. Moreover, for generalization, we assume that there exist M different clients, denote as $\{F_1, F_2, \cdots, F_M\}$, with data $\{\mathcal{D}_1^F = \{(\mathbf{x}_{i,j}, y_{i,j})\}_{j=1}^{m_1}, \mathcal{D}_2^F = \{(\mathbf{x}_{i,j}, y_{i,j})\}_{j=1}^{m_2}, \cdots, \mathcal{D}_N^F = \{(\mathbf{x}_{i,j}, y_{i,j})\}_{j=1}^{m_M}\}$. These M clients do not participate in training, and we hope f can also be able to perform well on these clients.

$$\min_f \frac{1}{M} \sum_{i=1}^{M} \frac{1}{m_i} \sum_{j=1}^{m_i} \ell(f(\mathbf{x}_{i,j}), y_{i,j}), \tag{2}$$

3.2 Preliminaries

CLIP CLIP, Contrastive Language Image Pre-training, is an efficient and scalable method of learning [36]. To compensate for the problems caused by the amount of data and model parameters, it trained a large model with over 4×10^8 pairs of data. With help of natural language supervision, CLIP can better understand concepts of visual images and better learn the semantic connections behind images. Usually, CLIP models contain more information and they might be more robust.

A simple CLIP model regularly contains two parts, an image encoder f^I and a text encoder f^T. In common models, labels are frequently represented as numbers or one-hot vectors. For CLIP, to better utilize semantic information, these labels are often transformed into sentences, e.g. 'A photo of dogs'. And then text feature vectors, $\mathbf{T}$ are extracted from these sentences via f^T. Concurrently, images are encoded into visual feature vectors, $\mathbf{I}$, via f^I. Cosine similarities between $\mathbf{T}$ and $\mathbf{I}$ are used to training and predicting.

FedAVG In FedAVG [33], each client trains f with local clients' data, and then parameters of updated models, w_i, are transmitted to the server. The server typically aggregates the parameters according to Eq. 3,

$$w^* = \sum_{i=1}^{N} \frac{n_i}{\sum_{j=1}^{N} n_j} w_i \tag{3}$$

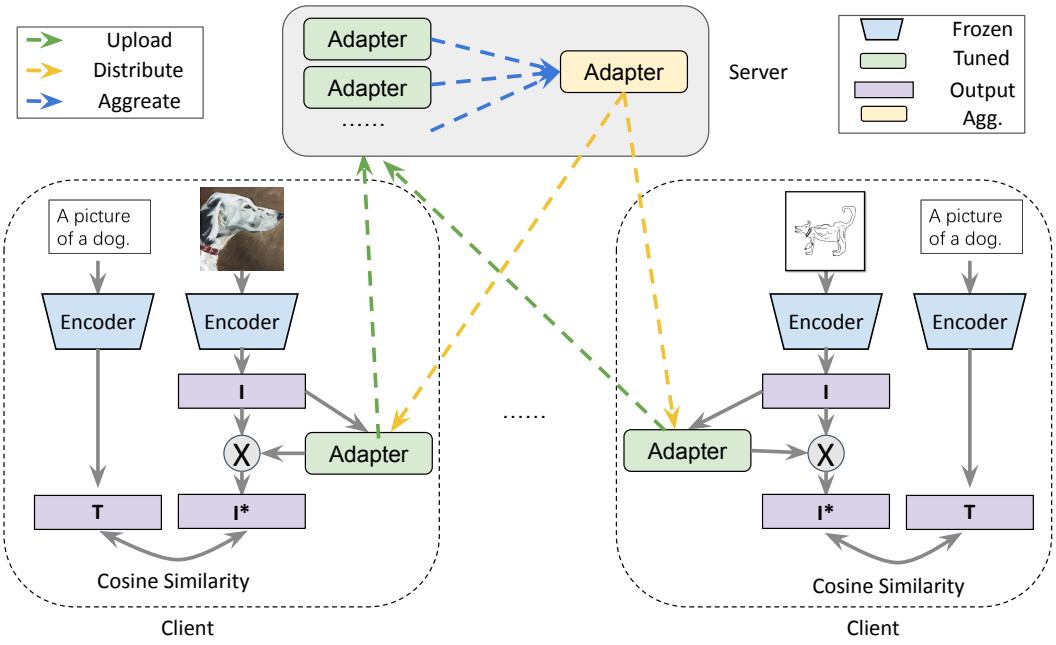


Figure 2: The framework of FedCLIP.

After aggregation, w^* is distributed. When $|w|$ is larger, the server cannot afford communication costs.

3.3 FedCLIP

To reduce computational costs and communications and make the most use of existing pretrained model information, we propose FedCLIP. Pretrained models already have abilities to extract robust and diversified features. Tuning whole networks with limited data can compromise the original ability of pretrained models. What we need to do is to try our best to preserve useful prior knowledge and let it be used to a suitable extent for our task. Besides, tuning large networks is impractical in federated learning due to limited resources in reality. Therefore, instead of operating on the whole model, FedCLIP concentrates on a simple attention-based adapter for the image encoder, AttAI.

Figure 2 gives the framework of FedCLIP. As shown in Figure 2, our method mainly contains four steps.

1. For Client i, we utilize a pretrained CLIP model to extract features of data, denoted as T_i and I_i.
2. In each client, we utilize $\mathcal{D}_i^{train}$ to train the corresponding adapter, g_i. And then we upload $\{g_i\}_{i=1}^N$ to the server.
3. In the server, the parameters of all g_i are weighted averaged and we can obtain g^*. The server then distributes g^* to each client and updates the parameters of each g_i.
4. Repeat Step 2 and Step 3 until convergence or reaching maximum rounds.

In step 1, we utilize the pretrained CLIP model to extract features. We consider the pretrained model is so powerful that we do not need to explore some other features. For $(\mathbf{x}, y)$, we can obtain corresponding features,

$$\mathbf{I} = f^I(\mathbf{x}), \mathbf{T} = f^T(y) \tag{4}$$

What we need to do next is to identify which parts of features are suitable for our specific tasks. Therefore, we introduce an attention-based adapter, g, to locate where we should concentrate on. Particularly, we utilize one linear layer, Tahn activation function, one linear layer, and Softmax activation function to construct g. The Softmax function is used to ensure our final outputs ranging from 0 to 1. Once we obtain the attention vector $att = g(\mathbf{I})$, we utilize it to update the visual feature via a dot multiply operation,

$$\mathbf{I}^* = g(\mathbf{I}) \cdot \mathbf{I}. \tag{5}$$

Then, similar to [36], we normalize $\mathbf{I}^*$ and $\mathbf{T}$ to compute the final logits.

$$\mathbf{I} = \frac{\mathbf{I}^*}{|I^*|}, \mathbf{T} = \frac{\mathbf{T}}{|\mathbf{I}|}, \tag{6}$$

$$\hat{\mathbf{I}} = s \times \mathbf{I} \times \mathbf{T}^T, \hat{\mathbf{T}} = \hat{\mathbf{I}}^T. \tag{7}$$

where s is a scale parameter.

Now, we can utilize the ground truth, a vector $\tilde{\mathbf{y}} = [0, 1, 2, 3, \cdots, B]$

$$\begin{aligned} \ell_{cls}^I &= \ell(\hat{\mathbf{I}}, \tilde{\mathbf{y}}), \\ \ell_{cls}^T &= \ell(\hat{\mathbf{T}}, \tilde{\mathbf{y}}), \end{aligned} \tag{8}$$

where ℓ is CrossEntropy loss [57] while B is the number of images in a batch.

We only exchange parameters of adapters, w^g, and therefore in the server, we replace Eq. 3 with Eq. 9.

$$w^{g,*} = \sum_{i=1}^{N} \frac{n_i}{\sum_{j=1}^{N} n_j} w_i^g. \tag{9}$$

Since w^g contains substantially less amount of trainable parameters than w, FedCLIP saves computational costs and communication costs.

3.4 Summary

For clarity, we give a detailed description of FedCLIP in Algorithm 2. In Line 1, directly obtaining generalized and diversified features with fixed CLIP make it possible to utilize more prior knowledge of pretrained models. In Line 2, with adapters, we can concentrate on valuable information and eliminate the influence of redundant information in specific tasks. Rich prior knowledge and targeted attention make the ultimately extracted features more robust, effective, and adaptable, resulting in our method having good generalization and personalization capabilities. From Line 2 to Line 5, performing computation and transmission merely with adapters can save a lot of resources and ensure the efficiency of our method.

3.5 Discussion

Adapter is a common technique in transfer learning [18]. It is at a small scale and has a plug-and-play implementation. In this paper, we mainly focus on adaptations to image encoders. Actually, we also can add adapters to text encoders. We can even change the inputs of text encoders to incorporate more semantic information.

4 Experiments

In this section, we extensively evaluate FedCLIP in three common visual image classification benchmarks.

Algorithm 2 FedCLIP

Input: N clients' datasets $\{\mathcal{D}_i\}_{i=1}^{N}$, a pretained CLIP model consist of an image encoder, f^I, and a text encoder, f^T

Output: An adapter g

1: For client i, computer the corresponding features $I_i = f^I(\mathbf{X}_i), T_i = f^T(\mathbf{Y}_i)$
2: For client i, train the local adapter, g_i, according to Eq.5 to Eq.9
3: Send the current adapter g_i to the server
4: Aggregate adapters' parameters via Eq. 9 and obtain w^{g*}
5: Transmit w^{g*} to each client
6: Repeat steps $2 \sim 5$ until convergence

4.1 Datasets

PACS PACS [23] is a popular object classification benchmark. It is composed of four sub-datasets, including photo, art-painting, cartoon, and sketch. There exist $9,991$ images in total and the dataset contains 7 classes, including dog, elephant, giraffe, guitar, horse, house, and person. Large discrepancies in image styles widely exist among different sub-datasets. In this paper, we view each sub-dataset as a client. We choose three sub-datasets as participated clients while the rest served as the target client to evaluate generalization ability. For each participated client, we split the corresponding sub-dataset into three parts, 60% for training, 20% for validation, and the rest 20% for testing. Validation parts of data are used for model selection.

VLCS VLCS [10] is another widely accepted public image classification benchmark. It also consists of four sub-datasets (VOC2007, LabelMe, Caltech10, and SUN09). It contains $10,729$ instances with 5 classes. Feature shifts exist generally among different sub-datasets. Similar to PACS, four sub-datasets correspond to four clients. Three sub-datasets play the roles of participants while the rest one act as an upcoming client.

Office-Home Office-Home [48] is a larger image classification benchmark, which contains 65 classes. Office-Home comprises four sub-datasets (Art, Clipart, Product, and Real_World) with about $15,500$ images. The feature shifts from Office-Home mainly come from image styles and viewpoints, but they are much smaller than PACS. We assess methods on Office-Home in a similar manner to PACS.

4.2 Implementation Details and Comparison Methods

For these three common image classification benchmarks, we use the CLIP pre-trained model with ViT-B/32 [9] as the image encoder. For model training, we utilize cross-entropy loss and Adam optimizer. The learning rate is tuned from 5×10^{-5} to 5×10^{-3}. We set local update epochs as $E = 1$ where E means the number of training epochs in one round while we set the total communication round number as $R = 200$. Since, at each time, we set one sub-dataset as the target, i.e. upcoming client, there exist four tasks for each benchmark. We run three trials to record the average results. To better illustrate the function and necessity of using larger pretrained models, we also utilize a related small architecture, AlexNet [21], to perform some base federated learning methods.

We compare our method with two methods including a common federated learning method, FedAVG, and a method designed for non-iid data, FedProx.

1. FedAVG [33]. The server aggregates all client models' parameters. FedAVG will aggregate networks with several layers for AlexNet while FedAVG will aggregate both image encoders and text encoders for CLIP.

2. FedProx [25]. It adds a proximal term to FedAVG and allows the existence of slight differences between clients and the server.

Table 2: Generalization accuracy. **Bold** means the best.

Dataset		PACS							Office-Home				
Backbone	Method	A	C	P	S	AVG	Backbone	Method	A	C	P	R	AVG
AlexNet	FedAVG	31.54	43.69	44.55	36.29	39.02	AlexNet	FedAVG	15.70	17.00	31.56	28.99	23.31
	FedProx	29.79	46.80	44.67	35.12	39.09		FedProx	16.48	17.66	29.83	27.98	22.99
CLIP	FedAVG	53.08	80.08	90.00	76.99	75.04	CLIP	FedAVG	65.60	57.64	71.64	75.42	67.57
	FedProx	66.06	87.33	91.68	78.42	80.87		FedProx	65.60	57.64	71.64	75.42	67.57
	Ours	**96.34**	**97.91**	**99.76**	**85.59**	**94.90**		Ours	**78.00**	**63.69**	**87.52**	**87.79**	**79.25**

Table 3: Personalization accuracy. **Bold** means the best.

Dataset			PACS							Office-Home				
Target	BackBone	Method	C	P	S	AVG	Target	BackBone	Method	C	P	R	AVG
A	AlexNet	FedAVG	72.86	61.08	78.22	70.72	A	AlexNet	FedAVG	50.74	63.47	38.81	51.01
		FedProx	71.37	56.89	81.53	69.93			FedProx	51.78	66.74	40.07	52.86
	CLIP	FedAVG	76.28	86.83	42.42	68.51		CLIP	FedAVG	64.38	79.14	78.76	74.09
		FedProx	90.81	90.42	63.95	81.73			FedProx	64.38	79.14	78.76	74.09
		Ours	**97.65**	**99.40**	**86.75**	**94.60**			Ours	**68.61**	**87.37**	**88.06**	**81.35**
			A	P	S	AVG				A	P	R	AVG
C	AlexNet	FedAVG	46.45	66.17	75.67	62.76	C	AlexNet	FedAVG	23.51	61.78	41.56	42.28
		FedProx	47.19	64.07	77.45	62.90			FedProx	24.54	64.04	40.18	42.92
	CLIP	FedAVG	84.11	92.81	81.02	85.98		CLIP	FedAVG	73.81	80.38	80.48	78.23
		FedProx	86.06	92.81	85.61	88.16			FedProx	73.81	80.38	80.48	78.23
		Ours	**96.33**	**99.10**	**86.88**	**94.10**			Ours	**78.97**	**87.60**	**87.60**	**84.72**
			A	C	S	AVG				A	C	R	AVG
P	AlexNet	FedAVG	37.65	75.00	81.53	64.73	R	AlexNet	FedAVG	23.30	49.94	40.87	38.04
		FedProx	35.45	73.93	83.57	64.32			FedProx	21.03	48.91	39.84	36.59
	CLIP	FedAVG	83.13	93.38	84.97	87.16		CLIP	FedAVG	70.93	**68.73**	77.73	72.46
		FedProx	83.86	93.59	88.54	88.66			FedProx	70.93	**68.73**	77.73	72.46
		Ours	**97.56**	**97.65**	**86.75**	**93.99**			Ours	**78.35**	68.38	**87.94**	**78.23**
			A	C	P	AVG				A	C	P	AVG
S	AlexNet	FedAVG	53.30	68.80	66.17	62.76	P	AlexNet	FedAVG	22.27	49.14	58.51	43.31
		FedProx	52.32	69.66	66.47	62.82			FedProx	20.21	50.06	58.29	42.85
	CLIP	FedAVG	90.71	94.02	94.91	93.21		CLIP	FedAVG	69.07	66.21	77.79	71.02
		FedProx	91.44	94.66	95.81	93.97			FedProx	69.07	66.21	77.79	71.02
		Ours	**97.31**	**97.65**	**99.40**	**98.12**			Ours	**78.56**	**68.50**	**87.37**	**78.14**

4.3 Results

Generalization Ability We first evaluate the generalization ability of each method via accuracy on clients that do not participate in training. Table 2 shows the generalization results for each task on PACS and Office-Home. We have the following observations from these results. 1) Our method achieves the best generalization ability on average with remarkable improvements (about 14% for PACS and about 12% for Office-Home). Moreover, our method achieves the best generalization ability in each task, which demonstrates the excellent generalization ability of our method. 2) Compared to methods with AlexNet as the backbone, methods with CLIP as the backbone can obtain better performance. It demonstrates that large well-trained models can be able to bring better generalization. 3) Compared to methods with CLIP as the backbone, our method has a further improvement, which demonstrates that our method leverages prior knowledge better.

Table 4: Comprehensive average accuracy. **Bold** means the best

Datasets	PACS					Office-Home					
Backbone	AlexNet		CLIP			Backbone	AlexNet		CLIP		
Methods	FedAVG	FedProx	FedAVG	FedProx	Ours	Methods	FedAVG	FedProx	FedAVG	FedProx	Ours
A	60.93	59.89	64.65	77.81	**95.04**	A	42.18	43.77	71.97	71.97	**80.51**
C	57.99	58.88	84.50	87.95	**95.06**	C	35.96	36.60	73.08	73.08	**79.46**
P	59.68	59.41	87.87	89.42	**95.43**	P	36.42	34.90	72.26	72.26	**80.55**
S	56.14	55.89	89.16	90.08	**94.99**	R	39.73	39.13	72.12	72.12	**80.55**
AVG	58.69	58.52	81.55	86.32	**95.13**	AVG	38.57	38.60	72.36	72.36	**80.27**

Table 5: Comprehensive average accuracy on VLCS. **Bold** means the best

Backbone	AlexNet		CLIP		
Methods	FedAVG	FedProx	FedAVG	FedProx	Ours
C	62.13	61.37	72.48	68.57	**83.68**
L	63.01	63.77	75.04	76.50	**82.62**
S	63.15	63.59	68.13	75.50	**82.82**
V	62.32	62.04	69.55	70.09	**83.30**
AVG	62.65	62.69	71.30	72.67	**83.11**

Personalization Ability Then, we evaluate the personalization ability of each method via the accuracy on test data of each participating client. Table 3 shows the personalization results for each task on PACS and Office-Home. We also have some insightful observations. 1)Although all clients share the same adapter in our method, our method still achieves the best average accuracy. Moreover, FedCLIP almost achieves the best performance on each client for every task. 2) Compared to methods with AlexNet, corresponding methods with CLIP perform better overall. For CLIP-based methods, results are quite sensitive to hyperparameters, e.g. learning rate. And FedAVG has disappointing results on some specific clients. 3) Our method has the most use of prior knowledge since it achieves the stablest results.

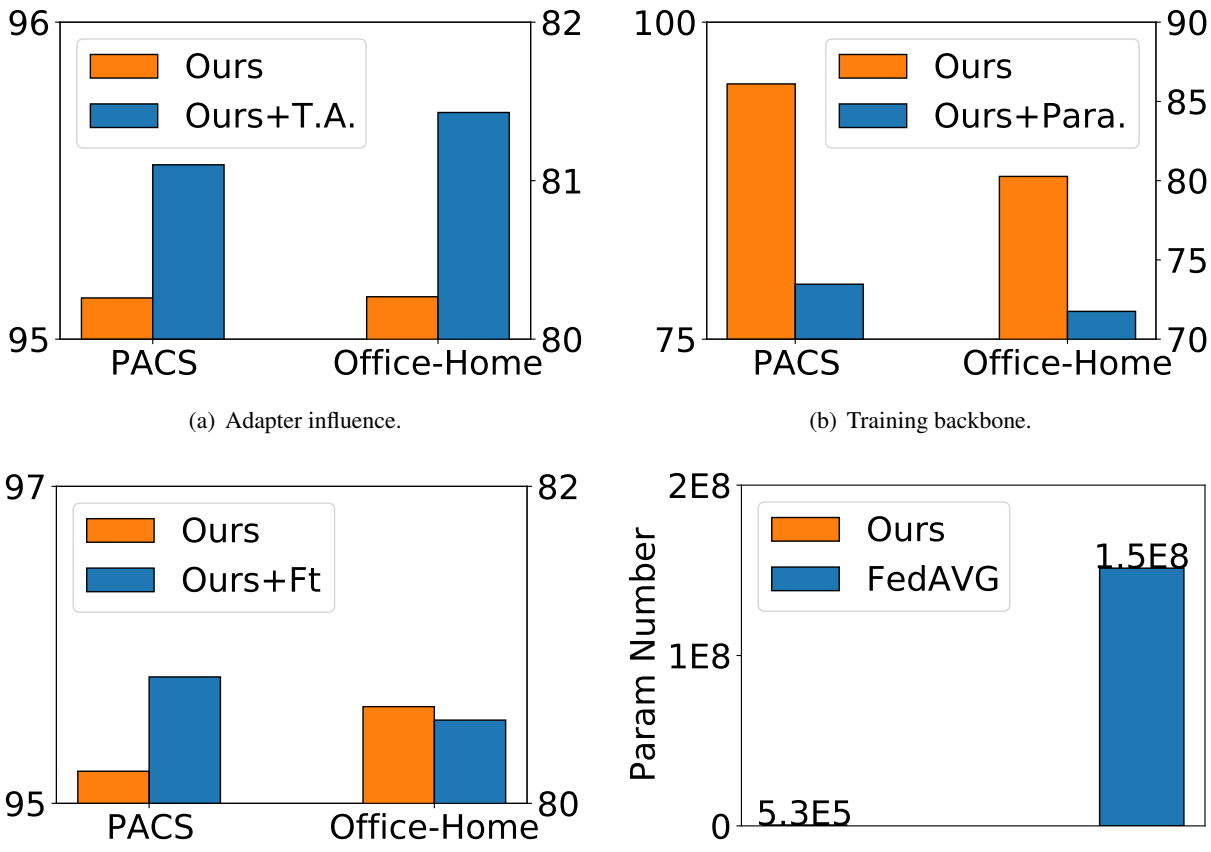
Comprehensive Ability Finally, taking into account the performance of both personalization and generalization, we provide an overall performance in Table 4. Without a doubt, our method achieves the best overall performance with significant improvements (about 9% for PACS and 8% for Office-Home). Compared to methods based on AlexNet, corresponding methods based on CLIP perform better.

More results on VLCS Due to space limitations, we only report comprehensive ability on VLCS. As shown in Table 5, our method still achieves the best performance with improvements of over 10%. Moreover, our method achieves the best in each task. The results prove the superiority of our method again.

4.4 Analysis

Can more adapters bring better performance? In our method, we only add one adapter to the image encoder. We can add another adapter to the text encoder. As shown in Figure 3(a), adding more adapters brings slight improvements. However, the improvements are so small that we need to assess whether it is necessary to do so since more adapters regularly mean more computational costs and more communication costs.

Can more trainable parameters bring better performance? If we train both adapters and the backbones, the results could be worse. Since CLIP models have a wealth of good information, it is not suitable to change



(a) Adapter influence.

(b) Training backbone.

(c) Finetune influence.

(d) Parameter counts.

Figure 3: Analysis on PACS.

parameters with only a few data for a specific task. Changes in CLIP with few data can destroy the feature extraction capabilities. As shown in Figure 3(b), we train more parameters but achieve worse performance.

Will finetuning bring better personalization? According to [54], finetuning can be a useful technique for better personalization. We also add experiments with finetune. As shown in Figure 3(c), finetune has no advance in personalization, which demonstrates that our method can be remarkable and robust when meeting non-iid.

Resource Cost Comparison The number of trainable parameters represents how many resources we need to cost in federated learning. As shown in Figure 3(d), our method merely has $5.3E5$ parameters while FedAVG with CLIP requires $1.5E8$ trainable parameters. Common methods via training whole networks have 283 times as many parameters as ours, which illustrates that our method is fast and resource-efficient.

5 Conclusion and Future Work

In this article, we propose FedCLIP, a fast generalization and personalization learning method for CLIP in federated learning. FedCLIP designs an attention based adapter to replace updating the whole model. Therefore, FedCLIP makes the most use of prior knowledge and saves computational costs and communication costs. Comprehensive experiments have demonstrated the superiority of FedCLIP. In the future, we plan to embed

FedCLIP into more architectures and design more flexible adapters for different tasks. We also plan to apply FedCLIP for heterogeneous architectures and more realistic applications.

References

[1] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. arXiv preprint arXiv:1803.01164, 2018.

[2] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz. Invariant risk minimization. stat, 1050:27, 2020.

[3] S. Banabilah, M. Aloqaily, E. Alsayed, N. Malik, and Y. Jararweh. Federated learning review: Fundamentals, enabling technologies, and future applications. Information processing & management, 59(6):103061, 2022.

[4] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258, 2021.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[6] D. Caldarola, B. Caputo, and M. Ciccone. Improving generalization in federated learning by seeking flat minima. In Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIII, pages 654–672. Springer, 2022.

[7] H.-Y. Chen and W.-L. Chao. On bridging generic and personalized federated learning for image classification. In International Conference on Learning Representations, 2022.

[8] Y. Chen, W. Lu, X. Qin, J. Wang, and X. Xie. Metafed: Federated learning among federations with cyclic knowledge distillation for personalized healthcare. arXiv preprint arXiv:2206.08516, 2022.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations, 2021.

[10] C. Fang, Y. Xu, and D. N. Rockmore. Unbiased metric learning: On the utilization of multiple datasets and web images for softening bias. In Proceedings of the IEEE International Conference on Computer Vision, pages 1657–1664, 2013.

[11] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In International Conference on Learning Representations, 2021.

[12] L. Gao, H. Fu, L. Li, Y. Chen, M. Xu, and C.-Z. Xu. Feddc: Federated learning with non-iid data via local drift decoupling and correction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10112–10121, 2022.

[13] M. W. Gardner and S. Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric environment, 32(14-15):2627–2636, 1998.

[14] T. Guo, S. Guo, J. Wang, and W. Xu. Promptfl: Let federated participants cooperatively learn prompts instead of models–federated learning in age of foundation model. arXiv preprint arXiv:2208.11625, 2022.

[15] S. Gupta, K. Ahuja, M. Havaei, N. Chatterjee, and Y. Bengio. Fl games: A federated learning framework for distribution shifts. In Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022), 2022.

[16] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, et al. A survey on vision transformer. IEEE transactions on pattern analysis and machine intelligence, 45(1):87–110, 2022.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[18] W. Hou, H. Zhu, Y. Wang, J. Wang, T. Qin, R. Xu, and T. Shinozaki. Exploiting adapters for cross-lingual low-resource speech recognition. IEEE ACM Trans. Audio Speech Lang. Process., 30:317–329, 2022.

[19] Z. Huang, H. Wang, E. P. Xing, and D. Huang. Self-challenging improves cross-domain generalization. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, pages 124–140. Springer, 2020.

[20] N. Inkster. China's cyber power. Routledge, 2018.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NeurIPS, volume 25, pages 1097–1105, 2012.

[22] S. Lee, H. Park, D. U. Kim, J. Kim, M. Boboev, and S. Baek. Image-free domain generalization via clip for 3d hand pose estimation. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 2934–2944, 2023.

[23] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales. Deeper, broader and artier domain generalization. In Proceedings of the IEEE international conference on computer vision, pages 5542–5550, 2017.

[24] L. Li, Y. Fan, M. Tse, and K.-Y. Lin. A review of applications in federated learning. Computers & Industrial Engineering, 149:106854, 2020.

[25] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. Proceedings of Machine Learning and Systems, 2:429–450, 2020.

[26] X. Li, M. JIANG, X. Zhang, M. Kamp, and Q. Dou. Fedbn: Federated learning on non-iid features via local batch normalization. In International Conference on Learning Representations, 2021.

[27] J. Liu, J. Huang, Y. Zhou, X. Li, S. Ji, H. Xiong, and D. Dou. From distributed machine learning to federated learning: A survey. Knowledge and Information Systems, 64(4):885–917, 2022.

[28] W. Liu, Y. Wen, B. Raj, R. Singh, and A. Weller. Sphereface revived: Unifying hyperspherical face recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(2):2458–2474, 2022.

[29] W. Lu, Y. Chen, J. Wang, and X. Qin. Cross-domain activity recognition via substructural optimal transport. Neurocomputing, 454:65–75, 2021.

[30] W. Lu, J. Wang, and Y. Chen. Local and global alignments for generalizable sensor-based human activity recognition. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022.

[31] W. Lu, J. Wang, Y. Chen, S. Pan, C. Hu, and X. Qin. Semantic-discriminative mixup for generalizable sensor-based cross-domain activity recognition. IMWUT, 2022.

[32] W. Lu, J. Wang, Y. Chen, X. Qin, R. Xu, D. Dimitriadis, and T. Qin. Personalized federated learning with adaptive batchnorm for healthcare. IEEE Transactions on Big Data, 2022.

[33] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics, pages 1273–1282. PMLR, 2017.

[34] N. Paluru, A. Dayal, H. B. Jenssen, T. Sakinis, L. R. Cenkeramaddi, J. Prakash, and P. K. Yalavarthy. Anam-net: Anamorphic depth embedding-based lightweight cnn for segmentation of anomalies in covid-19 chest ct images. IEEE Transactions on Neural Networks and Learning Systems, 32(3):932–946, 2021.

[35] Z. Qu, X. Li, R. Duan, Y. Liu, B. Tang, and Z. Lu. Generalized federated learning via sharpness aware minimization. In International Conference on Machine Learning, pages 18250–18280. PMLR, 2022.

[36] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In International conference on machine learning, pages 8748–8763. PMLR, 2021.

[37] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[38] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In International Conference on Machine Learning, pages 8821–8831. PMLR, 2021.

[39] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein, et al. The future of digital health with federated learning. NPJ digital medicine, 3(1):1–7, 2020.

[40] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámara. Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges. Information Fusion, 90:148–173, 2023.

[41] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning. arXiv, 2019.

[42] I. H. Sarker. Machine learning: Algorithms, real-world applications and research directions. SN computer science, 2(3):160, 2021.

[43] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek. Robust and communication-efficient federated learning from non-iid data. IEEE transactions on neural networks and learning systems, 31(9):3400–3413, 2019.

[44] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune bert for text classification? In Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18, pages 194–206. Springer, 2019.

[45] I. Tenison, S. A. Sreeramadas, V. Mugunthan, E. Oyallon, E. Belilovsky, and I. Rish. Gradient masked averaging for federated learning. arXiv preprint arXiv:2201.11986, 2022.

[46] I. Tenney, D. Das, and E. Pavlick. Bert rediscovers the classical nlp pipeline. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4593–4601, 2019.

[47] E. A. van Dis, J. Bollen, W. Zuidema, R. van Rooij, and C. L. Bockting. Chatgpt: five priorities for research. Nature, 614(7947):224–226, 2023.

[48] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan. Deep hashing network for unsupervised domain adaptation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5018–5027, 2017.

[49] P. Voigt and A. Von dem Bussche. The eu general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing, 10:3152676, 2017.

[50] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu. Deep learning for sensor-based activity recognition: A survey. Pattern Recognition Letters, 119:3–11, 2019.

[51] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng, and P. Yu. Generalizing to unseen domains: A survey on domain generalization. IEEE Transactions on Knowledge and Data Engineering, 2022.

[52] S. Warnat-Herresthal, H. Schultze, K. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Händler, P. Pickkers, N. A. Aziz, et al. Swarm learning for decentralized and confidential clinical machine learning. Nature, 594(7862):265–270, 2021.

[53] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST), 10(2):1–19, 2019.

[54] T. Yu, E. Bagdasaryan, and V. Shmatikov. Salvaging federated learning by local adaptation. arXiv preprint arXiv:2002.04758, 2020.

[55] H. Yuan, W. R. Morningstar, L. Ning, and K. Singhal. What do we mean by generalization in federated learning? In The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net, 2022.

[56] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In Proceedings of the IEEE/CVF international conference on computer vision, pages 558–567, 2021.

[57] Z. Zhang and M. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. Advances in neural information processing systems, 31, 2018.

Federated Truth Discovery for Mobile Crowdsensing with Privacy-Preserving Trustworthiness Assessment

Leye Wang[1,2], Guanghong Fan[1,2], Xiao Han[3,*]
[1]Key Lab of High Confidence Software Technologies (Peking University), Ministry of Education, China
[2]School of Computer Science, Peking University, China
[3]School of Information Management and Engineering, Shanghai University of Finance and Economics, China
leyewang@pku.edu.cn, fgh@stu.pku.edu.cn, xiaohan@mail.shufe.edu.cn

Abstract

With the prevalence of smart mobile devices empowered by considerable sensing capabilities, crowdsensing has become one promising way to sense urban phenomena (e.g., traffic and environment) at a large scale. In crowdsensing, a fundamental issue is discovering the truth from participants' noisy sensed data. Traditionally, participants need to upload their raw sensed data with locations for truth discovery, but this may leak participants' private information such as home and work locations. In this paper, we propose a federated truth discovery method that can learn the truth without collecting participants' sensed data and locations. Our method ensures that the obtained truth quality has no performance loss compared to the original truth discovery method if all the participants keep online; even if some participants lose connections unpredictably, our method can still learn the truth based on rest participants' data. Meanwhile, as participants' sensed data are unknown to the server, it is hard for the crowdsensing organizer to justify each participant's sensing trustworthiness. This brings difficulties to crowdsensing management such as participant recruitment and incentive allocation. We further propose a federated ranking mechanism to generate a leader-board for participants' trustworthiness, which can also tolerate participants' connection loss. Both theoretical analysis and real-data empirical evaluations have been done to verify the effectiveness of FedTruthFinder.

1 Introduction

With the popularity of smart mobile devices, such as smartphones, pads, and vehicles, crowdsensing has become one promising paradigm for sensing urban dynamics [3]. A typical crowdsensing process first recruits participants and then asks them to upload the data of interest to the central server. Afterward, the server aggregates the data from participants toward a synthetic sensed result [39, 11]. While each participant's sensed data may include noise, an important issue is how to ensure the accuracy of the aggregated sensed result, often called *truth discovery* [26, 12, 24].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Corresponding author

Many research studies have been devoted to the truth discovery for crowdsensing applications. Generally, most relevant studies model participant trustworthiness and sensed data confidence iteratively to obtain the final aggregated sensed result [36]. The basic idea is that a high-credit participant's sensed data should be assigned with high confidence; a user whose sensed data are more confident should be more trustworthy. While this has been verified to be effective, the iterative computation process needs a central server to collect every participant's raw sensed data, which may bring privacy threats to crowdsensing participants. For example, crowdsensing usually asks participants to do sensing tasks at specific locations, and thus most sensed data are associated with detailed location information [39]. The traditional truth discovery process would inevitably leak crowdsensing participants' visited locations during sensing periods, which may be exploited by malicious third parties to conduct serious location privacy breaches such as physical stalking [19]. With the user privacy and personal data regularization (e.g., General Data Protection Regulation[1]) becoming more and more important nowadays, we believe that a privacy-preserving truth discovery algorithm is urgently required for facilitating more extensive crowdsensing campaigns in practice.

Privacy-preserving truth discovery has been recently studied in crowdsensing. Existing studies are usually based on some hardly realized assumptions such as every participant being always online and/or non-colluding [13, 15, 14][2], or two non-colluding servers (or fog nodes) are required [23, 40, 38, 37]. These assumptions hinder the practical applications of the prior mechanisms. More importantly, almost all the prior studies do not discuss two fundamental issues in privacy-preserving truth discovery.

i) **Participants' Completed Tasks Protection.** Existing mechanisms focus on protecting participants' sensed data, but most assume that participants' completed tasks are known to the crowdsensing server [34, 13, 14, 40]. However, sometimes task information is more sensitive than sensed data. Suppose the tasks are air quality sensing at certain points of interest. If a participant's task completion information is disclosed (e.g., finish a sensing task at the Times Square NYC), then her location is revealed without the need to know her sensed air quality value.

ii) **Participants' Trustworthiness Assessment.** Trustworthiness assessment is a key part of crowdsensing for participant recruitment and incentive design [17], while privacy-preserving truth discovery needs to hide participants' trustworthiness scores for privacy protection. To address the dilemma, a privacy-preserving trustworthiness assessment method needs to be proposed.

In this paper, we aim to design a novel and practical privacy-preserving truth discovery mechanism for crowdsensing to overcome the pitfalls of prior studies. In particular, our design follows the federated learning (FL) paradigm [5, 35]. We thus call our method as *FedTruthFinder*. In general, FL requires user clients to do some local computation (e.g., learning the gradients for updating the parameters of the model) on their devices and then only upload the computation results instead of raw data. For a specific algorithm, the local computation and uploading process usually incorporates certain secure mechanisms (e.g., homomorphic encryption and secure multi-party computation) to ensure that no user privacy is leaked theoretically [2, 10]. Based on FL, we aim to consider the following specific issues in the design of FedTruthFinder.

No Accuracy Loss: We expect that FedTruthFinder will not hurt the accuracy of the aggregated sensed results compared to the centralized truth discovery. Without the loss of accuracy, crowdsensing organizers would be likely to adopt the method in practice.

No Third Party: Crowdsensing involves a central server and a set of participants' clients. To make FedTruthFinder easy to deploy, we do not want to introduce any more third parties which are usually hard to find in reality [1, 27].

Robustness against Unpredictable Connection Loss: While crowdsensing participants travel around the whole city, their device connection with the central server is not always stable. Hence, it is necessary to make FedTruthFinder effective when some participants lose connections suddenly.

[1] `https://gdpr.eu/eu-gdpr-personal-data`

[2] Some participants' relations can be very close such as family members, and thus it is non-reasonable to assume that they will not collude with each other.

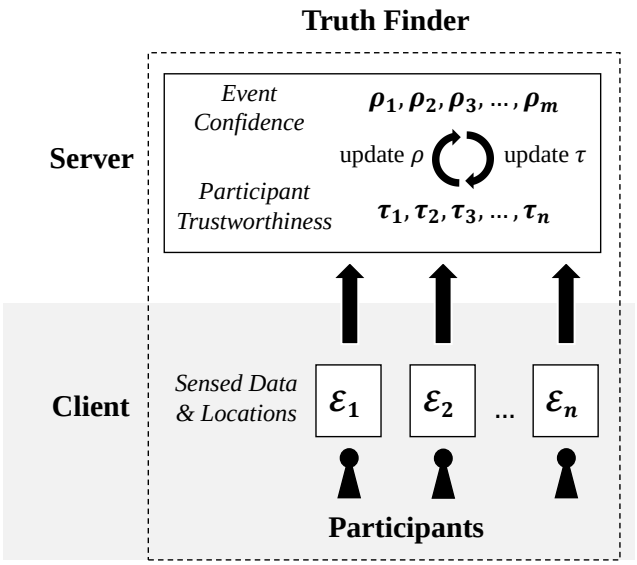


Figure 1: Overview of Iterative Truth Discovery

With the above issues in consideration, this paper makes the following contributions:

(1) To the best of our knowledge, this paper is the first study that addresses the problem of privacy-preserving crowdsensing truth discovery with (i) *participants' completed task protection*, and (ii) *participants' trustworthiness assessment*.

(2) We propose *FedTruthFinder*, a novel privacy-preserving truth discovery mechanism following the federated learning paradigm [35]. FedTruthFinder does not need any third party and can tolerate participants' unpredictable connection loss. The two key components of FedTruthFinder are (i) *federated confidence computation* to learn the probability of a sensed event, and (ii) *federated trustworthiness ranking* to assess participants' sensed data quality.

(3) We have conducted both theoretical analysis and empirical evaluations for FedTruthFinder. In particular, FedTruthFinder can reduce the system failure probability significantly compared to state-of-the-art privacy-preserving truth discovery approaches [1, 34], and achieve good detection accuracy.

2 Preliminary: Truth Discovery

Truth discovery algorithms usually follow an iterative method to calibrate user trustworthiness and data confidence alternatively until convergence [36]. Figure 1 shows the framework of iterative truth discovery methods. In this paper, for clarity, we assume that sensed data is a binary spatial event. That is, for a specific location, the sensed data can be 1 or 0. Our method can be easily extended to multi-class and continuous-value events (see Appendix).

As shown in Figure 1, first, participants upload all of their sensed data and locations $\mathcal{E}_i$ to the central server. The central server would assign an initial trustworthiness score τ_i to each participant u_i (e.g., 0.9 by assuming that 90% of the sensed data are accurate). Then, for each sensed event e_j, the truth discovery algorithm will calculate its confidence ρ_j (i.e., the probability of $e_j = 1$) by considering the users who have sensed e_j as:

$$\rho_j = F_\rho(\mathcal{U}_{j,1}, \mathcal{U}_{j,0}) \tag{10}$$

where $\mathcal{U}_{j,k}$ is the users who have sensed the event e_j with the reported data k; F_ρ is an event confidence calculation function which we will elaborate on later.

With ρ_j for each event e_j, we can then update the trustworthiness score τ_i of each participant u_i by:

$$\tau_i = F_\tau(\mathcal{E}_{i,1}, \mathcal{E}_{i,0}) \tag{11}$$

where $\mathcal{E}_{i,k}$ is the users' sensed event set with the reported data k; F_τ is a user trustworthiness calculation function which we will elaborate on later.

Once τ_i is updated for each user u_i, we can continue updating ρ_j for each event e_j according to Eq. 10, and so on, leading to an alternative updating process for both τ_i and ρ_j. This process can be terminated after a fixed number of iterations or until convergence. Next, we elaborate on the common choices of F_ρ and F_τ in literature.

Sum Function

An intuitive selection of the updating functions of F_ρ and F_τ is the weighted sum:

$$\rho_j = F_\rho(\mathcal{U}_{j,1}, \mathcal{U}_{j,0}) = \frac{\sum_{u_i \in \mathcal{U}_{j,1}} \tau_i}{\sum_{u_i \in \mathcal{U}_{j,1}} \tau_i + \sum_{u_k \in \mathcal{U}_{j,0}} \tau_k} \tag{12}$$

$$\tau_i = F_\tau(\mathcal{E}_{i,1}, \mathcal{E}_{i,0}) = \frac{\sum_{e_j \in \mathcal{E}_{i,1}} \rho_j + \sum_{e_k \in \mathcal{E}_{i,0}} 1 - \rho_k}{|\mathcal{E}_{i,1}| + |\mathcal{E}_{i,0}|} \tag{13}$$

Logistic Function

Another widely used updating function is the Logistic function [36]. Its basic idea is seeing every user independently, so that the probability of event happening, i.e., $e_j = 1$, can be formulated as:

$$\rho_j = 1 - \prod_{u_i \in \mathcal{U}_{j,1}} (1 - \tau_i) \tag{14}$$

As $1 - \tau_i$ may often be small and multiplying many of them may lead to underflow, prior studies proposed to use the logarithm to define a log-trustworthiness score of u_i as [36]:

$$\tau_i^* = -\ln(1 - \tau_i) \tag{15}$$

Similarly, a log-confidence score of event e_j is defined as:

$$\rho_j^* = -\ln(1 - \rho_i) \tag{16}$$

Then, we can infer

$$\rho_j^* = \sum_{u_i \in \mathcal{U}_{j,1}} \tau_i^* \tag{17}$$

The above equation does not consider the users' trustworthiness who report $e_j = 0$, and thus we refine it:

$$\rho_j^* = \sum_{u_i \in \mathcal{U}_{j,1}} \tau_i^* - \sum_{u_k \in \mathcal{U}_{j,0}} \tau_k^* \tag{18}$$

Finally, a logistic function is used to calculate the final confidence ρ_j of event e_j [36]:

$$\rho_j = F_\rho(\mathcal{U}_{j,1}, \mathcal{U}_{j,0}) = (1 + e^{-\rho_j^*})^{-1} \tag{19}$$

τ_i is updated same as Eq. 13.

3 Federated Truth Discovery: Overview and Key Issues

3.1 Overall Design

Figure 2 overviews the workflow of our method *FedTruthFinder*. The general design principle follows the federated learning paradigm [35], which requires the user clients to conduct local computations of their raw

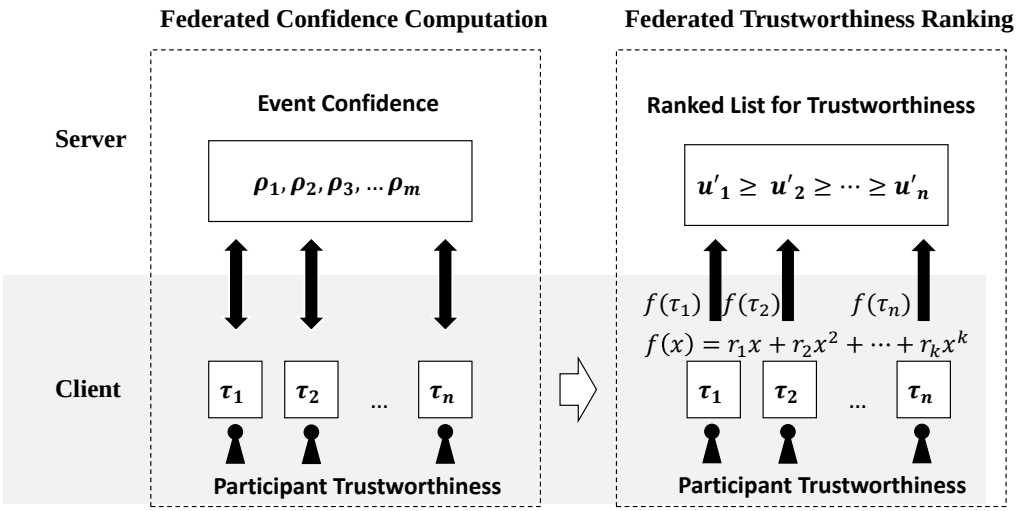


Figure 2: Overview of FedTruthFinder

data and then upload processed data that do not reveal the user's privacy to the server. With these uploaded privacy-preserving data, the server can still find the aggregate truth of the sensed events, the same as the server receiving the raw sensed data and locations from participants.

By analyzing the original truth discovery algorithm in the previous section, we note that there exist two alternative computation processes: (i) *ρ-computation*: updating the confidence ρ_j for each event e_j, and (ii) *τ-computation*: updating the trustworthiness τ_i for each participant u_i. In the two computation processes, τ-computation (e.g., Eq. 13) can be naturally offloaded to each participant u_i's device, as long as the server sends all the current $\rho_j, \forall e_j$ to the participants. However, ρ-computation needs to know each user's sensed data (and locations) and then do aggregation (e.g., sum). This needs a dedicated design to enable the privacy-preserving truth discovery, which will be illustrated in Sec. 4.

Besides, the trustworthiness of each participant's sensed data (i.e., τ_i) is a key metric in crowdsensing organization for participant recruitment and incentive allocation. Hence, we also design a federated privacy-preserving mechanism to rank participants' trustworthiness. Particularly, instead of transferring raw τ_i to the server, we leverage certain security mechanisms to upload $f(\tau_i)$ to the server, while $f(\tau_i)$ keeps the same ranking orders as τ_i. Particularly, in FedTruthFinder, $f(\tau_i) = r_1\tau_i + r_2\tau_i^2 + ... + r_k\tau_i^k$, where $r_i > 0$. In this regard, even though the server cannot know the specific τ_i of each participant u_i, the ranked list of participants according to the trustworthiness can still be learned with $f(\tau_i)$. Specifically, during the whole computation process, the server cannot know r_i, and each participant will also not know all the r_i, so that none of the server or participant can infer other participants' private τ_i. How to compute $f(\tau_i)$ securely will be introduced in Sec. 5.

3.2 Key Issues

Issue 1. Privacy-Preserving ρ-computation: *Suppose there are a set of crowdsensing participants $\mathcal{U}$ and a set of spatial events to sense $\mathcal{E}$, each participant $u_i(\in \mathcal{U})$ with sensed events $\mathcal{E}_{i,1}(\subseteq \mathcal{E})$ and $\mathcal{E}_{i,0}(\subseteq \mathcal{E})$ corresponding to the sensed value being 1 and 0, respectively. How to calculate confidence ρ_j for each event $e_j(\in \mathcal{E})$ while every participant u_i will not leak $\mathcal{E}_{i,1}$, $\mathcal{E}_{i,0}$, and $\mathcal{E}_{i,1} \cup \mathcal{E}_{i,0}$ to the server and other participants?*

Some factors need to be carefully considered:

(1) **Computation**: In ρ-computation, the value to share is a complicated equation instead of a single value, and the equation may even be varied depending on the truth discovery algorithm implementation (e.g., Eq. 12 or

Eq. 19).

(2) **Network Connection**: In crowdsensing, the network connections of mobile participants may not be always stable. Hence, our mechanism should tolerate the scenario when a few participants lose connections.

(3) **No Leakage of Task Completion**: For protecting participants' privacy, not only the sensed data but also the completed tasks (i.e., $\mathcal{E}_{i,1} \cup \mathcal{E}_{i,0}$) should not be disclosed.

Issue 2. Secure Trustworthiness Ranking: *Suppose there are a set of crowdsensing participants $\mathcal{U}$ and each participant $u_i (\in \mathcal{U})$ has a private trustworthiness score τ_i. How to rank participants according to τ_i while every participant u_i will not leak τ_i to the server and other participants?*

Addressing this issue also needs to consider the unstable network connections of participant clients as aforementioned.

Remark on security definition: In this work, we assume that the crowdsensing server and participants are *semi-honest (honest-but-curious)*: they will follow our designed protocol and not maliciously modify the inputs or outputs; however, the server and the participants will try their best to infer others' data from the data that they have received. Besides, our mechanism can defend against *collusion attacks* to a certain extent (i.e., some participants may collude with each other), which we will elaborate on later.

4 Federated Truth Computation

We first introduce a basic scheme for ρ-computation in a federated manner assuming no connection loss. Then, we improve the scheme to be against the participants' unpredictable connection loss.

4.1 Basic Scheme of ρ-Computation with SSS

In this section, we propose our basic scheme for the ρ-computation problem with the federated learning paradigm leveraging secret sharing. Given a spatial crowdsensing event e_j, we first consider ρ-computation with the sum function, i.e., Eq. 12.

4.1.1 ρ-computation with the sum function.

Our process includes three steps as follows:

Step 1 (Share Dispatching). Each participant u_i dispatches d_{ij} and s_{ij} with secret sharing to all the n participants ($\mathcal{U} = \{u_1 \cdots u_n\}$), where

$$d_{ij} = \begin{cases} \tau_i & e_j \in \mathcal{E}_{i,1} \\ 0 & e_j \in \mathcal{E} \setminus \mathcal{E}_{i,1} \end{cases} \tag{20}$$

$$s_{ij} = \begin{cases} \tau_i & e_j \in \mathcal{E}_{i,1} \cup \mathcal{E}_{i,0} \\ 0 & e_j \in \mathcal{E} \setminus (\mathcal{E}_{i,1} \cup \mathcal{E}_{i,0}) \end{cases} \tag{21}$$

Specifically, d_{ij} is divided into n shares

$$\{d_{ij}^1, \cdots, d_{ij}^n\}$$

where $d_{ij}^1, \cdots, d_{ij}^{n-1}$ are random numbers and

$$d_{ij}^n = d_{ij} - \sum_{k=1}^{n-1} d_{ij}^k$$

Hence, $\sum_{k=1}^{n} d_{ij}^k = d_{ij}$. Then, u_i sends d_{ij}^k to u_k. Similarly, s_{ij} is split to n shares

$$\{s_{ij}^1, \cdots, s_{ij}^n = s_{ij} - \sum_{k=1}^{n-1} s_{ij}^k\}$$

and u_k receives s_{ij}^k from u_i.

Step 2 (Client Summation). For each event e_j, an participant u_k uploads $\hat{d}_j^k = \sum_{i=1}^{n} d_{ij}^k$ and $\hat{s}_j^k = \sum_{i=1}^{n} s_{ij}^k$ to the server.

Step 3 (Server Aggregation). After receiving $\hat{d}_j^k$ and $\hat{s}_j^k$ from $\forall u_k \in \mathcal{U}$, the server can add them together:

$$d_j = \sum_{k=1}^{n} \hat{d}_j^k = \sum_{k=1}^{n}\sum_{i=1}^{n} d_{ij}^k = \sum_{i=1}^{n}\sum_{k=1}^{n} d_{ij}^k = \sum_{i=1}^{n} d_{ij} \tag{22}$$

$$s_j = \sum_{k=1}^{n} \hat{s}_j^k = \sum_{k=1}^{n}\sum_{i=1}^{n} s_{ij}^k = \sum_{i=1}^{n}\sum_{k=1}^{n} s_{ij}^k = \sum_{i=1}^{n} s_{ij} \tag{23}$$

Then, ρ_j is computed as:

$$\rho_j = \frac{d_j}{s_j} \tag{24}$$

4.1.2 ρ-computation with the logistic function.

The computation process with the logistic function is not much different from the one with the summation function. Actually, for the event e_j, we only need to modify d_{ij} to:

$$d_{ij} = \begin{cases} -\ln(1-\tau_i) & e_j \in \mathcal{E}_{i,1} \\ \ln(1-\tau_i) & e_j \in \mathcal{E}_{i,0} \\ 0 & e_j \in \mathcal{E} \setminus (\mathcal{E}_{i,1} \cup \mathcal{E}_{i,0}) \end{cases} \tag{25}$$

Besides, we do not need s_{ij}, so every participant u_k only receives d_{ij}^k from other $u_i \in \mathcal{U}$. Finally, in Step 3, we can compute ρ_j as:

$$\rho_j = (1 + e^{-d_j})^{-1} \tag{26}$$

Remark on our novelty. In our ρ-computation for an event e_j, the participant u_i who has not sensed e_j also needs to upload data to the server, e.g., $d_{ij} = s_{ij} = 0$ for the sum function. As d_{ij} and s_{ij} are sent by secret shares, the other participants and the server would not know whether u_i senses e_j or not. In comparison, prior studies usually assume that participants send only the data of their sensed events, which may disclose user privacy from event information (e.g., event locations) [13, 15, 14, 40, 41, 38].

4.2 Connection Robustness Improvement

The basic scheme can learn ρ_j in an ideal environment when all the participants are always online. In practice, participants move around and their network connections are often sporadic. This inspires us to make three improvements to the basic scheme design.

4.2.1 Bias-avoidance adaptive truth updating.

While participants may drop during the iterative truth discovery process due to bad connections, the original event confidence updating function would be ineffective. Specifically, if u_i loses the connection at the k^{th} iteration, u_i's data would not be considered in the ρ-computation (e.g., Eq. 12) from then on. This leads to unreliable ρ_j as the finally alive participants dominate the results. To address this pitfall, we propose an adaptive updating function for k^{th} iteration's $\rho_{j,k}$ as,

$$\rho_{j,k} = w_k F_\rho + (1 - w_k)\rho_{j,k-1} \tag{27}$$

where F_ρ is an original event confidence updating function (e.g., sum and logistic). With Eq. 27, the data contribution of the participants who drop at the k^{th} iteration can still be kept (by $\rho_{j,k-1}$) to avoid the truth bias toward alive participants. w_k can be set as,

$$w_k = (\frac{|\mathcal{U}_{alive,k}|}{|\mathcal{U}|})^\alpha \tag{28}$$

where $\mathcal{U}_{alive,k}$ is the alive participants at the k^{th} iteration. When alive participants decrease with more iterations, w_k becomes smaller, reflecting that fewer participants should occupy lower weights. From our experiments, we find that $\alpha = 3$ is a proper setting.

4.2.2 Server-coordinated communication structure.

In Sec. 4.1, we assume that one participant u_i can establish a secure communication channel with every other participant u_k so as to transfer the secret share d_{ij}^k and s_{ij}^k. Hence, each participant needs to establish $n-1$ channels with others. Considering the sporadic property of the mobile connections, this may not be easy for a participant to keep so many channels stable in practice. To alleviate this issue, we convert the peer-to-peer communication structure to a server-coordinated one. In the server-coordinated structure, every participant first transmits all the data to the server, and then the server dispatches the desired data to the corresponding participant. In this way, each participant needs to establish *only one* secure channel to the server.

To ensure that the transmitted data will not be directly observed by the server, we leverage a public-key encryption system to encrypt the data before the transmission. In particular, each participant u_i first generates a pair of keys, the public key pk_i and the private key sk_i. The public key pk_i is sent to all the other participants (e.g., through the server) at the beginning of the crowdsensing campaign. For details, readers can refer to **(author?)** [1].

Then, for computing ρ_j, each participant u_i first transmits $E_i = \{\textit{Encrypt}(d_{ij}^k, pk_k)|k = 1 \cdots n\}$ to the server.[3] After receiving n participants' E_i, the server re-organizes the received data and sends $\hat{E}_k = \{\textit{Encrypt}(d_{ij}^k, pk_k)|i = 1 \cdots n\}$ to each participant u_k. Afterward, u_k decrypts the received data with her private key, obtains $\{d_{ij}^k|i = 1 \cdots n\}$, and then uploads $\hat{d}_j^k = \sum_i d_{ij}^k$ to the server. The server can then recover $d_j = \sum_i d_{ij}$ from participants' uploaded $\hat{d}_j^k$ and computes ρ_j accordingly.

4.2.3 (t, n)-Shamir secret sharing (SSS)

While the server-coordinated communication structure reduces the burden of secure channel establishing for mobile participants. It may still fail if a user loses the connection during the campaign and cannot link back. For example, suppose a participant u_i has sent E_i to the server and then quit the crowdsensing campaign (e.g., u_i's mobile device runs out of battery). Then, in Step 3, the server will not be able to receive $\hat{d}_j^i$ from u_i, and thus cannot recover d_j.

[3] If s_{ij} is needed (e.g., for the sum function), it can be encoded in E_i same as d_{ij}.

To address this pitfall, in practical deployment, we can leverage the threshold secret sharing method proposed by Shamir [21], namely (t, n)-*Shamir secret sharing (SSS)*. With (t, n)-SSS, the server only needs to receive t ($t \leq n$) participants' $\hat{d}_j^i$ for recovering d_j. In particular, to leverage (t, n)-SSS to dispatch d_{ij}, we first create a $(t-1)$-polynomial:

$$D_{ij}(x) = d_{ij} + a_{ij1}x + a_{ij2}x^2 + \cdots + a_{ijt-1}x^{t-1} \tag{29}$$

where $a_{ij1}, \cdots, a_{ijt-1}$ are random numbers selected by u_i. Then, u_i dispatches $D_{ij}(k)$ to u_k. If we obtain more than t participants' $D_{ij}(k)$, according to linear algebra, we can infer d_{ij}.

Similar to Step 2 of our basic scheme, $\hat{d}_j^k = \sum_i D_{ij}(k)$ is uploaded to the server by each u_k. Then, in Step 3, after receiving more than t participants' $\hat{d}_j^k$, the server will be able to infer $d_j = \sum_i d_{ij}$ according to the *additive homomorphism property* of SSS [21].

Remark on our novelty. In the truth discovery part, the key advantage of our mechanism beyond literature is its robustness against connection loss. Preliminary privacy-preserving truth discovery papers rarely consider the connection loss issue [15]. Some work tries to deal with drop-out users by letting alive participants send extra information [38, 31]; however, when the connection condition is so bad that certain alive participants again lose connections during the extra information communication, this process would be uncontrolled and time-consuming. Recent work also adopts SSS to enhance connection robustness [34]. The basic idea is using the double-masking secure aggregation algorithms proposed by **(author?)** [1], and every participant needs *two* connections to do event confidence computation for one iteration. In comparison, our mechanism only needs every participant to connect *once* for one iteration of computation. Our numerical experiments (Sec. 6.1) will show that this connection reduction can lead to a significantly difference in the algorithm success probability (e.g., increasing the success probability from 1% to 99% under certain conditions).

4.3 Theoretical Analysis

4.3.1 Correctness

The process to calculate ρ_i in FedTruthFinder follows the original algorithm shown in Sec. 2. Hence, we can obtain the same aggregate truth results as the original algorithm, as long as the SSS scheme is valid. In this regard, the correctness of our algorithm is theoretically guaranteed.

4.3.2 Robustness to Connection Loss & Security

Setting t to a small value allows our mechanism to tolerate more users dropping the campaign due to connection losses. Meanwhile, a small t reduces the security level of our mechanism — if t participants collude with each other, they can recover the other participants' sensed data and locations, leading to privacy leakage.

Theorem 4.1. If there are $\leq n - t$ participants losing the connection in one iteration of ρ-computation, the server can learn the event confidence ρ_j.

Theorem 4.2. If $t'(< t)$ semi-honest users collude with each other, they cannot infer any other users' secret information.

The two theorems hold based on the property of (t, n)-SSS.

4.3.3 Complexity

We analyze the algorithm from both communication and computation complexity perspectives. Particularly, since the participant clients are more sensitive to the communication and computation overhead, our current analysis focuses on the client side, while the server part analysis is similar.

Communication Complexity - $O(nn_e)$. For each participant client, she needs to transfer n share pieces of the secret to the other participants and receive the corresponding shares from every other participant, so the complexity is $O(n)$ for one event. Suppose there are n_e events, the total communication complexity is $O(nn_e)$.

Computation Complexity - $O(nn_e)$. Each client needs to do two local computation processes. The first process is to generate the random coefficients for (t, n)-SSS and calculate the secret shares sent to all the other participants (Step 1), which is $O(nn_e)$. The second process is to do local summation (Step 2), which is also $O(nn_e)$. Hence, the total computation complexity is $O(nn_e)$.

5 Federated Trustworthiness Rank

While FedTruthFinder learns the integrated event truth in a privacy-preserving manner, it brings a challenge in justifying participants' trustworthiness. For example, to incentivize the crowdsensing participants, it is a common strategy to pay the high-trustworthy participants (i.e., high-quality sensing results) with higher incentives. However, in FedTruthFinder, the sensing quality of each participant, i.e., the trustworthiness score τ_i is kept at each participant side and unknown to the server. Hence, how to assess participants' trustworthiness is required and challenging for FedTruthFinder.

In this section, we first illustrate a concrete case to describe that τ_i cannot be directly known to the server, otherwise the server may infer which event u_i has sensed. As τ_i cannot be known to the server, we then design a secure ranking algorithm to let the server know every participant u_i's ranking position of τ_i among all the participants without leaking τ_i. Based on the ranked positions, the crowdsensing organizer can enable certain trustworthiness-aware incentive mechanisms, e.g., rewarding high-position participants with bonus, which can incentivize participants to compete with each other to get more high-quality sensed data [20].

5.1 Privacy Leakage by Trustworthiness τ_i

Here, we illustrate an example to show the risk of revealing τ_i to the server for leaking participant u_i's privacy.

Without the loss of generality, we assume that u_1's $\tau_1 = 0.9$, and other u_i's $\tau_i < 0.9$ $(i \neq 1)$. Suppose that one event e_j's $\rho_j = 0.9$ after truth discovery, then we can easily infer that u_1 has sensed the event e_j and the sensed result is 1. This reveals the fact that u_1 has visited the location of e_j, leaking u_1's location privacy.

Hence, participants cannot directly upload their τ_i to the server for incentive allocation. Next, we design a privacy-preserving method to enable trustworthiness-aware incentive allocation.

5.2 Secure Trustworthiness Leader-board

While revealing τ_i may leak participants' private information, we propose a secure ranking algorithm to learn a leader-board regarding participants' trustworthiness for facilitating trustworthiness-aware incentive allocation.

Secure ranking algorithms have been studied for decades; however, prior studies cannot be directly applied in our scenario for two reasons. First, the communication overheads are usually high. Second, prior studies mostly assume that all the network connections are stable for all the parties, but this is unrealistic for crowdsensing.

Our secure ranking algorithm generally follows the design of **(author?)** [22]. However, the original design [22] cannot tolerate any participants to lose the network connections. We thus enhance it to ensure that the ranking algorithm can still work when certain participants lose connections. The major steps of our federated trustworthiness leader-board generation mechanism are:

Step 1. First, we categorize all the participants into $(2t + 1)$ groups, and thus each group includes $n/(2t + 1)$ participants. We denote $gid(u)$ to refer to the group ID of participant u.

Step 2. For each user u_i, she shares τ_i, τ_i^2, ... , τ_i^{2t+1} with $(t + 1, 2t + 1)$-SSS to all the user groups. Specifically, a user u_j will receive the share piece regarding $gid(u_j)$, denoted as $\tau_{i_1}(gid(u_j))$, $\tau_{i_2}(gid(u_j))$, ... $\tau_{i_{2t+1}}(gid(u_j))$ for τ_i, τ_i^2, ... , τ_i^{2t+1}, respectively.

Step 3. For each user group g_k, it generates a random number $r_k(>0)$ and shares r_k with $(t+1, 2t+1)$-SSS to all the user groups. That is, u_j will receive r_k's share regarding $gid(u_j)$, denoted as $r_k(gid(u_j))$.

Step 4. For each participant u_j, she calculates the following number with the $\tau_{i_k}(gid(u_j))$ received from u_i:

$$h_i(gid(u_j)) = \lambda(gid(u_j)) \sum_{k=1}^{2t+1} r_k(gid(u_j))\tau_{i_k}(gid(u_j)) \tag{30}$$

$$= \lambda(gid(u_j))\gamma(gid(u_j)) \tag{31}$$

where

$$\begin{pmatrix} 1 & 1 & 1^2 & ... & 1^{2t} \\ 1 & 2 & 2^2 & ... & 2^{2t} \\ ... & ... & ... & ... & ... \\ 1 & 2t+1 & (2t+1)^2 & ... & (2t+1)^{2t} \end{pmatrix}^{-1}$$

$$= \begin{pmatrix} \lambda(1) & \lambda(2) & \lambda(3) & ... & \lambda(2t+1) \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \end{pmatrix}$$

Step 5. For each user group, we randomly select one participant u_j to share $\{h_i(gid(u_j))|i \in [1,n]\}$ with $(t+1, n)$-SSS to all the n participants. Each user u_k's received shares from all the groups are denoted as $\{h_i(g,k)|i \in [1,n], g \in [1, 2t+1]\}$.

Step 6. For each participant u_k, she computes:

$$h'_i(k) = \sum_{g=1}^{2t+1} h_i(g,k), \quad \forall i \in [1,n] \tag{32}$$

Each u_k sends $\{h'_i(k)|i \in [1,n]\}$ to the server.

Step 7. After receiving at least $t+1$ participants' $\{h'_i(k)|i \in [1,n]\}$, the server can recover:

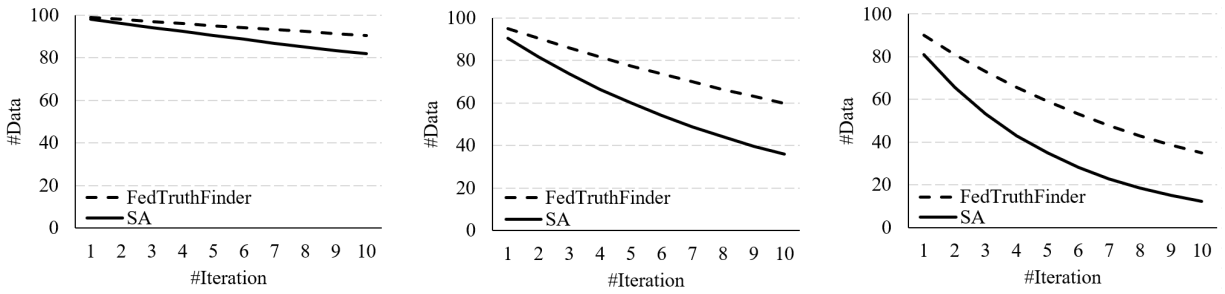
$$h_i = \sum_{k=1}^{2t+1} r_k \tau_i^k, \quad \forall i \in [1,n] \tag{33}$$

Step 8. The server ranks u_i according to h_i and the ranked list is the leader-board regarding trustworthiness τ_i.

Note that same as ρ-computation, we do not need to establish the peer-to-peer communication channels between every two participant clients and can use the crowdsensing server for coordination. To avoid redundancy, readers can refer to Sec. 4.2.2 for details.

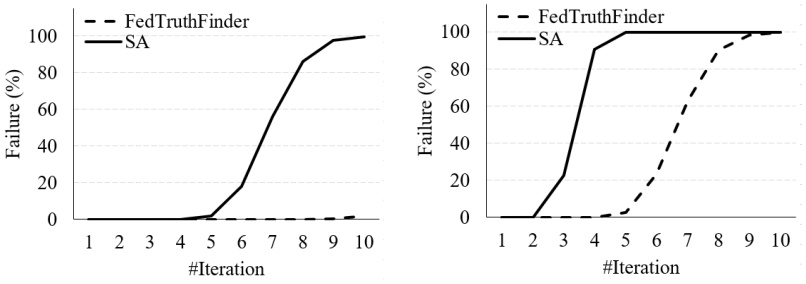
Remark on our novelty. The key improvement of our secure ranking algorithm compared to **(author?)** [22] is the enhanced robustness against participants' connection loss. In **(author?)** [22], every participant holds a r_i and we will randomly select $2t+1$ participants to share their r_i (Step 3) and h_i (Step 5). This process is easy to break if a selected online user (Step 3) loses the connection in Step 5. Our proposed algorithm first constructs user groups so that we only need at least one participant online in each group for both Step 3 and 5, reducing the failure possibility incurred by connection loss. It is worth noting that this algorithm can not only rank crowdsensing participants' trustworthiness, but also be applied to many other applications when privacy-preserving data ranking is needed under unstable network connections.

Remark on the ranked measurements. In the previous algorithm description, we suppose that τ_i needs to be ranked. In practice, crowdsensing organizers can use the same secure ranking mechanism to rank other key measurements of participants (e.g., the number of sensed events) to design better incentive mechanisms or participant recruitment strategies.



(a) $p_l = 0.01$ (b) $p_l = 0.05$ (c) $p_l = 0.1$

Figure 3: Number of data for each event's truth discovery by iterations.



(a) $p_l = 0.05$ (b) $p_l = 0.1$

Figure 4: Failure probability of truth discovery.

5.3 Theoretical Analysis

All the proofs are illustrated in Appendix.

5.3.1 Correctness

We first prove the correctness of our algorithm.

Lemma 5.1. $\sum_{k=1}^{2t+1} r_k(x)\tau_{i_k}(x)$ can be represented as:

$$h_i + a_{i1}x + a_{i2}x^2 + ... + a_{i2t}x^{2t}$$

where $h_i = \sum_{k=1}^{2t+1} r_k \tau_i^k$. [22]

Theorem 5.1. With $t+1$ participants' $h_i'(k)$, we can recover h_i.

Theorem 5.2. Ranking h_i is equivalent to ranking τ_i.

5.3.2 Robustness to Connection Loss

We analyze how our secure ranking algorithm can tolerate connection losses. We assume that before Step 2, there is no user connection loss.[4]

[4]If u_i loses the connection in Step 2 and cannot share τ_i^k with SSS, then there is no way to rank u_i's position because the server has no u_i's information.

Theorem 5.3. To finish Step 3-5, there needs at least one user online for each group. Suppose that every user has p_l probability to lose connection and there are n users, the success probability $\geq (1 - p_l^{\lfloor n/(2t+1) \rfloor})^{2t+1}$.

Theorem 5.4. To finish Step 6-8, $\geq t + 1$ users need to be online.

5.3.3 Security

Here, we analyze the security of our mechanism.

Theorem 5.5 If there are no more than t collusive participants, then these participants cannot recover all the other users' τ_i.

5.3.4 Complexity

We analyze the algorithm from communication and computation complexity perspectives for participant clients.

Communication Complexity - $O(tn)$. In Step 2, the communication overhead of one participant to send $\tau_i, \tau_i^2, ..., \tau_i^{2t+1}$ is $O(t^2)$, while each user received data is $O(tn)$. In Step 3, the complexity is $O(t)$. In Step 5, for sending data, the complexity is $O(n)$; for receiving data, the complexity is $O(tn)$. In Step 7, the complexity is $O(n)$. Combing them together, the communication complexity of the whole process is $O(tn)$ as $t < n$.

Computation Complexity - $O(tn)$. The main computation processes of each client include (1) calculating secret shares for $\tau_i, \tau_i^2, ..., \tau_i^{2t+1}$ with $(t + 1, 2t + 1)$-SSS in Step 2, which is $O(t^2)$, (2) calculating secret shares of r_k in Step 3, which is $O(t)$, (3) computing h_i in Step 4, which is $O(tn)$, and (4) calculating h_i' in Step 6, which is $O(tn)$. Hence, the final computation complexity is $O(tn)$.

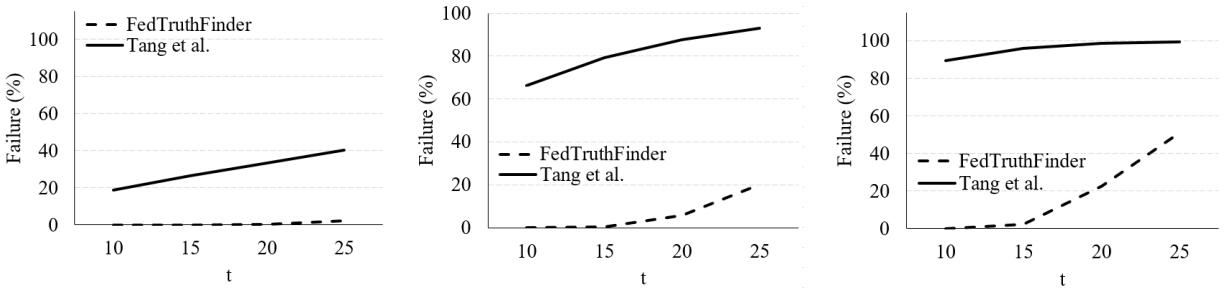
6 Evaluation

6.1 Numerical Analysis for Connection Loss

We have theoretically proven that our algorithm can learn the event confidence and trustworthiness ranking like the original centralized algorithms. This experiment then focuses on how the connection loss would impact FedTruthFinder quantitatively, since the unstable mobile network connection is a key characteristic for mobile crowdsensing. A practical mechanism should be able to fight against the unpredictable connection loss. In general, participants' connection loss may bring two types of negative impacts to the iterative truth discovery algorithm.

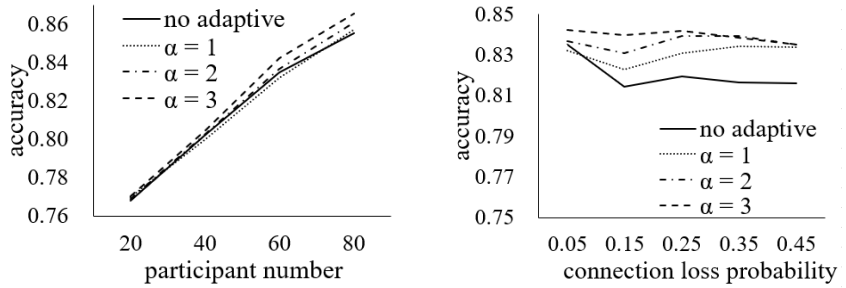
- **A small number of sensed data for truth discovery.** While FedTruthFinder can learn an aggregate truth as long as more than t participants are online, the data sources for the truth would be decreased. This would also affect the performance of the learned truth.
- **Possible failure of the whole algorithm.** If a large number of participants lose the connection and only fewer than t participants remain online, then the whole running process of FedTruthFinder would fail and no result can be learned.

Specifically, we conduct the numerical analysis for two parts of FedTruthFinder respectively, i.e., event confidence computation and participant trustworthiness ranking. We vary the probability of one participant losing the connection (denoted as p_l). If $p_l = 0.01$, a participant has 1% probability of dropping out of the crowdsensing campaign due to one-time connection loss. Then, if a participant needs to connect to the server for n times, it has $1 - (1 - p_l)^n$ probability to lose the connection. In the experiment, we test $p_l = 0.01/0.05/0.1$ to represent good/moderate/bad connection scenarios.



(a) $p_l = 0.01$ (b) $p_l = 0.05$ (c) $p_l = 0.1$

Figure 5: Failure probability of trustworthiness ranking.



(a) Participants number (b) Connection loss

Figure 6: Detection accuracy of FedTruthFinder.

6.1.1 Event Confidence Computation

To compare with FedTruthFinder, we consider the state-of-the-art way to do iterative truth discovery with an SSS-based secure aggregation (SA) protocol [1, 34], denoted as *SA*, which can also tolerate a certain level of participant connection loss. In brief, SA leverages a double-masking method to ensure that the truth discovery can run when some users lose the connection. However, not like FedTruthFinder which only needs a one-time connection for each participant to finish one iteration of ρ-computation, SA needs a two-time connection (double-masking).

Figure 3 shows the number of sensed data for truth discovery in each iteration for FedTruthFinder and SA (the total number of data is set to 100). Literature has shown that the number of iterations for truth discovery is often smaller than 10 [36] and thus we set the number of iterations up to 10. FedTruthFinder can always obtain more sensed data than SA as FedTruthFinder needs fewer connections. Especially, when the network connection condition is bad ($p_l = 0.1$), the performance improvement of FedTruthFinder over SA is more significant.

Figure 4 shows the algorithm failure probability (i.e., fewer than t users are online) for FedTruthFinder and SA (we set the number of participants to 100 and t to 50; we do not plot $p_l = 0.01$ as both methods are successful almost all the time). FedTruthFinder can significantly reduce the failure probability compared to SA. For example, when the network connection quality is moderate ($p_l = 0.05$), FedTruthFinder has around 99% probability to finish successfully for 10 iterations; however, SA has only around 1% probability. For the bad connection scenario ($p_l = 0.1$), SA will fail with more than 20% probability from iteration 3, while FedTruthFinder can keep working well until iteration 6. This reveals that, even if both FedTruthFinder and SA cannot finish all the ten iterations due to a bad network connection condition, FedTruthFinder can run a larger number of iterations, making the truth more reliable.

6.1.2 Participant Trustworthiness Rank

As none of the prior studies have addressed the privacy-preserving trustworthiness ranking problem, we cannot directly find a baseline method to compare. Meanwhile, our proposed trustworthiness ranking algorithm is inspired by the basic idea from **(author?)** [22] while significantly enhancing the capability to tolerate participants' connection loss. To this end, we compare FedTruthFinder and **(author?)** [22] when certain participants lose connections.

In federated trustworthiness ranking, t is the key parameter related to how many user groups are created, which significantly impacts the algorithm success probability (Theorem 5.3). Suppose the total number of users is 100, we set $t = 10/15/20/25$. The algorithm failure probability is shown in Figure 5. With the increase of t, the failure probability of FedTruthFinder rises. This fits our expectation as a larger t means that more user groups are generated and the user number per group is reduced. As FedTruthFinder needs at least one user online for each group, smaller user number per group means that the robustness against connection loss is weakened, leading to higher failure probability. Compared to **(author?)** [22], our algorithm significantly increases the success probability when connection is unstable. When $p_l = 0.1$ and $t = 10$, the failure probability of our ranking algorithm is 0.04%, but **(author?)** [22] is 96.18%. Hence, our algorithm could be an appropriate choice for ranking crowdsensing participants' trustworthiness scores considering the unstable mobile network connection environment.

6.2 Evaluation of Traffic Light Detection

We also test FedTruthFinder for traffic light detection, a representative crowdsensing task [16, 25]. We focus on the truth discovery accuracy and the runtime efficiency of FedTruthFinder, which has not been evaluated in the previous numerical analysis.

6.2.1 Data and Tasks

To evaluate FedTruthFinder on the traffic light detection task, we leverage a real-life open dataset including taxis' trajectories. Specifically, the dataset contains time-stamped GPS trajectories from 536 taxis in San Francisco, U.S. in one month of 2008 [18]. Following **(author?)** [16], we manually label 96 traffic light detection event positions using the Street View of Google Maps (Figure 7). Then, we randomly select some taxis as participants; their trajectories in the dataset are used to simulate their activities — if a taxi stops around an event's location, it may report the data. The report error rate (indicating trustworthiness) of each taxi is randomized in $[0, 0.5]$. The default participant number is 60 and the connection loss probability is 0.05. The event confidence function is set to 'logistic' as it performs better than 'sum'. t in SSS is set to half of the total participant number. To increase the randomness, each taxi randomly reports 20% of the events, and then each setting of the experiment is repeated by 50 times.

6.2.2 Experiment Platform

Our platform is an Alibaba cloud server with CPU of Intel Xeon Platinum 8163 (12 cores, 2.5GHz) and 24GB memory. The operating system is Ubuntu 20.04. FedTruthFinder is implemented by *Rust* 1.56. *Docker*[5] is adopted to simulate the crowdsensing server and participants.

6.2.3 Truth Discovery Accuracy

Figure 6(a) and 6(b) plot the accuracy regarding the number of participants and connection loss probability, respectively. Specifically, we compare FedTruthFinder with and without the adaptive truth updating technique

[5]https://www.docker.com/

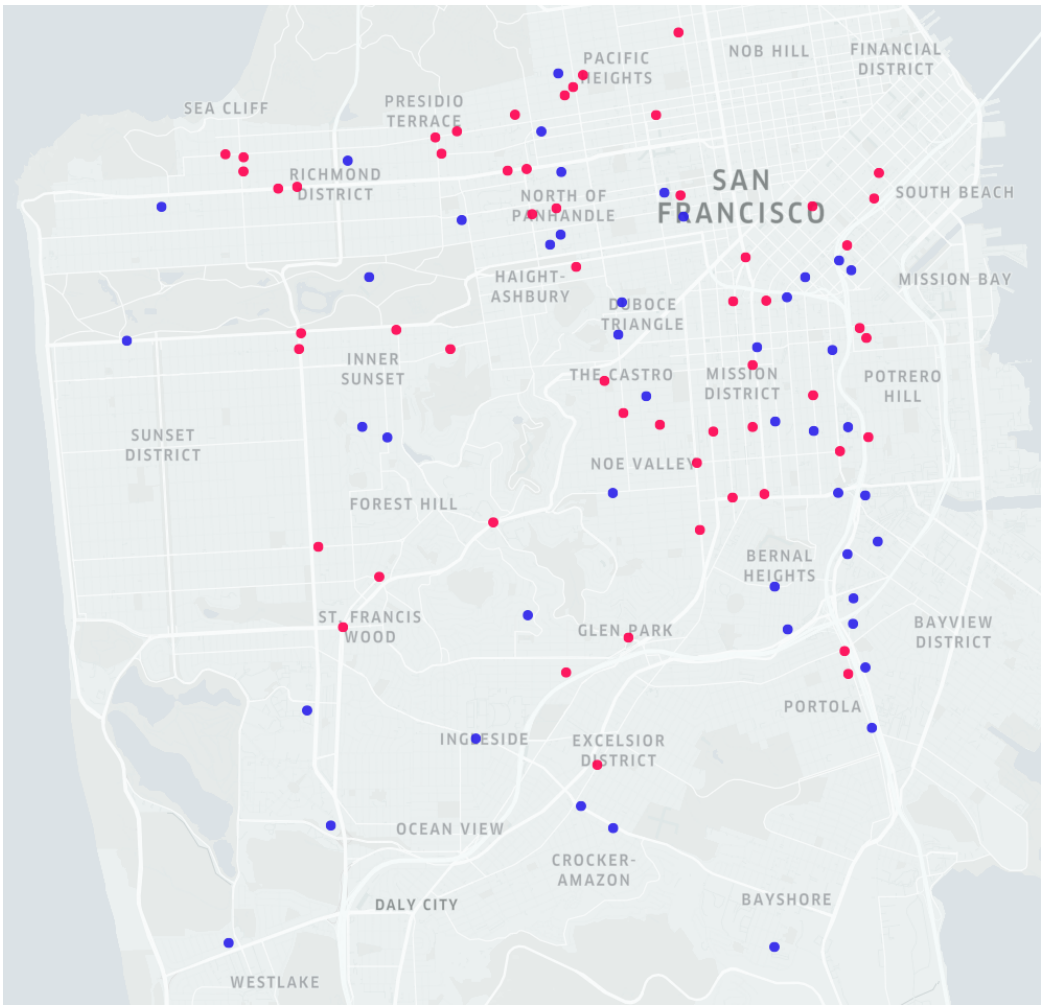
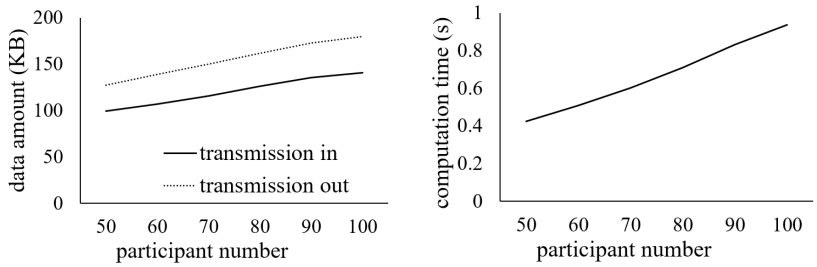


Figure 7: Traffic light event locations (red: true; blue: false). The red points represent the true event locations (i.e, with traffic lights) and the blue points mean the false event locations.

(Sec. 4.2.1). For the adaptive updating, we try $\alpha = 1/2/3$ (Eq. 28), and find $\alpha = 3$ performs the best. The adaptive updating ($\alpha = 3$) can consistently improve the accuracy with different participant numbers and connection losses. Specifically, with more participants and higher connection losses, the improvement is more significant. When the connection loss probability increases, the accuracy decreases gradually. This again verifies the effectiveness of FedTruthFinder over SA [34] — FedTruthFinder reduces the communication times per truth discover iteration compared to SA, which is conceptually equivalent to the reduction of connection losses in practice.

6.2.4 Runtime Efficiency

Figure 8(a) and 8(b) record each participant's data transmission amount and computation time, respectively. Note that the computation time is mostly spent in the truth finding step, while the trustworthiness ranking takes only $\sim$0.01s. The results show that the data transmission amount and computation time are both small, verifying the practicality of FedTruthFinder.



(a) Data transmission (b) Domputation time

Figure 8: Runtime efficiency of FedTruthFinder.

Table 6: Comparison of our work and representative related work. (NTP: No Third Party, AT: Assess Trustworthiness, CA: Collusion Attacks)

	Privacy Protection		**NTP**	**AT**	**Connection Loss**		**CA**
	Sensed Data	*Completed Tasks*			*Fault Tolerance*	*Bias Avoidance*	
(author?) [15]	√	×	×	×	×	×	×
(author?) [40]	√	×	×	×	√	×	×
(author?) [14]	√	×	√	×	√	×	×
(author?) [34]	√	×	√	×	√	×	√
(author?) [41]	√	×	√	×	√	×	√
(author?) [38]	√	×	×	×	√	×	×
Our Work	√	√	√	√	√	√	√

7 Related Work

Truth discovery is a traditional research direction as we may often receive diverse and even conflicting information about one event [7]. In the pioneering research [36], authors discuss the truth discovery problem when there are many conflicting facts about one subject on different websites. Besides information from websites, truth discovery is also important in many other areas such as social sensing [26] and crowdsourcing [6, 33].

Mobile crowdsensing [39], as a particular type of crowdsourcing that needs workers to do location-based sensing tasks, would also face the truth discovery problem [24]. Meanwhile, privacy protection is also an important issue to consider in crowdsensing, especially for location privacy [4, 30, 27, 32, 29, 28]. Most prior research focuses on protecting crowdsensing participants' location privacy in task allocation [27, 32, 28] or for particular crowdsensing tasks such as missing data inference [30, 29].

Recently, some studies investigate the privacy-preserving truth discovery in crowdsensing [13, 15, 14, 38, 37, 34]. One research direction is applying data perturbation methods such as differential privacy to participants' sensed data [8, 9], but these methods degrade the truth finding accuracy. Another research direction follows the federated learning [35] paradigm that participants' raw data will not be directly sent to the server with certain encryption techniques, while the aggregation results (i.e., detected truths) can be accurately learned. However, the existing privacy-preserving truth discovery methods usually suffer from certain assumptions which may not stand in reality, e.g., online/non-concluding participants [13, 15, 14], or third-party non-concluding servers [38, 37]. Moreover, no prior work considers hiding participants' completed tasks or tracking participants' trustworthiness in a privacy-preserving manner, which has been addressed by our work.

Table 6 summarizes the characteristics of our work and representative related work published in top venues recently. In particular, our work is the **first** privacy-preserving crowdsensing truth discovery research that considers (i) *providing a feasible solution to participant trustworthiness assessment* when the trustworthiness

scores are not revealed, and (ii) *hiding participants' completed tasks* to provide stronger privacy protection. Moreover, when dealing with the connection loss during the iterative crowdsensing truth discovery process, our work (i) proposes an adaptive event confidence updating function to reserve the data contributions of drop-out participants to avoid the truth bias toward alive participants, and (ii) designs an SSS-based scheme to defend against participants' collusion attacks while ensuring the high communication efficiency.

8 Conclusion

In this paper, we propose *FedTruthFinder*, a crowdsensing federated truth discovery mechanism that can not only find aggregate truth from multiple participants' sensed data, but also rank participants' trustworthiness in a privacy-preserving manner. The primary characteristic of FedTruthFinder is its capability to tolerate network connection loss of participants in both event confidence calculation and participant trustworthiness ranking. As a byproduct, our proposed federated ranking algorithm can also serve other applications when the privacy-preserving data ranking is needed and the network connections are unstable. Following most related papers, this work assumes participants to be semi-honest; in the future, we would explore the more challenging scenario that participants may behave maliciously.

References

[1] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. CCS, 2017.

[2] D. Chai, L. Wang, K. Chen, and Q. Yang. Secure federated matrix factorization. IEEE Intelligent Systems, 2020.

[3] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. IEEE Communications Magazine, 49(11):32–39, 2011.

[4] X. Han, L. Wang, and W. Fan. Is hidden safe? location protection against machine-learning prediction attacks in social networks. MIS Quarterly, 45(2):821–858, 2021.

[5] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. arXiv:1610.05492, 2016.

[6] H. Li, B. Zhao, and A. Fuxman. The wisdom of minority: discovering and targeting the right group of workers for crowdsourcing. In WWW, pages 165–176, 2014.

[7] Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, and J. Han. A survey on truth discovery. SIGKDD Explor. Newsl., 17(2):1–16, Feb. 2016.

[8] Y. Li, C. Miao, L. Su, J. Gao, Q. Li, B. Ding, Z. Qin, and K. Ren. An efficient two-layer mechanism for privacy-preserving truth discovery. KDD, 2018.

[9] Y. Li, H. Xiao, Z. Qin, C. Miao, L. Su, J. Gao, K. Ren, and B. Ding. Towards differentially private truth discovery for crowd sensing systems. ICDCS, pages 1156–1166, 2020.

[10] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang. A secure federated transfer learning framework. IEEE Intelligent Systems, 35(4):70–82, 2020.

[11] H. Ma, D. Zhao, and P. Yuan. Opportunities in mobile crowd sensing. IEEE Communications Magazine, 52(8):29–35, 2014.

[12] C. Meng, W. Jiang, Y. Li, J. Gao, L. Su, H. Ding, and Y. Cheng. Truth discovery on crowd sensing of correlated entities. In Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems, pages 169–182, 2015.

[13] C. Miao, W. Jiang, L. Su, Y. Li, S. Guo, Z. Qin, H. Xiao, J. Gao, and K. Ren. Cloud-enabled privacy-preserving truth discovery in crowd sensing systems. In SenSys, 2015.

[14] C. Miao, W. Jiang, L. Su, Y. Li, S. Guo, Z. Qin, H. Xiao, J. Gao, and K. Ren. Privacy-preserving truth discovery in crowd sensing systems. ACM Transactions on Sensor Networks, 15:1 – 32, 2019.

[15] C. Miao, L. Su, W. Jiang, Y. Li, and M. Tian. A lightweight privacy-preserving truth discovery framework for mobile crowd sensing systems. INFOCOM, pages 1–9, 2017.

[16] R. W. Ouyang, M. Srivastava, A. Toniolo, and T. J. Norman. Truth discovery in crowdsourced detection of spatial events. IEEE transactions on knowledge and data engineering, 28(4):1047–1060, 2015.

[17] D. Peng, F. Wu, and G. Chen. Data quality guided incentive mechanism design for crowdsensing. IEEE Transactions on Mobile Computing, 17:307–319, 2018.

[18] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD dataset epfl/mobility (v. 2009-02-24). Downloaded from `https://crawdad.org/epfl/mobility/20090224`, Feb. 2009.

[19] V. Primault, A. Boutet, S. B. Mokhtar, and L. Brunie. The long road to computational location privacy: A survey. IEEE Communications Surveys & Tutorials, 21:2772–2793, 2019.

[20] S. Reddy, D. Estrin, M. Hansen, and M. Srivastava. Examining micro-payments for participatory sensing data collections. UbiComp, 2010.

[21] A. Shamir. How to share a secret. Communications of the ACM, 22(11):612–613, 1979.

[22] C. Tang, G. Shi, and Z. Yao. Secure multi-party computation protocol for sequencing problem. SCIENTIA SINICA Informationis, 41(7):789–797, 2011.

[23] X. Tang, C. Wang, X. Yuan, and Q. Wang. Non-interactive privacy-preserving truth discovery in crowd sensing applications. INFOCOM, pages 1988–1996, 2018.

[24] D. Wang, T. Abdelzaher, and L. M. Kaplan. Surrogate mobile sensing. IEEE Communications Magazine, 52:36–41, 2014.

[25] D. Wang, L. Kaplan, T. Abdelzaher, and C. C. Aggarwal. On credibility estimation tradeoffs in assured social sensing. IEEE Journal on Selected Areas in Communications, 31(6):1026–1037, 2013.

[26] D. Wang, L. Kaplan, H. Le, and T. Abdelzaher. On truth discovery in social sensing: A maximum likelihood estimation approach. In IPSN, pages 233–244, 2012.

[27] L. Wang, D. Yang, X. Han, T. Wang, D. Zhang, and X. Ma. Location privacy-preserving task allocation for mobile crowdsensing with differential geo-obfuscation. WWW, 2017.

[28] L. Wang, D. Yang, X. Han, D. Zhang, and X. Ma. Mobile crowdsourcing task allocation with differential-and-distortion geo-obfuscation. IEEE Transactions on Dependable and Secure Computing, 18:967–981, 2021.

[29] L. Wang, D. Zhang, D. Yang, B. Y. Lim, X. Han, and X. Ma. Sparse mobile crowdsensing with differential and distortion location privacy. IEEE Transactions on Information Forensics and Security, 15:2735–2749, 2020.

[30] L. Wang, D. Zhang, D. Yang, B. Y. Lim, and X. Ma. Differential location privacy for sparse mobile crowdsensing. In ICDM, pages 1257–1262. IEEE, 2016.

[31] T. Wang, C. Lv, C. Wang, F. Chen, and Y. Luo. A secure truth discovery for data aggregation in mobile crowd sensing. Secur. Commun. Networks, 2021:2296386:1–2296386:15, 2021.

[32] Z. Wang, J. Hu, R. Lv, J. Wei, Q. Wang, D. Yang, and H. Qi. Personalized privacy-preserving task allocation for mobile crowdsensing. IEEE Transactions on Mobile Computing, 18:1330–1341, 2019.

[33] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In NeurIPS, 2009.

[34] G. Xu, H. Li, S. Liu, M. Wen, and R. Lu. Efficient and privacy-preserving truth discovery in mobile crowd sensing systems. IEEE Transactions on Vehicular Technology, 68:3854–3865, 2019.

[35] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology, 10(2):12, 2019.

[36] X. Yin, J. Han, and S. Y. Philip. Truth discovery with multiple conflicting information providers on the web. IEEE Transactions on Knowledge and Data Engineering, 20(6):796–808, 2008.

[37] C. Zhang, C. Xu, L. Zhu, Y. Li, C. Zhang, and H. Wu. An efficient and privacy-preserving truth discovery scheme in crowdsensing applications. Computers & Security, 97:101848, 2020.

[38] C. Zhang, L. Zhu, C. Xu, X. Liu, and K. Sharif. Reliable and privacy-preserving truth discovery for mobile crowdsensing systems. IEEE Transactions on Dependable and Secure Computing, 18:1245–1260, 2021.

[39] D. Zhang, L. Wang, H. Xiong, and B. Guo. 4w1h in mobile crowd sensing. IEEE Communications Magazine, 52(8):42–48, 2014.

[40] Y. Zheng, H. Duan, and C. Wang. Learning the truth privately and confidently: Encrypted confidence-aware truth discovery in mobile crowdsensing. IEEE Transactions on Information Forensics and Security, 13:2475–2489, 2018.

[41] Y. Zheng, H. Duan, X. Yuan, and C. Wang. Privacy-aware and efficient mobile crowdsensing with truth discovery. IEEE Transactions on Dependable and Secure Computing, 17:121–133, 2020.

A Appendix

A.1 Theoretical Proof

Proof of Lemma 5.1. It is clear that,

$$\sum_{k=1}^{2t+1} r_k(0)\tau_{i_k}(0) = \sum_{k=1}^{2t+1} r_k \tau_i^k \tag{34}$$

Besides, both $r_k(x)$ and $\tau_{i_k}(x)$ are t-degree polynomials, and thus the degree of $\sum_k r_k(x)\tau_{i_k}(x)$ is $2t$. □

Proof of Theorem 5.1. With Lemma 5.1, for N $(= 2t+1)$ groups, $\gamma(gid(u_j)) = \sum_{k=1}^{2t+1} r_k(gid(u_j))\tau_{i_k}(gid(u_j))$ (Step 4) is:

$$\begin{pmatrix} 1 & 1 & 1^2 & ... & 1^{2t} \\ 1 & 2 & 2^2 & ... & 2^{2t} \\ ... & ... & ... & ... & ... \\ 1 & N & N^2 & ... & N^{2t} \end{pmatrix} \begin{pmatrix} h_i \\ a_{i1} \\ ... \\ a_{i2t} \end{pmatrix} = \begin{pmatrix} \gamma(1) \\ \gamma(2) \\ ... \\ \gamma(N) \end{pmatrix}$$

then,

$$\begin{pmatrix} h_i \\ a_{i1} \\ ... \\ a_{i2t} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1^2 & ... & 1^{2t} \\ 1 & 2 & 2^2 & ... & 2^{2t} \\ ... & ... & ... & ... & ... \\ 1 & N & N^2 & ... & N^{2t} \end{pmatrix}^{-1} \begin{pmatrix} \gamma(1) \\ \gamma(2) \\ ... \\ \gamma(N) \end{pmatrix}$$

so,

$$h_i = \sum_{g=1}^{N} \lambda(g)\gamma(g)$$

In Step 5, $h_i(g) = \lambda(g)\gamma(g)$ is shared with $(t+1, n)$-SSS to all the participants from every group $g \in [1, 2t+1]$. Hence, according to the additive homomorphism property of SSS [21], we can easily recover h_i by receiving $t+1$ participants' $h_i'(k) = \sum_{g=1}^{2t+1} h_i(g, k)$. □

Proof of Theorem 5.2. As $\tau_i > 0$ and $r_k > 0$, $h_i = \sum_k r_k \tau_i^k$ will keep the same ranking as τ_i. □

Proof of Theorem 5.3. For Step 3 to 5, if there is at least one user in every group, then the process can continue. So the probability of failure incurred by one specific group g is all the users in g losing the connections, i.e., $p^{n_g} \leq p_l^{\lfloor n/(2t+1) \rfloor}$ (n_g is the user number in g). So for g, the probability of at least one user online $\geq 1 - p_l^{\lfloor n/(2t+1) \rfloor}$. With $2t+1$ groups, the success probability $\geq (1 - p_l^{\lfloor n/(2t+1) \rfloor})^{2t+1}$. □

Proof of Theorem 5.4. This is based on the property of $(t+1, n)$-SSS in Step 5. □

Proof of Theorem 5.5. In Step 2, $\tau_i^k (k = 1...2t+1)$ is shared with $(t+1, 2t+1)$-SSS. So, if t participants collude, they can get at most $t \cdot (2t+1)$ equations when t participants are from t groups. However, the number of unknown parameters (including τ_i and t random coefficients for sharing each τ_i^k) is $t \cdot (2t+1) + 1$. Hence, these t collusive participants cannot recover other participants' τ_i. □

A.2 Mechanism Extension to Multi-class and Continuous-value Events

Multi-class Events. For a multi-class event (m classes), we can see it as m binary events, so that our method can be directly applied.

Continuous-value Events. For continuous-value events, following the literature, we may adopt other proper event confidence and participant trustworthiness updating functions such as CRH [34, 41]. Specifically, suppose that the discovered truth sensed value of a continuous event e_j is ρ_j, and u_i's sensed data of e_j is $\hat{\rho}_{ij}$, then the event truth (confidence) and participant trustworthiness updating functions can be:

$$\rho_j = \frac{\sum_{u_i \in \mathcal{U}_{e_j}} \tau_i \cdot \hat{\rho}_{ij}}{\sum_{u_i \in \mathcal{U}_{e_j}} \tau_i} \tag{35}$$

$$\tau_i = \log(\sum_{u_i \in \mathcal{U}} \sum_{e_j \in \mathcal{E}_{u_i}} \frac{(\rho_j - \hat{\rho}_{ij})^2}{|\mathcal{E}_{u_i}|}) - \log(\sum_{e_j \in \mathcal{E}_{u_i}} \frac{(\rho_j - \hat{\rho}_{ij})^2}{|\mathcal{E}_{u_i}|}) \tag{36}$$

where $\mathcal{U}_{e_j}$ is the set of users who sense e_j, and $\mathcal{E}_{u_i}$ is the set of events that u_i has sensed. For ρ-computation, following Sec. 4.1, we can just adapt d_{ij} and s_{ij} according to Eq. 35 (the participant $u_i \notin \mathcal{U}_{e_j}$ can still send $d_{ij} =$

$s_{ij} = 0$ to protect her task completion information). For τ-computation, Eq. 36 requires $\sum_{u_i \in \mathcal{U}} \sum_{e_j \in \mathcal{E}_{u_i}} \frac{(\rho_j - \hat{\rho}_{ij})^2}{|\mathcal{E}_{u_i}|}$, which can be done with the same SSS-based method as ρ-computation. In particular, each participant u_i can send $\sum_{e_j \in \mathcal{E}_{u_i}} \frac{(\rho_j - \hat{\rho}_{ij})^2}{|\mathcal{E}_{u_i}|}$ by secret shares, and then the server can compute the sum in a privacy-preserving manner. In a word, for continuous-value events, our mechanism can still work without revealing each participant's raw sensed data and completed tasks.

Federated Ensemble Learning: Increasing the capacity of label private recommendation systems

Abstract

Despite proven effectiveness of federated learning (FL) as a solution to private model training, FL has had limited success in the domains of ranking and recommendation systems. This is primarily due to the fact that modern recommendation systems, particularly in the context of on-line advertising, rely on large neural networks that cannot be feasibly trained on user devices. However, given increasing privacy regulations and evolving users' preferences, it is imperative for advertising platforms to invest in private learning solutions like FL that support training highly accurate models with strong privacy guarantees.

In this paper, we propose Federated Ensemble Learning (FEL) as a solution to address the large memory requirement for recommendation systems subject to label privacy. FEL enables scalable label-private recommendation model training by simultaneously training multiple smaller FL models on disjoint carefully selected clusters of client devices. The output of these models is aggregated with a neural network trained either on server-side data or via a second stage of private on-device training. Our experimental results demonstrate that FEL leads to 0.43–2.31% model quality improvement over traditional on-device federated learning — a significant improvement for ranking and recommendation system use cases.

A Introduction

Federated learning (FL) has emerged as an effective approach to address consumer privacy needs by allowing edge devices to collaboratively train a machine learning (ML) model, while the raw data samples remain on-device [5, 20]. While FL has been deployed for a variety of machine learning tasks, such as smart keyboard [1], personalized assistant services [17], computer vision [27], healthcare [45], it has seen limited adoption for ranking and recommendation tasks [32, 42]. This is due to the fact that constrained client resources in FL prevent training a large, high-accuracy recommendation model (often in the order of GBs or TBs [30, 53]), while meeting the privacy requirement. However, recommendation models need to achieve a strong user privacy and a high accuracy at the same time [1],

Furthermore, in contrast to conventional privacy settings of FL, a unique characteristic of modern recommendation systems is that models are usually trained on public features but with private labels. For instance, features such as user profiles and product catalogs are known to advertising platforms, but the labels, i.e., transaction details, are considered third-party (private) data. Matching cross-site or cross-app data is privacy-sensitive and can be subject to regulation and user-device policies. This new label-only privacy setting has gained significant interest from both academia and industry recently [13, 33].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] A recent study from Baidu [53] indicates that even a 0.1% accuracy drop is considered unacceptable for its ranking and recommendation tasks.

Focusing on the new label-only privacy requirement, we propose a new FL framework called Federated Ensemble Learning (FEL). FEL addresses the key limitations of FL for ranking systems in the novel setting of label privacy.

FEL proceeds in three stages: first, users are clustered into groups of similar behaviors using public features; second, a leaf model is learned for every cluster of users, simultaneously with other clusters using FL with global differential privacy (DP); and third, the leaf models (or parts of them) are concatenated to form a back-bone for a larger server model that can be trained on public data on the server or on private data on user devices. The insight behind FEL is that if client device clusters are appropriately formed, leaf models can learn distinct, potentially complementary, representations from each cluster, which are later combined to enhance the overall prediction capability. At each stage, the FEL framework ensures that the resulting ensemble satisfies the privacy goals by adapting the noise characteristics of DP for each leaf. We use composition theory to estimate the privacy cost of the ensemble aggregation layer and to automatically tune the noise added.

FEL excels when the number of observations per user is small, but the number of users is high (e.g., in the order of billions) — a setting that is common for recommendation and ranking tasks. We have deployed FEL in the production environment of a large-scale ranking system. We show that the deployed recommendation task achieves 0.43% precision gain compared to the vanilla FL baseline in the production environment, which is significant in the context of production ranking and recommendation systems. We observe significant gains of 1.55–2.31% using the open-source datasets, Ads click prediction [46] and image classification [29]. In addition, we show that FEL outperforms standard FL in the presence of differential privacy (DP) noise by 0.66–1.93%.

B Background: Federated Learning and Privacy Assumptions

FL trains a model collaboratively using client devices without requiring the raw data to be shared with service providers. However, the key constraint in using vanilla FL training for recommendation and ranking tasks is the limited model size it can support. In many federated recommendation and ranking tasks, the input space is large, e.g., over 1,000 features, and the data distribution is multi-modal. To achieve high accuracy in this setting, we have to increase the overall model capacity. However, FL can be applied only to sufficiently small models that can be transmitted and trained on end-user devices.

Prior work studied increasing model capacity by leveraging client heterogeneity: training larger models on devices that are more powerful, while sending smaller models to less capable devices. The smaller model can be a subsampled model that is later aggregated to the supernet [6, 19, 9], or a different model that later transfers its learned knowledge to the larger model with knowledge distillation [26, 23]. These approaches still limit the model capacity as the model size is capped by the most powerful client devices that still cannot train GB-size models.

Privacy Assumptions of FEL: FEL targets the label-only privacy setting, where the input is public (i.e., accessible to the service provider) while the label is private. Several advertising, recommendation, and survey/analytics applications fall into this category [13, 33, 37, 43].

Many recommendation tasks use features that are public from the perspective of the recommender system. Public features include user attributes that are explicitly shared at sign-up time (e.g., age or gender) [46], externally observable user behavior (e.g., public movie reviews) [15], or item information that is provided by the item vendors [38]. On the other hand, the labels of recommendation tasks may be considered private, e.g., user conversion behavior [40, 13]. It is also possible to use a mixture of public and private features. FEL can be further extended to incorporate private features (Section C).

While user labels must be kept private, i.e., unknown to the service provider, there can be opt-in users who consent to sharing their private label information with the service provider to improve the service quality. FEL does not require the presence of opt-in users; however, having some opt-in user population can simplify the training algorithm (Section C) and lower the privacy cost (Section C.2).

To avoid statistical inference attacks targeting FL, differentially private (DP) noise is added either on device or during the aggregation step [12, 48, 51]. We target the user-level differential privacy, in which the trained model weights are similarly distributed with or without a particular user [34]. We assume an honest-but-curious provider for training FL, which uses either hardware-based encryption in a trusted enclave, or software-based encryption via multiparty computation for FL aggregation [36, 25].

C Proposed Design: Federated Ensemble Learning (FEL)

Rather than training one model across all users, FEL proposes to train a distinct leaf model per user cluster and later aggregate the leaf models on the server to obtain a larger-capacity model. Ensemble methods similar in spirit have shown promising potential with classical machine learning algorithms, e.g., in the form of AdaBoost, random forest, and XGBoost [22]. We explored different variations in how the clients are clustered and how the leaf models are aggregated.

C.1 Federated Ensemble Learning

Figure 1 illustrates the proposed design of Federated Ensemble Learning (FEL). First, we cluster the clients into a desired number of clusters (Figure 1, Step 1). Then, we train a leaf model for each cluster using traditional FL (Figure 1, Step 2). After training, the server uses the ensemble of the leaf models for inference requests (Figure 1, Step 3). On each inference request, the server passes the public input features through all the leaf models. Then, the output of each leaf model and/or the output of the last hidden layer of each leaf model is passed to the ensemble aggregation layer, which outputs the final prediction. Below, we lay out how FEL forms clusters and implements ensemble aggregation, and how FEL can be extended to incorporate private features.

Forming Clusters Client clusters can be formed based on distinctive characteristics of the users, e.g., a user's age, location, or past preferences. Clusters can also be obtained by simple hashing, or through popular clustering approaches such as k-means or Gaussian mixture methods. Marketers have been forming clusters of clients to target each cluster more effectively, and those well-studied clustering methods can be adopted [4].

Rudimentary Ensemble Aggregation: A simplest way to implement ensemble aggregation is to collect the prediction output of each leaf model and perform typical aggregation methods, such as *mean*, *median*, or *max*, to generate the final prediction. This approach is similar to bagging technique leveraged in random forest [44].

Neural Network (NN)-based Ensemble Aggregation: NN-based ensemble aggregation uses a separate neural network that takes in the prediction and the output of the last hidden layer of all leaf models as input to generate the final prediction. We call this additional neural network model *the over-arch model*. The over-arch model is trained after all the leaf models are trained. The over-arch model can be trained in several different ways. In the presence of opt-in users, the server can simply use them to train the over-arch model. Otherwise, the server can again use FL to train the over-arch model on each client, where the server sends the output of the leaf models to the client, and the client uses it as an input to train the over-arch model locally.

Extending to Private Features FEL can be extended to support private features only known to the clients. This can be done by (privately) training a separate leaf model (private leaf model) that only takes in private client-side features. The output of the private leaf model is used as an input to the ensemble aggregation layer along with other leaf models. When the private leaf model is used, part of the inference must happen on-device instead of entirely on the server. Specifically, the server sends outputs of the leaf models to each client, which ensembles them with its private leaf model output on-device for prediction. By performing the ensemble on-device, the input/output of the private leaf model is never exposed to the server. The private leaf model is trained with conventional FL as well.

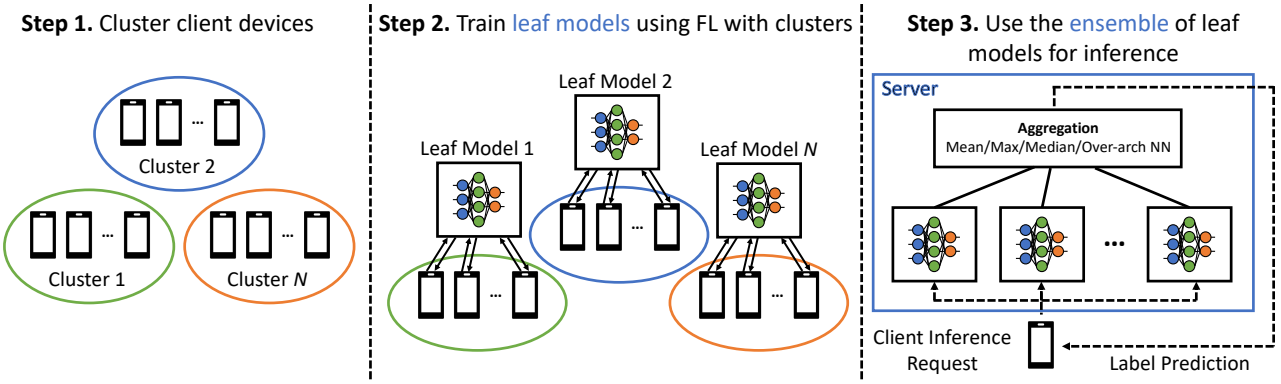


Figure 1: Federated Ensemble Learning architectures

C.2 Privacy Analysis for Federated Ensemble Learning

Ensuring data privacy is the utmost requirement and this work is the first to conduct privacy analysis for federated ensemble learning. Differential privacy for federated learning bounds how much the distribution of model parameters change between two datasets that differ in labels contributed by a single user [34]. We use a generalization of differential privacy [11] based on the Rényi divergence:

Definition 1 (Renyi Differential Privacy (RDP) [35]). A *randomized mechanism* M *with domain* D *is* (α, ϵ)*-RDP with order* $\alpha \in (1, \infty)$ *iff for any two neighboring datasets* $D, D' \in \mathrm{D}$*:*

$$\mathrm{D}_\alpha(\mathrm{M}(D)||\mathrm{M}(D')) := \frac{1}{\alpha - 1} \log E_{x \sim \mathrm{M}(D')} \left[\left(\frac{\Pr[\mathrm{M}(D) = x]}{\Pr[\mathrm{M}(D') = x]} \right)^\alpha \right] \leq \epsilon.$$

FEL is a multi-step process. To analyze the privacy bounds of a multi-step process, a common approach in differential privacy is to evaluate each process individually, then calculate the overall privacy bounds by composing all of the steps. In particular, we focus on two main composition theorems in differential privacy, sequential and parallel composition [10]. Formally we define:

Theorem 1 (Sequential Composition). *Let there be n RDP-mechanisms* M_i *with* (α, ϵ_i)*-RDP when being computed on a dataset* D *of the input domain* D*. Then, the composition of n mechanisms* $\mathrm{M}(\mathrm{M}_1(D), \ldots, \mathrm{M}_n(D))$ *is* $(\alpha, \sum_{i=1}^n \epsilon_i)$*-RDP*

Theorem 2 (Parallel Composition). *Let there be n RDP-mechanisms* M_i *with* (α, ϵ_i)*-RDP when being computed on disjoint subset* D_i *of the input domain* D*. Then, the composition of n mechanisms* $\mathrm{M}(\mathrm{M}_1(D), \ldots, \mathrm{M}_n(D))$ *is* $(\alpha, \max_{i=1}^n \epsilon_i)$*-RDP*

Sequential composition considers the case where a task uses the same users (even if different steps use different parts of the user's data) in different steps of an algorithm. For example, if the algorithm has four steps each with a privacy cost ε and uses the same users in all the steps, the total privacy cost of the algorithm would become 4ε. Parallel composition considers a case where each step is applied to different users. From the earlier example, if four disjoint sets of users were used for the four steps, the overall privacy cost would be $\max(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4)$, where ε_k is the privacy cost of the k-th step. The compositions are tracked using RDP, which leads to tighter bounds than (ϵ, δ)-DP.

When analyzing the privacy cost of training the leaf models, the parallel composition theorem applies as each leaf model is trained with a disjoint set of users. If training a leaf model with cluster $i \in \{1, \ldots, N\}$ has a privacy cost e_i, the privacy cost of the entire leaf model is $e_{\text{leafs}} = \max(e_1, \ldots, e_N)$.

The privacy cost of the ensemble aggregation layer e_{agg} depends on the aggregation method that is used (mean, max, median, or NN-based). When using the mean, max, and median ensemble aggregation, $e_{\text{agg}} = 0$ as both theorems show that privacy cost increases only when the additional step (ensemble aggregation) uses user data. In this case, the total privacy cost e_{tot} is simply e_{leafs}.

When the NN-based approach is used, however, user data is used to train the over-arch NN layer and $e_{\text{agg}} > 0$. Depending on exactly how the over-arch NN layer is trained, e_{tot} can be calculated in the following way. First, if the over-arch NN layer is trained with the users that trained the leaf models, sequential composition theorem applies ($e_{\text{tot}} = e_{\text{agg}} + e_{\text{leafs}}$). Second, if the over-arch NN layer is trained with a completely different set of users, the parallel composition theorem applies ($e_{\text{tot}} = \max(e_{\text{agg}}, e_{\text{leafs}})$). Finally, if the over-arch NN layer is trained with opt-in users, $e_{\text{agg}} = 0$ because no private data is used, and $e_{\text{tot}} = e_{\text{leafs}}$. In all cases, the privacy cost of FEL does not significantly deteriorate over the vanilla FL (which is similar to e_{leafs}).

D Evaluation

In this section we would like to provide more details about our experiments. We first provide details about each dataset that was used. We then provide details about the model architectures that we used for each dataset.

D.1 Datasets for FEL

We explored three datasets for FEL: An internal production dataset that represents a real-world ads-ranking system, an external dataset that addresses a similar ads-ranking task, and an image classification dataset.

D.1.1 Production Dataset

Production dataset is an internal dataset that captures whether a user installs a mobile application after being shown a relevant advertisement item. A few hundred features are used as an input (the exact number cannot be disclosed) to predict a binary label (install/not-install). All the input features are public. For training, we use advertisement data from a random sample of 35 million users over a period of one month. Randomly selected 15 million users from the following week were used for testing.

D.1.2 Taobao CTR Dataset

The Taobao dataset contains 26 million interactions (click/non-click when an Ad was shown) between 1.14 million users and 847 thousand items across an 8-day period. The dataset uses 9 user features (e.g., gender or occupation), 6 item features (e.g., price or brand), and two contextual features (e.g., the day of week), which we assume to be all public to the service provider.

In the Taobao CTR dataset, 16 out of the 17 features are sparse, with a categorical value encoding instead of a continuous, floating point value. While server-based recommendation models use large embedding tables to convert these sparse features into a floating point embedding [54, 38, 8], training such embedding tables on device is complicated because of the large memory capacity requirement (e.g., in the order of GB to TB [53, 2, 52, 30]) and can leak private information more easily through gradients [40]. Thus, we assume an architecture where embedding tables are pre-trained with opt-in users and are hosted on the server, while the rest of the model is trained with FEL using sparse features translated through the pre-trained tables. We randomly selected 10% of the users as opt-in.

Our setup cannot achieve the accuracy that can be reached when we fully train the embedding tables, as we pre-train the embeddings and fix their weight during FL. However, our setup represents a practical FL setup where training embedding tables on-device is prohibitive, due to client resource limitations [39] and privacy concerns [40].

D.1.3 CelebA Smile Prediction Dataset

While FEL is originally designed for recommendation and ranking tasks, we study its generality to non-recommendation models with CelebFaces Attributes Dataset (CelebA) [29]. CelebA consists of $200,288$ images

belonging to $9,343$ unique celebrities. Each image has 40 binary facial attribute annotations (e.g., bald, long hair, attractive, etc) and covers large pose variations and backgrounds. We defined distinguishing between smiling/non-smiling images as our target task.

The Taobao dataset contains 26 million interactions (click/non-click when an Ad was shown) between 1.14 million users and 847 thousand items across an 8-day period. The dataset uses 9 user features (e.g., gender or occupation), 6 item features (e.g., price or brand), and two contextual features (e.g., the day of week), which we assume to be all public to the service provider. The details on how we preprocess the dataset can be found in the appendix.

D.2 Model Architectures

Production/Taobao Dataset: For recommendation datasets (production/Taobao CTR), we use a model that consists of 3 fully-connected hidden layers. The number of units at each hidden layer is decreasing exponentially with a parameter K. For instance, if $K = 4$ and the input layer has 512 features, our neural network would have $[512, 128, 32, 8, 1]$ neurons. For each dataset, we tune K to obtain a resulting model of approximately 10MB. By doing so, it allows us to train a neural network even on older, low-tier devices with more limited memory capacity. ReLu is used as an activation function after each layer apart from the last one, where Sigmoid and binary cross-entropy was used.

For both datasets, we use synchronous FL with FedAvg [34]. We used the following hyperparameters for the Taobao dataset from an extensive hyperparameter search: client batch size of 32, 5 local epochs, 4096 clients per round, and a learning rate of 0.579 with SGD. Clients are selected at random and each only participates once (1 global epoch). The production dataset used similar hyperparameters.

For Taobao dataset's server-side pre-trained embedding table, we use an embedding dimension of 32, and train it with the 10% opt-in users for 1 epoch using AdaGrad optimizer with learning rate of 0.01.

CelebA Dataset: For CelebA, we follow the setup of prior work [39] and use a four layer CNN with dropout rate of 0.1, stride of 1, and padding of 2. We preprocess all images in train/validation/test sets; each image is resized and cropped to 32×32 pixels, then normalized by 0.5 mean and 0.5 standard deviation. We use asynchronous FL with a client batch size of 32 samples, 1 local epoch, 30 global epochs, and a learning rate of 0.899 with SGD.

D.3 Evaluation Methodology

Our evaluation aims to answer the following questions:

- Can FEL improve the model prediction quality over vanilla FL? [Section D.4]
- How do different ensemble aggregation methods affect the model accuracy? [Section D.4]
- How do different clustering methods affect the model accuracy? [Section D.5]
- How does FEL affect privacy compared to vanilla FL? [Section D.6]

To answer these questions we used the three datasets presented in Section D.1. To study recommendation and ranking tasks, we used a production dataset and an open-source, Taobao's Click-Through-Rate (CTR) prediction dataset [24]. To study the effect of FEL on non-recommendation use-cases, we additionally studied the LEAF CelebA Smile Prediction dataset [29]. More details about these datasets and their associated model architecture can be found in Section D.1.

Both the FL baseline and the FEL leaf models used the same set of hyperparameters. The FL baseline is trained using all the available client data. In FEL, the client data is clustered, and one leaf model is trained for each cluster. We vary the number of clusters from 3–10 and evaluate different clustering methods. When training the over-arch NN layer, a small subset of opt-in users is used.

Table 7: Explanation of different cluster methods in Figure 2 (right).

Dataset	Config	Feature	# clusters
Production	Clustering 1	Age	5
	Clustering 2	App	5
	Clustering 3	Location	4
	Clustering 4	Click ratio	10
Taobao [46]	Clustering 1	Age	7
	Clustering 2	Consumption	4
	Clustering 3	City level	5
CelebA [29]	Clustering 1	# Attributes	3
	Clustering 2	K-means	3
	Clustering 3	K-means	5

Table 8: FEL's prediction accuracy improvement over the baseline FL for different datasets. Following common practice of each dataset, Taobao uses AUC and CelebA uses accuracy as their metric. Production data's baseline accuracy is not disclosed.

	Production	Taobao [46] AUC	CelebA [29] accuracy
Baseline FL	-	0.5418 [3]	90.75
FEL (Mean Best)	(+0.27%)	0.5522 (+1.92%)	91.68 (+1.02%)
FEL (Median Best)	(+0.29%)	0.5459 (+0.74%)	91.35 (+0.66%)
FEL (Max Best)	(-0.06%)	0.5418 (-0.1%)	91.46 (+0.78%)
FEL (NN-based Best)	**(+0.43%)**	**0.5544 (+2.31%)**	**92.16 (+1.55%)**

D.4 Prediction Quality Improvement of FEL

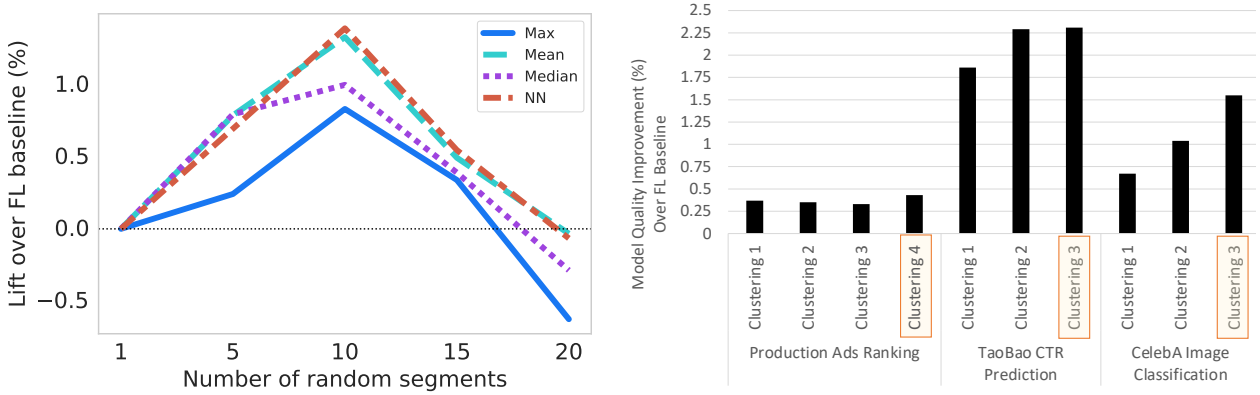
Overall, FEL achieves **0.43%** and **2.31%** prediction quality improvement over vanilla FL for production and Taobao datasets, respectively – a significant improvement for ranking and recommendation system use cases[2]. For non-recommendation tasks (CelebA), FEL shows similar improvement of **1.55%**, indicating that FEL can be generalized to non-recommendation use-cases as well. Table 8 summarizes the resulting prediction quality improvement of FEL compared to the baseline FL. We used accuracy for CelebA [39] and ROC-AUC (AUC) for Taobao [46]. We used normalized entropy for the production dataset, which we cannot disclose and only show the relative improvement. For different ensemble aggregation methods, we vary the clustering methods and report the best-accuracy results. Among the different ensemble aggregation methods, adding an over-arch NN layer provided the best prediction quality improvement, followed by mean and median.

D.5 Prediction Quality Improvement of Different Clustering Methods

Effects of the Number of Clusters: To understand the effect of the number of client clusters in the final model quality improvement, we varied the number of clusters in the Taobao dataset while using random clustering. Figure 2 (left) summarizes the result. There is an optimal setting for the number of clusters used in FEL. Going beyond the optimal setting for the number of clusters results in worse model accuracy. When the number of clusters is too small, the final model capacity is limited as there are not enough leaf models to ensemble. If the number of clusters is too large, each leaf model cannot learn enough information as the clients in each cluster are too few. The optimal number of clusters depends on the number of available devices that participate within each

[2] [53] mentioned 0.1% model quality improvement as significant and [50] considered 0.23% as impactful in similar recommendation and ranking use-cases.

[3] Taobao's baseline AUC is 0.26% less than the baseline FL result presented at [40], potentially due to simpler model architecture and freezed pre-trained embedding tables.



(a) (b)

Figure 2: Accuracy improvement for different number of clusters (segments) for each ensemble aggregation method (left), and different clustering methods when using over-arch NN layer (right). Different clustering methods are explain in Table 7

Table 9: Taobao dataset with DP. Percentage of FEL's accuracy improvement over the FL baseline with the same level of DP noise is shown. Table 7 explains the clustering configurations.

Config	ϵ	Mean	Median	Max	Over-Arch NN
Clustering 1	∞	+1.27%	-0.23%	-0.10%	**+1.86%**
	3.78	+0.63%	-0.14%	+0.11%	**+0.66%**
	1.56	+0.32%	-0.23%	+0.34%	**+0.68%**
Clustering 2	∞	+1.92%	-0.46%	-1.64%	**+2.29%**
	3.78	+0.75%	-0.21%	+0.03%	**+1.38%**
	1.56	+0.69%	-0.11%	+0.74%	**+0.76%**
Clustering 3	∞	+1.86%	+0.74%	-2.07%	**+2.31%**
	3.78	+1.49%	-0.09%	+1.03%	**+1.93%**
	1.56	+0.71%	-0.37%	+1.26%	**+1.02%**

cluster and, here, the number of partitions can be treated as a hyperparameter [21].

Effects of Features Used in Clustering: We also varied the clustering methods for each dataset and observed the effect on the final accuracy. We explored different clustering methods for different datasets and presented the best performing methods. Table 7 (Section D.3) summarizes the clustering methods. Here, we show the result for the best performing over-arch NN-based ensemble aggregation for brevity. For the production dataset, we used user age, the app category where the ad was displayed, location (larger geographic regions), and previous click ratio of the users to cluster the users. For Taobao, we used user age, city level, and consumption level. For CelebA, we clustered the 40 binary attributes of each user using K-means clustering or simply used the number of present attributes.

Figure 2 (right) shows that clustering can affect the final model accuracy significantly. For the production dataset, clustering using the click ratio (Clustering 4) showed the best accuracy. For Taobao, clustering with city level showed the best accuracy (Clustering 3). For CelebA, using K-means clustering was the best (Clustering 3). The results show that clustering methods as well as the number of clusters are two important hyperparameters of FEL.

D.6 Evaluation Results with Differential Privacy

Table 9 shows the accuracy improvement of FEL compared to vanilla FL for two different levels of DP noise, along with the case of no DP noise ($\epsilon = \infty$). We assume the over-arch NN layer was trained with opt-in data and no DP noise added when training the over-arch NN layer. Table 9 shows that even when DP noise is added, FEL shows meaningful accuracy improvement over vanilla FL. Again, we observe that the over-arch NN layer and mean aggregations still provide the most significant gains. However, smaller ϵ leads to reduced accuracy gain, possibly due to larger injected noise. Another interesting observation is that the max ensemble aggregation improves the accuracy when DP noise is added, unlike the no-DP-noise case where it did not show any improvement. One possible reason is that DP noise mitigates the effects of outliers in training.

E Related Work

Ensemble Distillation. Lin et al. [26] relied on unlabeled data generated by a generative model to aggregate knowledge from all heterogeneous client models, Gong et al. [14] focused on communication efficiency and privacy guarantee with one-shot offline knowledge distillation.

Boosted Federated Learning. Boosting and bagging are two prominent approaches for model ensemble learning. In Hamer et al. [16], an ensemble of pre-trained based predictors is fine tuned via federated learning, thus saving on communication costs. Luo et al. [31] suggest gradient boosting decision tree (GBDT) method, which takes the average gradient of similar samples and its own gradient as a new gradient to improve the accuracy of the local model.

Local Ensemble Learning. FedEnsemble uses random permutations to update a group of K models, and then obtains predictions through model averaging, instead of aggregating local models to update a single global model [47]. Attota et al. [3] propose a multi-view ensemble learning approach aimed at maximizing the learning efficiency of different classes of attacks for intrusion detection tasks.

Ensemble Aggregation. FedBE takes a Bayesian inference perspective by sampling and combining higher-quality global models via Bayesian ensemble for robust aggregation [7]. FedGRU uses both secure parameter aggregation and cluster ensembles to scale [28]. Orhobor et al. [41] assigned users into pre-specified bins and trained different regressors on each bin, which were later ensembled.

Although these methods have their own merits, they do not address the problem of the recommender and ranking systems use cases, in which each user has only a small number of examples, and require user-level privacy guarantee. As a result, none of these studies leverages the variation across users and diversity of behavior in their proposals. Our approach trains models on separate user clusters, leveraging a large user base in recommender and ranking systems. Furthermore, our over-arch model approach provides extra gains both in terms of precision and privacy budget.

F Conclusion

While Federated learning (FL) has achieved considerable success as a privacy-preserving solution for model training, its impact on ranking and recommendation systems, particularly in the context of digital advertising, remains limited. We introduce Federated Ensemble Learning (FEL) to increases the learning capacity of FL. FEL can be trained efficiently without introducing significant privacy concerns and can improve the prediction accuracy meaningfully compared to vanilla FL. This work demonstrates that FEL enables FL for demanding ranking and recommendation tasks. As future work, we plan to integrate unsupervised clustering approaches, so that the segmentation can happen automatically to optimize FEL's learning performance. Finally, to minimize the cost of managing a number of leaf models, we plan to explore ways to automatically assess the quality of leaf models to pinpoint under-represented clusters and seek possible mitigation such as dynamic clustering and leaf retraining.

%bibliographystyleabbrv

References

[1] S. AbdulRahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani. A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. IEEE Internet of Things Journal, 8(7):5476–5497, 2020.

[2] B. Acun, M. Murphy, X. Wang, J. Nie, C.-J. Wu, and K. Hazelwood. Understanding training efficiency of deep learning recommendation models at scale. In 2021 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2021.

[3] D. C. Attota, V. Mothukuri, R. M. Parizi, and S. Pouriyeh. An ensemble multi-view federated learning intrusion detection for IoT. IEEE Access, 9:117734–117745, 2021.

[4] T. Beane and D. Ennis. Market segmentation: a review. European journal of marketing, 1987.

[5] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, page 1175–1191, 2017.

[6] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. arXiv preprint arXiv:1812.07210, 2018.

[7] H.-Y. Chen and W.-L. Chao. FedBE: Making Bayesian model ensemble applicable to federated learning. arXiv preprint arXiv:2009.01974, 2020.

[8] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In Proceedings of the 1st workshop on deep learning for recommender systems, pages 7–10, 2016.

[9] E. Diao, J. Ding, and V. Tarokh. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.

[10] C. Dwork and J. Lei. Differential privacy and robust statistics. In Proceedings of the forty-first annual ACM symposium on Theory of computing, pages 371–380, 2009.

[11] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3, pages 265–284. Springer, 2006.

[12] R. C. Geyer, T. Klein, and M. Nabi. Differentially private federated learning: A client level perspective. CoRR, abs/1712.07557, 2017.

[13] B. Ghazi, N. Golowich, R. Kumar, P. Manurangsi, and C. Zhang. Deep learning with label differential privacy. In Advances in Neural Information Processing Systems (NeurIPS), 2021.

[14] X. Gong, A. Sharma, S. Karanam, Z. Wu, T. Chen, D. Doermann, and A. Innanje. Preserving privacy in federated learning with ensemble cross-domain knowledge distillation. 2022.

[15] Grouplens. MovieLens 20M dataset, 2016.

[16] J. Hamer, M. Mohri, and A. T. Suresh. FedBoost: A communication-efficient algorithm for federated learning. In International Conference on Machine Learning, pages 3973–3983. PMLR, 2020.

[17] K. Hao. How Apple personalizes Siri without hoovering up your data. Technology Review, 2020.

[18] C. He, M. Annavaram, and S. Avestimehr. Group knowledge transfer: Federated learning of large CNNs at the edge. Advances in Neural Information Processing Systems, 33:14068–14080, 2020.

[19] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane. FjORD: Fair and accurate federated learning under heterogeneous targets with ordered dropout. Advances in Neural Information Processing Systems, 34, 2021.

[20] D. Huba, J. Nguyen, K. Malik, R. Zhu, M. Rabbat, A. Yousefpour, C.-J. Wu, H. Zhan, P. Ustinov, H. Srinivas, et al. Papaya: Practical, private, and scalable federated learning. Proceedings of Machine Learning and Systems, 4, 2022.

[21] Y. G. Kim and C.-J. Wu. AutoFL: Enabling heterogeneity-aware energy efficient federated learning. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '21, 2021.

[22] N. K. Le, Y. Liu, Q. M. Nguyen, Q. Liu, F. Liu, Q. Cai, and S. Hirche. FedXGBoost: Privacy-preserving XGBoost for federated learning. arXiv preprint arXiv:2106.10662, 2021. Presented at International Workshop on Federated and Transfer Learning for Data Sparsity and Confidentiality (FTL-IJCAI'21).

[23] D. Li and J. Wang. FedMD: Heterogenous federated learning via model distillation. arXiv preprint arXiv:1910.03581, 2019.

[24] L. Li, J. Hong, S. Min, and Y. Xue. A novel CTR prediction model based on DeepFM for Taobao data. In 2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID), pages 184–187. IEEE, 2021.

[25] Y. Li, Y. Zhou, A. Jolfaei, D. Yu, G. Xu, and X. Zheng. Privacy-preserving federated learning framework based on chained secure multiparty computing. IEEE Internet of Things Journal, 8(8):6178–6186, 2020.

[26] T. Lin, L. Kong, S. U. Stich, and M. Jaggi. Ensemble distillation for robust model fusion in federated learning. Advances in Neural Information Processing Systems, 33:2351–2363, 2020.

[27] Y. Liu, A. Huang, Y. Luo, H. Huang, Y. Liu, Y. Chen, L. Feng, T. Chen, H. Yu, and Q. Yang. FedVision: An online visual object detection platform powered by federated learning. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 13172–13179, 2020.

[28] Y. Liu, J. James, J. Kang, D. Niyato, and S. Zhang. Privacy-preserving traffic flow prediction: A federated learning approach. IEEE Internet of Things Journal, 7(8):7751–7763, 2020.

[29] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In Proceedings of the IEEE international conference on computer vision, pages 3730–3738, 2015.

[30] M. Lui, Y. Yetim, z. Özkan, Z. Zhao, S.-Y. Tsai, C.-J. Wu, and M. Hempstead. Understanding capacity-driven scale-out neural recommendation inference. In 2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2021.

[31] C. Luo, X. Chen, J. Xu, and S. Zhang. Research on privacy protection of multi source data based on improved GBDT federated ensemble method with different metrics. Physical Communication, 49:101347, 2021.

[32] K. Maeng, H. Lu, L. Melis, J. Nguyen, M. Rabbat, and C.-J. Wu. Towards fair federated recommendation learning: Characterizing the inter-dependence of system and data heterogeneity. arXiv preprint arXiv:2206.02633, 2022.

[33] M. Malek, I. Mironov, K. Prasad, I. Shilov, and F. Tramer. Antipodes of label differential privacy: PATE and ALIBI. In Advances in Neural Information Processing Systems (NeurIPS), 2021.

[34] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. arXiv preprint arXiv:1710.06963, 2017.

[35] I. Mironov. Rényi differential privacy. In 2017 IEEE 30th computer security foundations symposium (CSF), pages 263–275. IEEE, 2017.

[36] A. Mondal, Y. More, R. H. Rooparaghunath, and D. Gupta. Flatee: Federated learning across trusted execution environments. arXiv preprint arXiv:2111.06867, 2021.

[37] M. Nalpas and S. Dutton. A more private way to measure ad conversions, the event conversion measurement API, Oct. 2020.

[38] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. arXiv preprint arXiv:1906.00091, 2019.

[39] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba. Federated learning with buffered asynchronous aggregation. arXiv preprint arXiv:2106.06639, 2021.

[40] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen. Billion-scale federated learning on mobile clients: A submodel design with tunable privacy. In Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, pages 1–14, 2020.

[41] O. I. Orhobor, L. N. Soldatova, and R. D. King. Federated ensemble regression using classification. In International Conference on Discovery Science, pages 325–339. Springer, 2020.

[42] V. Perifanis and P. S. Efraimidis. Federated neural collaborative filtering. Knowledge-Based Systems, 242:108441, 2022.

[43] J. J. Pfeiffer III, D. Charles, D. Gilton, Y. H. Jung, M. Parsana, and E. Anderson. Masked LARk: Masked learning, aggregation and reporting workflow. arXiv preprint arXiv:2110.14794, 2021.

[44] A. M. Prasad, L. R. Iverson, and A. Liaw. Newer classification and regression tree techniques: bagging and random forests for ecological prediction. Ecosystems, 9(2):181–199, 2006.

[45] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein, et al. The future of digital health with federated learning. NPJ digital medicine, 3(1):1–7, 2020.

[46] P. Sabnagapati. Ad display/click data on taobao.com, 2020.

[47] N. Shi, F. Lai, R. A. Kontar, and M. Chowdhury. Fed-ensemble: Improving generalization through model ensembling in federated learning. arXiv preprint arXiv:2107.10663, 2021.

[48] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou. A hybrid approach to privacy-preserving federated learning - (extended abstract). Inform. Spektrum, 42(5):356–357, 2019.

[49] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564, 2018.

[50] R. Wang, B. Fu, G. Fu, and M. Wang. Deep & cross network for ad click predictions. In Proceedings of the ADKDD'17, pages 1–7. 2017.

[51] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor. Federated learning with differential privacy: Algorithms and performance analysis. IEEE Transactions on Information Forensics and Security, 15:3454–3469, 2020.

[52] M. Wilkening, U. Gupta, S. Hsia, C. Trippel, C.-J. Wu, D. Brooks, and G.-Y. Wei. RecSSD: Near data processing for solid state drive based recommendation inference. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021.

[53] W. Zhao, D. Xie, R. Jia, Y. Qian, R. Ding, M. Sun, and P. Li. Distributed hierarchical GPU parameter server for massive scale deep learning ads systems. Proceedings of Machine Learning and Systems, 2:412–428, 2020.

[54] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai. Deep interest network for click-through rate prediction. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1059–1068, 2018.

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314

Non-profit Org.
U.S. Postage
PAID
Los Alamitos, CA
Permit 1398