VF^{2}Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning

Fangcheng Fu, Yingxia Shao, Lele Yu, Jiawei Jiang, Huanran Xue, Yangyu Tao, Bin Cui
[1,3,7]Department of Computer Science & Key Lab of High Confidence Software Technologies (MOE), Peking University
[7]Institute of Computational Social Science, Peking University (Qingdao), China
[2]School of Computer Science, BUPT [4]ETH Zurich [1,5,6]Tencent Inc.
[1,3,7]{ccchengff,leleyu,bin.cui}@pku.edu.cn [2]shaoyx@bupt.edu.cn [4]jiawei.jiang@inf.ethz.ch
[1,5,6]{fangchengfu,huanranxue,brucetao}@tencent.com

ABSTRACT

With the ever-evolving concerns on privacy protection, vertical federated learning (FL), where participants own non-overlapping features for the same set of instances, is becoming a heated topic since it enables multiple enterprises to strengthen the machine learning models collaboratively with privacy guarantees. Nevertheless, to achieve privacy preservation, vertical FL algorithms involve complicated training routines and time-consuming cryptography operations, leading to slow training speed.

This paper explores the efficiency of the gradient boosting decision tree (GBDT) algorithm under the vertical FL setting. Specifically, we introduce VF^{2}Boost, a novel and efficient vertical federated GBDT system. Significant solutions are developed to tackle the major bottlenecks. First, to handle the deficiency caused by frequent mutual-waiting in federated training, we propose a concurrent training protocol to reduce the idle periods. Second, to speed up the cryptography operations, we analyze the characteristics of the algorithm and propose customized operations. Empirical results show that our system can be 12.8-18.9× faster than the existing vertical federated implementations and support much larger datasets.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**;
• **Computing methodologies** → **Machine learning algorithms**;
• **Security and privacy** → *Cryptography*.

KEYWORDS

Vertical Federated Learning; Gradient Boosting Decision Tree

ACM Reference Format:
Fangcheng Fu, Yingxia Shao, Lele Yu, Jiawei Jiang, Huanran Xue, Yangyu Tao, Bin Cui. 2021. VF^{2}Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3448016.3457241

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457241

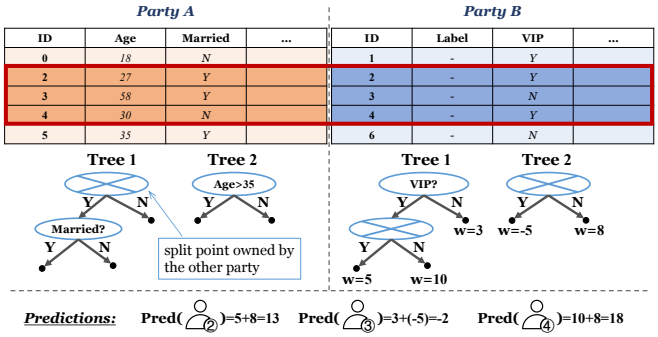

Figure 1: An illustration of vertical federated GBDT.

1 INTRODUCTION

Recent years have witnessed the unprecedented success of machine learning (ML) in the whole world. With the explosive surge of data volume, researchers and data scientists eagerly design more sophisticated ML models and make remarkable improvements on various applications [17, 35, 57, 81]. Meanwhile, many enterprises are ambitious to collect more data, such as users' pictures, messages, and daily routes, to strengthen the models. It brings a growing concern about the privacy of users at the same time. As such, the society has raised intensive attention to the protection of user data and the supervision on potential risks to information leakage. A compelling example is the establishment of the General Data Protection Regulation (GDPR) [77], which protects the individual data in a lawful way. Consequently, it is harder for enterprises to collect a large amount of data into a data center for ML training.

Motivated by this problem, federated learning (FL) [32, 44, 45, 49, 83] is attracting high attention in both the academic and industrial communities since it conveys a possibility to train ML models over physically distributed data with confidentiality guarantees. FL can be categorized into different areas according to the data layout. Specifically, vertical FL describes the case where data are vertically partitioned by features. As depicted in Figure 1, two parties (typically, two enterprises)[1] own two datasets that contain different columns (features) but have overlapping row indexes (instances). Only *Party B* owns the label information, which is not allowed to be

[1]In practice, there can be two or more parties that work as *Party A*, and the vertical federated GBDT algorithm can be easily generalized to more *Party A*'s. However, we only describe the case of only one *Party A* for simplicity, whilst conduct experiments for multi-party training in Section 6.

disclosed to *Party A*. Then the intersection can be viewed as a virtually joined dataset and provided as the input of vertical FL tasks to obtain a jointly trained federated model. Finally, the federated model is utilized to give predictions in real applications. Enterprises, especially those sweepingly affected by the privacy preservation regulations, have an arousing interest in this mode of FL.

(Motivating Scenario) One of our industrial partners, Tencent Inc., runs popular social applications with billions of users and is able to gather a rich set of user data. Many collaborators wish to enhance the performance of their ML tasks with the help of the additional data owned by Tencent. Especially, we consider the gradient boosting decision tree (GBDT) algorithm [25, 26].

GBDT is a popular algorithm in both data science competitions and industrial applications [10, 14, 19, 36, 43, 74]. Owing to its high accuracy and strong interpretability, previous works have studied how to migrate GBDT into the FL setting [15, 47, 48, 53]. Notably, Cheng et al. [15] propose a *lossless and privacy preserving* vertical federated GBDT algorithm, namely SecureBoost. As illustrated in Figure 1, the federated model makes use of the data in both parties to give more precise predictions. There are three essential steps to let *Party A* join the training process: (1) **Gradient statistics encryption and communication**: *Party B* encrypts the gradients and hessians calculated on training instances, and sends the ciphers to *Party A*. (2) **Histogram construction**: based on the encrypted gradient statistics, *Party A* builds gradient histograms, a core data structure in GBDT that summarizes the features, via the homomorphic addition technique [23, 66]. (3) **Histogram communication and decryption**: *Party B* receives the histograms from *Party A*, which are then decrypted and used for split finding of decision tree nodes. SecureBoost can achieve comparable model performance as the model obtained in the non-federated fashion on co-located datasets, and is analyzed to be secure since the data in both parties will not be leaked [15]. To this end, our industrial partner attempts to adopt the algorithm in the collaborative tasks.

Nevertheless, our industrial partner experienced extremely poor efficiency when deploying the vertical federated GBDT algorithm to collaborative tasks. First, the existing implementations require a sophisticated setup and manage the input (datasets) and output (models) in a highly encapsulated way, which is impractical for many productive applications. Second, since the existing algorithm protocol suffers from frequent mutual-waiting between the parties, the training process encounters a low efficiency. Worse still, it results in a huge waste of computational resources. Third, in order to achieve privacy preservation, the vertical federated GBDT involves time-consuming cryptography operations. It inevitably slows down the training speed if we naïvely migrate these expensive operations into the vertical federated GBDT.

(Summary of Contributions) In order to cope with these problems, we develop VF^{2}Boost, a novel and efficient vertical federated GBDT system. To boost the training performance, we analyze the major bottlenecks and propose specific optimization techniques. Our contributions are summarized below.

Proposal of VF^{2}Boost. We develop VF^{2}Boost, a brand new vertical federated GBDT system for efficient cross-enterprise collaboration. Inside each party, we implement the training routines in a scheduler-worker architecture that supports efficient distributed training. Between the parties, we set up message queues for scalable, robust, and confidential cross-party communication. Finally, we implement a high-performance library following the principles of Paillier homomorphic cryptosystem [61] to alleviate the burden of cryptography operations. VF^{2}Boost has been integrated into the production pipelines of Tencent Inc. and deployed in many real-world federated applications.

Concurrent Training Protocol. According to our analysis, in the existing training protocol for vertical federated GBDT, the key procedures are performed sequentially, making both parties suffer from long periods of idle time to wait for each other. Obviously, such a protocol results in a huge waste of computational resources. Consequently, we derive a new training protocol that allows both parties to run concurrently so that the idle time can be reduced. For the root node, a blaster-style encryption scheme is introduced to parallelize the encryption, public network communication, and histogram construction phases. For the subsequent layers of the decision tree, an optimistic node-splitting strategy is developed to overlap the decryption and histogram construction phases.

Customized Cryptography for GBDT. Although the Paillier cryptosystem provides a guarantee on data confidentiality, it is time-consuming and inevitably deteriorates the overall efficiency. To tackle this bottleneck, we carefully anatomize the essential cryptography operations and discover that we are able to optimize the performance by customizing them to adapt the characteristics of the GBDT algorithm. To be specific, we (1) accelerate the histogram construction by a re-ordered accumulation technique, which helps avoid tremendous cipher scaling operations; (2) propose a histogram packing method that packs several histogram bins (ciphers) into one in a lossless manner, by which we can boost the efficiency of histogram communication and decryption to a large extent.

System Evaluation. Comprehensive experiments are performed to verify the power of VF^{2}Boost. We first conduct an ablation study to assess the effectiveness of each aforementioned optimization individually. Then, end-to-end evaluation on extensive datasets shows that VF^{2}Boost can be 12.8-18.9× faster than existing implementations and is able to handle much larger datasets with tens of millions of training instances and tens of thousands of feature dimensionalities. To the best of our knowledge, this is the first vertical federated GBDT implementation that supports such a scale of datasets. Finally, we evaluate the scalability of VF^{2}Boost and scale the system to multi-party scenarios.

2 PRELIMINARIES

In this section, we briefly introduce the preliminary literature related to our work. We first present the notations used in this paper.

- N: number of instances (a.k.a. samples).
- D: number of features.
- d: average number of nonzero values of each instance.
- s: number of histogram bins for each feature.
- S: number of bits of the private key in Paillier cryptosystem.
- $[\![v]\!]$: the encrypted form (cipher) of v in Paillier cryptosystem.

2.1 The GBDT Algorithm

Gradient boosting decision tree [25, 26] is an ensemble algorithm that trains a series of decision trees as weak learners and enhances

the model ability with a boosting strategy [69, 71, 89]. Given a dataset $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$ where $\boldsymbol{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$ represent the features and label of the i-th instance, and a model with T decision trees $\{f_t(\boldsymbol{x})\}_{t=1}^T$. GBDT sums up the predictions of all trees, i.e., $\hat{y}_i = \hat{y}_i^{(T)} = \sum_{t=1}^T \eta f_t(\boldsymbol{x}_i)$, where η is the learning rate.

The essence of GBDT comes from how it boosts the decision trees sequentially. After $t-1$ trees are trained, the objection of the t-th tree is to minimize the following function:

$$F^{(t)} = \sum_{i=1}^N \mathcal{L}(y_i, \hat{y}_i^{(t)}) + \Phi(f_t) = \sum_{i=1}^N \mathcal{L}(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)) + \Phi(f_t).$$

$\mathcal{L}$ is a twice differentiable convex loss function that outputs the loss given prediction and label. For instance, a common choice for classification tasks is the logistic loss $\mathcal{L}(y, \hat{y}) = y \log(\delta(\hat{y})) + (1-y)\log(1-\delta(\hat{y}))$, where δ is the Sigmoid function. Φ is the regularized function to prevent the model from over-fitting. In this paper, we follow [14] to set $\Phi(f_t) = \gamma J_t + \lambda ||\omega_t||_2^2/2$, where ω_t is the weight vector of the J_t tree leaf predictions, and γ, λ are hyper-parameters to control the regularization. Friedman et al. [25] proposes to expand the objective function by

$$F^{(t)} \approx \sum_{i=1}^N \left[\mathcal{L}(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i)\right] + \Phi(f_t),$$

where $g_i = \partial_{\hat{y}_i^{(t-1)}} \mathcal{L}(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}_i^{(t-1)}} \mathcal{L}(y_i, \hat{y}_i^{(t-1)})$ are the gradient and hessian of the i-th instance after $t-1$ trees. Considering the j-th tree leaf, which owns a set of instances $I_j = \{i | \boldsymbol{x}_i \in \text{leaf}_j\}$, omitting the constant terms, we should minimize

$$\widetilde{F}^{(t)} = \sum_{j=1}^{J_t} \left[\left(\sum_{i \in I_j} g_i\right)\omega_j + \left(\sum_{i \in I_j} h_i + \lambda\right)\omega_j^2 \right] + \gamma J_t.$$

If the tree cannot be expanded (e.g., reaches maximum depth or there are no splits able to bring reduction in loss), we can obtain its optimal weight vector and minimal loss by

$$\omega_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \quad \widetilde{F}^{(t)*} = -\frac{1}{2}\sum_{j=1}^{J_t} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma J_t. \tag{1}$$

Equation 1 can be reckoned as the impurity function in the GBDT algorithm. In order to minimize the loss after the t-th tree, we iteratively find the best split (a split feature and a split value) for the tree leaves until the t-th tree finishes. We call a split is the best if it achieves the maximal split gain, which is defined as

$$SplitGain = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma,$$

where I_L and I_R are the sets of instances of the left and right child nodes, respectively.

How to fast and accurately locate the best split is vital to the efficiency of GBDT. Obviously, a brute force enumeration over all possible splits is impractical. In contrast, a common method is to adopt the histogram-based split finding method, which is illustrated in Figure 2. At initialization, several candidate splits are proposed for each feature (e.g., using the percentiles of each feature column [29, 33, 42]), which form s bins. For a target tree node, we enumerate the instances on it and accumulate the gradients and hessians into histograms according to their feature values. Based on the histograms, we can compute the gains of candidate splits efficiently, and select the one with the highest gain.

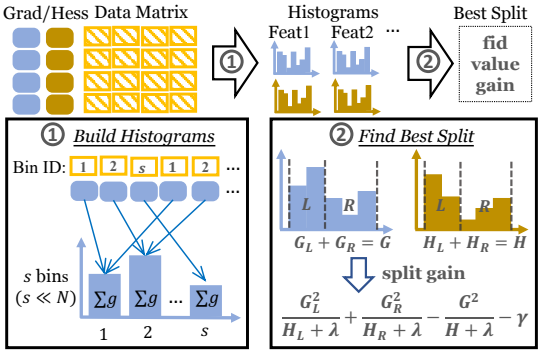

Figure 2: Illustration of histogram-based split finding.

To learn a decision tree, we recursively find the best split of one or more tree nodes (starting with the root node) and split them accordingly. Finally, when the tree stops growing (i.e., there are no more splits), the optimal weights of tree leaves are computed via Equation 1 and the predictions of all instances are updated.

2.2 Paillier Homomorphic Cryptosystem

Homomorphic Encryption (HE) [23, 66] describes the cryptographic methods that allow arithmetic computation in the space of ciphers. There are different kinds of HE, such as fully HE, additive HE, and multiplicative HE [6, 16, 21, 22, 30, 40, 61, 67, 68].

In this work, we focus on additive HE since GBDT requires the addition arithmetic for histogram construction. Specifically, we consider the Paillier cryptosystem [61], which is a well-known additive HE method and has been used in many FL algorithms [4, 5, 15, 34, 80, 84, 88, 90]. Paillier cryptosystem initializes with a S-bit modulus $n = pq$ where p, q are two big primes. (Accordingly, the ciphers are $2S$-bit long.) For instance, $S = 2048$ is reckoned to be safe as recommended by [9]. Given an integer V, we denote $[\![V]\!]$ as the corresponding cipher. Interested readers are referred to [61] for more details of the encryption and decryption phases. Paillier cryptosystem also supports the homomorphic addition (**HAdd**) and scalar multiplication (**SMul**) operations:

- (**HAdd**) $[\![U]\!] \oplus [\![V]\!] := ([\![U]\!] \times [\![V]\!]) \bmod n^2 = [\![U+V]\!]$;
- (**SMul**) $U \otimes [\![V]\!] := [\![V]\!]^U \bmod n^2 = [\![U \times V]\!]$.

Given a floating-point value v, we usually encode it into a tuple of integers $\langle e, V\rangle$ satisfying $V = \text{round}(v \times B^e) + \mathbb{1}(v < 0) \times n$, where e is the exponential term[2], V is the corresponding big integer representation, and B is a factor for encoding (e.g., $B = 16$). Intuitively, floating-point values are encoded into a much larger space with up to 2^S integers, and positive and negative values are separated in different ranges. Then V will be encrypted into a cipher $[\![V]\!]$. For simplicity, we denote $[\![v]\!] = \langle e, [\![V]\!]\rangle$, and denote $[\![u]\!] \oplus [\![v]\!], u \otimes [\![v]\!]$ as the **HAdd** and **SMul** operations of encrypted floating-point values. For instance, the **HAdd** involves a scaling operation if two ciphers are different in the exponential terms:

$$[\![u]\!] \oplus [\![v]\!] = \begin{cases} \langle e_v, (B^{e_v - e_u} \otimes [\![U]\!]) \oplus [\![V]\!]\rangle, & \text{if } e_u < e_v \\ \langle e_u, [\![U]\!] \oplus [\![V]\!]\rangle, & \text{if } e_u = e_v \\ \langle e_u, [\![U]\!] \oplus (B^{e_u - e_v} \otimes [\![V]\!])\rangle, & \text{if } e_u > e_v \end{cases}.$$

[2]The exponential term e can be non-deterministic in order to obfuscate the range of v.

2.3 The Vertical Federated GBDT Algorithm

Obviously, *Party A* can reveal the label information if gradients or hessians are exposed. For instance, the gradient of logistic loss is given as $g = \text{sigmoid}(\hat{y}) - y$. Since each gradient is positive when $y = 0$ and negative when $y = 1$, it will cause privacy leakage if *Party B* discloses the gradient statistics to *Party A*.

To this end, SecureBoost [15] proposes to leverage the additive HE property of Paillier cryptosystem. Specifically, *Party A* receives the encrypted gradient statistics $\{[\![g]\!], [\![h]\!]\}$ from *Party B* and constructs histograms $\{\sum[\![g]\!], \sum[\![h]\!]\}$, then *Party B* receives and deciphers encrypted histograms in order to search the best split in *Party A*. Cheng et al. [15] analyze the security of SecureBoost under the semi-honest scenario (all parties honestly follow the protocol but one or more parties try to learn knowledges of the others, a.k.a. the honest-but-curious scenario). First, the gradient statistics are encrypted before sent to *Party A*, which protects the labels. Second, *Party B* cannot recover the features from the histograms of *Party A*.

By means of additive homomorphism, the split gains can be computed in a lossless manner even though the gradient statistics are encrypted before histogram construction. Consequently, such a vertical federated GBDT algorithm is proven to achieve the same model accuracy as the non-federated training on co-located datasets, and maintains privacy preservation in the meantime.

2.4 Bottleneck Analysis and Our Solutions

Although the vertical federated GBDT algorithm conveys a possibility to jointly train a model with privacy guarantee, the overall performance is unsatisfactory owing to two critical bottlenecks.

Bottleneck 1. In the existing training protocol, *Party A* and *Party B* execute their key procedures sequentially, so both parties inevitably wait for each other idly in turns: (1) *Party A* sits idle when *Party B* performs encryption or decryption, leading to long idle periods in *Party A*; (2) since homomorphic addition can be hundreds of times slower than vanilla floating-point addition, *Party A* has to spend more time on histogram construction whilst *Party B* cannot proceed to next phase but stalls idly. Obviously, such a mutual-waiting results in a huge waste of resources and a low efficiency.

Bottleneck 2. In spite of the secure property of Paillier cryptosystem, the cryptography operations are time-consuming and dominate the overall training time. To this end, we thoroughly investigate how GBDT runs the cryptography operations and find out two improper issues: (1) during histogram construction, we consistently accumulate ciphers into the histogram bins without taking their exponential terms into account, which leads to enormous unnecessary scaling operations; (2) since the number of tree nodes grows exponentially w.r.t. tree depth, the size of histograms also increases exponentially for the deeper tree layers, so as the time cost of histogram communication and decryption.

Our Solutions. In order to tackle these bottlenecks, we propose a novel training protocol for concurrent execution in both parties to address bottleneck 1 (Section 4), and customize the cryptography operations for GBDT to address bottleneck 2 (Section 5).

3 SYSTEM OVERVIEW

In this section, we present an overview of our vertical federated GBDT system, namely VF^{2}Boost.

3.1 Architecture

Figure 3 illustrates the architecture of VF^{2}Boost. There are three major components, as introduced below.

Scheduler. The scheduler manages the whole training routine, dispatches operations to workers, and collects the computation results from workers. It is also responsible for the maintenance of GBDT model and the synchronization with the opposite party.

Worker. Inside each party, the training dataset is distributed among workers by instances (a.k.a. data parallel). In vertical FL, the two parties hold the same instance set. We further let the instances be aligned between the parties at the worker level. That is to say, if Worker #1 of *Party A* owns 100 instances, then Worker #1 of *Party B* owns exactly the same 100 instances. Each worker runs the operations dispatched from the scheduler on its own training data shard and exchange necessary messages with the opposite party.

Channel. Since *Party A* and *Party B* stand for different enterprises and locate in separate data centers, there is restricted network access between them. Both the scheduler and workers inside one party are organized by an in-data-center cluster, which is usually forbidden from connecting with the public network in order to avoid external attacks. Therefore, we set up message queues on several gateway machines to route the cross-party communication. Because we align the instances in the worker level as described above, each worker needs to communicate with only one worker in the opposite party. Therefore, the scheduler and workers simply register to a specific channel to produce and consume messages.

3.2 Execution Workflow

Below we describe the key procedures to train one decision tree.

Gradient statistics encryption. For each decision tree, *Party B* calculates the gradient statistics based on the current predictions. To preserve the label privacy, gradient statistics are encrypted before sent to *Party A*. Specifically, we introduce a blaster-style encryption scheme to speed up this phase, as we will discuss in Section 4.1.

Histogram construction. Both parties build histograms individually based on their own feature columns. As we will introduce in Section 5.1, we accelerate the histogram construction in *Party A* by a re-ordered accumulation technique. Since instances are scattered among workers inside each party, the local histograms built by workers are further aggregated into global ones. In this work, we simply let each worker be in charge of a few features, and each worker sends the local histograms to corresponding workers for aggregation. There are more methods studied for histogram aggregation, such as All-Reduce, Reduce-Scatter, and Parameter-Server [14, 38, 43]. However, it is orthogonal to our work since these methods can also be applied in our context.

Split finding. Then we find the best splits over the aggregated histograms. Since the histograms in *Party A* need to be transferred to *Party B* for decryption and split finding, to accelerate the communication and decryption of histograms, we introduce a polynomial-based histogram packing technique in Section 5.2. Finally, the scheduler of *Party B* summarizes the split points found by all workers, and chooses the global best one. If the best split belongs to *Party A*, since *Party B* only knows the index of histogram bin for the best split, the index will be sent to *Party A* to recover the split feature

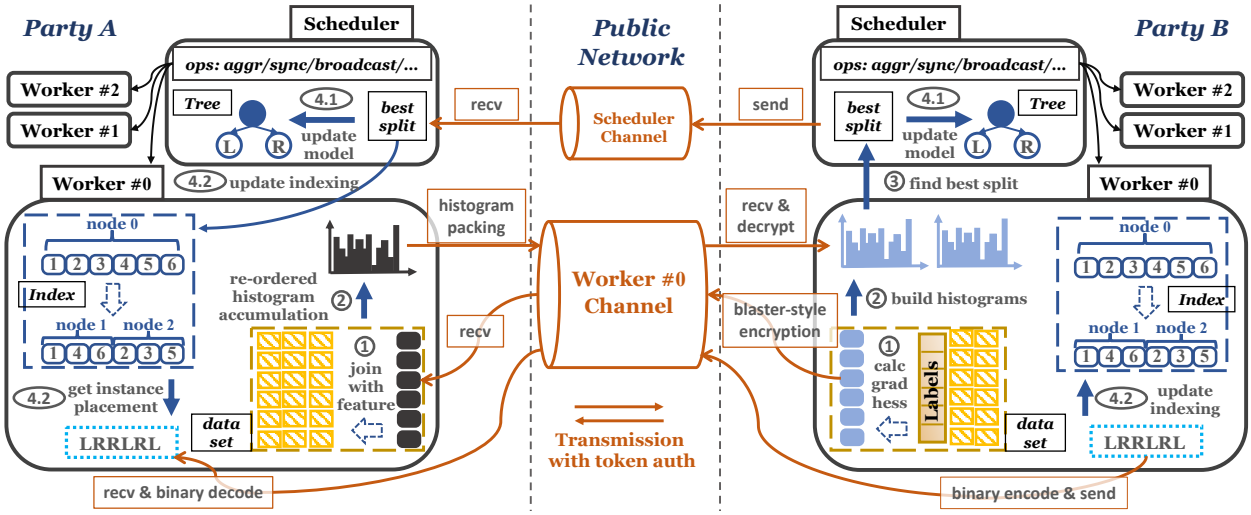

Figure 3: System overview of VF^{2}Boost.

and split value. This ensures that only one party knows the actual split information whilst the opposite party does not.

Node splitting. In this phase, the schedulers update the tree models and workers update the indexing structures between tree nodes and instances. As aforementioned, only one party knows the actual split information. Thus, the owner party should notify the other party of the instance placement after node splitting, i.e., whether each instance is classified to the left or right child node. The same as SecureBoost [15], the instance placement only reveals how the best split partitions the instances, which is not enough to recover the features of the owner party. We follow [2, 28] to encode the instance placement into a bitmap so that the communication overhead can be lowered greatly. As we will analyze in Section 4.2, since the node splitting phase incurs a synchronization between the parties, which leads to non-trivial idle periods, we derive an optimistic node-splitting strategy to improve the overall performance.

3.3 Implementation Details

When implementing the vertical federated GBDT system, we emphasize the following significant properties.

Production-friendly. Inside each party, we leverage Apache Spark [87], a popular distributed engine for big data processing. Specifically, we run the scheduler on the Spark driver and wrap each worker as one RDD partition [86]. Federated jobs are submitted to Yarn or Kubernetes [11, 76] for resource allocation. We load datasets from and save ML models to HDFS. By doing so, our system can be integrated into real productive pipelines easily.

Scalable & Robust. We establish the message queues for the geo-distributed cross-party communication via Apache Pulsar, a scalable and robust messaging system. The effectively-once semantics of Pulsar avoids message missing or duplication in geo-distributed transmission. Moreover, it supports token authentication, which facilitates confidential message passing.

High-performance. To speed up the cryptography operations, we develop an efficient library for the Paillier cryptosystem, which supports parallel processing with the OpenMP library.

With these properties, our system outperforms the existing vertical federated GBDT implementations by an order of magnitude and supports much larger datasets, as demonstrated in Section 6.

4 CONCURRENT TRAINING PROTOCOL

As aforementioned, one drawback of the vertical federated GBDT protocol is that both parties have frequent idly waiting periods. Obviously, it leads to extremely low efficiency and a huge waste of computational resources. After we carefully analyze the existing protocol, we find that the root cause is that the key procedures in both parties are performed sequentially. Therefore, we propose a concurrent training protocol to reduce the idle time in this section.

Methodology. To achieve this goal, we analyze the schedule of different procedures in training a decision tree via Gantt charts. Specifically, we focus on four types of actions that are related to cryptography operations: (1) encryption of gradients and hessians in *Party B*; (2) histogram construction in *Party A* (abbrv. BuildHistA) which involves homomorphic addition; (3) split finding for *Party A* in *Party B* (abbrv. FindSplitA) which involves decryption; (4) public network communication of ciphers including encrypted gradient statistics and histograms (abbrv. CipherComm). We do not stress other actions, such as split finding for *Party B* (abbrv. FindSplitB, including histogram construction) and communication of instance placement, since they usually spend much less time.

4.1 Blaster-Style Encryption

Observation. We observe that the processing of the root node usually takes a large portion of the total time cost. To find out the reason, we draw the Gantt chart of the root node with the existing protocol on the top of Figure 4. Before *Party A* can build histograms for the root node, *Party B* must encrypt the gradients and hessians for all training instances and send the ciphers across the public network. According to the Gantt chart, we know that *Party A* and the network channel sit idle for a long time. Undoubtedly, this results in a waste of both computational and network resources.

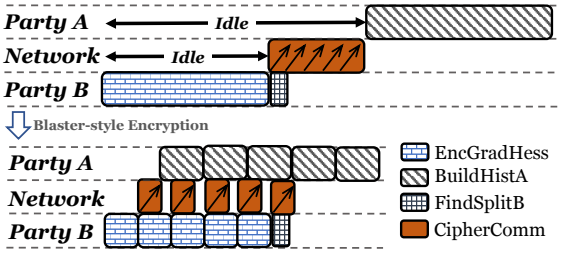

Figure 4: Gantt chart of the root node processing. Top: procedure of existing protocol. Bottom: procedure of blaster-style encryption.

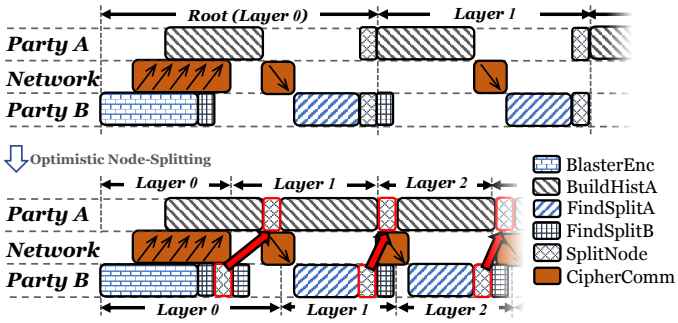

Figure 5: Gantt chart of tree processing. Top: procedure of existing protocol. Bottom: procedure of optimistic node-splitting under an ideal case where the splits of *Party B* are always better than those of *Party A*.

Solution. To solve this problem, we propose a blaster-style encryption scheme, as illustrated on the bottom of Figure 4. The intuition is that the histogram construction is essentially the accumulation of the gradient statistics, so different instances can be processed independently. Thus, we divide the training instances into small batches and process each batch individually. For each batch, *Party B* encrypts the gradient statistics, blasts the ciphers to *Party A* in a background thread, and proceeds to the next batch. Meanwhile, *Party A* accumulates the encrypted gradient statistics to histograms upon receiving every batch. By doing so, the encryption in *Party B*, cipher transmission across the public network, and the histogram construction in *Party A* can be executed concurrently.

Discussion. In practice, the blaster-style encryption scheme is also beneficial to the public network transmission. In many cases, we use more workers than the gateway machines, therefore, the total network bandwidth of workers is larger than that of gateway machines. In the existing protocol, the message queue would be congested due to the bulk of transmission caused by all encrypted gradients and hessians. In contrast, with the blaster-style communication, the burden can be alleviated since we do not need to trigger a large amount of cipher transmission in a short period of time.

4.2 Optimistic Node-Splitting

Observation. In addition to the root node, there are also many non-trivial idle periods in both parties caused by the synchronization of node splitting (abbrv. SplitNode). As shown on the top of Figure 5, after *Party A* has finished histogram construction for one tree layer, the histograms are transmitted to *Party B* for decryption and split finding. Then both parties jointly split the tree node(s). We observe that *Party A* and *Party B* wait for each other by turns. First, since BuildHistA involves enormous homomorphic addition operations, it is much slower than FindSplitB and *Party B* is blocked by the receiving of histograms. Second, since SplitNode requires synchronizing the instance placement, *Party A* must standby when *Party B* is performing FindSplitA, which involves the time-consuming decryption of histograms.

Motivation. Readers might wonder whether we can make use of the idle time as we have done in blaster-style encryption. In other words, can we overlap the decryption of histograms in *Party B* with the construction of histograms in *Party A*? In fact, the reason why both parties wait mutually is that they need to synchronize the instance placement so that each party knows the correct splitting of every tree node. Nevertheless, we discover that it is actually a waste of time if we synchronize a tree node where *Party B* has a better split than *Party A*. Motivated as such, we borrow the idea of optimistic locks [46] in database management and propose an optimistic node-splitting strategy. With such a strategy, *Party B* optimistically splits tree nodes and proceeds to the next layer ahead of *Party A*, rather than waits for synchronization.

Illustration of ideal case. To help readers better understand the optimistic strategy, we first consider an ideal case where the best splits using features of *Party B* always have higher gains than those of *Party A*, i.e., the instance placement is always computed by *Party B*. The workflow of the optimistic strategy under such an ideal case are shown on the bottom of Figure 5 and described below.

Our first idea is to make use of the idle time of *Party B* to process the next layer optimistically. Once FindSplitB of the layer l is done, we let *Party B* optimistically split the tree nodes and head to the optimistic layer $l+1$, even though *Party A* is still in the BuildHistA phase of layer l. Only after FindSplitB of layer $l+1$ is done will *Party B* pause and wait to receive the histograms of layer l from *Party A*. The phase of FindSplitA is akin to a validation of the optimistic node-splitting. Under our ideal assumption that the splits of *Party B* are always better than *Party A*, we will verify that tree nodes in layer l are correctly split, i.e., the instances on the optimistic layer $l+1$ are correctly classified. Thus, the histograms built for layer $l+1$ are also accurate for split finding.

Second, we occupy *Party A* with the histogram construction of optimistic layers. Under our ideal assumption, the instance placement of node splitting must be sent from *Party B* to *Party A*, which is indicated as the red arrows on the bottom of Figure 5. Moreover, since FindSplitB is much faster than BuildHistA, *Party B* always finishes the node splitting phase earlier than *Party A*. Therefore, whenever BuildHistA of layer l commits, *Party A* immediately splits the tree nodes according to the instance placement shared by *Party B*, and starts the BuildHistA phase of layer $l+1$. Meanwhile, the histograms of layer l are transmitted to *Party B* in order to validate the optimistic node splitting. Consequently, the histogram construction of optimistic layer $l+1$ in *Party A* and the histogram communication and decryption of layer l are executed concurrently.

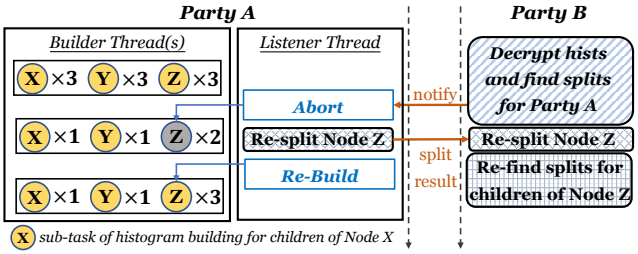

Figure 6: An illustration of the mechanism to roll-back and re-do the dirty nodes in optimistic node-splitting.

Mechanism to roll-back and re-do dirty nodes. Indisputably, the ideal case described above is unrealistic since there must be tree nodes where *Party A* owns better splits, unless otherwise *Party A* having zero contribution to the federated model. We say a tree node is dirty if its best split is owned by *Party A* but erroneously split by *Party B*. If we do not correct the dirty nodes, all splits will belong to *Party B*, i.e., the model will be equivalent to training with *Party B* only, since BuildHistA is far slower than FindSplitB. As a result, a mechanism to roll-back and re-do the dirty nodes is necessary to maintain the model accuracy. Borrowing the ideas from the field of concurrency control in database management, we determine to abort the dirty nodes and re-do the phases of node-splitting and histogram construction. We present the main procedures with an example in Figure 6 and go through the details below.

Initially, *Party A* has constructed and sent the histograms of Node X, Y, Z to *Party B*, and performed optimistic node-splitting. Then, *Party A* divides the histogram construction of the children nodes into small sub-tasks which can be processed in parallel. Meanwhile, *Party B* decrypts the histograms and looks for the best splits of *Party A*. Later, it unfortunately finds out that *Party A* has a better split for Node Z, which means that the optimistic splitting is invalid. As shown in Figure 6, *Party B* sends the best split of Node Z to notify *Party A* immediately. To save computational resources, the sub-tasks for children of Node Z are aborted at once. Node Z will be re-split by both parties to ensure the correctness of the model. Finally, Node Z is no longer dirty and its children nodes are re-processed. It is worthy to note that since FindSplitB is much faster than BuildHistA, we can still apply the optimistic node-splitting strategy to the children nodes of Node Z.

Discussion. The efficiency of the optimistic node-splitting strategy depends on the frequency of dirty nodes. In expectation, the failure possibility of the optimistic splits is $D_A/(D_A + D_B)$, where D_A, D_B are the number of features in the two parties. The failure possibility goes lower when *Party B* owns more features, and vice versa, which is also verified by our experiments in Section 6. Compared to the traditional protocol where both parties wait idly in turns, the proposed strategy utilizes the idle time to finish considerable workloads in advance at the price of some extra computation for dirty nodes. Furthermore, in order to reduce the aborted works, we slice the histogram construction of all nodes into smaller tasks rather than process them one by one. As we will show in our experiments, our system is able to achieve consistent model accuracy as the non-federated model obtained on co-located datasets, which validates the correctness of the proposed mechanism for dirty nodes.

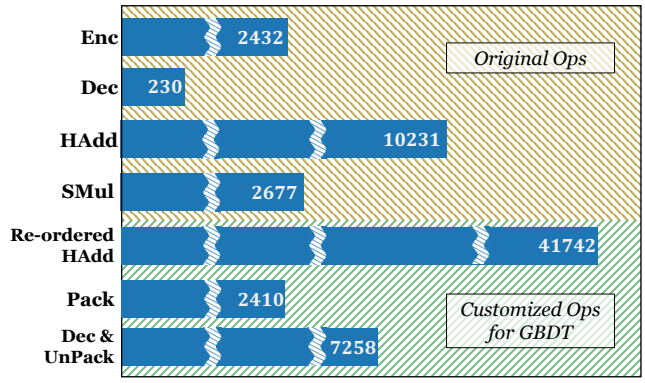

Figure 7: Throughputs (#operations per second) of different cryptography operations using one thread. We set $S = 2048$ and generate the values from a normal distribution.

5 CUSTOMIZED CRYPTOGRAPHY FOR GBDT

Although we have introduced the concurrent protocol to improve the resource usage ratio in both parties, the overall performance is still unsatisfactory as the cryptography operations are extremely expensive. Thus, we dive into the core operations and propose specific customization matching the characteristics of GBDT algorithm.

Cost model. Focusing on the cryptography-related operations, we denote the unit time cost of encryption, decryption, homomorphic addition, scalar multiplication, and cipher communication across the public network as $T_{ENC}, T_{DEC}, T_{HADD}, T_{SMUL}$, and T_{COMM}, respectively. Furthermore, Figure 7 illustrates the throughputs of different types of operations as a reference.

5.1 Re-ordered Histogram Accumulation

Observation. For the root node, *Party B* encrypts and sends the gradient statistics for all instances, whilst *Party A* adds the ciphers to corresponding histogram bins by scanning the non-zero feature values. Thanks to the blaster-style encryption scheme, we overlap the computation in both parties and the communication across public network. Asymptotically speaking, we have a time complexity of $O(N \times \max\{T_{ENC}, d \times T_{HADD}, T_{COMM}\})$ for the histogram construction of the root node. In practice, we observe that the time cost of histogram construction, which contains umpteenth homomorphic addition operations, usually dominates, which is also shown in our experiments. As a result, we place more emphasis on the efficiency of histogram construction than the encryption of gradient statistics, which is also beneficial to the subsequent layers.

Re-ordered accumulation. Recall that the homomorphic addition performs a scaling if the addends are different in the exponential term. During histogram construction, the encryption gradient statistics are accumulated into histogram bins one by one, and the exponential term of each histogram bin is determined by the max one ever seen. We notice that the total number of scaling operations is influenced by the ordering of instances. As shown in Figure 8, if we sequentially accumulate the five encrypted gradient statistics, there will be four scaling operations. However, if we re-order the ciphers, it can be reduced to two scaling operations.

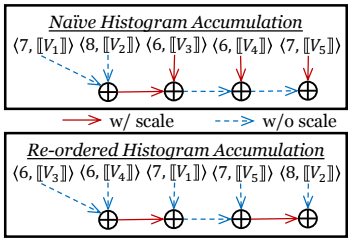

Figure 8: Comparison of Naïve and Re-ordered Histogram Accumulation. Red solid line means the cipher is scaled before addition, while blue dashed line vice versa.

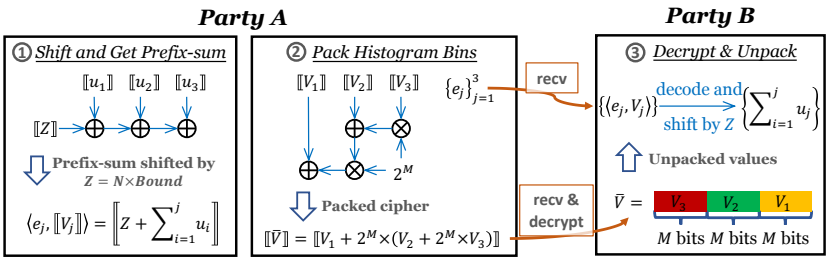

Figure 9: Illustration of Polynomial-based Histogram Packing.

Undoubtedly, the efficiency is affected by the number of scaling operations. As a result, we determine to re-schedule the ordering of accumulation according to the exponential terms, as shown in Figure 8. By such means, we only need $E-1$ scaling operations, where E is the number of unique values in the exponential terms. Compared to the $O(N \times (E-1)/E)$ asymptotic complexity when the instances are in a random order, the re-ordered accumulation scheme avoids tremendous scaling operations. In practice, we find that there are very few unique exponential values (ranging from 4 to 8). Therefore, the time cost of the $E-1$ scaling operations is negligible. As shown in Figure 7, the throughput of homomorphic addition increases by 4.08× if we take the exponential terms into account and re-schedule the ordering of accumulation accordingly.

Integration with histograms. Although the re-ordered accumulation scheme reduces the number of scaling operations sharply, it incurs a non-trivial overhead for sorting the encrypted gradient statistics w.r.t. the exponential terms for every tree node. As a result, in practice, we allocate individual workspaces for different exponential values temporarily, and accumulate the gradient statistics to the corresponding one. Finally, all workspaces are summed into the final histograms. As aforementioned, E is very small, so the auxiliary memory cost is acceptable.

Discussion. Our re-ordered technique suits for GBDT since the histogram construction contains mostly the addition operations. However, it can also be utilized in more scenarios where we need to sum a set of ciphers. For instance, for the vertical federated LR [84], we can accelerate the reduction of encrypted gradients in a mini-batch by the re-ordered accumulation technique.

5.2 Polynomial-based Histogram Packing

Then we proceed to handle the subsequent layers. For *Party A*, since the total number of instances remains unchanged for all layers, the time complexity of histogram construction is $O(N \times d \times T_{HADD})$ per layer. For *Party B*, the major cost is brought by the decryption of histograms, whose size is $O(D \times s)$ for each tree node. However, since we usually build the histograms for the tree nodes on the same layer concurrently, the time complexity on the l-th layer should be $O(D \times s \times 2^l \times \max\{T_{DEC}, T_{COMM}\})$ for *Party B*, and $O(N \times d \times T_{HADD} + D \times s \times 2^l \times T_{COMM})$ for *Party A*, respectively.

At the first sight, we might concern that the histogram construction dominates since $N \gg D$ in most cases. Nevertheless, according to Figure 7, the decryption operation is far slower than the homomorphic addition. Worst still, since the number of tree nodes grows exponentially w.r.t. tree depth, the complexity of decryption and communication gradually dominates as the tree goes deeper. For many complicated tasks, users would like to train deeper trees to improve the model ability. Therefore, it is necessary to accelerate the communication and decryption of histograms.

Motivation. Jiang et al. [38] propose to reduce the communication overhead by quantizing the histograms, i.e., turn the floating-point values into low-precision representations, on account of the fact that gradients and hessians have a small value range in most time. Although such a method cannot be applied to our system since histogram bins are ciphers, it motivates us that the encoded integer V of each histogram value v locates in a very narrow range compared with the huge encoding space. Mathematically speaking, V can be represented by M bits if $v \in (0, 2^M/B^e)$, whilst the encoding space is determined by the S-bit modulus n. In practice, we have $M \leq 64 \ll S = 2048$ when $v \leq 2^{32}$, $B = 16$, and $e = 8$.

Cipher packing. Motivated as such, we propose to pack multiple ciphers into one to reduce the total size, which is depicted in Step 2 and Step 3 of Figure 9. Suppose there are t ciphers $\{[\![V_i]\!]\}_{i=1}^t$ for integers $V_i \in (0, 2^M)$, we pack them by a polynomial transformation $[\![\overline{V}]\!] = [\![V_1]\!] \oplus 2^M \otimes ([\![V_2]\!] \oplus 2^M \otimes ([\![V_3]\!] \oplus \cdots))$. To recover the original values, only one decryption is needed to obtain $\overline{V} = V_1 + 2^M \times (V_2 + 2^M \times (V_3 + \cdots))$. Finally, $\overline{V}$ will be unpacked by being sliced into the M-bit integers $\{V_i\}_{i=1}^t$. Obviously, if we pack t ciphers into one, both the size of ciphers and the time cost of decryption shrink to $1/t$ at the price of $O((t-1) \times (T_{HADD} + T_{SMUL}))$. As presented in Figure 7, the overhead of packing is worthy given the near 32× of improvement in decryption.

Integration with histograms. As described in Section 2.2, positive and negative values are encoded into different ranges. Therefore, the packing technique requires the actual values of all ciphers to be positive. However, *Party A* cannot tell the sign of each histogram bin since it could reveal the label information. Fortunately, unlike model gradients in LR or DNNs, the histogram bins are guaranteed to be lower bounded. For instance, the gradient of logistic loss is bounded by $[-1, 1]$ while the hessian is non-negative. (We can also apply an L1 regularization to bound the gradients without harming the model accuracy.) Thus, we let *Party A* shift the histogram bins by $N \times Bound$ before packing. As illustrated in Figure 9, we shift the first histogram bin before prefix-sum calculation so that

Table 1: Breakdown of the blaster-style encryption scheme (BlasterEnc) and the re-ordered histogram accumulation technique (Re-ordered). The metric is the time cost (in seconds) of building the histograms of the root node. We fix the number of features in both parties as 25K and vary the number of instances.

#Instances	Baseline				+ BlasterEnc	+ Re-ordered	+ BlasterEnc + Re-ordered
	Enc	Comm	HAdd	Total			
2.5M	116	44	248	398	256 (1.55×)	339 (1.17×)	177 (2.25×)
5M	235	93	490	818	539 (1.52×)	642 (1.27×)	369 (2.22×)
10M	486	204	957	1647	1043 (1.58×)	1360 (1.21×)	709 (2.32×)

Table 2: Breakdown of the optimistic node-splitting strategy (OptimSplit) and the polynomial-based histogram packing method (HistPack). The metric is the time cost (in seconds) of building one decision tree. We fix the number of instances as 10M and vary the number of features in both parties.

#Features (A/B)	Ratio of Splits in *Party B*	Baseline	+ OptimSplit	+ HistPack	+ OptimSplit + HistPack
40K/10K	22.03%	9143	7142 (1.28×)	5482 (1.67×)	4128 (2.21×)
25K/25K	53.60%	4286	3250 (1.32×)	2953 (1.45×)	1982 (2.16×)
10K/40K	84.19%	2068	1426 (1.45×)	1668 (1.24×)	1087 (1.90×)

the overhead is further reduced to $O(D \times T_{HADD})$. In practice, we set $M = 64$ and pack 32 histogram bins together if $S = 2048$. With such a technique, we can reduce the time cost of communication and decryption to a great extent with a small overhead.

Discussion. In vertical FL, gradients are encrypted into large ciphers (e.g., 4096 bits) regardless of the original value range to ensure security, making the existing low-precision training approaches [3, 27] infeasible. Nevertheless, our packing technique conveys a possibility to speedup the transmission and decryption phases for many algorithms other than GBDT. For instance, model gradients can be bounded by regularization techniques in vertical federated LR or DNNs [84, 90] so that our packing technique can be applied. We would like to leave it as our future work.

6 EXPERIMENTAL EVALUATION

In this section, we empirically evaluate the performance of VF^{2}Boost.

6.1 Setup

To begin with, the experimental setup is introduced below.

Environment. We run all the experiments on two clusters in different data centers to stand for different parties. Each machine is equipped with 16 cores and 64GB RAM. Inside each cluster, we have 10Gbps Ethernet, whilst the public network bandwidth between the two clusters is 300Mbps. For each party, we establish message queues on three machines for cross-party communication.

Data Preparation. Since we focus on the training phase, we preprocess the datasets via the private set intersection (PSI) technique [13, 18, 24, 51] to align the instances in all parties.

Protocol. Both VF^{2}Boost and the competitors achieve comparable model accuracies against the non-federated models trained on co-located datasets, as we will show in Section 6.3. Therefore, we focus on the improvement of training speed. Throughout the experiments, we let $T = 20$ (# trees), $\eta = 0.1$ (learning rate), $L = 7$ (# tree layers), and $s = 20$ (# bins per histogram).

6.2 Effectiveness of Proposed Methods

We first investigate the effectiveness of the proposed methods in Section 4 and Section 5. To achieve this goal, we generate synthetic datasets following Section 5.2 of [28], and assess the performance improvement of each method individually.

Blaster-style encryption. We first assess the effect of our optimization on encryption by comparing the running time of gradient statistics encryption and histogram construction for the root node on datasets with varying N. The results are shown in Table 1. Without the blaster-style encryption technique, the encryption, communication, and histogram construction are sequential, so we dissect the time cost of each part respectively. We notice that after applying the proposed technique, the total time cost is similar to that of histogram construction, which verifies that the three parts can be overlapped well. In short, the proposed encryption technique gives 1.52-1.58× of improvement on the root node processing.

Re-ordered histogram accumulation. Next, we evaluate the re-ordered accumulation technique. Since the re-ordered technique only accelerates the HAdd part without overlapping the three parts, it gives 1.17-1.27× of speedup alone. However, the two optimization techniques complement each other and together contribute 2.22-2.32× of speedup for the root node. Moreover, it is worthy to note that the proposed re-ordered technique is also beneficial to other tree nodes, bringing higher performance boosting.

Optimistic node-splitting. With the hope of overlapping the decryption and construction of histograms, we design the optimistic node-splitting strategy and the roll-back-and-re-do mechanism for invalid splits. To inspect the proposed algorithm protocol, we vary the number of features in both parties and record the training performance in Table 2. We observe that the ratio of best splits in *Party B* is proportional to the ratio of feature dimensionalities in both parties, which is consistent with our discussion in Section 4.2. In general, the optimistic node-splitting strategy provisions 1.28-1.45× of speedup by reducing the idle time in both parties. When *Party B* owns more features, the failure possibility drops and the

Table 3: Description of datasets for end-to-end evaluation.

Dataset	#Instances	#Features (A/B)	Density
census	22K	78/70	8.78%
a9a	32K	73/50	11.28%
susy	5M	9/9	100%
epsilon	400K	1K/1K	100%
rcv1	697K	23K/23K	0.15%
synthesis	10M	25K/25K	0.20%
industry	55M	50K/50K	0.03%

optimistic strategy performs better. In many real cases, *Party B* can extract more features targeting at the labels via feature engineering, so it holds more data that are closely relevant to the training task than *Party A*. As a result, the proposed optimistic strategy suits the scenario of vertical federated GBDT well.

Polynomial-based histogram packing. As presented in Table 2, the histogram packing technique brings 1.24-1.67× of speedup alone since the time cost of communication and decryption of histograms shrinks by 32×. Furthermore, it improvs by 1.31-1.73× over OptimSplit since *Party B* can discover the invalid optimistic splits earlier, saving more time from the dirty nodes. To summarize, the 1.90-2.21× of speedup is achieved via the two techniques.

Resource utilization. Finally, we briefly investigate the effects of the proposed techniques on CPU utilization and public network transmission. First, by reducing the idle time in both parties, the CPU utilization is improved by a large extent. For instance, the average CPU utilization in *Party A* bumps from 670% to 1056%, a 58% improvement. Second, our histogram packing technique helps to reduce the total public network transmission per tree from 3.2GB to 1.1GB, a 66% saving.

6.3 End-to-End Evaluation

Next, we start to evaluate the end-to-end training performance of VF^{2}Boost and more competitors on various datasets.

Competitors. Four implementations are chosen as the competitors of VF^{2}Boost, which are listed as followed:

- SecureBoost [15]: The vertical federated GBDT implementation in FATE[3], which is a popular open-source FL system.
- Fedlearner[4]: An open-source FL system that implements the vertical federated GBDT algorithm based on SecureBoost. However, it does not support distributed training inside a party.
- VF-GBDT: Our self-developed vertical federated GBDT system without the optimization techniques in Section 4 and Section 5.
- VF-MOCK: A variant of VF-GBDT that mocks the cryptography operations, i.e., the computation is carried out in plaintexts.
- XGBoost[5] [14]: One of the most widely used GBDT libraries. We take it as a non-federated baseline. Specifically, we use XGBoost to train in two modes: on co-located datasets and on datasets of *Party B* only.

Datasets. As listed in Table 3, we conduct the experiments on seven datasets, including five public datasets, one synthetic dataset, and one industrial dataset. We randomly split 80% and 20% as training and validation sets, respectively. For the two small-scale datasets *census* and *a9a*, we train on a single machine in each party. For the rest datasets, eight workers are used in each party.

Results on small-scale datasets. First, we compare the performance of all federated competitors on the two small-scale datasets *census* and *a9a*. The results are demonstrated in Figure 10.

Convergence. To validate the power of FL, we also conduct experiments with XGBoost in the non-federated fashion. Figure 10 shows that all the federated competitors achieve better model performance than training with datasets of *Party B* only, and is comparable to training on co-located datasets. This verifies the lossless property of the vertical federated GBDT algorithm.

Training speed. In general, our self-developed systems can be an order of magnitude faster than the other baselines. VF-GBDT outperforms SecureBoost by 12.11× and 12.85× on *census* and *a9a*, respectively. Fedlearner runs faster than SecureBoost since it accelerates the histogram construction by matrix operations, however, it is still 8.61× and 9.20× slower than VF-GBDT. This is unsurprising since our system supports efficient parallel processing and has a well-optimized cryptography library, whilst the two competitors are implemented in Pythonic language, which is hard to achieve as good computation efficiency as ours. With the techniques in Section 4 and Section 5, VF^{2}Boost gains extra 1.41-1.47× of speedup compared with VF-GBDT. In summary, VF^{2}Boost is 12.8-18.9× faster than the existing vertical federated GBDT implementations.

Results on large-scale datasets. We show the results on the larger datasets in Table 4. Since SecureBoost either runs out of memory or fails to accomplish the training tasks in a reasonable time, whilst Fedlearner does not support distributed training inside each party, we do not compare with these two competitors.

Convergence. Consistent with the convergence results on the small-scale datasets, federated training achieves similar AUC metrics as training on co-located datasets and significantly outperforms training inside a single party. It reveals the superiority of vertical FL that it unites the valuable data in different parties (enterprises) to boost the overall model ability.

Training speed. To anatomize the performance more in depth, we record the related changes in running time in Table 4. We observe that VF-MOCK is 1.71-10.38× slower than XGBoost due to the overhead of cross-party execution, whilst the cryptography operations incur extra 69.31-157.33× slowdown. Although the efficiencies of the FL competitors are not comparable to XGBoost, it is still worthy as FL strengthens the models with the augmentation in data.

For the two low-dimensional dense datasets, *susy* and *epsilon*, since the total number of non-zero feature values ($N \times d$) is very large compared to the size of histograms ($D \times s$), the performance improvement mainly comes from the re-ordered histogram accumulation method. Since *susy* has much more number of instances, the blaster-style encryption scheme also contributes to the speedup. For the three high-dimensional sparse datasets, *rcv1*, *synthesis*, and *industry*, the optimistic node-splitting strategy and polynomial-based histogram packing techniques play important roles to boost the training speed. To summarize, VF^{2}Boost speedups by 1.38-2.71× through our optimization techniques.

Summary. The empirical results verify that VF^{2}Boost is able to outperform the existing works by 12.8-18.9× and support much

[3]https://github.com/FederatedAI/FATE
[4]https://github.com/bytedance/fedlearner
[5]https://github.com/dmlc/xgboost

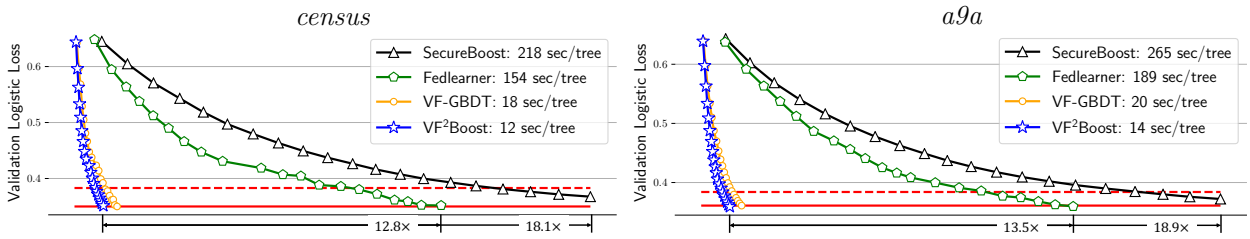

Figure 10: Logistic loss metrics (the lower the better) in terms of running time. The results show that VF^{2}Boost can be 12.8-18.9× faster than the existing vertical federated GBDT implementations. We draw the solid and dashed lines (in red) to indicate the logistic loss metrics of XGBoost by training on co-located datasets and on datasets of *Party B* only, respectively.

Table 4: Average running time and AUC metrics. We dissect the effects by showing the related slowdown (↓) or improvement (↑) to the previous competitor (Prev) and XGBoost (XGB). Both SecureBoost and Fedlearner fail to support these datasets.

Dataset	Average running time per tree (in seconds)									AUC metrics	
	XGB	VF-MOCK		VF-GBDT			VF^{2}Boost			VF^{2}Boost	XGB (co-located vs. *Party B*)
	Time	Time	vs. XGB	Time	vs. Prev	vs. XGB	Time	vs. Prev	vs. XGB		
susy	1.52	15.77	↓ 10.38×	204	↓ 12.94×	↓ 134.21×	117	↑ 1.74×	↓ 76.97×	0.865	0.865 vs. 0.858 (-0.007)
epsilon	9.89	32.38	↓ 3.27×	1556	↓ 48.05×	↓ 157.33×	1124	↑ 1.38×	↓ 113.65×	0.856	0.857 vs. 0.823 (-0.034)
rcv1	38.06	105.30	↓ 2.77×	3302	↓ 31.36×	↓ 86.76×	1419	↑ 2.33×	↓ 37.28×	0.972	0.971 vs. 0.948 (-0.023)
synthesis	50.80	130.62	↓ 2.57×	5364	↓ 41.07×	↓ 105.59×	1982	↑ 2.71×	↓ 39.02×	0.533	0.534 vs. 0.525 (-0.009)
industry	62.43	106.75	↓ 1.71×	4327	↓ 40.53×	↓ 69.31×	1958	↑ 2.21×	↓ 31.36×	0.762	0.762 vs. 0.724 (-0.038)

larger datasets with tens of millions of instances and tens of thousands of features. To the best of our knowledge, this is the first vertical federated GBDT system that supports such a scale of datasets.

6.4 Scalability

Finally, we empirically explore the scalability of VF^{2}Boost along two dimensions, i.e., how VF^{2}Boost scales when the number of workers or parties increases.

Scalability w.r.t. #Workers. We first conduct experiments with varying numbers of workers in order to evaluate the scalability of VF^{2}Boost. We present the results in Table 5. Overall, VF^{2}Boost runs faster given more workers. However, linear speedup is not observed on the four datasets, since the cost of some operations, e.g., public network transmission of ciphers and the node-splitting phase cannot be scaled linearly. For *susy* and *epsilon*, the speedups are close since they are dominated by histogram accumulation. When the number of workers increases from 4 to 8, the speedup is considerable since the computation cost can be alleviated owing to the benefit of data parallelism. When it further increases to 16, the speedup is not so significant since the cost of histogram aggregation inside each party grows as the number of workers increases. For the two high-dimensional sparse datasets *rcv1* and *synthesis*, other than histogram accumulation, the time cost of histogram decryption is also able to be amortized among workers. Thus, VF^{2}Boost achieves higher speedups on the two datasets when using 16 workers.

Scalability w.r.t. #Parties. Although our discussion focuses on the two-party scenario, it can be easily generalized to the cases of multiple parties, i.e., there are two or more parties that work as *Party A*'s and contribute their features to fit the tasks of *Party B*.

Table 5: Scalability w.r.t. number of workers (scaled by the training speed of 4 workers).

#Workers	Speedup			
	susy	*epsilon*	*rcv1*	*synthesis*
4	1.00×	1.00×	1.00×	1.00×
8	1.65×	1.64×	1.40×	1.57×
16	1.85×	1.87×	1.91×	2.23×

Thus, we evaluate VF^{2}Boost using 2-4 parties[6], with eight workers inside each party. The *epsilon* and *rcv1* datasets are used in the experiments. To be specific, we randomly divide the features into four subsets on average and assign them to different parties. For instance, each party owns 500 features for the *epsilon* dataset. The results are shown in Table 6. In general, when there are more parties, the model achieves higher AUC metric since the total amount of features increases. For the training speed, since *Party B* requires to send the encrypted gradient statistics to more destinations and decrypt more histograms, it takes slightly more time to finish the training when there are more parties. Nevertheless, VF^{2}Boost is able to scale to more parties within a reasonable time increment (within 10%) given the improvement of model ability.

7 RELATED WORKS

The GBDT Algorithm. GBDT is a popular ensemble model and there have been numerous implementations [2, 14, 19, 28, 38, 39,

[6]In practice, vertical FL with four or more parties is rare since it is hard to unite so many enterprises, so we do not conduct experiments with more than four parties.

Table 6: Scalability w.r.t. number of parties (scaled by the training speed of 2 parties) and the respective AUC metrics.

#Parties	Speedup		Validation AUC	
	epsilon	*rcv1*	*epsilon*	*rcv1*
Party B only	-	-	0.769	0.926
2	1.00×	1.00×	0.825	0.944
3	0.96×	0.93×	0.837	0.959
4	0.93×	0.90×	0.856	0.972

43, 56, 62, 63, 65]. XGBoost [14] is a shoo-in in various data analytic competitions and has been widely used in many commodity applications. Jiang et al. [38] systematically analyze different kinds of collective communication methods [73] for histogram aggregation. Abuzaid et al. [2] and Fu et al. [28] investigate how the data layout (partitioning schemes and storage patterns of training datasets) impacts the efficiency. All of these prior works aim at the non-federated version of GBDT and the proposed methods can be combined with our work by simply tweaking the procedures inside each party. Panda et al. [62] treats each tree node as one task and parallelize different nodes, whilst we process tree nodes layer-by-layer for two reasons: (1) we can aggregate the histograms and sends them across parties for multiple nodes together, which avoids frequent synchronization in distributed execution; (2) the histogram subtraction technique can be applied if the sibling nodes are processed together. Moreover, our optimistic node-splitting method can be applied to [62] for the shallow layers.

Horizontal FL. Due to the rising concerns on data protection, FL has become a widely discussed topic. One of the most important fields of FL is the horizontal federated learning, where all parties have the same feature columns but their instance sets are disjoint [44, 45, 47, 48, 50, 53, 55, 83]. Notably, [47, 48, 53] study the GBDT algorithm under the horizontal federated setting. Our work differs from these works in goal since we consider the GBDT algorithm in the vertical federated scenario.

Vertical FL. Many works are also raised for the case where two or more parties expect to jointly train on vertically partitioned data [15, 37, 52, 75, 78, 80, 84, 90]. Among them, [37, 78] assume that the labels of *Party B* can be shared with *Party A* and do not encrypt the gradient statistics, which is infeasible as it disobeys the goal of privacy protection. SecureBoost [15] proposes a lossless and privacy preserving vertical federated GBDT algorithm in the semi-honest scenario. Our work focuses on such a circumstance and develops a novel and efficient system with a series of optimization approaches. There are also works that study decision tree models in vertical FL. Nevertheless, our work differs from them. PPID3 [75] uses set intersection techniques to count the numbers of different classes for ID3 [64], which is infeasible for summing the gradient statistics in GBDT. FederatedForest [52] accomplishes vertical federated random forest (RF) [7] with a third-party trusted server, whilst we do not require such a third-party server. Pivot [80] studies CART [8] and extends to ensemble models such as RF and GBDT. Unlike our work, Pivot records the instance placement by encrypted mask vectors. However, its complexity is exponential times larger when the tree grows deeper since it needs to process all instances on every tree node. Moreover, these works dedicate to proposing safe and accurate algorithms rather than specific techniques to tackle the bottlenecks in vertical FL and improve the overall training speed. Therefore, they are orthogonal to our work.

Privacy Preservation. In FL, privacy preservation is one of the most essential properties. Several techniques are studied in depth to achieve meaningful guarantees to the data and model security. Homomorphic Encryption [23, 66] is applied in a wide range of works [4, 5, 15, 31, 34, 84] since cryptographic methods convey a high-level security guarantee. For instance, a 2048-bit Paillier cryptosystem [61] will take us decades if we try to decipher in a brute force manner. Secure Multi-Party Computation (MPC) [85] also delivers a considerable privacy guarantee. However, since the generic MPC framework involves many sophisticated computation protocols to achieve zero knowledge disclosure, it is extremely inefficient for ML algorithms. Thus, researchers adapt the security models and protocols to some pre-defined assumptions on the honesty and maliciousness of parties [58, 59, 92]. Differential Privacy (DP) [20] is also a well-known technique for privacy protection. Many works have adopted the idea of DP to obscure the model parameters or gradient updates by adding noises so that the individual privacy can be made indistinguishable [1, 12, 55, 72, 79].

Other Related Studies. In addition to machine learning algorithms, there is a surge of interests to more kinds of federated computing. For instance, the private set intersection [13, 18, 24, 51] helps vertical FL securely extract the overlapping instances in different parties. Another line of research tries to deploy the trusted execution environment (e.g., Intel SGX [54] and AMD memory encryption [41]) to federated computing, including machine learning [60] and database query [91]. However, it requires special hardware support and might be vulnerable to certain attacks [70, 82], which dampens the guarantee of security to private data.

8 CONCLUSION AND FUTURE WORKS

This work developed VF^{2}Boost, a novel and efficient vertical federated GBDT system for cross-enterprise collaboration. To address the deficiency in federated training, we systematically analyzed the training workflow and introduced a novel concurrent protocol to reduce the idle time. To accelerate the cryptography operations, we studied the essential operations in depth and proposed specific customization to match the characteristics of GBDT. Experimental results show that VF^{2}Boost runs faster than the existing works by an order of magnitude and is able to support much larger datasets.

As a possible future direction, we wish to optimize our mechanism for dirty nodes to skip instances that are already correctly classified so that we can save more time on histogram construction. Furthermore, we would also like to speed up the cryptography operations with the power of GPUs.

ACKNOWLEDGMENTS

This work is supported by National Key R&D Program of China (No. 2018YFB1004403), National Natural Science Foundation of China (NSFC) (No. 61832001, U1936104), PKU-Tencent joint research Lab, Beijing Academy of Artificial Intelligence (BAAI), and The Fundamental Research Funds for the Central Universities 2020RC25. Yinxia Shao and Bin Cui are the corresponding authors.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 308–318.

[2] Firas Abuzaid, Joseph K Bradley, Feynman T Liang, Andrew Feng, Lee Yang, Matei Zaharia, and Ameet S Talwalkar. 2016. Yggdrasil: An optimized system for training deep decision trees at scale. In *Advances in Neural Information Processing Systems*. 3817–3825.

[3] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 1709–1720. https://proceedings.neurips.cc/paper/2017/hash/6c340f25839e6acdc73414517203f5f0-Abstract.html

[4] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. 2016. Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. 142–144.

[5] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. 2017. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security* 13, 5 (2017), 1333–1345.

[6] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. 2015. A Guide to Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* 2015 (2015), 1192.

[7] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[8] Leo Breiman. 2017. *Classification and regression trees*. Routledge.

[9] BSI 2020. *Cryptographic Mechanisms: Recommendations and Key Lengths*. Technical Report TR-02102-1 v2020-01. BSI.

[10] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[11] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, omega, and kubernetes. *Queue* 14, 1 (2016), 70–93.

[12] Kamalika Chaudhuri and Claire Monteleoni. 2009. Privacy-preserving logistic regression. In *Advances in neural information processing systems*. 289–296.

[13] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1243–1255.

[14] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[15] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. 2019. Secureboost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755* (2019).

[16] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 409–437.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[18] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 789–800.

[19] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).

[20] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*. Springer, 1–19.

[21] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.

[22] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* 2012 (2012), 144.

[23] Caroline Fontaine and Fabien Galand. 2007. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security* 2007 (2007), 1–10.

[24] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*. Springer, 1–19.

[25] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. 2000. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics* 28, 2 (2000), 337–407.

[26] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[27] Fangcheng Fu, Yuzheng Hu, Yihan He, Jiawei Jiang, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Don't Waste Your Bits! Squeeze Activations and Gradients for Deep Neural Networks via TinyScript. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 3304–3314. http://proceedings.mlr.press/v119/fu20c.html

[28] Fangeheng Fu, Jiawei Jiang, Yingxia Shao, and Bin Cui. 2019. An experimental evaluation of large scale gbdt systems. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1357–1370.

[29] Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. 2018. Moment-Based Quantile Sketches for Efficient High Cardinality Aggregation Queries. *Proceedings of the VLDB Endowment* 11, 11 (2018).

[30] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[31] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*. 201–210.

[32] Vincent Grari, Boris Ruf, Sylvain Lamprier, and Marcin Detyniecki. 2020. Achieving fairness with decision trees: An adversarial approach. *Data Science and Engineering* 5 (2020), 99–110.

[33] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record* 30, 2 (2001), 58–66.

[34] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677* (2017).

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[36] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. ACM, 1–9.

[37] Yaochen Hu, Di Niu, Jianming Yang, and Shengping Zhou. 2019. FDML: A collaborative machine learning framework for distributed features. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2232–2240.

[38] Jiawei Jiang, Bin Cui, Ce Zhang, and Fangcheng Fu. 2018. Dimboost: Boosting gradient boosting decision tree to higher dimensions. In *Proceedings of the 2018 International Conference on Management of Data*. 1363–1376.

[39] Jie Jiang, Jiawei Jiang, Bin Cui, and Ce Zhang. 2017. TencentBoost: A Gradient Boosting Tree System with Parameter Server. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. 281–284.

[40] Lin Jiao, Yonglin Hao, and Dengguo Feng. 2020. Stream cipher designs: a review. *Science China Information Sciences* 63, 3 (2020), 1–25.

[41] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).

[42] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*. IEEE, 71–78.

[43] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*. 3146–3154.

[44] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527* (2016).

[45] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[46] Hsiang-Tsung Kung and John T Robinson. 1981. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)* 6, 2 (1981), 213–226.

[47] Qinbin Li, Zeyi Wen, and Bingsheng He. 2020. Practical Federated Gradient Boosting Decision Trees.. In *AAAI*. 4642–4649.

[48] Qinbin Li, Zhaomin Wu, Zeyi Wen, and Bingsheng He. 2020. Privacy-Preserving Gradient Boosting Decision Trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 784–791.

[49] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.

[50] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2018. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127* (2018).

[51] Gang Liang and Sudarshan S Chawathe. 2004. Privacy-preserving inter-database operations. In *International Conference on Intelligence and Security Informatics*. Springer, 66–82.

[52] Yang Liu, Yingting Liu, Zhijie Liu, Yuxuan Liang, Chuishi Meng, Junbo Zhang, and Yu Zheng. 2020. Federated forest. *IEEE Transactions on Big Data* (2020).

[53] Yang Liu, Zhuo Ma, Ximeng Liu, Siqi Ma, Surya Nepal, and Robert Deng. 2019. Boosting privately: Privacy-preserving federated extreme boosting for mobile

crowdsensing. *arXiv preprint arXiv:1907.10218* (2019).

[54] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution. *Hasp@ isca* 10, 1 (2013).

[55] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJ0hF1Z0b

[56] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[57] X. Miao, L. Ma, Z. Yang, Y. Shao, B. Cui, L. Yu, and J. Jiang. 2020. CuWide: Towards Efficient Flow-based Training for Sparse Wide Models on GPUs. *IEEE Transactions on Knowledge and Data Engineering* (2020). https://doi.org/10.1109/TKDE.2020.3038109

[58] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 35–52.

[59] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–38.

[60] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. USENIX Association, 619–636. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[61] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*. Springer, 223–238.

[62] Biswanath Panda, Joshua Herbach, Sugato Basu, and Roberto J. Bayardo. 2009. PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce. *Proc. VLDB Endow.* 2, 2 (2009), 1426–1437. https://doi.org/10.14778/1687553.1687569

[63] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[64] J. Ross Quinlan. 1986. Induction of decision trees. *Machine learning* 1, 1 (1986), 81–106.

[65] Greg Ridgeway. 2007. Generalized Boosted Models: A guide to the gbm package. *Update* 1, 1 (2007), 2007.

[66] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. 1978. On data banks and privacy homomorphisms. *Foundations of secure computation* 4, 11 (1978), 169–180.

[67] Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.

[68] Arosha Rodrigo, Miyuru Dayarathna, and Sanath Jayasena. 2019. Latency-Aware Secure Elastic Stream Processing with Homomorphic Encryption. *Data Science and Engineering* 4, 3 (2019), 223–239.

[69] Robert E Schapire. 1999. A brief introduction to boosting. In *Ijcai*, Vol. 99. Citeseer, 1401–1406.

[70] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs.. In *NDSS*.

[71] Dimitri P Solomatine and Durga L Shrestha. 2004. AdaBoost. RT: a boosting algorithm for regression problems. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, Vol. 2. IEEE, 1163–1168.

[72] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. 2013. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*. IEEE, 245–248.

[73] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

[74] Stephen Tyree, Kilian Q Weinberger, Kunal Agrawal, and Jennifer Paykin. 2011. Parallel boosted regression trees for web search ranking. In *Proceedings of the 20th international conference on World wide web*. ACM, 387–396.

[75] Jaideep Vaidya, Chris Clifton, Murat Kantarcioglu, and A Scott Patterson. 2008. Privacy-preserving decision trees over vertically partitioned data. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 2, 3 (2008), 1–27.

[76] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. 1–16.

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

[78] Li Wan, Wee Keong Ng, Shuguo Han, and Vincent CS Lee. 2007. Privacy-preservation for gradient descent methods. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 775–783.

[79] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. 2020. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security* (2020).

[80] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy Preserving Vertical Federated Learning for Tree-based Models. *Proceedings of the VLDB Endowment* 13, 11 (2020).

[81] Xu Xie, Fei Sun, Xiaoyong Yang, Zhao Yang, Jinyang Gao, Wenwu Ou, and Bin Cui. 2021. Explore User Neighborhood for Real-time E-commerce Recommendation. *arXiv preprint arXiv:2103.00442* (2021).

[82] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 640–656.

[83] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.

[84] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu. 2019. Parallel distributed logistic regression for vertical federated learning without third-party coordinator. *arXiv preprint arXiv:1911.09824* (2019).

[85] Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 160–164.

[86] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*. USENIX Association, 15–28. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia

[87] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.

[88] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. 2020. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. In *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*. USENIX Association, 493–506. https://www.usenix.org/conference/atc20/presentation/zhang-chengliang

[89] Wentao Zhang, Jiawei Jiang, Yingxia Shao, and Bin Cui. 2020. Snapshot boosting: a fast ensemble framework for deep neural networks. *Science China Information Sciences* 63, 1 (2020), 1–12.

[90] Yifei Zhang and Hao Zhu. 2020. Additively Homomorphical Encryption based Deep Neural Network for Asymmetrically Collaborative Machine Learning. *arXiv preprint arXiv:2007.06849* (2020).

[91] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. USENIX Association, 283–298. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng

[92] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2019. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 724–738.