Improving Availability of Vertical Federated Learning: Relaxing Inference on Non-overlapping Data

ZHENGHANG REN, LIU YANG, and KAI CHEN, Hong Kong University of Science and Technology, China

Vertical Federated Learning (VFL) enables multiple parties to collaboratively train a machine learning model over vertically distributed datasets without data privacy leakage. However, there is a limitation of the current VFL solutions: current VFL models fail to conduct inference on non-overlapping samples during inference. This limitation seriously damages the VFL model's availability because, in practice, overlapping samples may only take up a small portion of the whole data at each party which means a large part of inference tasks will fail. In this article, we propose a novel VFL framework which enables federated inference on non-overlapping data. Our framework regards the distributed features as privileged information which is available in the training period but disappears during inference. We distill the knowledge of such privileged features and transfer them to the parties' local model which only processes local features. Furthermore, we adopt Oblivious Transfer (OT) to preserve data ID privacy during training and inference. Empirically, we evaluate the model on the real-world dataset collected from Criteo and Taobao. Besides, we also provide a security analysis of the proposed framework.

CCS Concepts: • **Security and privacy** → **Privacy-preserving protocols**; • **Computing methodologies** → **Learning paradigms;**

Additional Key Words and Phrases: Availability, privacy, vertical federated learning

ACM Reference format:
Zhenghang Ren, Liu Yang, and Kai Chen. 2022. Improving Availability of Vertical Federated Learning: Relaxing Inference on Non-overlapping Data. *ACM Trans. Intell. Syst. Technol.* 13, 4, Article 58 (June 2022), 20 pages.
https://doi.org/10.1145/3501817

1 INTRODUCTION

In **Vertical Federated Learning (VFL)** [7, 15, 20, 37], the model is trained across multiple parties on vertically partitioned dataset without exposing the training data to other parties. Introduced by [37] in 2019, with the increasing concerns about privacy and new legislation for privacy protection [34], VFL has drawn great attention both in academic and industry, especially for collaborative training between companies. For example, one company may wish to improve the machine learning model's accuracy with the help of the dataset from another company that has some other

This work is supported in part by the Key-Area Research and Development Program of Guangdong Province (2021B0101400001) and the Hong Kong RGC TRS T41-603/20-R, GRF 16213621, GRF 16215119, and the Turing AI Computing Cloud (TACC) [38].
Authors' address: Z. Ren, L. Yang, and K. Chen, Hong Kong University of Science and Technology, China; emails: zrenak@cse.ust.hk, lyangau@cse.ust.hk, kaichen@cse.ust.hk.

2157-6904/2022/06-ART58 $15.00
https://doi.org/10.1145/3501817

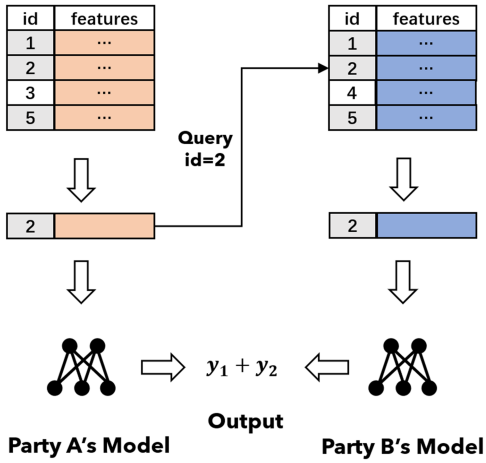

Fig. 1. Inference in VFL model. During inference, party A and party B find the features of the same instance with id = 2. Then the features are entered into models for inference. The results are then aggregated together as the final result.

features. In this case, the two companies need to train a machine learning model that processes both features from its dataset and the other company's dataset.

Training a machine learning model with outsourced features without privacy leakage is challenging both in theory and practice. Existing solutions mainly use cryptographic tools including **homomorphic encryption (HE)** [6, 20, 27] and secure **multi-party computation (MPL)**[9]. After training is finished, the parties will publish their part of VFL models for online serving, and the inference is conducted by multiple parties aggregating their inferences of the same sample. See Figure 1 for a simple example.

However, the aggregation of the local model's output during inference leads to two main problems: availability and data ID privacy [17].

— Availability: in the online-serving stage, the inference task may fail because the data sample with the same ID may be absent in one of the parties and this problem is unique in VFL since VFL involves collaboration during inference. See Figure 2 for an example.
— Data ID privacy: the inference task will expose the data ID to all parties, which may violate some party's privacy restrictions. The root of these problems is the heterogeneity of different datasets that distribute among multiple parties.

To solve these problems, one of the naive solutions to ensure the success of inference is to centralize all datasets to one server for online serving, but that will undoubtedly violate the privacy restrictions. We may also adapt **Federated Transfer Learning (FTL)** [11, 21, 22, 37] to transfer the knowledge from the passive party's model to the active party's model. However, the source domain of FTL is limited to one party and thus are not feasible when we want to train a model that combines both datasets from two parties.

To preserve privacy during inference, we may apply **Oblivious Transfer (OT)** [16] to send all inference results without the sender knowing the data ID, which leads to tremendous cost in computation and network. All these challenges can be summarized as follows:

— The balance between availability and privacy: centralizing all datasets for training and inference will ensure inference success but violate privacy, while decentralized private solutions will not guarantee inference success.

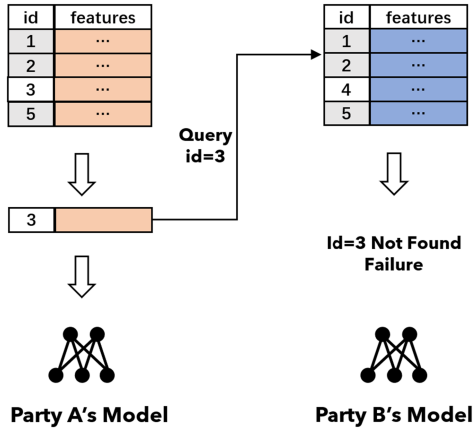

Fig. 2. VFL inference failure. The failure happens because the query for id in B's database is failed.

— The balance between efficiency and privacy: the centralized solution is efficient for both training and inference yet violates privacy. In contrast, decentralized private solutions in the online-serving stage, which involves OT will greatly harm the efficiency during inference.

1.1 Motivation

The key observation to solve these problems is that the VFL model can supervise the "student model" which only requires a local dataset for inference. However, the VFL model is still preferred during inference because it is trained on a larger dataset across multiple parties. So the desired solution will be conducting inference on the student model after the inference on the VFL model has failed.

Based on the observation, we propose a VFL solution with high availability and data ID privacy. First, we ensure inference success by training a student model with the party's local data and soft labels marked by the federated learning model. Second, we address the inference stage's privacy issue by constructing an oblivious inference protocol across all parties so that the sender remains oblivious about the data ID. Moreover, we strike a balance between privacy and efficiency in oblivious inference by constructing hash tables in all parties and moving the preparation of ciphertexts to the offline stage so that the computation and transmission of ciphertexts are greatly reduced. We tested the solution based on the VFL method proposed by [37], but our solution can also be applied to other VFL solutions such as SecureML [25], which is based on secret sharing.

1.2 Contribution

Our contributions can be summarized as follow:

— We propose the training of the student model in VFL that ensures the success of inference by conducting knowledge distillation from the VFL model to the party's local model. When the VFL model fails because of the absence of a data ID, the local model can still give inference.
— We propose a privacy-preserving inference protocol which preserves the privacy of data ID when conducting inference task.
— The efficiency in online inference is improved by constructing a hash table and moving encryption operations to the preparation stage.

We tested our construction of the student model in federated learning algorithms in real-world datasets. It shows that the student model's performance has increased with the supervision of the VFL model. Also, we analyzed the cost for oblivious inference in online serving. Compared to the naive application of OT, our solution has a much lower cost during inference.

2 BACKGROUND

Federated learning was first proposed by Google in 2016 mainly to address the privacy concern during the training of machine learning models in mobile devices such as cell phones [14, 23]. Since then, many other scenarios have been added to federated learning, including federated learning over large datasets owned by companies. In this case, each party has a relatively large dataset and features compared to model devices. Usually, these features are different between parties, and they are not allowed to exchange data because of privacy restrictions. So VFL is proposed to train a model with outsourced features across multiple parties.

2.1 Vertical Federated Learning

Our work is suitable for all VFL algorithms, including [24, 25, 37]. To simplify the narration, we build our algorithm based on Paillier homomorphic encryption [27]. VFL with HE classifies all parties into three classes: **passive party** who has datasets that contain only features, **active party** who has datasets that contain labels, and **coordinator** played by the federated learning platform which is responsible for orchestrating the learning process and generating Paillier key pairs. Note that there may be multiple passive parties in the training but only one active party. This article assumes that there is only one passive party, and it's easy to extend to multiple passive parties' scenario. The training is conducted by multiple parties with the following steps:

— First, the datasets in active party and passive party are aligned to find the overlapping samples to construct the training set. The overlapping samples will be the training set in later steps. See Figure 3 for an example.
— The federated learning platform (coordinator) generates a pair of Paillier keys and sends the public key to the active party and passive party.
— The active party and the passive party start training. In every iteration, the passive party loads a batch of data, calculates the output of its local model, encrypts the output, and sends it to the active party.
— The active party gets the ciphertext of the passive party's output and adds its own model's output on the same batch of data to get the encrypted output of the VFL model.
— The active and the passive party jointly calculate the encrypted loss and gradient and decrypted by the coordinator. Their local models are then updated with the gradients.

As we see in federated learning with HE in Figure 4, the local models only process local features.

2.2 Tussle of Privacy and Efficiency

In this section, we give a detailed explanation of the problems and challenges during the inference of the VFL model.

Due to the distributed features and privacy restrictions in VFL, the VFL models are also vertically distributed across multiple parties, so the inference task must be conducted by a series of queries to the parties and aggregate the inference results. In practice, the parties may not want to expose the data ID that is non-overlapping to other parties. Naturally, some parties (companies) have a secret dataset that is not allowed to be exposed to other parties due to privacy restrictions. So the problems in VFL can be summarized as the failure of inference and privacy leakage.

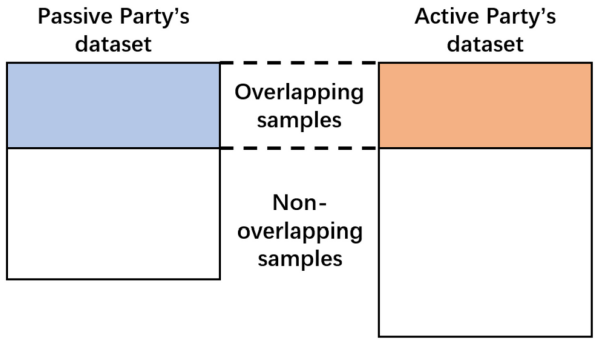

Fig. 3. Data distribution in VFL assumption. The subset marked with orange and blue have overlapping data IDs.

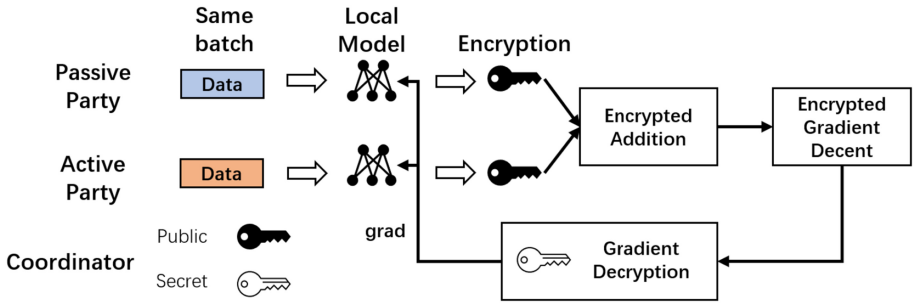

Fig. 4. Vertical federated learning based on homomorphic encrytion.

One of the naive solutions to improve the availability of the VFL model during inference is to centralize all datasets from different parties to one server. When a new entry is created in the dataset, all features from parties are also filled. In this case, no query failure will happen, and it is equal to a model deployed in a single party. This solution requires a fully trusted and honest party, but it is impossible to achieve this goal in the real world.

Besides the concerns about the inference success, the privacy of data ID in the inference should also be considered. In practice, one party may not want other parties to know the data ID in its dataset, especially for non-overlapping data. A trivial solution to this problem is to perform OT when requesting inference results from other parties. However, the efficiency is harmed because of computation and communication costs in OT, especially when the dataset grows very large.

3 PRELIMINARIES

In this section, we introduce several techniques that help to understand our solution, including **partially homomorphic encryption** (**PHE**) and OT.

3.1 Partially Homomorphic Encryption

HE specifies a series of cryptosystems that support computing over encrypted data [1, 12, 13]. PHE supports part of the operations on ciphertext. In this article, we use Paillier Homomorphic Encryption [27, 28], which supports add operation over ciphertexts. Although only part of operations is supported, the training function can still be carried out because unsupported operations can be done in plaintext locally and transmitted after encryption. In this article, we use the following operations to construct our solution.

— Key generation: $(pk, sk) = Gen(keylength)$ which generates public key pk and private (secret) key sk with specified key length.
— Encryption of plaintext x: $Enc(x, pk)$ returns the encrypted plaintext x with public key pk.
— Decryption of ciphertext: $Dec(c, sk)$ returns the decrypted ciphertext c with private key sk.

In the rest of this article, we denote the encrypted text x as $[[x]]$. There are also some evaluations supported by Paillier, which are listed below:

— Addition of ciphertext: $Add([[x]], [[y]])$ which gives the addition of two ciphertext $[[x + y]]$ and it is implemented by multiplying two ciphertexts.
— Multiplication of ciphertext and plaintext: $Mul(x, [[y]])$ which gives encrypted multiplication $[[xy]]$ and this operation is implemented by the modular power $[[y]]^x$.

With the above operations, we can conduct collaborative training with encrypted loss functions and gradients. Take the vertical linear regression model for an example, with parameter θ, the loss and gradient are calculated by a batch of samples:

$$Loss = \sum_{batch} (y_{pred} - y)^2 + \frac{\Lambda}{2}\theta\theta^T \tag{1}$$

$$grad_{local} = \alpha * \left(\sum_{batch} x_{local}(y_{pred} - y) + \Lambda\theta \right). \tag{2}$$

In the above equations, y_{pred} is encrypted, and even if the direct calculation of y^2_{pred} is not supported, we can still let one party who has y_{pred} in plaintext to calculate and encrypt it. And the solution remains the same when calculating the local model's gradient.

3.2 Oblivious Transfer

OT [16] is an important cryptographic primitive to construct the secure MPL programs and can be applied in collaborative training. In OT protocol, we define a sender S and receiver R. At the beginning of OT, the sender S has two messages m_0, m_1 and the receiver has a choice bit $r \in 0, 1$. At the end of this protocol, the sender S learns nothing about r, and the receiver gets m_r but learns nothing about m_{1-r}.

OT protocols have been extended into the choice from a set of elements [18]. That is, the basic 1-out-of-2 OT is extended to 1-out-of-n OT. In extended OT, the sender S has a set X with n elements while the receiver has a m-bit string t with $m = \lceil logn \rceil$. After OT completes, the receiver gets the element indicated by its bit string as an index $X[t]$ but learns nothing about the other elements. At the same time, the sender remains oblivious about the choice of the receiver. The main operations of OT are denoted as:

— Sending operation $Send(X, n)$ which send elements to the receiver with specified size n.
— Receiving operation $Recv(choice, n)$ which get the element in the set indicated by $choice$.

A naive extension from basic 1-out-of-2 OT to 1-out-of-n OT is to conduct $\lceil logn \rceil$ OT operations. But some optimized protocols have been proposed to reduce the cost. Reference [2] has proposed an OT protocol that performs only k basic OTs for 1-out-of-2 OT. Here, k is the secure parameters that often take 128 or 256. Reference [18] has given the solution to perform m 1-out-of-n OTs effectively for short secrets.

In the extended OT, the transfer is divided into two stages. During the first stage, the sender and receiver conduct a backward OT to share information for later OT operations. In the second stage, the two parties conduct real transfers, and the receiver gets the desired message. See Figure 5 for illustration.

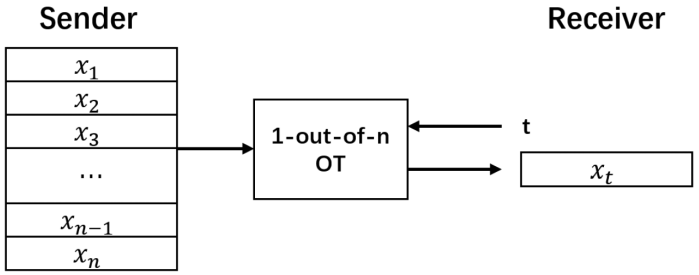

Fig. 5. 1-out-of-n OT. The sender has dataset $X = x_1, x_2 \ldots x_n$ while the receiver holds choice index t. At the end of protocol, the receiver gets x_t and the sender learns nothing about t.

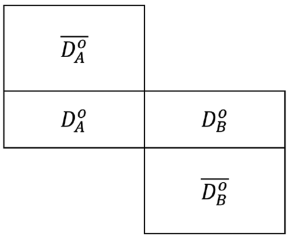

Fig. 6. Distribution of 2 parties' datasets. $D^o = D_A^o \cup D_B^o$ is the training set for VFL model.

Table 1. List of Notations used in Training Section

Notation	Description
λ	The weight for soft label from VFL model
L_s, L_d	Loss function from real labels and soft labels
f_s, f_t	Output of student model and teacher model
X_B^o	Overlapping data in the active party
X^o	Overlapping data across two parties
W_s, W_t	Parameters for the teacher model and student model

In our solution, we use 1-out-of-n OT as a basic tool to protect data ID during inference so that the choice (data ID) remains oblivious to the dataset owners. Moreover, we have made further optimizations by moving the ciphertext generation to the preparation phase without privacy leakage.

4 METHODOLOGY

Before dividing into our solution, we first formalize the notations used in the training. Table 1 shows the notations used throughout training.

The training is divided into two stages. In the first stage, the VFL model is trained with the VFL method. Then the VFL model will give encrypted soft labels on its training data to supervise the student model in the second stage. In the second stage, the student model is trained with the loss from both the real labels and soft labels marked by the VFL model in stage one. The training of the VFL model remains the same as in Section 2.1.

Besides training the VFL model, we adapt knowledge distillation to transfer the federated model's knowledge to the party's local model. Assume that we have a passive party and an active party with dataset D_A and D_B, respectively. D^o denotes the overlapping samples. $\overline{D_A^o} = D_A - D^o$ and $\overline{D_B^o} = D_B - D^o$. See Figure 6 for detail.

In previous studies of machine learning, knowledge distillation is used to transform the knowledge from a large model to a smaller one so that the inference is accelerated without loss of accuracy. Two models are trained during knowledge distillation: a student model and a teacher model. Typically the training is divided into two stages. The teacher model is trained in the first stage. While in the second stage, the student model is trained supervised by the teacher model's output (called soft labels) as well as the labels of training data. While in VFL, we transfer the knowledge from the VFL model to the active party's student model, which processes the active party's local features. Since these two models process different features, the objective function is revised as follow:

$$\min_{W_s}(1-\lambda) * L_s(y, f_s(X_B^o, W_s)) + \lambda * L_d(f_t(X^o, W_t), f_s(X_B^o, W_s)). \tag{3}$$

In the above function, the output of the student model is calculated with X_B^o, which means that the student model only processes the active party's local features. On the other hand, the VFL model's output is calculated with X^o, which consists of both parties' features.

4.1 VFL Model Training

Here in stage one, we regard the training of the VFL model as a black box, and the detail of the VFL model training will not affect our solution. The parties train a VFL model collaboratively with the protocol they have chosen. Here comes the first step of our solution: unlike previous VFL solutions that end here, the VFL model is used to predict each data sample in the training set, and the predictions are encrypted with the same public key during training. We call the predictions soft labels and denote them as $\tilde{y}$.

4.2 Student Model Training

The loss of student model is from real labels and the supervision of the VFL model. The soft labels are produced by the VFL model using D^o, so they may contain private information of the passive party's data. So, in addition to the knowledge transfer, we also need to preserve privacy during training which is solved by applying PHE to the soft labels.

Similar to stage one, the training of the student model also involves three parties: passive party, active party, and coordinator. On the one hand, at the beginning of training a student model, the passive party first generates encrypted soft labels with the public key and sends them to the active party. Then in each iteration of training, the active party calculates the encrypted loss and gradient, adds some noise (mask) to the gradient, and sends them to the coordinator, who will decrypt the loss and gradient and send the gradient back to the active party. Finally, the active party eliminates (unmask) the noise from the gradient and updates the student model. The detailed algorithm of the active party is shown in Algorithm 2.

In Algorithm 2, the active party mainly calculates encrypted loss and gradients and sends them to the coordinator in every iteration. Note that the gradient is applied with a random mask so that no private information leakage will happen at the coordinator. Also, all parties are expected to be honest but curious, which means they stick to the protocol.

On the other hand, in every iteration, the coordinator decrypts *Loss* and *mask_grad* and checks if the model has converged. The coordinator's algorithm is shown in Algorithm 1.

In the Algorithm 1, the coordinator decrypts all losses and gradients sent by the active party and sends them back for model updates. At the same time, the coordinator judges if the model has converged and sends signals to all parties. Note that the gradients are masked after decrypted. So the coordinator will not get any information from the decrypted gradient.

ALGORITHM 1: Student Model Training in the Coordinator

Result: Student model W_s
Input: max iterations t, converge threshold c;
Initialize student model parameter W_s;
i = 0, pred_loss = 0;
while $i < t$ **do**
  Recv [[*Loss*]] from active party;
  Loss = Decrypt([[*Loss*]]);
  Recv [[*mask_grad*]] from active party;
  mask_grad = Decrypt([[*Loss*]]);
  send mask_grad to active party;
  if *abs(Loss - pred_loss)* < *c* **then**
    converged = true;
    send converged to active party;
  else
    converged = false;
    send converged to active party;
    i = i + 1;
  end
end

ALGORITHM 2: Student Model Training in Active Party

Result: Student model W_s
Input: VFL model f_t with parameter W_t, hard label y, encrypted soft label $[[\tilde{y}]]$, soft label squares $[[\tilde{y}^2]]$, X_B^o, max iterations t, balance factor λ;
Initialize student model parameter W_s;
i = 0;
while $i < t$ **do**
  $[[Loss]] = \lambda * L_s(y, f_s(W_s, X_B^o) + (1-\lambda) * L_d([[\tilde{y}]], f_s(W_s, X_B^o))$;
  send [[*Loss*]] to Coordinator;
  $[[grad]] = \frac{\partial Loss}{\partial W_s}$;
  generate rand_mask;
  $[[mask\_grad]] = [[mask\_grad]] + rand\_mask$;
  send [[*mask_grad*]] to Coordinator;
  Recv mask_grad from Coordinator;
  $grad = mask\_grad - rand\_mask$;
  $W_s = W_s - \alpha * grad$;
  Recv converged from Coordinator;
  if *converged* **then**
    Return W_s;
  else
    i = i + 1;
  end
end

Table 2. List of Notations in During Inference

Notation	Description
$\binom{n}{1} - OT$	1-out-of-n OT
N	Bucket size
m	Inference result or an indicator for failure
key	Encryption/Decryption key
$Enc(x, key)$	Encryption with plaintext x and key
$Dec(c, key)$	Decryption with ciphertext c and key
$M_k = \{m_0, m_1, \ldots, m_{N-1}\}$	Inference result in bucket k
l	Size of ciphertext

4.3 ID Oblivious Inference

In this section, we describe our solution to protect the data ID during inference. We construct the protocol based on the 1-out-of-n OT protocol. With carefully designed precomputations, we divide our protocol into two phases: OT preparation phase and online phase. The preparation phase can be conducted before all inference tasks, and the online phase is conducted in every inference. First, we give an overview of our solution.

- We map all parties' data samples into several buckets with size bucket size N and a predefined hash function and give a unique address $(bucket\_id, offset)$ to all data ID.
- The passive party performs inference on its data samples and puts the inference results into the same location in the hash table as the data samples.
- When the active party needs to perform an inference task, it first calculates the $bucket\_id$ and $offset$, and then the active party gives the bucket id to the passive party, who will then perform a 1-out-of-n OT with the active party. The active party plays as the receiver with $offset$, and the passive party plays as the sender with the inference results in the bucket.

At the end of the protocol, the active party gets the inference result given by the passive party, and the passive party only knows the bucket that may contain the id. Note that the inference result could be a message indicating failure because the passive party does not have the matching ID. In that case, the active party has to conduct inference with the student model on the local dataset.

4.4 Security Definition and Notation

In this article, we use the semi-honest adversary model, also called passive adversary model. In this assumption, the parties all stick to the protocol and will not compromise with each other. However, the parties will try to extract more information from the messages during protocol execution. Compared to the stronger malicious adversary model, this assumption is weaker but highly efficient. Besides, the passive adversary model is helpful for the cooperation between companies and organizations.

Next, we give the notation we use in our description. Table 2 lists the notations used in our article.

4.5 OT Preparation

The OT preparation of Oblivious Inference is composed of two operations: the construction of a hash table and ciphertext generation.

Building the hash table. The two parties calculate the hash values of their data IDs. Here, we choose a simple hash function $H(x) = \lfloor x/N \rfloor$ where N is decided by the agreement of two parties.

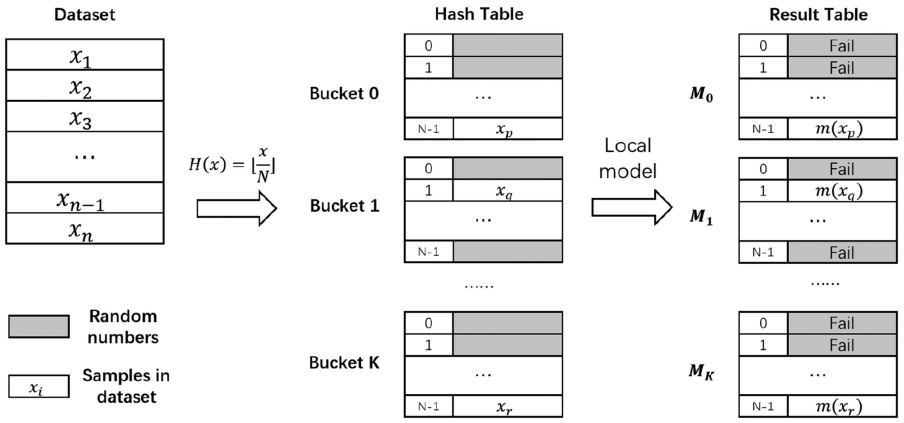

Fig. 7. OT Preparation—Hash table setup.

The collision in the hash table is solved by putting the collided elements into an array with index $offset = x \bmod N$.

At the begining, each party agrees with the number N. Then the parties put their elements into the hash table with hash function $H(x) = \lfloor x/N \rfloor$. After that the parties fill their empty entries in the buckets with random numbers. At the end of table building, each element x in the dataset is put into mth entry at n bucket, where $m = x \bmod N$ and $n = \lfloor x/N \rfloor$. See Figure 7 for example.

Note that the number of buckets may vary across parties because the largest bucket numbers $M = \lfloor \max_{x \in Dataset} x/N \rfloor$ are different across parties. After the mapping completes, the passive party performs inferences on elements in real samples in the hash table and gives **FAIL** indicator on the random masks. The result, as shown in Figure 7, is the result hash table with K buckets named $M_0, M_1, \ldots, M_K$.

Ciphertext Generation. In this operation, the active party and passive party exchange keys and generate ciphertexts for later OTs. First, the passive party generates $N \times B \times 2$ keys and the cryptography scheme could be AES or RSA or other preferred protocols. Here, $B = \lceil logN \rceil$ and the keys are given indexes (i, j, k) where $i \in \{0, \ldots, N-1\}, j \in \{0, \ldots, B-1\}, k \in \{0, 1\}$. Then for every bucket, the passive party performs recursive encryption on the elements based on the binary representation of the elements' indexes within the bucket. To be exact, if the binary representation of element's index is $\{b_0, b_1, \ldots, b_{\lceil logN \rceil - 1}\}$, then we encrypt it with the key at $(i, j, b_j), i \in \{0, \ldots, N-1\}, j \in \{0, \ldots, B-1\}$. After the encryption, each element in the bucket will have N corresponding ciphertexts, see Algorithm 3 for detail.

At the end of the ciphertext generation, the passive party has a set of encrypted buckets, and the elements in the bucket are recursively encrypted with the keys that have the same binary representation in array as the elements, see Figure 8 for illustration.

For the active party, it first generates a random permutation of $\{0, 1, \ldots, N-1\}$ as $R = \{r_0, r_1, \ldots, r_{N-1}\}$. Then for every element, r in R, the active party and the passive party conduct $\binom{1}{2}-OT$ for $\lceil logN \rceil$ times where the active party plays as the receiver, and the passive party plays as the sender. The receiver uses the binary presentation of $r_i, i \in \{0 \ldots N-1\}$ as the choice string, and the passive party uses the $i-th$ key matrix $keys(i, :, :)$ as the message. At the end of the OT operations, for $r_i \in R$, it has an array of keys that are $keys(i, j, b_j)$ where j is the $j-th$ bit in i's binary representation.

To summarize the OT preparation for oblivious inference, the passive party generates a set of keys and generates N encrypted buckets for each bucket of inference results. The active party first

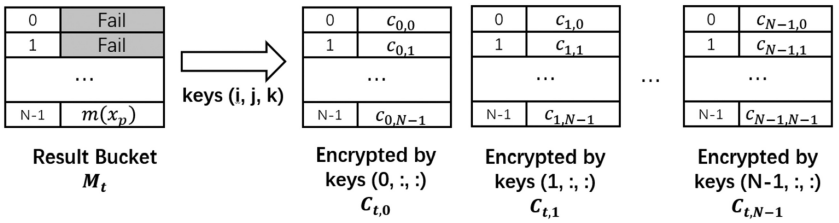

Fig. 8. Encrypted matrix for result bucket M_t.

ALGORITHM 3: Generate Encrypted elements for x_i

Result: Encrypted array of elements $C = \{c_0, c_1, \ldots, c_{N-1}\}$
Input: Element x_i, $keys$ and bucket size N;
i = 0;
set b as the binary representation of i;
while $i < N$ **do**
  tmp = x_i j = 0;
  while $j < \lceil logN \rceil$ **do**
    tmp = Enc($keys(i, j, b_j)$, tmp);
    j += 1;
  end
  $c_i = tmp$;
  i += 1;
end

generates a permutation of $\{0, 1, \ldots, N-1\}$ as R. Then it conducts OT to get an array of keys for every element in R. We will use these data for queries in the online stage.

4.6 Online Phase

The online stage starts when the active party wants to conduct an inference task with data $id = x$. At the beginning, the active party calculates the bucket index and offset with the hash function $bkt\_id = \lceil x/N \rceil$ and $offset = x \bmod N$. Then, the active party find the index t of random permutation R so that $R[t] = offset$. Next, the active party sends bkt_id and t to the passive party and the passive party sends $C_{bkt\_id,t}$ to the active party. At the end of this operation, the active party gets a encrypted bucket.

The next step is decryption, with $R[t]$ and its corresponding array of keys $\{k_0, k_1, \ldots, k_{\lceil logN \rceil}\}$, we perform decryption recursively with the array of keys. At the end of decryption, only one element (x_{offset})in the encrypted bucket is decrypted successfully because other elements in the bucket are encrypted with different combinations of keys. See Figure 9 for illustration.

We give the following assertion and proof to show that only x_{offset} is correctly decrypted, and the passive party cannot learn anything about the value of $offset$.

Theorem 4.1. *In the online stage, x_{offset} is correctly decrypted and $x_t, t! = offset$ cannot be decrypted. Moreover, the passive party learns nothing about $offset$.*

Proof. First, x_{offset} in encrypted bucket $C_{bkt\_id,t}$ is encrypted with the unique combination of keys $keys(t, j, b_j)$ where j goes from 0 to N and b_j is the $j-th$ bit of binary representation of $offset$. So only x_{offset} is correctly decrypted by the active party.

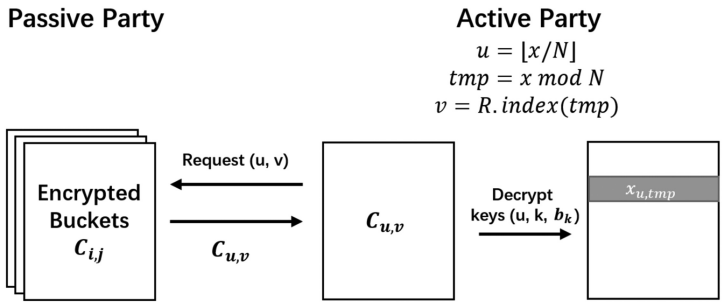

Fig. 9. The passive party sends the encrypted bucket with index u,v to the active party for decryption.

Second, the passive party only knows the index of $offset$ in the active party's random permutation. Because of the OT operation in the preparation phase, the passive party learns nothing about the combination of keys that the active party has chosen. □

The last step for the active party is to check the result of inference. If the inference is successful, then the inference from the passive party is aggregated to produce the VFL model's inference. Otherwise, if the result is an indicator for failure, the active party must conduct inference on the student model.

5 EVALUATION

This section describes our experiments showing the efficiency and performance of the student model. We conduct our experiments on two datasets: the click-through dataset collected from Criteo [8] and Taobao [31]. The results of the experiments showed the performance of the student model under different portions of the overlapping dataset, and also showed the limitations of our method.

5.1 Experiment Setup

First, we briefly describe the datasets. The first dataset is collected from Criteo. It contains a label indicating whether the advertisement is clicked by the user and features of the advertisements displayed in several days. The second dataset is collected from Taobao, it records whether the product displayed in the webpage is clicked by the user, together with the feature of products and users.

Note that these two datasets have great potential in VFL applications because the records of advertisements are naturally distributed among multiple websites and multiple companies. These distributed datasets may contain different features and cannot be directly shared because of privacy restrictions. For example, Taobao dataset consists of three data sheets: the feature of the products, the feature of users, and the click-through records. With the restriction of privacy, one can get the click-through dataset and the features of product but cannot get the dataset of users. In this case VFL is needed.

Then we give a short introduction to the experimental platform. We construct and conduct experiments on FATE [35] which supports VFL on two parties. After specifying the dataset, FATE will perform private intersection and VFL. We built our solution as an plugin for VFL model in FATE. As we have defined in Equation (3), when λ equals zero, the model is equivalent to the local model.

In our experiment, the training set is vertically partitioned into two datasets: the click-through dataset and the user dataset. A logistic regression model is trained on these two datasets vertically. We also consider different portions of data samples overlapping between two parties.

Table 3. Student Model's AUC under $\lambda = 0.5$ and Different Portions of Overlapping Samples

α	0.1	0.25	0.5	0.75	0.9
Criteo local model	0.6642	0.6642	0.6642	0.6642	0.6642
Criteo student model	**0.7234**	**0.7304**	**0.7341**	**0.7351**	**0.7349**
Criteo VFL model	0.7289	0.7476	0.7534	0.7536	0.7603
Taobao local model	0.5109	0.5109	0.5109	0.5109	0.5109
Taobao student model	**0.5104**	**0.5136**	**0.5274**	**0.5448**	**0.5439**
Taobao VFL model	0.5744	0.5965	0.6308	0.6405	0.6509

The bolded entries in the table show the performance of student model which surpasses local model in most cases.

To implement our solution with the data distribution illustrated in Figure 6, first the dataset is loaded into FATE data table on two parties, respectively. Then the intersection dataset is calculated by FATE with secure intersection [19]. The intersection dataset is then fed into VFL training module. A student model will then be trained with the active party's local dataset and the supervision of VFL model based on target function 3. We have tried different values of λ and also considered different portions of overlapping samples in VFL training.

Note that during the training of the student model, we generate batches of training data where half of the data samples are from the overlapping dataset with soft labels, and the other half of data samples do not have soft labels. This ensures that we conduct an equal number of iterations of training over different types of data samples.

5.2 Student Model Performance

We evaluate the performance of models with AUC (area under the ROC curve) which is widely used in the classification problem. The closer AUC is near to 1, the better the model's performance is.

Denote the portion of overlapping samples in the training set as α. First, we evaluate and compare the AUC of the VFL model, local model, and non-federated model (local model). The VFL model is trained collaboratively with two parties' datasets; the local model is trained on local dataset only; the student model is trained on local dataset together with the soft labels. Then the experiments with combination of several different values of λ in Equation (3) and different values α are conducted to measure the effect of supervision by the VFL model. We trained a Logistic Regression model on the dataset to predict the "clicked" label. As shown in Table 3, the student model's performance surpassed the local model's performance.

From the Table 3, we observed that our method performs good on Criteo dataset but gives less improvement on Taobao dataset. This is mainly because the click-through dataset in Taobao has few features relevent to the labels. This reminds us that the supervision of the VFL model has no guarantee in performance because the features in local dataset still dominate the model's performance. In this case, we still need to conduct collaborative inference.

The choice of λ depends on the distribution of features. In our experiment, we divide features with equal possibility so that the features in two parties are almost equally important. In Table 3, we set $\lambda = 0.5$ so that the loss from real labels and loss from VFL models are equally important. In practice, the choice of λ should be larger if there is more important features in the other party's dataset, which can be decided by the improvement of AUC in VFL model compared to local model.

In Figure 10, the Criteo student model's AUC with different α and λ are shown. We observed that:

— The AUC of the student model increases with the portion of overlapping samples.
— The AUC of the student model increases as λ increases, which is the weight of the loss with soft label marked by the VFL model.

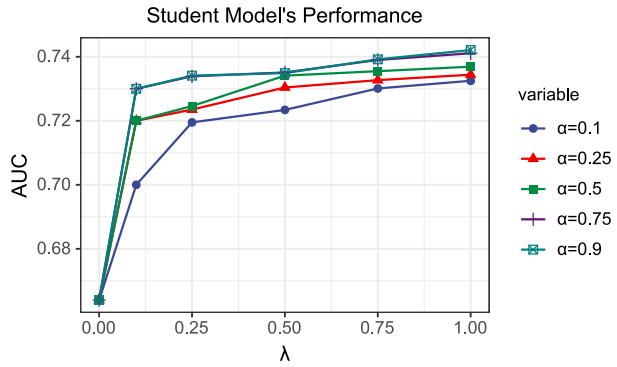

Fig. 10. Criteo student model's AUC under different overlapping portions and λ.

Table 4. Comparison of the Cost between Oblivious Inference and Naive OT

	OT Preparation		Online Phase		
Method	Comp.	Comm.	Comp.	Comm.	Rounds
$\binom{MN}{1} - OT$	–	–	$2\lceil logMN \rceil + MN$	$2l\lceil logMN \rceil + lMN$	2
$\binom{N}{1} - OT$	–	–	$2\lceil logN \rceil + N$	$2l\lceil logN \rceil + lN$	2
Oblivious Inference	$N^2M\lceil logN \rceil$	$lN\lceil logN \rceil$	$\lceil logN \rceil$	$N * l$	1

The AUC of the student model increases as the portion of overlapping samples increases. This is because, on the one hand, the more overlapping data, the more soft labels produced by the VFL model to supervise the student model's training. On the other hand, when the overlapping sample increases, the VFL model also gets more training data, thus gives better performance. Although the AUC of student model is less than the AUC of VFL model, the improvement is verified.

In the figure, it can also be concluded that AUC also increases as λ increases. This can be explained by how the soft labels influence the training of the student model. The soft labels produced by VFL models are the probabilities that the samples are predicted to be 1 (clicked). So compared to real labels, which take only 0 and 1, the soft labels give continuous target to fit the model. Moreover, when λ reaches 1, the only targets are soft labels. Note that when $\lambda = 0$, the model is trained without soft labels, and it is equal to training on the local dataset. However, this does not necessary mean that increasing λ leads to a better student model, since soft labels are not always the same as real labels.

5.3 Oblivious Inference Cost

To compare the cost of our solution to the cost of naive $\binom{1}{n} - OT$, we give the following table to show the optimization, assuming that the OT is implemented by Bellare-Micali OT [5]. See Table 4 for statistics.

In the table, we define the number of the bucket as M, and every bucket has N elements. l is the length of the ciphertext. We compare our method with two other implementations: the $\binom{MN}{1} - OT$ and $\binom{N}{1} - OT$ solution with bucket_id in plaintext.

In the naive $\binom{MN}{1} - OT$ solution, the choice of active party is completely hidden to the passive party. So to privately transfer the inference result, the passive party conducts $\binom{MN}{1} - OT$ with the active party where the active party holds the data ID and the passive party holds $M \times N$ inference

results. This method has the strongest privacy for data ID but needs to transfer tremendous amounts of data in every inference task. The $\binom{MN}{1} = OT$ is implemented by performing $\binom{2}{1} = OT$ for $\lceil logN \rceil$ times. Note that these OT operations can be executed in parallel, so only two rounds are counted (OT and sending ciphertext).

On the other hand, if we chose to expose bucket_id to the passive party and conduct a $\binom{N}{1} - OT$ which is composed of $\lceil logN \rceil$ times of $\binom{2}{1} - OT$, we will get much fewer data to transfer during the online stage. But still, we need to conduct OT during online inference. For one OT operation, based on the implementation of Bellare-Micali, the cost of communication is dependent on the number of ciphertexts, which is the total length of keys plus the length of encrypted buckets. Since, we have N elements in each bucket, the total cost of communication is $2l\lceil logN \rceil + lN$.

In our solution, based on the observation that the keys generation, encryption, and OT operations on keys can be done before the inference begins, those operations are moved to OT Preparation. Though we need to compute and store N times more ciphertexts than $\binom{n}{1} - OT$ solution, the computation and communication cost of the online phase is greatly reduced. Note that the data ID is still unknown to the passive party in this solution, but the passive party knows whether the data IDs in different inference tasks are the same based on the index of the requested encrypted bucket. The trade-off between privacy and efficiency is worthy given the acceleration.

6 RELATED WORK

6.1 Feature Distributed Machine Learning

Feature Distributed Machine Learning (**FDML**) defines the training of a machine learning model on a dataset whose features are distributed among many clients. FDML is a special case of distributed machine learning. In previous studies of FDML, the messages between parties are plaintext, so privacy is at risk during training.

Compared to data-parallel distributed machine learning, which aggregates the gradients in the parameter server during training, FDML tries to learn embedding on the client, and with the embedding transmitted to the central server [15], a global model is trained on the central server with the embedding as the input. The main difference between FDML and VFL lies in privacy protection. In VFL, the dataset is private for the clients, so the messages exchanged during training shall not expose private information to other parties. While in FDML, although the parties are not allowed to centralize their dataset to a single server, they still know the data ID of other parties, and the embedding is not encrypted. Generally, VFL can be regarded as a FDML with more strict privacy restrictions.

Other studies of VFL also focus on the architecture design during training. VAFL [7] discussed the asynchronous pattern of training in VFL. VAFL adapted the parameter server architecture for VFL training, and the local model of every party is updated multiple times before it is pushed to the parameter server for asynchronous updates. Different from our work which focuses on availability and privacy, VAFL focuses on a new training paradigm which reduces the communication cost during training.

6.2 Federated Transfer Learning

FTL defines the problem of privately training a model with a relatively small dataset or a small number of labels in the target [21, 30]. Basically, FTL is the transfer learning in federated setting [29]. FTL trains the model by transferring the knowledge from a different but related model to the target model. The performance of the model trained in FTL is closely related to how relevant the two domains are. The studies of FTL include the FTL framework, and FTL applications in healthcare, finances [30], and so on.

Both FTL and our framework focus on training a machine learning model collaboratively with few intersection labels. The main difference between FTL and our framework is that FTL transfers the knowledge from the source domain to the target domain and the two domains reside on different parties, while our framework focuses on transferring knowledge from VFL model to the active party's local model. And our experiments have shown that transferring from VFL model is more effective to improve the performance of the active party's local model.

6.3 Privileged Features Distillation

Our solution is mainly inspired by knowledge distillation and privileged feature distillation in machine learning. Knowledge distillation is known as a solution to transfer the knowledge from a large model (called teacher model) to a smaller one (called student model). Generally, this technique works by supervising the learning process of the student model by the teacher model.

Also, there have been studies of privileged information that defines the information that is available only in the training stage but unavailable in the test or inference stage. Reference [33] gives the paradigm for learning with privileged information. Moreover, some studies give new paradigms for generating new features or information of the sample, such as [32]. Similarly, some studies of privileged features involve the distillation from the model trained with privileged features to the model without privileged features [36].

6.4 OT in Machine Learning

OT allows the sender to transmit one of its elements to the receiver so that the sender remains oblivious about the choice of receiver and the receiver learns just the element it chose. OT has been an important tool for constructing secure protocols, and it is very useful for constructing privacy-preserving machine learning protocols.

Many studies focus on improving the efficiency of OT protocols, and our solution is also inspired by OT with precomputation [3]. OT extension [2, 4, 16] is proposed, which performs base OTs with a constant number of rounds so that the cost in computation and transmission is reduced. Moreover, some special conditions in OT are also exploited for optimization like **correlated OT** (**C-OT**) [2] and **random OT** (**R-OT**) [26].

OT has been used in the construction of privacy-preserving machine learning protocols. ABY [10] and ABY3 [24] supports training machine learning models on arbitrarily partitioned datasets using secret sharing and OT.

7 SUMMARY

Current VFL models cannot guarantee the success of inference in the online stage because some parties' datasets may not have the data instance with specified data ID. Also, sending the data ID to all parties may violate privacy constraints. Our work proposes a new training and inference algorithm in VFL that improves availability and privacy. This idea is inspired by knowledge distillation from a large model to a smaller model in traditional machine learning. Unlike knowledge distillation in machine learning which relies on the student model's inference, the inference is first conducted on the VFL model (teacher model), and when it fails because of an absence of data ID in some party, the inference will be carried out by the student model. Moreover, we protect the information about data ID during inference by performing OT across multiple parties. We also improve the efficiency by constructing a hash table for queries and moving the encryption operation to the preparation stage.

Our experiments on the Crieto CTR dataset show that the knowledge transfer from the federated model to the student model will improve the performance on testing datasets. This is because the federated model got more knowledge during training with extra features from other parties. With

the federated model's supervision, the local model tends to give inferences close to the VFL model and thus performs better than training on local datasets only.

8 DISCUSSION AND FUTURE WORK

8.1 Additional Overhead

Our framework introduced an extra step to ensure the success of inference which may lead to additional latency in inference task. However, the overhead can be relieved by paralleling the inference on VFL model and student model. In practice, the inference on VFL model and student model can be conducted simultaneously. Since conducting inference on VFL model gives more latency because of extra communication and computation, the total inference latency is almost equal to conducting inference on VFL model only.

8.2 Other VFL Solutions

In this article, we build our solution based on Paillier homomorphic encryption, which supports addition operation on ciphertexts. But there are other protocols that can be applied in federated learning, such as secure MPC, **fully homomorphic encryption** (**FHE**), and **Trusted Execution Environment** (**TEE**). The main difference between these solutions lies in cryptographic tools and security definition. Here, we illustrate that our solution is suitable for all existing VFL protocols.

First, our framework does not relies on certain cryptographic tools. The only feature we used in the protocol is additive homomorphic, which is also supported by other VFL solutions. MPC supports addition on ciphertexts with arithmetic secret sharing; TEE supports addition on secret values with hardware enclave; and FHE supports addition on ciphertexts. Our framework can be implemented on all these backends since they all support addition on secret values.

Second, the only assumptions of our framework is the feature distribution of dataset. We assume that the active party has at least one feature for conducting inference locally. This assumption will not break the framework if we change the cryptographic backend to MPC, FHE, or TEE.

REFERENCES

[1] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. 2015. A Guide to Fully Homomorphic Encryption. Cryptology ePrint Archive Report 2015/1192. https://eprint.iacr.org/2015/1192.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, New York, NY, 535–548. DOI: https://doi.org/10.1145/2508859.2516738

[3] Donald Beaver. 1995. Precomputing oblivious transfer. In *Proceedings of the Advances in Cryptology*, Don Coppersmith (Ed.). Springer Berlin, Berlin, 97–109.

[4] Donald Beaver. 1996. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, New York, NY, 479–488. DOI: https://doi.org/10.1145/237814.237996

[5] Mihir Bellare and Silvio Micali. 1990. Non-interactive oblivious transfer and applications. In *Proceedings of the Advances in Cryptology*, Gilles Brassard (Ed.). Springer New York, New York, 547–557.

[6] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. 2021. Secure federated matrix factorization. *IEEE Intelligent Systems* 36, 5 (2021), 11–20. DOI: https://doi.org/10.1109/MIS.2020.3014880

[7] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. 2020. VAFL: a Method of Vertical Asynchronous Federated Learning. arXiv:2007.06081. Retrieved from https://arxiv.org/abs/2007.06081.

[8] Criteo Challenge. 2014. Criteo Display Advertising Challenge. https://www.kaggle.com/c/criteo-display-ad-challenge/data. Access on 20 Feb. 2021.

[9] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical covertly secure MPC for dishonest majority – or: Breaking the SPDZ limits. In *Proceedings of the Computer Security*, Jason Crampton, Sushil Jajodia, and Keith Mayes (Eds.). Springer Berlin, Berlin, 1–18.

[10] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation. In *Proceedings of the NDSS*.

[11] Dashan Gao, Yang Liu, Anbu Huang, Ce Ju, Han Yu, and Qiang Yang. 2019. Privacy-preserving heterogeneous federated transfer learning. In *Proceedings of the 2019 IEEE International Conference on Big Data*. 2552–2559. DOI: https://doi.org/10.1109/BigData47090.2019.9005992

[12] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, New York, NY, 169–178. DOI: https://doi.org/10.1145/1536414.1536440

[13] Chun Sheng Gu. 2015. Fully homomorphic encryption from approximate ideal lattices. *Ruan Jian Xue Bao/Journal of Software* 26, 10 (2015), 2696–2719. DOI: https://doi.org/10.13328/j.cnki.jos.004808

[14] Andrew Hard, Chloé M. Kiddon, Daniel Ramage, Francoise Beaufays, Hubert Eichner, Kanishka Rao, Rajiv Mathews, and Sean Augenstein. 2018. Federated Learning for Mobile Keyboard Prediction. https://doi.org/10.48550/ARXIV.1811.03604

[15] Yaochen Hu, Di Niu, Jianming Yang, and Shengping Zhou. 2018. FDML: A collaborative machine learning framework for distributed features. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), 2232–2240.

[16] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *Proceedings of the Advances in Cryptology*, Dan Boneh (Ed.). Springer Berlin, Berlin, 145–161.

[17] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, and D'Oliveira RG. 2019. Advances and open problems in federated learning. arXiv:1912.04977. Retrieved from https://arxiv.org/abs/1912.04977.

[18] Vladimir Kolesnikov and Ranjit Kumaresan. 2013. Improved ot extension for transferring short secrets. In *Proceedings of the Advances in Cryptology*, Ran Canetti and Juan A. Garay (Eds.). Springer Berlin, Berlin, 54–70.

[19] Gang Liang and Sudarshan S. Chawathe. 2004. Privacy-preserving inter-database operations. In *Proceedings of the Intelligence and Security Informatics*, Hsinchun Chen, Reagan Moore, Daniel D. Zeng, and John Leavitt (Eds.). Springer Berlin, Berlin, 66–82.

[20] Yang Liu, Yan Kang, Xin wei Zhang, Liping Li, Yong Cheng, Tianjian Chen, M. Hong, and Q. Yang. 2019. A communication efficient collaborative learning framework for distributed features. https://doi.org/10.48550/ARXIV.1912.11187

[21] Sudipan Saha and Tahir Ahmad. 2020. Federated Transfer Learning: concept and applications. https://doi.org/10.48550/ARXIV.2010.15561

[22] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. 2020. A secure federated transfer learning framework. *IEEE Intelligent Systems* 35, 4 (2020), 70–82. DOI: https://doi.org/10.1109/MIS.2020.2988525

[23] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Aarti Singh and Jerry Zhu (Eds.). PMLR, Fort Lauderdale, FL, 1273–1282. Retrieved from http://proceedings.mlr.press/v54/mcmahan17a.html.

[24] Payman Mohassel and Peter Rindal. 2018. ABY^3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, New York, NY, 35–52. DOI: https://doi.org/10.1145/3243734.3243760

[25] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy*, 19–38. DOI: https://doi.org/10.1109/SP.2017.12

[26] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. 2012. A new approach to practical active-secure two-party computation. In *Proceedings of the Advances in Cryptology*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer Berlin, Berlin, 681–700.

[27] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the Advances in Cryptology*, Jacques Stern (Ed.). Springer Berlin, Berlin, 223–238.

[28] Pascal Paillier and David Pointcheval. 1999. Efficient public-key cryptosystems provably secure against active adversaries. In *Proceedings of the Advances in Cryptology*, Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing (Eds.). Springer Berlin, Berlin, 165–179.

[29] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2009), 1345–1359.

[30] Shreya Sharma, Chaoping Xing, Yang Liu, and Yan Kang. 2019. Secure and efficient federated transfer learning. In *Proceedings of the 2019 IEEE International Conference on Big Data*. IEEE, 2569–2576.

[31] Taobao. 2018. Taobao Display/Click Dataset. https://tianchi.aliyun.com/dataset/dataDetail?dataId=56, Access on 23 Feb. 2021.

[32] Vladimir Vapnik and Rauf Izmailov. 2015. Learning using privileged information: Similarity control and knowledge transfer. *Journal of Machine Learning Research* 16, 1 (2015), 2023–2049.

[33] Vladimir Vapnik and Akshay Vashist. 2009. A new learning paradigm: Learning using privileged information. *Neural Networks* 22, 5 (2009), 544–557. DOI: https://doi.org/10.1016/j.neunet.2009.06.042

[34] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10, 3152676 (2017), 10–55.

[35] Webank. 2019. FATE: An Industrial Grade Federated Learning Framework. Retrieved from https://github.com/FederatedAI/FATE, Access on 20 Feb. 2021.

[36] Chen Xu, Quan Li, Junfeng Ge, Jinyang Gao, Xiaoyong Yang, Changhua Pei, Fei Sun, Jian Wu, Hanxiao Sun, and Wenwu Ou. 2020. Privileged features distillation at taobao recommendations. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, 2590–2598. DOI: https://doi.org/10.1145/3394486.3403309

[37] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. arXiv 10, 2 (2019), 1–19.

[38] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. 2021. TACC: A Full-stack Cloud Computing Infrastructure for Machine Learning Tasks. https://doi.org/10.48550/arXiv.2110.01556

Received April 2021; revised August 2021; accepted November 2021