Towards Privacy-Preserving Collaborative Gradient Boosted Decision Trees

Chester Leung, Andrew Law, Octavian Sima
UC Berkeley
{chester.leung, andrewlaw8, octavian.sima}@berkeley.edu

Abstract

With the wide availability of data in recent years, machine learning has taken off as an effective general solution with an abundance of use cases. Collaborative learning can yield benefits for all parties involved, but parties may be unwilling to share sensitive data in plaintext. In this paper, we explore the collaborative training of gradient boosted decision trees while hiding each party's data from all other parties. We do this by extending the popular XGBoost framework to two different modes. Federated mode leverages a modified communication pattern to offer more privacy than vanilla XGBoost and has similar performance, but still leaks some information. Coopetitive mode leverages secure enclaves and novel oblivious algorithms to leak no information, but has an order of magnitude worse performance than vanilla XGBoost.

1. Introduction

There has recently been growing interest in collaborative machine learning, where multiple parties work together to perform a machine learning task such as training or inference over their collective data. Collaboration among parties often yields significant benefits; having more data that may be complimentary or of different distributions may yield more robust models. However, parties may not be willing to reveal their own data to other parties for reasons including legal regulation, privacy policies, or business competition. This presents a need to develop systems that can jointly compute on sensitive data sets belonging to different parties all while hiding the contents of the data from all parties. Below, we provide two concrete uses cases where the benefits of jointly computing on sensitive data are evident.

Banks today attempt to detect fraud by training models on customer transaction data, but criminals often mask their actions by moving their assets across different banks. As a result, trained models will be much weaker if only trained on one bank's data – this elevates the need for banks to share data. However, customer financial data is sensitive and cannot be shared in plaintext. Joint computation can occur only if the shared data is not revealed to any party.

Hospitals also may want to collaborate to obtain more effective models for diagnoses or treatment plans. This collaboration requires the sharing of sensitive patient data, which should not be done in plaintext. Secure collaboration is ideal for this situation – hospitals will be able to leverage the data of other hospitals to improve their own models and consequently patient care, but will also not see the exact contents of the data.

There are currently many machine learning methods out there, each suitable for their own tasks. One such method is decision trees, a powerful algorithm that can efficiently and accurately model non-linear relationships in data. A powerful extension to decision trees is gradient boosted decision trees, a popular machine learning algorithm that has given state-of-the-art results in both production environments and in machine learning and data mining competitions. Facebook uses it to predict clicks on ads [12], while XGBoost [8], an existing gradient boosted decision tree framework, has yielded state-of-the-art performance on Kaggle, a machine learning competition website: of the 29 challenge winning solutions published on machine learning competition website Kaggle's blog in 2015, 17 used XGBoost. Since its inception in 2016, XGBoost has become incredibly popular, garnering attention on popular Medium blogs [21, 25, 19], adoption on Amazon's managed cloud machine learning platform SageMaker, and nearly 18k stars on GitHub.

Our work aims to bring the benefits of XGBoost to the secure collaborative setting. First, we improve XGBoost's security by redesigning it to run in the federated setting [20]. This enables multiple parties to collaborate without having to send data offsite, meaning that the potentially sensitive data owned by each party will never be exposed to another party. Instead, Federated XGBoost relies on a central trusted aggregator to coordinate training among all parties; the aggregator requires only data summaries to update the model during training, as opposed to all data. Second, we employ hardware enclaves [10] and obliviousness to bring even greater security to XGBoost. We redesign XGBoost

under the hood to perform all functionality on sensitive data inside the enclave, making all data processing opaque to even the host. Therefore, even if data is transferred across the network to foreign sites, even the parties at those sites have no way to access the data. Hardware enclaves, however, have side channel leakage, and consequently are prone to side channel attacks [7]. To counteract this, we modify the existing XGBoost algorithms' access patterns to make the algorithms oblivious, i.e. independent of the input data, which in turn makes our system resistant to access pattern attacks. Following prior work [28], we define this version of XGBoost to be *coopetitive* XGBoost.

To summarize, our contributions are as follows:

- federated XGBoost with a trusted centralized aggregator and mutually distrustful parties
- coopetitive XGBoost with hardware enclaves and obliviousness that provides security against even a malicious adversary
- a Python interface supporting both federated and coopetitive mode that enables data scientists and developers to easily adopt our system

2. Background

2.1. XGBoost

Gradient boosting is a technique used to build a decision tree ensemble by iteratively improving on the errors of the last "weak learner" (singular tree) that was produced. Together, this ensemble of decision trees can perform better than just one. XGBoost, created by Tianqi Chen, is a gradient boosting decision tree framework which aims to provide a "Scalable, Portable and Distributed Gradient Boosting (GBM, GBRT, GBDT) Library." It has recently gained widespread popularity – the algorithm is used by many winning teams in machine learning competitions.

XGBoost supports both single machine and distributed workloads. In the distributed version of XGBoost, an "approximate" split finding algorithm is run during training at every node to find the decision tree splits optimal for that node's partition of data. All nodes' splits are then aggregated to obtain the optimal split over all data. Details on the split finding algorithm and XGBoost's "weighted quantile sketch" can be found in the original XGBoost paper.

2.2. Secure Enclaves

Secure enclaves provide private regions of memory isolated from the rest of the host. Other processes, the hypervisor, and even the host kernel cannot access this secure region of memory. The trusted execution environment that secure enclaves create provide confidentiality and integrity guarantees, making them fit for sensitive data processing.

There are currently a selection of hardware enclaves available on the market, e.g. Intel SGX [10], AMD Memory Encryption [14]. Microsoft Azure Confidential Computing [24] and IBM Cloud [15] also offer cloud offerings of enclaves.

2.3. Obliviousness

Ideally, secure enclaves provide confidentiality and integrity for applications running inside of them. However, recent work has shown that secure enclaves are vulnerable to many forms of side channel attacks.

We employ *oblivious algorithms* [27] to make program execution independent of data inputs, removing side channel attacks that exploit memory access patterns. As a naive example of an oblivious algorithm, an oblivious array access may scan the entire array rather than accessing the desired index. This way, the desired index is hidden from attackers exploiting memory access side channels. Concretely, the compiled code should not have conditional branches dependent on data inputs.

3. Related Works

3.1. Decision Trees

XGBoost was designed to improve upon existing tree-boosting frameworks such as those provided by scikit-learn and R.gbm. Both of these two libraries support the exact greedy algorithm for split finding, but not the approximate histogram algorithm. This gives XGBoost greater performance with little cost in accuracy. In the original XGBoost paper [8], there are figures outlining the drastic difference in runtime for a variety of inputs comparing XGBoost to these other two choices.

Another widely used tree-boosting algorithm is Microsoft's LightGBM [16], developed after XGBoost for the purpose of enhancing the speed and performance of existing tree-boosting frameworks. To do this, LightGBM employs two novel techniques called *Gradient-based One-Side Sampling* (GOSS) and *Exclusive Feature Bundling* (EFB). These methods are intended to improve both the exact greedy algorithm and the histogram based algorithm that XGBoost uses. GOSS works by downsampling data instances with small gradients, since those are typically well trained with small training error. EFB takes advantage of sparse feature spaces with many features that are almost exclusive to create fewer, denser features that drastically reduce training time. Bundling effectively transforms the complexity of histogram building from *O(#data * #features)* to *O(#data * #bundles)* with proven negligible effect on accuracy. Determining how to implement obliviousness for these modifications would be an interesting challenge.

SecureBoost [9] is similar to our Federated XGBoost – it extends gradient boosted decision trees to the federated

setting. It assumes that each party in the federation holds a *vertically* partitioned subset of the data; our federated mode assumes that each party holds a *horizontal* partition. Our system in coopetitive mode also provides greater security against more powerful adversaries. Additionally, we were unable to find any open source implementation of it.

3.2. Cryptographic Approaches

ID3 is a general algorithm used to create a decision tree from a given dataset. The idea is that the tree is constructed in a top-down approach; at each node, the attribute that is chosen is done so in order to maximize the information gain. Lindell et al. [17] examines a method to compute ID3 in a privacy preserving, horizontally distributed manner for two parties. Some obstacles of a traditional secure MPC approach are that Yao's generic protocol and using an oblivious transfer protocol for each bit of input would both be too inefficient for exceptionally large datasets. Computing entropy values in the ID3 protocol also requires taking logarithms, which is problematic when dealing with computation over finite fields. Instead, the paper demonstrates calculations done using an oblivious polynomial evaluation where the logarithm function is approximated using the Taylor approximation.

Hoogh et al. [11] outlines a secure ID3 algorithm implemented using the Virtual Ideal Functionality Framework (VIFF), a general software framework for doing secure MPC with Shamir secret sharing. This paper improves upon the efficiency of Lindel et al [17]. while maintaining an identical quality of decision trees as provided by vanilla ID3. The protocol works with $n \geq 3$ parties, and has the threat model of semi-honest servers where no more than $n/2$ will collude in an attempt to learn more information. The methods described work for settings where the data is horizontally distributed among the parties, much like our modifications to distributed XGBoost.

3.3. Hardware Approaches

Hardware enclaves provide a trusted execution environment that is protected against a malicious operating system. However, side channel leakage is still possible at the memory layer. Ohrimenko et al. [22] outlines oblivious variations for various machine learning algorithms, including decision trees *inference*. Oblivious primitives for assignment, comparisons, array accesses, and sorting are used to provide the building block for these new algorithms. Namely, evaluating a decision tree without obliviousness can leak the path taken through the tree. This enables an observer to infer information such as the instance itself and the size of the tree. This is prevented by storing each decision tree as a 2D array, where each individual row corresponds to the nodes at that depth. The tree is traversed by using an oblivious lookup at each one of these rows. In the case that a leaf is reached before the deepest depth, dummy nodes are traversed on the remaining layers to hide this. For ensemble tree methods, all tree decisions are accumulated obliviously into an array; the final output is the prediction with the largest weight.

4. System Overview and Threat Models

Our system can be run in two modes – federated mode and coopetitive mode. The modes represent a tradeoff between performance and security. Federated mode offers lower latencies but provides less security, while coopetitive mode has greater security guarantees but higher latencies.

4.1. Federated Mode

In federated mode, we consider a federation of size N, separated into 1 trusted aggregator and $N-1$ parties. The aggregator may or may not have data, while all other parties own sensitive data.

Together, the parties run a federated version of XGBoost to collaboratively train a model. As in federated learning, no data ever leaves its original site. Instead, over iterations, the aggregator broadcasts the current global model, which all other parties individually use their data to locally train. All local updates are sent to the aggregator, who then aggregates the $N-1$ updates into one global update and applies it to the global model.

4.2. Coopetitive Mode

In coopetitive mode, we introduce secure enclaves at every node. All computation and data processing running at the node executes inside the enclave, making them opaque to the host. Thus, even an attacker who has compromised the root and has root access cannot observe training.

However, because hardware enclaves are susceptible to side channel attacks, coopetitive mode modifies the existing XGBoost algorithms to make them oblivious. Oblivious algorithms eliminate access pattern leakage, erasing a set of attacks [13, 18] in which the adversary can infer characteristics of the data by observing access patterns.

Because all computation is hidden by the enclave, coopetitive mode requires no trust in any party. Therefore, the training paradigm can be flexible. For example, coopetitive mode is suitable for centralized training – parties can encrypt and send all their data to a centralized enclave cluster, which then aggregates all the data into one large dataset and subsequently trains on this large dataset. Coopetitive mode can also be coupled with federated mode to hide intermediate summaries and intermediate models, as the aggregation of summaries on the aggregator and the training on intermediate models all occur inside the enclave.

Depending on the use case, the final model can or cannot be inspected by all parties. Predictions can also be requested by parties. What is allowed and not allowed is a

matter of policy, and out of the scope of this paper. We assume that parties all agree on a shared policy that determines what parties can and cannot request and receive before initiating training.

4.3. Threat Models

We next present threat models of each mode of our system. Out of scope for both modes are adversaries that can launch model poisoning attacks or denial of service attacks.

4.3.1 Federated Mode Threat Model

In federated mode, we assume the presence of a trusted party that serves as the centralized aggregator during federated learning. All parties trust this party to follow the protocol, properly aggregate update summaries, and redistribute the updated model. They also trust the aggregator with summaries of their own data.

All parties other than the aggregator are honest-but-curious. They all agree to follow the federated learning protocol but are eager to observe the data of other parties.

4.3.2 Coopetitive Mode Threat Model

In coopetitive mode, we assume a powerful adversary that can compromise nearly the entire software stack of the host, including the operating system, the hypervisor, and other processes. The adversary can also gain root access to the host and modify any data or program outside the enclave. Additionally, the adversary can monitor the memory access patterns of the enclave to untrusted memory, as well as monitor network traffic, and thus can launch access pattern attacks. Other side channel attacks based on timing or power analysis are out of scope.

We assume the adversary cannot observe any processing occurring within the trusted hardware and cannot gain access to the memory of trusted hardware. We also assume that the enclave's private key is unknown to the adversary. In summary, we assume a malicious adversary with full control of the host but who has not compromised the secure enclave.

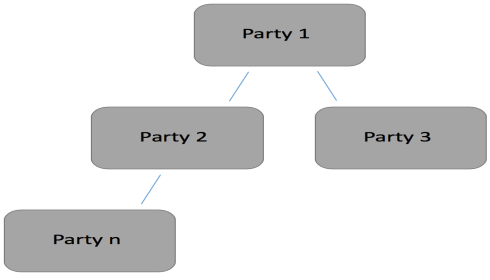

Figure 1. Tree Topology

Figure 2. Ring Topology

5. Federated Mode

The design of federated mode involved architecture and communication modifications to the original distributed XGBoost system. We do not change the underlying decision tree or gradient boosting algorithms.

5.1. Architecture

Our system requires 1 aggregator to collect summary statistics of the $N-1$ workers. To build our federated mode, we changed the node topology that determines node communication patterns. In original distributed XGBoost, node communication can either happen in a tree structure (Figure 1) or a ring structure (Figure 2). We replace both of these with a star structure, in which no node is able to communicate with another node. (Note that a party is logically a node in the federated setting). Rather, parties communicate only with the aggregator. This prevents all other parties (except the aggregator) from seeing the data summaries of any one party but allows the trusted aggregator to compute a global model during training (Figure 3).

5.2. Communication

Communication happens in two stages. On the aggregator, a process deemed the *tracker* is initialized to set up the communication pattern. First, the tracker assigns each party a numbered rank representing its position in the topology and sends that party relevant information about the federation, e.g. the URI of its parent node and the federation size. After the tracker sends this information out, the nodes all initialize socket communication with the aggregator. The second stage leverages the Allreduce paradigm [23] for parallel learning. The training process described in §4.1 is performed with Allreduce. All communication happens over TLS to protect against a network eavesdropper. Communication ends when a certain threshold of model improvement fails to be reached, at which point the workers may receive the final, global model from the master.

5.3. Information Leakage

We informally discuss the security guarantees provided by federated mode. Federated mode enables global computation over sensitive data through the transfer of summaries of model updates over the network, meaning that all data stays on site. Summaries, though observed by the aggregator, by definition contain less information than the data itself. As a result, federated mode provides greater security

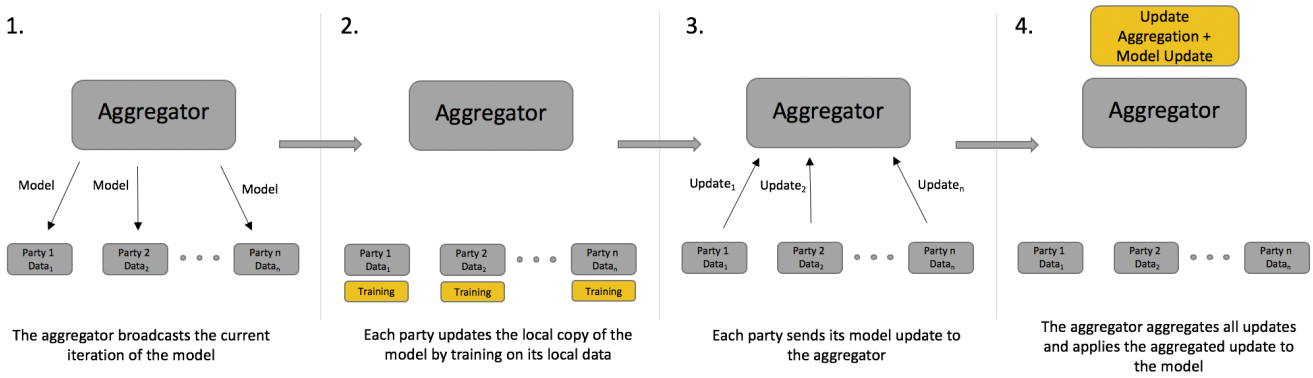

Figure 3. Federated Workflow

than a multiparty XGBoost that would involve sending data to a centralized location.

In federated mode, there are two main sources of information leakage: intermediate updates and intermediate models. Intermediate updates are sent from each party to the aggregator in each training iteration. Intermediate models are broadcasted by the aggregator to all parties, leaking the intermediate models. Therefore, all intermediate updates from all parties are seen by the aggregator, and the intermediate global model at each iteration is observed by all parties.

6. Coopetitive Mode

6.1. Architecture

In coopetitive mode, at least one enclave is present at each node. This enables all computation on sensitive data to be run in a trusted environment such that the data and the processing can never be observed by the host. The inability of the host to see the computation gives us flexibility when designing the architecture – the nodes can be logically positioned in a variety of ways, leveraging hardware enclaves for security. We present two possible architectures in the multiparty setting that take advantage of secure enclaves to provide malicious security.

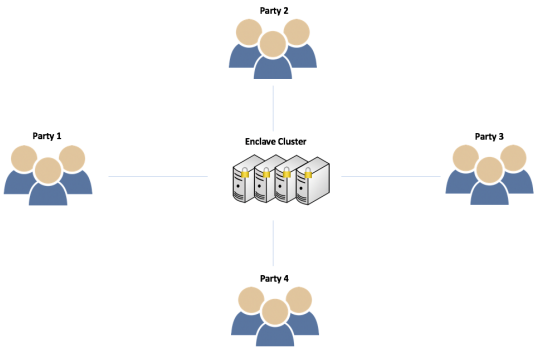

Figure 4. Centralized Coopetitive Architecture

Figure 4 showcases a centralized training scenario. All parties send their encrypted data to a centralized enclave cluster, where the data is loaded into the enclave, decrypted, aggregated, and used to train. All training occurs within the cluster, meaning that all data must leave its original site. This architecture involves a one time transfer of data between parties and the central cluster and contains a central point of attack.

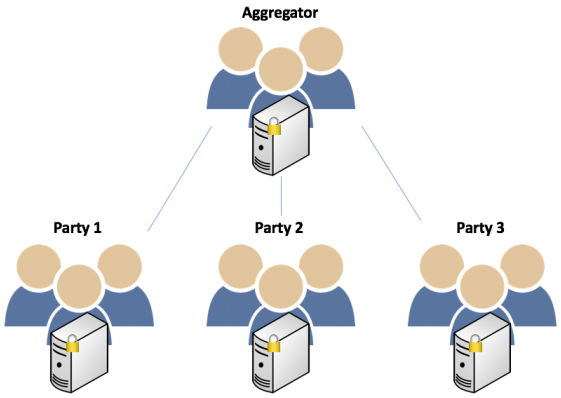

Figure 5. Federated Coopetitive Architecture

Figure 5 demonstrates hardware enclaves on top of our federated mode described in §5. Every party has an enclave: at the aggregator, all summary aggregation occurs within the enclave, and at all other parties, all intermediate models are received and further trained inside the enclave. The aggregator doesn't see the intermediate summaries, and no party sees any intermediate model.

6.2. Enclaves

6.2.1 Pre Computation

In this section we describe the process of data transfer from each party to a centralized enclave cluster. This is relevant for an architecture like the centralized training architecture mentioned in the previous section.

Before data transfer, remote attestation [26] must be performed to verify that each enclave is genuine, has not been tampered with, and that it is running at the latest security level. Second, if the enclave cluster can be trusted, data must be sent to the enclave cluster so that training can occur. This process of data transfer happens with a five step protocol.

1. Each party encrypts its data with a symmetric key k and sends it over the network with TLS to the enclave cluster.
2. Each party requests remote attestation from each enclave, which responds with a report of its identity and its public key PK_e.
3. Each party verifies each enclave report locally. This completes the process of remote attestation.
4. Each party encrypts k with PK_e, signs the ciphertext with its own public key PK_p, and sends the ciphertext and the signature to the enclave.
5. Each enclave verifies the ciphertext, decrypts the ciphertext to get k, then decyrpts the encrypted data with k

6.2.2 Computation

We leverage hardware enclaves to perform secure computation. We move computation on data that occurs in vanilla XGBoost into enclaves, making it unobservable to the host. Thus far, we support the following functionality inside enclaves:

- Model initialization
- Data matrix loading
- Model parameter setting
- Model training
- Model evaluation
- Model serialization
- Model deserialization
- Inference

6.3. Obliviousness

To remove possible side channel attacks on the Coopetitive Mode system, we modify the XGBoost split finding algorithm and summary merge operations to be data oblivious. Our contributions toward this effort thus far have been in identifying and implementing the oblivious primitives that the novel oblivious algorithms require. Specifically, XGBoost relies on an oblivious sort operation, which is built on oblivious compares, assignment, and array accesses.

Below is a code snippet for an oblivious assignment function supporting integral types, written in inline assembly.

```
// Returns t_val if pred is true else f_val
template <typename T>
T o_assign(bool pred, T t_val, T f_val) {
    T result;
    __asm__ volatile (
            "mov %2, %0;"
            "test %1, %1;"
            "cmovz %3, %0;"
            : "=&r" (result)
            : "r" (pred),
              "r" (t_val),
              "r" (f_val)
            : "cc"
            );
    return result;
}
```

The motivation behind writing this function lies in our desire to achieve the behavior of the following pseudocode:

```
if (pred)
    return x;
else
    return y;
```

However, we want to achieve this without having branches in our compiled code. One direction we tried was to use the ternary operator,

```
return pred ? x : y;
```

but when compiled with x86-64 gcc 9.2 in the online compiler explorer, Godbolt, the resulting assembly contained jump instructions. This motivates our need for custom written assembly to implement the oblivious primitives. In particular, we use the CMOVcc (conditional move) x86 instruction to implement oblivious conditional assignment. This approach was discussed in [22].

We plan to optimize these primitives, as well as implement efficient oblivious array access using vectorized instructions.

6.4. Information Leakage

Because all computation on sensitive data occurs within enclaves, no information is leaked. Furthermore, obliviousness makes the system resistant to access pattern leakage attacks, eliminating a set of potential inference attacks on side channels. Though other side channels like timing and power analysis attacks exist, they are out of the scope of this paper.

7. Implementation

Thus far, we've implemented prototypes for both federated and coopetitive mode. Below we discuss in more detail the implementation of each.

7.1. Federated Mode

Federated mode is written in Python and C++. Much of the tracker code for topology initialization is written in Python, while Allreduce and the XGBoost algorithms are in C++. We provide a Python API as an interface to our system.

```
# Instantiate FederatedXGBoost
fxgb = FederatedXGBoost()

# Get number of federating parties
print(fxgb.get_num_parties())

# Load training data
fxgb.load_training_data('hb_train.csv')

# Train a model
params = {'max_depth': 3}
num_rounds = 40
fxgb.train(params, num_rounds)

# Load the test data
fxgb.load_test_data('hb_test.csv')

# Evaluate the model
print(fxgb.eval())

# Get predictions
ypred = fxgb.predict()

# Save the model
fxgb.save_model("fxgb_model.model")

# Shutdown
fxgb.shutdown()
```

Federated mode is logically split into two stages. First, the aggregator initializes connections with all parties and sends a request to start training. Second, all parties perform federated training. Stage 1 leverages gRPC [1] to enable the aggregator to remotely commence training on all parties' machines. Stage 2 leverages Mbed TLS [2] to encrypt all communication between the aggregator and the parties.

Thus far, Federated XGBoost has required about an additional 100 lines of code over the existing XGBoost system. However, we do not yet have a fully functioning integration of TLS.

7.2. Coopetitive Mode

Coopetitive mode is written in Python, C++, and x86 assembly. We support a Python frontend for ease of use with a C++ backend – the C++ encapsulates all the cryptography, the data processing functionality within enclaves, and the remote attestation logic. We implement some of the oblivious primitives with assembly code. Below is example usage of our Python API

```
enclave = xgb.Enclave("xgboost_enclave.signed")

# Remote Attestation
enclave.get_remote_report_with_pubkey()
enclave.verify_remote_report_and_set_pubkey()

dtrain = xgb.DMatrix("train.encrypted")
dtest = xgb.DMatrix("test.encrypted")

booster = xgb.Booster()
booster.set_param(params)

for i in range(n_trees):
  booster.update(dtrain, i)
  booster.eval_set([(dtrain, "train"), (dtest, "
    test")], i)

booster.predict(dtest)
```

In our implementation, we use the Open Enclave SDK [3] to integrate enclaves with XGBoost, Mbed TLS [2] for cryptography and gRPC [1] to enable party/central-cluster architecture. We've also tested our system on Microsoft Azure Confidential Computing, a cloud offering of Intel SGX enclaves.

We have not yet fully added TLS to coopetitive mode. Obliviousness has also not been completely implemented. The distributed version of coopetitive XGBoost is also still buggy. Our implementation thus far has required about 4,000 lines of additional code.

8. Evaluation

We use the Higgs Boson [4] and Allstate Insurance Claim [5] datasets for evaluation, as the original XGBoost paper [8] also did. The Higgs Boson dataset has 27 features, while the Allstate Insurance Claim dataset has 34 features. We present both accuracy and performance numbers for federated and coopetitive mode.

8.1. Federated Mode

Our evaluations regarding federated mode pertain to measuring both accuracy and training time for the star topology and comparing metrics to training using the original tree topology. These results are simulated locally on a Macbook Pro running Catalina 10.15.1 with 16 GB of RAM and a 2.3 GHz 8-Core Intel Core i9-9880H CPU by running 7 "workers" in different processes. As a result, metrics in this experiment do not include network latencies. Each worker node has around 140,000 rows of data, randomly selected from the Higgs Boson dataset[4].

Figure 6 shows the results of these experiments. Both training time and accuracy in federated mode are comparable to those of vanilla XGboost.

	Training Time	Accuracy (AUC)
Tree Topology	6.4056327343 secs	0.7901876899
Star Topology	6.3726212978 secs	0.7899275457

Figure 6. Tree vs Star Topology Metrics

8.2. Coopetitive Mode

Since our coopetitive mode implementation does not yet support obliviousness, the numbers we present for coopetitive mode do not include the overhead of obliviousness. Usually, obliviousness adds a 2x-10x overhead to existing performance.

We ran our coopetitive experiments on Microsoft Azure Confidential Computing, on machines with 4 vCPUs and 16 GB RAM. Secure XGBoost ran on Standard DC4s machines with Intel SGX enclaves, while vanilla XGBoost ran on Standard D4s v3 machines.

Because our enclave integration and oblivious algorithms do not touch the learning part of XGboost, we do not expect the accuracy to differ at all. This is reflected in our measurements, as the accuracies we obtain for the Higgs Boson data and the Allstate Insurance claim data on test sets are the same for coopetitive XGboost vs vanilla XGBoost. For Higgs Boson, both our coopetitive implementation and the vanilla implementation achieve an AUC of 0.7461556 after training on 1 million rows. For the Allstate dataset, both our coopetitive implementation and the vanilla implementation achieve an AUC of 0.7830182 after training on 1 million rows.

We also present performance metrics for both the Higgs Boson and Allstate Insurance claim datasets for our coopetitive implementation. We compare training times of our coopetitive implementation (with Intel SGX hardware enclaves but no obliviousness) to the vanilla implementation on various data sizes. Our results are in Figure 7 and Figure 8.

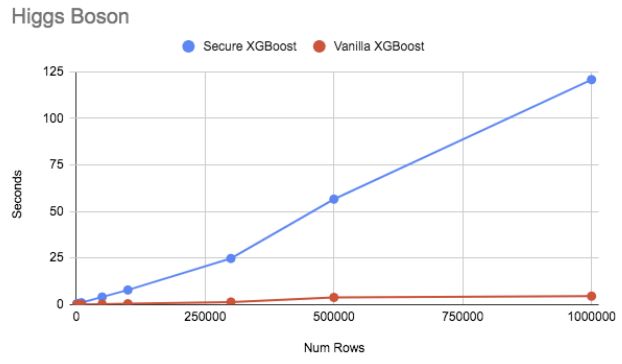

Figure 7. Benchmarking training times on Higgs Boson dataset, comparing Secure XGBoost vs Vanilla XGBoost. We measure the training time on a varying number of rows.

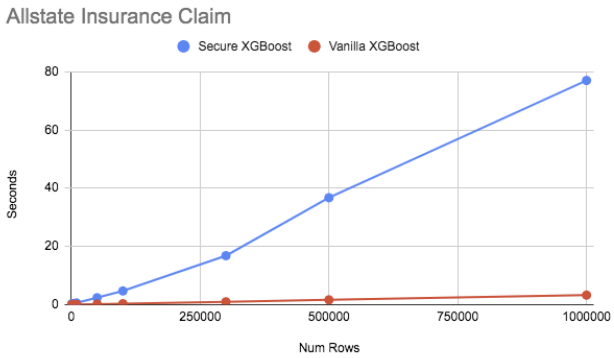

Figure 8. Benchmarking training times on Allstate Insurance Claim dataset, comparing Secure XGBoost vs Vanilla XGBoost. We measure the training time on a varying number of rows.

9. Real World Use Cases

We've been fortunate enough to collaborate with some folks in industry, in the realms of banking and network service providers. Unfortunately, due to NDA, we cannot discuss the details of our collaboration in this paper.

10. Future Work

Our first next steps are to finish implementations of both federated mode and coopetitive mode. Federated mode is nearly complete – we just need to finish supporting encrypted communication. Coopetitive mode requires a bit more work – we need to finish adding obliviousness, debugging our distributed implementation, and support encrypted communication.

Once our implementations are complete, we'll need to optimize our systems for more practical performance. Though our systems do provide strong security guarantees, the faster we can make them run, the better.

The issue of authentication is also present in our system. In federated mode, how does the aggregator ensure that the parties are indeed part of the federation? How do the parties know that the aggregator is their trusted aggregator? In coopetitive mode, some of the same issues arise, but in a party-to-party manner. Either our system needs to adopt and integrate a public key infrastructure, or we need to rely on an existing one and trust that the users of our system can properly use it.

The field of secure collaborative learning has much potential for exciting new work. To us, there are two main directions that secure collaborative learning is heading – towards hardware enclaves and towards secure multiparty computation (MPC) [6]. Some directions to go include redesigning other existing machine learning frameworks to run inside hardware enclaves, designing MPC frameworks that support specific machine learning algorithms, or even designing new machine learning libraries with security in

mind. We believe that current and novel work in secure collaborative learning will yield great benefits, enabling data that could previously not be shared to be leveraged to produce better performing models.

11. Conclusion

In this paper we presented Secure XGBoost, a secure system built on top of XGBoost that enables collaborative learning by multiple parties on sensitive data. Our system offers two modes: federated mode, which requires a trusted centralized aggregator but offers better performance, and coopetitive mode, which leverages secure enclaves and obliviousness to protect against malicious adversaries. Development on both systems is currently in progress, and the code is open source at https://github.com/mc2-project/mc2.

12. Acknowledgement

Special thanks to Rishabh Poddar, Wenting Zheng, and Raluca Ada Popa for their work and guidance on this project.

13. Individual Contribution

As this paper was submitted for a class project, Chester must discuss his contributions to the project below.

- Federated mode design
- Coopetitive mode design
- Integrating Open Enclave with existing C++ XGBoost code
- Linking a Python interface on top of C++ enclave-integrated code to provide a simple API for users
- Writing a Python API for the client to perform remote attestation, send his keys to the enclave, and initiate training in coopetitive mode with a centralized enclave cluster
- Coopetitive mode benchmarking
- Prepared a tutorial on this project for RISE camp (https://risecamp.berkeley.edu/)
- Designing the entire poster for the 262A poster session
- Writing most of this paper
- Maintaining open source MC^2 repo (https://github.com/mc2-project/mc2)
- Collaborating with multiple industry users through meetings

References

[1] https://github.com/grpc/grpc. 7

[2] https://github.com/ARMmbed/mbedtls. 7

[3] https://github.com/openenclave/openenclave. 7

[4] https://archive.ics.uci.edu/ml/datasets/HIGGS. 7

[5] https://www.kaggle.com/c/ClaimPredictionChallenge. 7

[6] David W Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P Smart, and Rebecca N Wright. From keys to databases—real-world applications of secure multi-party computation. *The Computer Journal*, 61(12):1749–1771, 2018. 8

[7] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *2010 IEEE Symposium on Security and Privacy*, pages 191–206. IEEE, 2010. 2

[8] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016. 1, 2, 7

[9] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. Secureboost: A lossless federated learning framework. *CoRR*, abs/1901.08755, 2019. 2

[10] Victor Costan and Srinivas Devadas. Intel sgx explained. 1, 2

[11] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. Practical secure decision tree learning in a teletreatment application. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, pages 179–194, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. 3

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

[13] Tara Merin John, Syed Kamran Haider, Hamza Omar, and Marten Van Dijk. Connecting the dots: Privacy leakage via write-access patterns to the main memory. *IEEE Transactions on Dependable and Secure Computing*, 2017. 3

[14] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. white paper. 2016. 2

[15] Pratheek Karnati. Data-in-use protection on ibm cloud using intel sgx. 2

[16] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, 2017. 2

[17] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *IACR Cryptology ePrint Archive*, 2008:197, 11 2008. 3

[18] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-An Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265:176–188, 2014. 3

[19] Scott Lundberg. Interpretable machine learning with xgboost. 1

[20] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2016. 1

[21] Vishal Morde. Xgboost algorithm: Long may she reign!, 2019. 1

[22] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, Austin, TX, Aug. 2016. USENIX Association. 3, 6

[23] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009. 4

[24] Mark Russinovich. The rise of confidential computing. 2

[25] George Seif. A beginner's guide to xgboost. 1

[26] Hiie Vill. Sgx attestation process. 6

[27] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 283–298, 2017. 2

[28] Wenting Zheng, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. *CoRR*, abs/1907.07212, 2019. 2