Privacy-Preserving Tree-Based Inference with Fully Homomorphic Encryption

Jordan Frery, Andrei Stoian, Roman Bredehoft, Luis Montero, Celia Kherfallah, Benoit Chevallier-Mames, and Arthur Meyre

Zama **

Abstract. Privacy enhancing technologies (PETs) have been proposed as a way to protect the privacy of data while still allowing for data analysis. In this work, we focus on Fully Homomorphic Encryption (FHE), a powerful tool that allows for arbitrary computations to be performed on encrypted data. FHE has received lots of attention in the past few years and has reached realistic execution times and correctness.

More precisely, we explain in this paper how we apply FHE to tree-based models and get state-of-the-art solutions over encrypted tabular data. We show that our method is applicable to a wide range of tree-based models, including decision trees, random forests, and gradient boosted trees, and has been implemented within the Concrete-ML library, which is open-source at https://github.com/zama-ai/concrete-ml. With a selected set of use-cases, we demonstrate that our FHE version is very close to the unprotected version in terms of accuracy.

1 Introduction

Over the past decade, machine learning (ML) has become a powerful tool to solve various types of problems, such as facial recognition, image classification, and text prediction, among many others. This is due to the advancements in hardware and data availability, which have allowed for training of increasingly complex models. Additionally, the development of deep learning techniques such as convolutional neural networks has played a significant role in the success of ML in various domains.

On the importance of tree-based models in Machine Learning. In ML, tree-based models have a place of choice, since they are particularly fit for tabular-data [GOV22]. They are particularly easy to train as they do not need specific feature engineering or pre-processing. Additionaly, they are scale-invariant unlike neural networks where input features scale is very important. Furthermore, open-source libraries such as scikit-learn or xgboost make it even easier for machine learning practitioners to use them.

** `hello@zama.ai` http://zama.ai

The need for privacy. In their wide range of applications, machine learning models sometimes need to deal with sensitive data. For instance, facial recognition systems use people's pictures, which are usually considered confidential information. Another critical example is healthcare applications, which are still years behind the state-of-the-art due to confidentiality issues. Most hospitals have restrictions on sharing patient data with third parties, which often prevent them from using the state-of-the-art in data processing. Performing inference on encrypted data would be a powerful ability to protect the privacy of data while still allowing for accurate predictions.

Existing art. The goal of privacy-enhancing technologies (PET) — also named privacy-preserving ML (PPML) —[ARC19], is to allow for computations to be performed on encrypted data. There are several directions to handle privacy-preserving ML, including multi-party computation (MPC) [Gol98] and fully homomorphic encryption (FHE) [Gen09b].[1] In our paper, we focus on FHE, where encryption functions are designed to enable complex computations to be carried out on ciphertexts, without the need for any secret information. This allows for the execution of virtually complex computations on untrusted servers.

The first realization of fully homomorphic encryption (FHE) was introduced by Gentry [Gen09b,Gen09a] in 2009, and notably mentions for the first time a very important feature for FHE, namely the *bootstrapping*. Bootstrapping allows to reduce the noise of a ciphertext, once a certain number of homomorphic operations have been done, to be able to do more operations. Thanks to this operation, homomorphic schemes were no more limited to a given number of operations, and it was virtually possible to apply an infinite number of operations without losing correctness.

A number of methods have been proposed for performing tree-based inference with FHE. (see, e.g., [ALR$^+$22,TBK20]). Most of these methods rely on CKKS [CKKS17] for approximate arithmetic and BGV [BGV12] or BFV for modular arithmetic. One drawback of these approaches is that the machine learning model has to be approximated using polynomial approximations. This is a strong constraint whereas in TFHE — the cryptographic scheme that we use [CGGI16,CGGI17,CGGI20] —, we can replace any non-linear function (e.g. comparisons) by so-called *programmable bootstrapping* [CJP21]. Other attempts such as [MF20] rely on additive homomorphic encryption [Pai99] and rely on order-preserving encryption which allows them to compare encrypted data in the clear. This raises many privacy flaws [BCO11] in the context of machine learning where encrypted data could be analysed by the server directly.

Our contribution. In this paper, we present a new method to run tree-based inference with FHE. Our technique can easily be used — amongst other models — with any tree-based model. We implemented our method in Concrete-ML

[1] In this paper, we restrict ourselves to secure *inferences*. Secure *training* is also a field of research, using additional techniques such as Differential Privacy [Dwo06] or Federated Learning [BEG$^+$19].

library [MCMF+2], which provides an easy-to-use interface for working with FHE-friendly machine-learning models. Important to say, Concrete-ML is open source and free-to-use for research and non-commercial uses.

We demonstrate our method on a variety of tree-based models, including decision trees, random forests, and gradient boosted trees. We believe that it can be of interest to practitioners who wish to perform privacy-preserving inference on tabular data, a setting where ensemble methods based on trees still yield state-of-the-art results.

The plan of our paper is as follows: We first give an overview of FHE, TFHE and tree-based models. We then describe our method for tree-based inference with FHE. After that, we show our experimental results on a wide variety of datasets. We conclude with a discussion.

2 Background

2.1 Fully Homomorphic Encryption

What is FHE. An encryption scheme f is said to have *homomorphic* properties when there exist two operations $\cdot$ and $\blacksquare$ such that

$$f(a \cdot b) = f(a) \blacksquare f(b),$$

for all valid messages a and b. Often $\cdot$ and $\blacksquare$ are the same operations. Homomorphic schemes are known since the beginning of public-key cryptography, since for example RSA has multiplicative homomorphic property [RSA78]. Additive homomorphic schemes as Paillier are also known [Pai99] and have been widely used, for example in voting schemes.

Fully homomorphic encryption schemes are homomorphic for more than one operation, and ideally, for a wide set of operators. It has been much more complicated for the cryptographic community to build such a scheme, and one had to wait for Gentry's breakthrough [Gen09b,Gen09a] to have first implementations of FHE. One key ingredient of his construction (and of those which followed) is the so-called *bootstrapping*, which allows to reduce the noise in a ciphertext. Then several generations of FHE schemes which are both secure and practical have been proposed, notably BFV [Bra12,FV12], GSW [GSW13], BGV [BGV12,BGV14], FHEW [DM15], CKKS [CKKS17], and TFHE [CGGI16,CGGI17,CGGI20].

FHE and ML. FHE has been identified as a great tool for privacy-enhancing technology, notably thanks to its security and the fact that it doesn't really change the protocol (as opposed to, for example, other techniques such as multi party computations). In the TFHE scheme, programmable bootstrapping (PBS) allows for the utilization of table lookup (TLU) during the bootstrapping process without incurring additional cost, as noted in [CJP21]. Thus, with PBS, it is possible to replace non-linear functions — typically activations — by TLUs.

2.2 Our Use of TFHE and Corresponding Constraints.

TFHE is a very convenient cryptographic scheme, but, as the other FHE schemes, it comes under some constraints. We refer the reader to [Joy21] for more information about the reasons, and just list the most important points for us here.

1. Every input value and intermediate value within the model must be an integer type. This is not a hard constraint of TFHE (which works over the reals modulo 1) but a choice was have made, to use an *exact approach*,[2] i.e., to have computations in FHE which are always exact.[3]
2. The maximum precision we can handle is 16 bits.[4] This is also true for all intermediate values (e.g., accumulators where the dot products are computed).
3. Conditional operations are not possible with FHE scheme in general, and so, with TFHE in particular. We explain in Section 3.2 how we are able to handle this difficulty with the PBS, which is a tool only available in TFHE (so far).
4. Due to the linear homomorphism in TFHE, additions, subtraction and multiplications by constants are easy to do directly on ciphertexts. All univariate functions are also easy to do with PBS.[5]. Multivariate functions (typically, MaxPool in ML) are more complicated to handle, and are out-of-scope of this paper.

2.3 Decision Tree, Random Forest and XGBoost

Tree-based models are a popular class of machine learning models that are used for a variety of tasks, including classification and regression. They are attractive as they are relatively interpretable [GBY+18], easy to use (thanks to popular libraries such as SCIKIT-LEARN), and still state-of-the-art models when it comes to accuracy over tabular datasets [SZA22]. Decision trees are a specific type of tree-based model that are commonly used for classification or regression tasks in which each internal node represents a test on an input feature, each branch represents the outcome of the test, and each leaf node represents a class label.

Decision trees can be easily trained using a variety of algorithms, including the popular CART algorithm [BFOS17]. There are a lot of different popular algorithms that train decision trees or an ensemble of decision trees such as random forest or gradient boosting [CG16]. Once trained, tree-based models can be used for inference by traversing the trees from the root node to a leaf node, making a prediction based on the class and then applying a weighted sum of the trees outputs.

[2] By opposition to the so-called *approximate approach*. We refer the interested reader to "Approximate vs Exact approaches" Section in [Zam22a] for more information about this choice.

[3] Always because "always except a very small probability".

[4] This limitation comes from the foundations of our dependency, namely the CONCRETE-LIBRARY [Zam21]

[5] As long as the input range is small enough to be on 16 bits.

2.4 Quantization

Principle. Quantization is a process of converting a continuous signal into a discrete signal. A common example of quantization is converting a signal from an analog format to a digital format. In the context of neural networks, quantization refers to the process of converting weights and activations from floating point values to integer values. This can be done using a variety of methods, in particular uniform quantization [JKC+18]. Quantization can be used to improve the efficiency of neural networks, both in terms of memory usage and computational speed. It can also be used to reduce the amount of data that needs to be stored and transmitted, which is important for applications such as mobile devices or embedded systems. In our context, the interest is to obtain a final model that contains and operates only over integers.

Symmetric quantization. The straightforward approach is to use uniform quantization as follows:

$$q(x) = \mathsf{round}\left(\frac{x}{\Delta}\right) \tag{1}$$

where x is a real number, $q(x)$ is the *quantized value* and Δ is the step size also called the *scale*.

An appropriate step size taking the max and min in consideration can be computed as follows:

$$\Delta = \frac{\mathsf{max}(x) - \mathsf{min}(x)}{2^p - 1} \tag{2}$$

where p is the number of bits that will be used to represent the quantized values. In this case, the quantized values will be integers between -2^{p-1} and $2^{p-1} - 1$.

These equations satisfy the following property:

$$q(0) = 0 \tag{3}$$

or in other words, the quantization of the values zero in floating point is equal to zero once quantized. This is an important property when working with binary matrices or with sparse models (e.g. neural network subject to pruning).

Asymmetric quantization. Symmetric quantization is great as long as the distribution of the floating point values are symmetric around zero. In some cases, the distribution does not satisfy this property and we rather use asymmetric quantization with a *zero point* value, defined as follows:

$$q_a(x) = \mathrm{round}\left(\frac{x}{\Delta} + z_p\right) \tag{4}$$

The zero point is typically chosen such that the minimum of x becomes the integer 0 after quantization, i.e., $q_a(\mathsf{min}(x)) = 0$.

This asymmetric quantization allows better use of the available precision as it take into account the whole range.

3 Transforming Tree-Based Models into FHE

In this section, we present our technique for converting tree-based models into a privacy-preserving setting using Fully Homomorphic Encryption (FHE). As outlined in Section 2.1, FHE imposes certain limitations and constraints, particularly in regard to condition and flow-operations.

Our method consists of three steps:

- quantizing float variables to integers, as detailed in Section 3.1.
- replacing conditions with table lookups, as discussed in Section 3.2.
- performing all computations in a vectorized manner, as outlined in Section 3.3.

In Section 3.3, we demonstrate how our technique can be applied to a tree-based models to make it FHE-compatible. Our implementation and experimental results can be found in Section 4 and are reproducible using the open-source repository Concrete-ML.

3.1 Our Use of Quantization

In this section, we describe how we utilize quantization in our algorithm.

Choosing asymmetric quantization. Asymmetric quantization is preferred for its greater precision, as it does not assume a symmetrical distribution of values around zero. This approach is used for converting input features.

Since a tree-based model does not perform linear combinations of the inputs, like linear models or a neural networks, we can safely quantize every single feature independently of each other. This allows us to have a scale and zero point for each feature which is a great advantage when the input dimensions follow different distributions.

Tree-based model quantization. Our input space is fully quantized, leaving us with integers only. This means that our tree-based model can be trained on this new input space, resulting in quantized split decision thresholds. In practice, when a split is chosen, the training algorithm (e.g. GINIGAIN, ENTROPY or XGBOOST) selects the floating point numbers between two observed input feature values and then applies a "strictly greater" or "strictly lower" comparison. To convert this into an integer-only problem, we use the ceil or floor function and convert the comparison into a *greater or equal* and *lower or equal* comparison depending on the algorithm at use. Finally, we apply another asymmetric quantization to each terminal leaf value.

The quantization of operations in neural networks requires adherence to specific rules [JKC+18], making the process complex. However, in tree-based models, this complexity is avoided as the tree is learned over quantized input directly. As a result, the issue of propagating the scale and zero point through quantized operations to facilitate dequantization is not present in tree-based models.

At this point, our FHE tree-based model is fully quantized. However, as we mentioned previously, control-flow operations are not directly possible in FHE, and require a different approach, which is explained in next section.

3.2 Conditional Operations

Conditional operations are not directly possible in FHE. To circumvent this limit, we use the table lookup (TLU) operation [CJP21], which is currently an exclusive feature of TFHE.

Consider a two-dimensional integer input space where each data point $x \in X$ belongs to the set $[0, 2^p)^2$. Here, p represents the number of bits used to encode the features of x. Let the first feature of x be denoted as $x^{(1)}$ and the second feature as $x^{(2)}$, with $p = 3$ in this example. Our goal is to classify each data point $x \in X$ into two classes, C_0 and C_1, by finding a function f such that x is assigned to $C_{f(x)}$ by our algorithm.

A simple boundary such as $x^{(2)} > 3$ can be represented by the decision stump (tree depth of 1) and expressed as follows:

$$f(x) = \begin{cases} 0, & \text{if } x^{(2)} > 3 \\ 1, & \text{otherwise} \end{cases}$$

Such an algorithm uses an if, which is not FHE compatible, but it turns out that a lookup table can achieve similar results:

$$f(x) = T[x], \text{ with } T = [1, 1, 1, 1, 0, 0, 0, 0]$$

In TFHE, such TLU is directly converted into a programmable bootstrapping

3.3 Tree Traversal conversion to Tensor Operations

The tree traversal approach is common to run a tree inference. However, in FHE, it is not possible as control-flow operations are not supported. Selecting which branch to run from the encrypted data is thus not possible. To overcome this, we compute every branch simultaneously by converting the tree traversal into tensor operations. Replacing a circuit with control-flow operations by a circuit without any branch is something which is already done to accelerate computation over specific hardware (e.g. GPU). Thus, we use such implementation [Mic22] for the conversion in our work.

Algorithm 1 lists the tensors used in the conversion process as well as the process itself.

Algorithm 1 GEMM Strategy for Tree Scoring

Require

- X: input features to internal nodes
- A: threshold value of each internal node
- B: threshold value
- C: relationship between leaf nodes and internal nodes
- D: count of internal nodes in the path from a leaf node to the tree root
- E: mapping between leaf nodes and class labels

Step 1: $P \leftarrow X \cdot A$
Step 2: $Q \leftarrow P < B$
Step 3: $R \leftarrow Q \cdot C$
Step 4: $S \leftarrow R == D$
Step 5: $T \leftarrow S \cdot E$
Return: T

Q and S are matrices of booleans, while P, R and T are matrices of integers. A, B, C, D, E are obtained from the training and are also matrices of integers.

The algorithm involves five main steps:

Step 1: Creation of the input path tensor P. This involves multiplying the input tensor X with a tensor A that captures the relationship between input features and internal nodes.

Step 2: The result is then compared with a tensor B representing the values of the internal nodes. The comparison operation essentially assigns a binary value to each element in the input path tensor, indicating whether the corresponding internal node is satisfied. This creates tensor Q.

Step 3: Creation of the output path tensor R. This involves multiplying the tensor result of the previous operation Q with a tensor C that captures the relationship between the internal node and the left/right sub-tree.

Step 4: Tensor R, representing the output path, is compared to tensor D that keeps track of the count of left child nodes along the path from a leaf node to the root of the decision tree. The comparison yields tensor S, which indicates matching paths.

Step 5: Prediction generation. This involves multiplying the matching paths tensor S with a tensor E that maps the leaf nodes to produce the final prediction.

4 Experimental Results

In this section we describe the experiments we performed to evaluate our method using Concrete-ML .[6]

[6] At the time of writing we used the available public version 0.6.1.

Accuracy of quantized models. The first important point is to show the accuracy of quantized models as defined in Section 3.1 vs the floating point (FP32) models built with the original library.

FHE inference. Fully homomorphic inference has a main drawback of having a significantly longer inference time than the clear model inference (see also Section 4.4). This execution time depends mainly on two factors: the parameters associated to the cryptography security and the complexity of the FHE circuit. As the security level is constant,[7] cryptographic parameters depends mainly on the precision we use (i.e., the upper value of the bit width of intermediate values) while the complexity of the FHE circuit is directly correlated to the complexity of the model, or in other words, the hyper-parameters.

We provide experiments for three different types of tree-based models for classification as they are the most popular within the machine learning community [BRAN19]:

- DecisionTreeClassifier from SCIKIT-LEARN library.
- RandomForestClassifier from SCIKIT-LEARN library.
- XGBoostClassifier from the XGBOOST library [CG16].

4.1 Implementation in Concrete-ML

In CONCRETE-ML, the flow has been made easy for any data-scientist to have their usual tasks remain unchanged. User can use the API to train models and predict similarly as in the SCIKIT-LEARN. In summary, the process of converting the model to its FHE equivalent involves the following steps.

A tree-based model is trained over quantized data using one of the parent library (namely, SCIKIT-LEARN or XGBOOST). The model is then exported to ONNX using HUMMINGBIRD, decision thresholds are converted to integer and prediction values (in terminal leaves) are quantized. Both input and output quantizer remain at the user's disposal as they are needed to pre/post process the data before and after FHE execution. The ONNX model is then converted to NUMPY functions and given to CONCRETE-NUMPY for compilation and bounds measurements. An optimizer is run to compute the optimal cryptographic security parameters (a necessary information for public/private key generation). The FHE binary is produced by CONCRETE-COMPILER that implements every FHE operations.

Once the model is trained and compiled, the user can use CONCRETE-ML Python API to encrypt, quantize/dequantize and run the FHE execution easily. One can refer to our [Zam22b] available in our repository, where we handle Spam detection in FHE on a real dataset.

[7] Currently in CONCRETE-ML, security level is forced at 128-bit in CONCRETE-ML.

4.2 Quantization Precision And FHE Inference Time.

First we study the impact of quantization on models accuracy. In Figure 1, we compute the f1 score and average precision (AP) for different bit width (quantized precision) and plot it along with the metrics from the floating point value model. For the three different tree-based model, we can observe a convergence of the quantized model toward the full precision model.

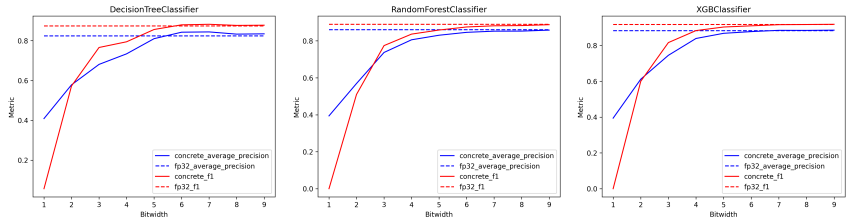

Fig. 1. Experiment reporting the f1-score and average precision with varying precision on the spambase dataset.

Such a behavior is expected and the natural choice would be obviously to select the highest bit width. However, FHE has its execution time impacted by a change in the precision. To have a better understanding of this quantization parameter impact, we run an experiment in Figure 2 where we compute the FHE inference time for the three models at different quantization precision.

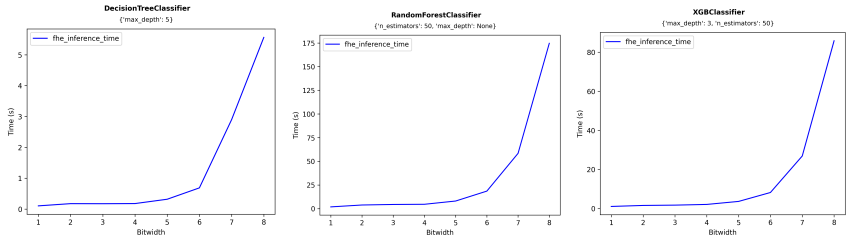

Fig. 2. FHE inference time for different bit widths.

Figures 1 and 2 provide a good overview for the trade-off between model accuracy vs. FHE inference time. We can see that a great increase in FHE inference time is found to be between starting at 7 bits. On the other hand, 5 and 6 bits gives us metric very close to the FP32 model: ¡2% drop in the metrics reported for 6 bits precision. In the following section, we use 6 bits as our quantization precision for both the input features and output tree values.

4.3 Experiments on Various Datasets

Table 1 presents the average and standard deviations (with the format AVERAGE, STD in the table) results of 5-fold cross validation repeated 3 times (or 15 runs in total per model per dataset).

Hyper-parameters used for both the FHE and sklearn models are:

- n_estimators is set to 50 and defines the number of trees in the ensemble for both Random Forest (RF) and Gradient Boosting (XGB).
- max_depth is set to 5 for both the decision tree (DT) and XGB. Random forest is left to None such that the tree can fully expand. This parameter defines the maximum depth a tree can have once fully trained.

		accuracy	f1	AP	Time (s)	FHE/FP32 ratio
spambase	FHE-DT	0.91, 0.01	0.88, 0.01	0.84, 0.02	0.687	344x
(#features: 57)	FP32-DT	0.9, 0.01	0.87, 0.01	0.82, 0.02	0.002	
	FHE-XGB	0.93, 0.01	0.91, 0.01	0.88, 0.02	7.950	3975x
	FP32-XGB	0.94, 0.01	0.92, 0.01	0.88, 0.01	0.002	
	FHE-RF	0.91, 0.01	0.88, 0.02	0.85, 0.02	16.589	8294x
	FP32-RF	0.92, 0.01	0.89, 0.01	0.86, 0.02	0.002	
adults	FHE-DT	0.85, 0.0	0.62, 0.01	0.52, 0.01	0.794	794x
(#features: 14)	FP32-DT	0.85, 0.0	0.63, 0.01	0.52, 0.01	0.001	
	FHE-XGB	0.85, 0.0	0.63, 0.01	0.53, 0.01	8.772	8772x
	FP32-XGB	0.85, 0.0	0.64, 0.01	0.54, 0.01	0.001	
	FHE-RF	0.83, 0.01	0.52, 0.02	0.47, 0.01	17.540	17540x
	FP32-RF	0.84, 0.01	0.53, 0.02	0.48, 0.01	0.001	
wine	FHE-DT	0.92, 0.04	0.89, 0.05	0.82, 0.08	0.346	346x
(#features: 13)	FP32-DT	0.95, 0.03	0.93, 0.05	0.88, 0.09	0.001	
	FHE-XGB	0.97, 0.02	0.96, 0.02	0.94, 0.04	5.121	5121x
	FP32-XGB	0.97, 0.02	0.96, 0.03	0.94, 0.05	0.001	
	FHE-RF	0.99, 0.01	0.98, 0.02	0.97, 0.04	8.531	8531x
	FP32-RF	0.99, 0.01	0.98, 0.02	0.97, 0.04	0.001	

Table 1. FHE vs FP32 tree-based experiments. The accuracy, f1-score and average precision (AP) are averaged over 15 runs and reported along with the standard deviation. The inference time per model is reported in the Time columns and finally the execution time ratio between FHE and FP32 model are reported in FHE/FP32 ratio column.

Accuracy between FP32 and FHE is closely matched for every dataset. The wine dataset shows the largest variance in all metrics. This can be caused by its limited sample size of only 178 examples which makes it challenging properly represent with quantization as only a fraction of these examples are presented to the model for each run. This is reflected by the higher standard deviation seen in all models and metrics for this particular dataset. For optimal quantization outcomes, input features should exhibit a balanced distribution and have a proper representation of the data in the training set.

Execution time is a crucial factor for machine learning practitioner when considering the use of FHE. The experiment shows interesting properties of the tree-based models. Decision tree, with its limited depth of 5 and single tree structure, has the fastest average FHE execution time with 0.7s. Wine again shows a faster execution time (0.3s), likely due to its few data points available as the maximum depth of 5 is not necessary for complete dataset classification. The ratio FHE/FP32 time execution is roughly 1000x. On the other hand, the random forest model exhibits the slowest FHE execution time, being roughly 10000 times slower than its FP32 counterpart. This outcome is predictable, as the unrestricted depth in the construction of random forests results in a substantial number of internal nodes. In contrast, gradient boosting offers the best of both worlds, combining small trees that makes the inference faster and yet as accurate as random forest if not more.

4.4 Take Aways

Our solution is a big step to achieve privacy-preserving ML for tree-based algorithm, with goodness of fit which is very close to the SCIKIT-LEARN original one. We provide a thorough analysis of its strengths and limitations.

Cons. First, we have to use large elements. One drawback of the GEMM approach to tree-based inference is that the size of the matrices grows linearly with the number of internal nodes. This can make the matrices very large, and can be a problem when there are many trees in the ensemble or if the depth is too high. For example, a single fully developed tree of depth 12 has $2^{12} - 1$ internal nodes and 2^{12} terminal nodes. This creates matrices of size 4096×4095. In an ensemble tree-based model, if trees all have the same depth and are fully developed, the matrix dimensions grows linearly with the number of trees. That being said, trees are very rarely, if not never, fully developed. Gradient boosting models such as XGBOOST should be preferred as they tend to group trees of small depth compared to e.g. random forest models.

Second, FHE in general and PBS in particular can be slow depending on the precision asked and hardware used. In the future, we plan to use GPU, FPGA and other hardware accelerations[8] to have execution times which are much more comfortable.

Pros. Our solution offers numerous advantages. To the best of our knowledge, it is the first tree-based solution to offer (i) deep customization, (ii) compatibility with ensemble methods, and (iii) outstanding accuracy.

Easy to use: it shouldn't take more than a couple of minutes for a data scientist to master the CONCRETE-ML package [MCMF+2], since its APIs were designed to be very close to well-known SCIKIT-LEARN. We hope that having a very user-friendly framework (where one does not need to known anything about cryptography, notably) is really a plus.

[8] Such that optics [Opt21]

Security is directly handled under the hood: cryptographic parameters don't have to be set by hand, which is very fastidious and can lead to big problems, since badly chosen parameters are insecure. In our tools, as opposed to others, the user doesn't have to take care of this.

Finally, our solution demonstrate remarkable accuracy, closely approximating the goodness of fit found in SCIKIT-LEARN. Any decrease in accuracy is primarily due to quantization, rather than FHE. This makes it feasible to implement privacy-preserving tree-based models in production, albeit not always adapted for real-time applications for complex models given the slow speed. However, it is perfectly suitable for occasional use cases such as cancer detection, image modification, or financial services.

4.5 Future Work

In this study, we adopt a rudimentary approach to quantization of tree-based models by uniformly quantizing the input features. While this affords the user some control over quantization, further investigation into more sophisticated quantization methods is warranted to maximize the benefits of the tree-based model. One promising avenue for exploration is training the model with floating-point value features to learn the decision splits in the tree, and then using these splits to quantize the input space. This would eliminate the impact of quantization on the model's accuracy. However, various considerations must be taken into account, so we have reserved this topic for further study in a future version of CONCRETE-ML.

5 Conclusion

The present study offers a method for the conversion of tree-based models into their fully homomorphic encryption (FHE) equivalent, thus providing a secure mechanism for the deployment of machine learning models by service providers. By encrypting user data beforehand, this method ensures a high level of privacy for the users. Although the conversion process may pose certain technical challenges (quantization and conversion to the FHE dialects) these have been mitigated through the implementation of the method in the open-source library CONCRETE-ML, making it as accessible as other commonly used libraries such as SCIKIT-LEARN. In summary, this method represents a significant advancement in the field of machine learning and privacy protection.

References

[ALR+22] Adi Akavia, Max Leibovich, Yehezkel S Resheff, Roey Ron, Moni Shahar, and Margarita Vald. Privacy-preserving decision trees training and prediction. *ACM Transactions on Privacy and Security*, 25(3):1–30, 2022.

[ARC19] Mohammad Al-Rubaie and J Morris Chang. Privacy-preserving machine learning: Threats and solutions. *IEEE Security & Privacy*, 17(2):49–58, 2019.

[BCO11] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Advances in Cryptology–CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings 31*, pages 578–595. Springer, 2011.

[BEG+19] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1:374–388, 2019.

[BFOS17] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325, 2012.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):13:1–13:36, 2014.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, volume 7417, 2012.

[BRAN19] Mousumi Banerjee, Evan Reynolds, Hedvig B Andersson, and Brahmajee K Nallamothu. Tree-based analysis: a practical approach to create clinical decision-making tools. *Circulation: Cardiovascular Quality and Outcomes*, 12(5):e004879, 2019.

[CG16] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology – ASIACRYPT 2016, Part I*, pages 3–33, 2016.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 377–408. Springer, 2017.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Cyber Security Cryptography and Machine Learning (CSCML 2021)*, pages 1–19, 2021.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology - ASIACRYPT 2017, Part I*, pages 409–437, 2017.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology – EUROCRYPT 2015, Part I*, pages 617–640, 2015.

[Dwo06] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33*, pages 1–12. Springer, 2006.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://ia.cr/2012/144.

[GBY+18] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[Gol98] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78(110), 1998.

[GOV22] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042, pages 75–92, 2013.

[JKC+18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[Joy21] Marc Joye. Guide to fully homomorphic encryption over the [discretized] torus. Cryptology ePrint Archive, Paper 2021/1402, 2021. https://eprint.iacr.org/2021/1402.

[MCMF+2] Arthur Meyre, Benoit Chevallier-Mames, Jordan Frery, Andrei Stoian, Roman Bredehoft, Luis Montero, and Celia Kherfallah. Concrete-ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists, 2022-*. https://github.com/zama-ai/concrete-ml.

[MF20] Xianrui Meng and Joan Feigenbaum. Privacy-preserving xgboost inference. *arXiv preprint arXiv:2011.04789*, 2020.

[Mic22] Microsoft. Hummingbird library, 2022. https://github.com/microsoft/hummingbird.

[Opt21] Optalysis. What we do (and why we do it), 2021. https://medium.com/optalysys/optalysys-what-we-do-and-why-we-do-it-20ab416c5ad0.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT '99*, pages 223–238, 1999.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[SZA22] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[TBK20] Anselme Tueno, Yordan Boev, and Florian Kerschbaum. Non-interactive private decision tree evaluation. In *Data and Applications Security and Privacy XXXIV: 34th Annual IFIP WG 11.3 Conference, DBSec 2020, Regensburg, Germany, June 25–26, 2020, Proceedings 34*, pages 174–194. Springer, 2020.

[Zam21] Zama. Concrete library, 2021. https://github.com/zama-ai/concrete.

[Zam22a] Zama. Announcing concrete numpy. Zama Blog, 2022. https://www.zama.ai/post/announcing-concrete-numpy.

[Zam22b] Zama. Decision tree classifier on spam classification task, 2022. https://github.com/zama-ai/concrete-ml/blob/main/docs/user/advanced_examples/DecisionTreeClassifier.ipynb.