FedTree

Xtra Computing

Jun 11, 2023

CONTENTS:

1 Installation 3

2 Quick Start 7

3 APIs/Parameters 9

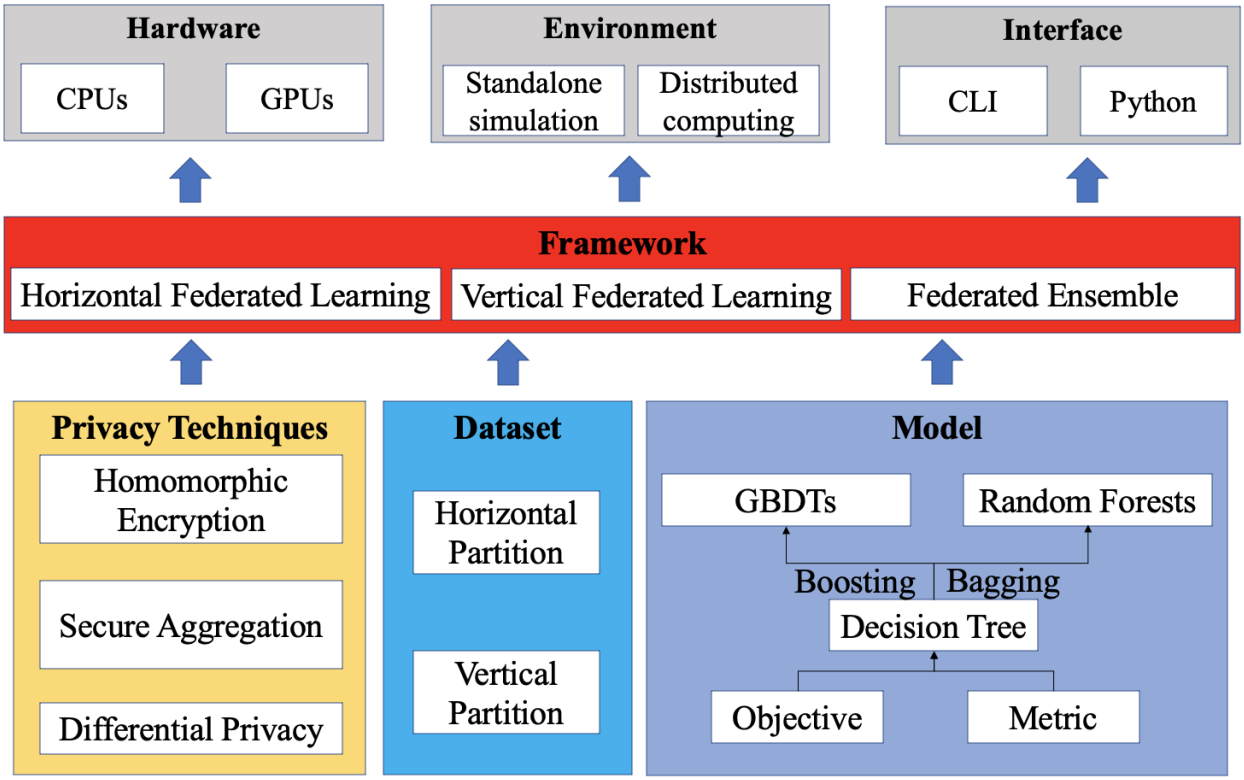
4 Examples 13

5 Frameworks 17

6 Experiments 23

FedTree is a federated learning system for tree-based models. It is designed to be highly **efficient**, **effective**, and **secure**. It has the following features.

- Parallel computing on multi-core CPUs and GPUs.
- Stand-alone simulation and distributed learning.
- Support of homomorphic encryption, secure aggregation, and differential privacy.
- Federated training algorithms of gradient boosting decision trees and random forests.



CHAPTER

ONE

INSTALLATION

Here is the guide for the installation of FedTree.

Contents

- *Prerequisites*
- *Install Fedtree*
- *Build on Linux*
- *Build on MacOS*

1.1 Prerequisites

- CMake 3.15 or above
- GMP library
- NTL library
- gRPC 1.50.0 (required for distributed version)

You can follow the following commands to install NTL library.

```
wget https://libntl.org/ntl-11.5.1.tar.gz
tar -xvf ntl-11.5.1.tar.gz
cd ntl-11.5.1/src
./configure SHARED=on
make
make check
sudo make install
```

If you install the NTL library at another location, please pass the location to the *NTL_PATH* when building the library (e.g., *cmake .. -DNTL_PATH="PATH_TO_NTL"*).

For gRPC, please remember to add the local bin folder to your path variable after installation, e.g.,

```
export PATH="$gRPC_INSTALL_DIR/bin:$PATH"
```

We suggest you install gPRC 1.50.0, i.e., using *-b v1.50.0* when cloning gRPC repo.

If your gRPC version is not 1.50.0, you need to go to *src/FedTree/grpc* directory and run

```
protoc -I ./ --grpc_out=. --plugin=protoc-gen-grpc=`which grpc_cpp_plugin` ./
↪fedtree.proto
protoc -I ./ --cpp_out=. ./fedtree.proto
```

1.2 Clone Install submodules

Run the following commands:

```
git clone https://github.com/Xtra-Computing/FedTree
cd FedTree
git submodule init
git submodule update
```

1.3 Build on Linux (Recommended)

Run the following commands:

```
# under the directory of FedTree
mkdir build && cd build
cmake ..
make -j
```

1.4 Build on MacOS

On MacOS, you can use Apple Clang to build FedTree.

1.4.1 Build with Apple Clang

Install *libomp* if you haven't:

```
brew install libomp
```

Run the following commands:

```
mkdir build
cd build
cmake -DOpenMP_C_FLAGS="-Xpreprocessor -fopenmp -I/usr/local/opt/libomp/include
↪" \
  -DOpenMP_C_LIB_NAMES=omp \
  -DOpenMP_CXX_FLAGS="-Xpreprocessor -fopenmp -I/usr/local/opt/libomp/include"␣
↪\
  -DOpenMP_CXX_LIB_NAMES=omp \
  -DOpenMP_omp_LIBRARY=/usr/local/opt/libomp/lib/libomp.dylib \
  ..
make -j
```

1.5 Building Options

There are the following building options passing with cmake.

- `USE_CUDA` [default = `OFF`]: Whether using GPU to accelerate homomorphic encryption or not. It is available only when setting `DISTRIBUTED` to `OFF`.
- `DISTRIBUTED` [default = `ON`]: Whether building distributed version of FedTree or not.
- `NTL_PATH` [default = `/usr/local`]: The PATH to the NTL library.

For example, if you want to build a version with GPU acceleration, distributed version with NTL library under /home/NTL directory, you can use the following command.

```
cmake .. -DUSE_CUDA=ON -DDISTRIBUTED=ON -DNTL_PATH="/home/NTL"
make -j
```

CHAPTER

TWO

QUICK START

Here we present an example to simulate vertical federated learning with FedTree to help you understand the procedure of using FedTree.

2.1 Prepare a dataset / datasets

You can either prepare a global dataset to simulate the federated setting by partitioning in FedTree or prepare a local dataset for each party.

For the data format, FedTree supports svmlight/libsvm format (each row is an instance with `label feature_id1:feature_value1 feature_id2:feature_value2 ...`) and csv format (the first row is the header `id,label,feature_id1,feature_id2,...` and the other rows are the corresponding values). See here for an example of libsvm format dataset and here for an example of csv format dataset.

For classification task, please ensure that the labels of the dataset are organized as `0 1 2 ...` (e.g., use labels 0 and 1 for binary classification).

2.2 Configure the Parameters

You can set the parameters in a file, e.g., `machine.conf` under `dataset` subdirectory. For example, we can set the following example parameters to run vertical federated learning using homomorphic encryption to protect the communicated message. For more details about the parameters, please refer to here.

```
data=./dataset/test_dataset.txt
test_data=./dataset/test_dataset.txt
partition_mode=vertical
n_parties=4
mode=vertical
privacy_tech=he
n_trees=40
depth=6
learning_rate=0.2
```

2.3 Run FedTree

After you install FedTree, you can simply run the following commands under `FedTree` directory to simulate vertical federated learning in a single machine.

```
./build/bin/FedTree-train ./examples/vertical_example.conf
./build/bin/FedTree-predict ./examples/prediction.conf
```

CHAPTER

THREE

APIS/PARAMETERS

We provide two kinds of APIs: command-line interface (CLI) and Python interface. For CLI, users only need to prepare a configuration file specifying the parameters and call FedTree in a one-line command. For Python interface, users can define two classes *FLClassifier* and *FLRegressor* with the parameters and use them in a scikit-learn style (see here). The parameters are below.

Contents

- *Parameters for Federated Setting*
- *Parameters for GBDTs*
- *Parameters for Privacy Protection*

3.1 Parameters for Federated Setting

- **mode [default = horizontal, type=string]**
  - horizontal: horizontal federated learning
  - vertical: vertical federated learning
- **num_parties [default = 10, type = int, alias: num_clients, num_devices]**
  - Number of parties
- **partition [default = 0, type = bool]**
  - 0: each party has a prepared local dataset
  - 1: there is a global dataset and users require FedTree to partition it to multiple subsets to simulate federated setting.
- **partition_mode [default=``horizontal``, type=string]**
  - horizontal: horizontal data partitioning
  - vertical: vertical data partitioning
- **ip_address [default=``localhost``, type=string, alias: server_ip_address]**
  - The ip address of the server in distributed FedTree.
- **data_format [default=``libsvm``, type=string]**
  - libsvm: the input data is in a libsvm format (label feature_id1:feature_value1 feature_id2:feature_value2). See here for an example.
  - csv: the input data is in a csv format (the first row is the header and the other rows are feature values). See here for an example.

- **n_features [default=-1, type=int]**
  - Number of features of the datasets. It needs to be specified when conducting horizontal FedTree with sparse datasets.
- **propose_split [default=``server``, type=string]**
  - server: the server proposes candidate split points according to the range of each feature in horizontal FedTree.
  - party: the parties propose possible split points. Then, the server merge them and sample at most num_max_bin candidate split points in horizontal FedTree.
- **key_length [default=512, type=int]**
  - Number of bits of the key used in encryption.
- **pred_output [default=``predictions.txt``, type=string]**
  - The file to save the predicted labels when using FedTree-predict

3.2 Parameters for GBDTs

- **data [default=``../dataset/test_dataset.txt``, type=string, alias: path]**
  - The path to the training dataset(s). In simulation, if multiple datasets need to be loaded where each dataset represents a party, specify the paths seperated with comma.
- **model_path [default=``fedtree.model``, type=string]**
  - The path to save/load the model.
- **verbose [default=1, type=int]**
  - Printing information: 0 for silence, 1 for key information and 2 for more information.
- depth [default=6, type=int]
  - The maximum depth of the decision trees. Shallow trees tend to have better generality, and deep trees are more likely to overfit the training data.
- n_trees [default=40, type=int]
  - The number of training iterations. n_trees equals to the number of trees in GBDTs.
- max_num_bin [default=32, type=int]
  - The maximum number of bins in a histogram. The value needs to be smaller than 256.
- learning_rate [default=1, type=float, alias: eta]
  - Valid domain: [0,1]. This option is to set the weight of newly trained tree. Use eta < 1 to mitigate overfitting.
- objective [default=``reg:linear``, type=string]
  - Valid options include reg:linear, reg:logistic, binary:logistic, multi:softprob, multi:softmax, rank:pairwise and rank:ndcg.
  - reg:linear is for regression, reg:logistic and binary:logistic are for binary classification.
  - multi:softprob and multi:softmax are for multi-class classification. multi:softprob outputs probability for each class, and multi:softmax outputs the label only.
  - rank:pairwise and rank:ndcg are for ranking problems.

- **num_class [default=1, type=int]**
  - Set the number of classes in the multi-class classification.
- min_child_weight [default=1, type=float]
  - The minimum sum of instance weight (measured by the second order derivative) needed in a child node.
- lambda [default=1, type=float, alias: lambda_tgbm or reg_lambda]
  - L2 regularization term on weights.
- gamma [default=1, type=float, alias: min_split_loss]
  - The minimum loss reduction required to make a further split on a leaf node of the tree. gamma is used in the pruning stage.

3.3 Parameters for Privacy Protection

- **privacy_method [default = none, type=string]**
  - none: no additional method is used to protect the communicated messages (raw data is not transferred).
  - he: use homomorphic encryption to protect the communicated messages (for vertical FedTree).
  - sa: use secure aggregation to protect the communicated messages (for horizontal FedTree).
  - dp: use differential privacy to protect the communicated messages (currently only works for vertical FL with single machine simulation).
- **privacy_budget [default=10, type=float]**
  - Total privacy budget if using differential privacy.

CHAPTER

FOUR

EXAMPLES

Here we present several additional examples of using FedTree.

4.1 Distributed Horizontal FedTree with Secure Aggregation

In the horizontal FedTree, the parties have their local datasets with the same feature space but different sample spaces. Also, in each machine, a configuration file needs to be prepared. We take UCI Adult dataset as an example (partitioned data provided in here).

In the server machine, the configuration file *server.conf* can be:

```
test_data=./dataset/adult/a9a_horizontal_test
n_parties=2
objective=binary:logistic
mode=horizontal
partition=0
privacy_tech=sa
learning_rate=0.1
max_depth=6
n_trees=50
```

In the above configuration file, it needs to specifies number of parties, objective function, mode, privacy techniques, and other parameters for the GBDT model. The *test_data* specifies the dataset for testing.

Supposing the ip address of the server is a.b.c.d, in the party machine 1, the configuration file *party1.conf* can be:

```
data=./dataset/adult/a9a_horizontal_p0
test_data=./dataset/adult/a9a_horizontal_test
model_path=p1.model
n_parties=2
objective=binary:logistic
mode=horizontal
partition=0
privacy_tech=sa
learning_rate=0.1
max_depth=6
n_trees=50
ip_address=a.b.c.d
```

The difference between *party1.conf* and *server.conf* is that *party1.conf* needs to specify the path to the local data and the ip address of the server. Similarly, we can have a configuration file for each party machine by changing the *data* (and *model_path* if needed). Then, we can run the following commands in the corresponding machines.

```
# under 'FedTree' directory
# under server machine
./build/bin/FedTree-distributed-server ./server.conf
# under party machine 1
./build/bin/FedTree-distributed-party ./party1.conf 0
# under party machine 2
./build/bin/FedTree-distributed-party ./party2.conf 1
......
```

In the above commands, the party machines need to add an additional input ID starting from 0 as its party ID.

4.2 Distributed Vertical FedTree with Homomorphic Encryption

In the vertical FedTree, the parties have their local datasets with the same sample space but different feature spaces. Moreover, at least one party has the labels of the samples. We need to specify one of the parties that has labels as the aggregator. Suppose party machine 1 is the aggregator. Then, we need to write a server configuration file *server.conf*, e.g.,

```
data=./dataset/adult/a9a_vertical_p0
test_data=./dataset/adult/a9a_vertical_test
n_parties=2
mode=vertical
partition=0
reorder_label=1
objective=binary:logistic
privacy_tech=he
learning_rate=0.1
max_depth=6
n_trees=50
```

For each party machine, supposing the ip address of the aggregator is a.b.c.d, we need to write a configuration file, e.g., *party1.conf* in party 1

```
data=./dataset/adult/a9a_vertical_p0
test_data=./dataset/adult/a9a_vertical_test
model_path=p1.model
n_parties=2
mode=vertical
partition=0
reorder_label=1
objective=binary:logistic
privacy_tech=he
learning_rate=0.1
max_depth=6
n_trees=50
ip_address=a.b.c.d
```

Then, we can run the following commands in the corresponding machines:

```
#under aggregator machine (i.e., party machine 1)
./build/bin/FedTree-distributed-server ./server.conf
#under party machine 1
```

(continues on next page)

(continued from previous page)

```
./build/bin/FedTree-distributed-party ./party1.conf 0
#under party machine 2
./build/bin/FedTree-distributed-party ./party2.conf 1
```

CHAPTER

FIVE

FRAMEWORKS

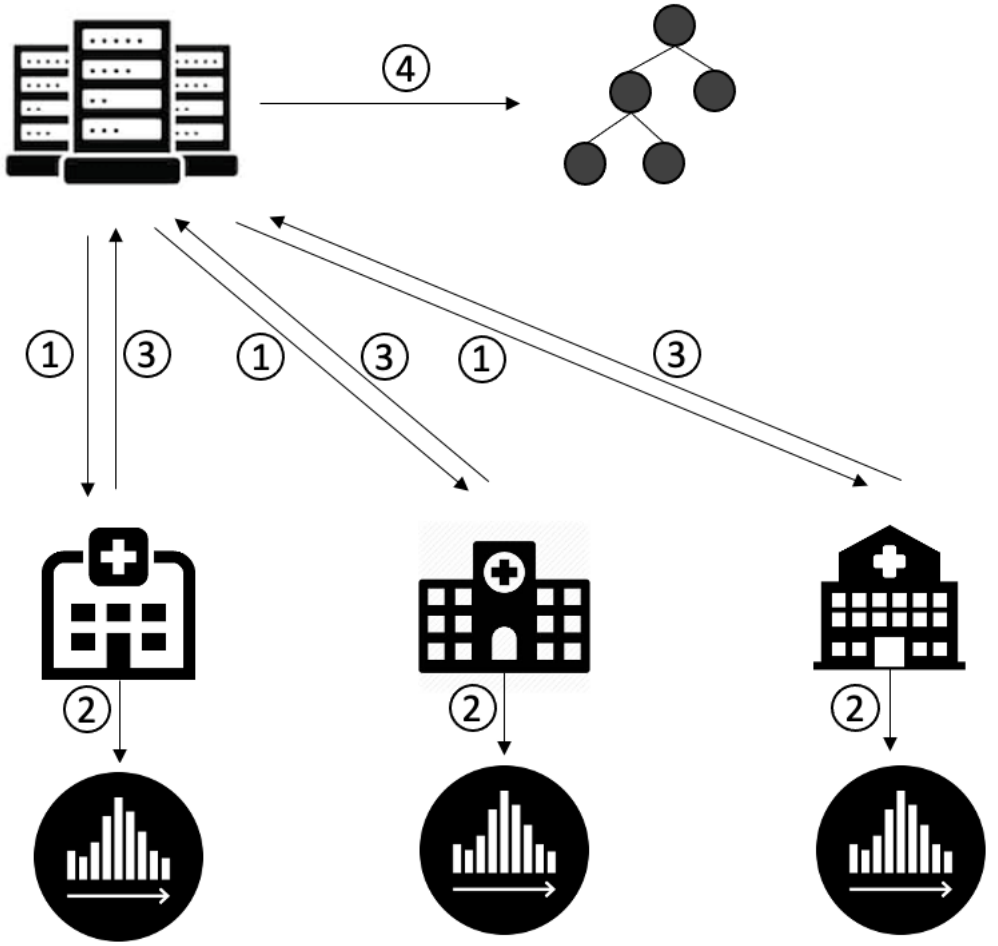
Here is an introduction of FedTree algorithms.

Contents

- *Horizontal Federated GBDTs*
- *Vertical Federated GBDTs*
- *Build on Linux*
- *Build on MacOS*

5.1 Horizontal Federated GBDTs

In the horizontal FedTree, the parties have their local datasets with the same feature space but different sample spaces. The framework of horizontal federated GBDTs training is shown below. There are four steps in each round.



1. The server sends the initialization parameters (#round = 1) or sends the new tree (#round > 1) to the parties.
2. The parties update the gradient histogram.
3. The parties send the gradient histogram to the server.
4. The server merges the histogram and boosts a new tree.

We repeat the above steps until reach the given number of trees.

We provide the option to adopt the secure aggregation method to protect the exchanged histograms. In the beginning of training, the clients and the server use Diffie-Hellman key exchange to share a secret key for each pair of clients. Then, before transfering the gradient histogram, each client generates random noises for each other client, encrypts the noises by the shared key of the corresponding client, and sends the encrypted noises to the server. Then, the server sends back the encrypted noises to the clients. The clients decrypts the noises with the shared keys. Then, the clients add the generated noises and subtract the decrypted noises to the local histogram. The injected noises of each client cancel each other out and the aggregates histogram remains unchanged.

The detailed algorithm is shown below.

Algorithm 1: FedTree-Hori with secure aggregation.

Input: Parties $P_1, ..., P_n$, number of trees T, tree depth d
Output: The final GBDT model f.

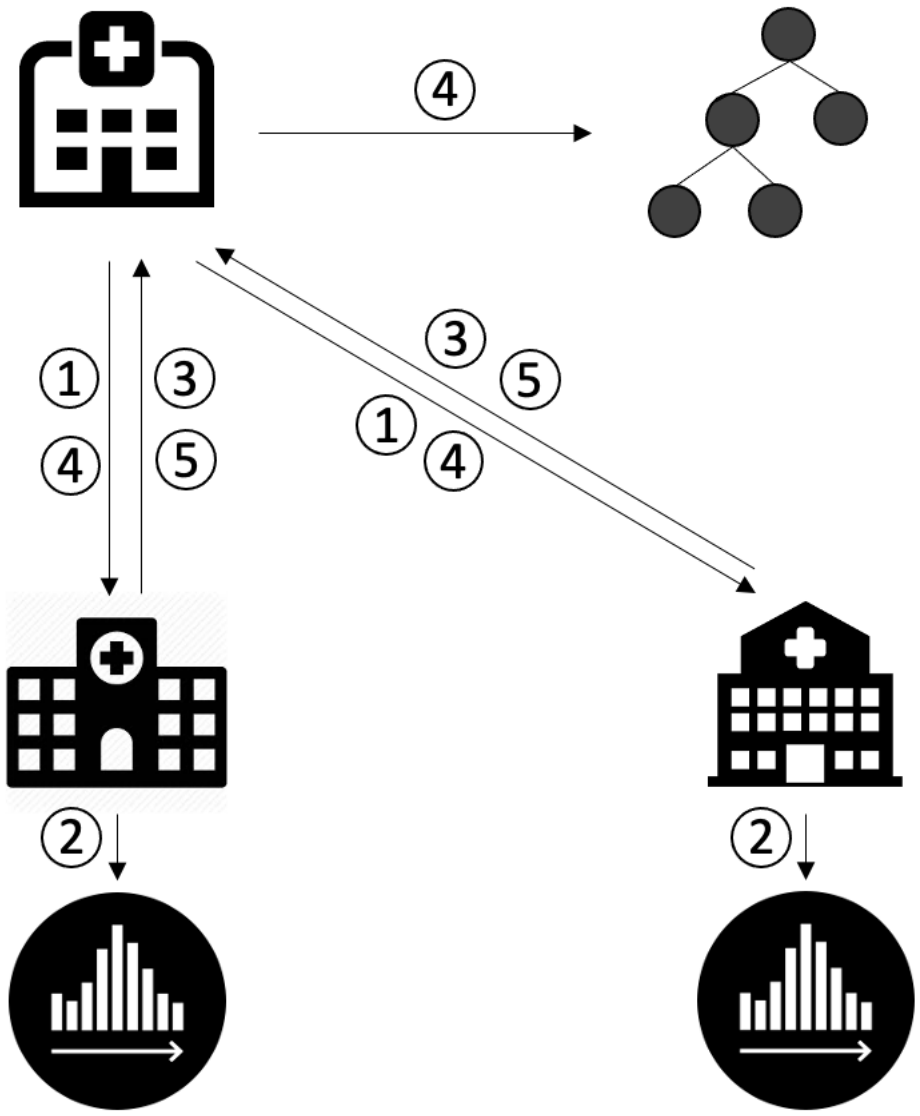
$S \leftarrow ProposeSplitCandidates()$
Each pair of party (P_i, P_j) shares a secret key K_{ij} using Diffie-Hellman key exchange
for $i = 1, ..., T$ **do**
    `/* Conduct on each party */`
    $g, h \leftarrow UpdateGradients()$
    for $j = 1, ..., d$ **do**
        for $m = 1, ..., n$ **do**
            `/* Conduct on party` P_m `*/`
            $G, H \leftarrow ComputeHistogram(g, h, S)$
            $[G], [H] \leftarrow G + \sum_z K_{z*} - \sum_z K_{*z}, H + \sum_z K_{z*} - \sum_z K_{*z}$
            Send $[G], [H]$ to server.
        `/* Conduct on server */`
        $G_{sum}, H_{sum} \leftarrow \sum[G], \sum[H]$
        Update depth j of tree f_i using the histograms G_{sum}, H_{sum}.
    `/* Conduct on server */`
    Send f_i to each party.

If adopting differential privacy, the server will train a differentially private tree in the fourth step using Laplace mechanism and exponential mechanism.

5.2 Vertical Federated GBDTs

In the vertical FedTree, the parties have their local datasets with the same sample space but different feature spaces. Moreover, at least one party has the labels of the samples. We specify one party that has the labels as the host party (i.e., aggregator).

The framework of vertical federated GBDTs training is shown below. There are four steps in each round.



1. The host party (i.e., the party with the labels) updates the gradients and sends the gradients to the other parties.

For each depth:

2. The parties computes the local gradient histograms.
3. The parties send their local histograms to the host party.
4. The host party aggregate the histograms, computes the best split point, and ask the corresponding party (including itself) to update the node.
5. The parties send back the nodes to the host party.

Here 2-4 steps are done for each depth of a tree until reaching the given maximum depth. The above steps are repeated until reaching the given number of trees. If homomorphic encryption is applied, the host party sends the encrypted gradients in the first step and decrypts the histogram in the fourth step.

We provide the option to adopt additive homomorphic encryption to protect the exchanged gradients. Specifically, the host party generates public and private keys before the training. Then, it uses the public key to encrypt the gradients before sending them. After receiving local histograms from the parties, the host party uses privacy key to decrypt the histograms before further computation.

The detailed algorithm is shown below.

Algorithm 2: FedTree-Verti with Homomorphic Encryption (HE)

Input: Parties $P_1, ..., P_n$, number of trees T, tree depth d. Assume P_1 has the labels.
Output: The final GBDT model f.

$S \leftarrow ProposeSplitCandidates()$
`/* Conduct on` P_1 `*/`
P_1 generates HE key pair (K_{pub}, K_{pri}).
P_1 sends public keys to the other parties.
for $i = 1, ..., T$ **do**
  `/* Conduct on party` P_1 `*/`
  $g, h \leftarrow UpdateGradients()$
  $[g], [h] \leftarrow Encrypt(g, K_{pub}), Encrypt(h, K_{pub})$
  Send $[g], [h]$ to the other parties.
  `/* Conduct on party` P_i `*/`
  for $j = 1, ..., d$ **do**
    for $k = 1, ..., n$ **do**
      `/* Conduct on each party` P_k `*/`
      $[G]_k, [H]_k \leftarrow ComputeHistogram([g], [h], S_k)$
      Send $[G]_k, [H]_k$ to P_1.
    `/* Conduct on` P_1 `*/`
    $G, H \leftarrow Decrypt([G]_*, K_{pri}), Decrypt([H]_*, K_{pri})$
    Update depth j of tree f_i using G, H with the parties that have the corresponding split features for each node

If differential privacy is applied, the host party updates the tree using Laplace mechanism and exponential mechanism.

CHAPTER

SIX

EXPERIMENTS

Here we present some preliminary experimental results. We use two UCI datasets, adult and abalone for experiments. The adult dataset is a classification dataset and the abalone is a regression dataset. We use FedTree-Hori to denote the horizontal FedTree and FedTree-Verti to denote the vertical FedTree.

Baselines: Homo-SecureBoost and Hetero-SecureBoost. Both approaches are from FATE.

6.1 Standalone Simulation

For the standalone simulation, we use a machine with 64*Intel Xeon Gold 6226R CPUs and 8*NVIDIA GeForce RTX 3090 to conduct experiments. We allocate each experiment with 16 threads. By default, we set the number of parties to 2, the number of trees to 50, learning rate to 0.1, the maximum depth of tree to 6, and the maximum number of bins to 255. The other parameters of all approaches are set to the default setting of FedTree.

6.1.1 Effectiveness

We first compare the accuracy of federated training and centralized training using XGBoost and ThunderGBM. The results are shown below. We report AUC for adult and RMSE for abalone. We can observe that the performance of FedTree is same as ThunderGBM. Also, SA and HE do not affect the model performance.

datasets	XG-Boost	Thun-derGBM	FedTree-Hori	FedTree-Hori+SA	FedTree-Verti	FedTree-Verti+HE	Homo-SecureBoost	Hetero-SecureBoost
a9a	0.914	0.914	0.914	0.914	0.914	0.914	0.912	0.914
abalone	1.53	1.57	1.57	1.57	1.56	1.57	1.56	0.001

6.1.2 Efficiency

We compare the efficiency of FedTree-Hori with Homo-SecureBoost of FATE. The results are shown below. We present the trainig time (s) per tree. Note that FedTree-Hori+SA achieves the same security guarantee as Homo-SecureBoost. The speedup is the computed by the improvement of FedTree-Hori+SA over Homo-SecureBoost, which is quite significant.

datasets	FedTree-Hori	FedTree-Hori+SA	Homo-SecureBoost	Speedup
a9a	0.09	0.098	8.76	89.4
abalone	0.11	0.19	7.7	40.5

We compare the efficiency of FedTree-Verti with Hetero-SecureBoost of FATE. We present the trainig time (s) per tree. Note that FedTree-Verti+HE achieves the same security guarantee as SecureBoost. The speedup is the improvement of

FedTree-Verti + HE (CPU) over FATE. FedTree is still much faster than SecureBoost. Moreover, FedTree can utilize GPU to accelerate the HE computation.

datasets	FedTree-Verti	FedTree-Verti+HE (CPU)	FedTree-Verti+HE (GPU)	Hetero-SecureBoost	Speedup
a9a	0.11	5.25	3.24	34.02	6.48
abalone	0.05	7.43	6.5	15.7	2.11

6.2 Distributed Computing

For distributed setting, we use a cluster with 5 machines, where each machine has two Intel Xeon E5-2680 14 core CPUs. We set the number of parties to 4, where each party hosts a machine. The results are shown below. Here Homo-SecureBoost (from FATE) and FedTree-Hori+SA have the same security level. We can observe that both horizontal and vertical FedTree are faster than FATE.

datasets	Homo-SecureBoost	FedTree-Hori + SA	Speedup	SecureBoost	FedTree-Verti+HE	Speedup
a9a	214.7	124.4	1.7	505.4	93.2	5.4
abalone	256.3	156.8	1.6	299.8	143.5	2.1