`XorSHAP`: Privacy-Preserving Explainable AI for Decision Tree Models

Dimitar Jetchev[⋆] and Marius Vuille[⋆⋆]

Inpher Sàrl, Lausanne, Switzerland

Abstract. *Explainable AI (XAI)* refers to the development of AI systems and machine learning models in a way that humans can understand, interpret and trust the predictions, decisions and outputs of these models. A common approach to explainability is *feature importance*, that is, determining which input features of the model have the most significant impact on the model prediction. Two major techniques for computing feature importance are LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations). While very generic, these methods are computationally expensive even in plaintext. Applying them in the privacy-preserving setting when part or all of the input data is private is therefore a major computational challenge. In this paper, we present `XorSHAP` - the first practical privacy-preserving algorithm for computing SHAP values for decision tree ensemble models in the semi-honest Secure Multiparty Computation (SMPC) setting with full threshold. Our algorithm has complexity $O(T\widetilde{M}D2^D)$, where T is the number of decision trees in the ensemble, D is the depth of the decision trees and $\widetilde{M}$ is the maximum of the number of features M and 2^D (the number of leaf nodes of a tree), and scales to real-world datasets. Our implementation is based on Inpher's `Manticore` framework and simultaneously computes (in the SMPC setting) the SHAP values for 100 samples for an ensemble of $T = 60$ trees of depth $D = 4$ and $M = 100$ features in just 7.5 minutes, meaning that the SHAP values for a single prediction are computed in just 4.5 seconds for the same decision tree ensemble model. Additionally, it is parallelization-friendly, thus, enabling future work on massive hardware acceleration with GPUs.

Keywords: Explainable AI, model explainability, gradient boosting decision trees, SHAP values, secure multiparty computation

1 Introduction

1.1 Motivation

Explainable AI (XAI) refers to the design and development of AI systems and machine learning methods that allow for human understanding and interpretation of the decisions and predictions made by these models [Mol22]. It provides

[⋆] dimitar@inpher.io

[⋆⋆] marius@inpher.io

insights into how a model generates its predictions, which features it considers important as well as the reasoning behind the decisions. The main challenge of explainable AI is thus to create explainable models without compromising model accuracy. This is particularly important in areas such as finance, healthcare, manufacturing, etc.

While explaining the decision may be transparent for simple models such as linear and logistic regression, the more complex the model becomes, the more difficult it is to explain its predictions.

There are several approaches to model explainability:

- *Feature importance:* this approach determines the relative importance or contribution of each feature in the prediction of a model.
- *Local explainability:* a method that provides explanations for individual predictions, showing which features had the most significant impact on the specific instance.
- *Rule extraction:* deriving interpretable rules or decision trees that mimic complex machine learning models.

Additionally, techniques such as LIME (Local Interpretable Model-agnostic Explanations) [RSG16] (as well as the official `github` repository[1] and SHAP (SHapley Additive exPlanations) [LL17] generate explanability metrics at the instance level considering contributions of individual features.

Explainable AI is an important aspect of *Trustworthy AI*, a broader concept aiming at developing, building and deploying reliable, ethical, transparent and accountable AI systems. Trustworthty AI encapsulates various principles and guidelines to address the following objectives:

1. Ethical and responsible AI,
2. Transparency and explainability,
3. Fairness and bias mitigation,
4. Robustness and reliability,
5. Privacy and data protection,
6. Accountability and governance.

The recently voted EU AI Act [Act23] is a regulatory framework proposed by the European Union (EU) to ensure the proper use of Trustworthy AI. It emphasizes the importance of providing explanations for AI decisions, especially in high-risk applications and recognizes that individuals should have the right to understand the logic, significance, and consequences of automated decisions that affect them. This is particularly relevant in areas such as finance, healthcare, industrials and legal domains where the impact of AI decisions can be significant.

In the context of the EU AI Act, using techniques like SHAP values can help fulfill the requirement of providing meaningful explanations for AI decisions. By understanding explainability metrics such as SHAP values, stakeholders can gain insights into how specific features influence the model's predictions and

[1] `https://github.com/marcotcr/lime`

make informed judgments about the fairness, bias, or potential risks associated with the AI system.

The computation of SHAP values can be rather expensive since it involves computing predictions with models built on every possible subset of features. This can become particularly challenging and even infeasible for models with large number of features and samples. One often uses approximation methods such as Monte Carlo or sampling subsets of features [LEL18, SK14]. The complexities can become even higher if the input data is sensitive - in use case that occurs very often in healthcare, financial technology as well as manufacturing where the combination of explainability and privacy of the input data is particularly relevant.

Thus, the problem of designing and implementing efficient and scalable privacy-preserving algorithms for computing feature importance metrics such as SHAP values is of primary importance for Trustworthy AI.

1.2 Our contributions

In this paper, we address the above problem for decision tree ensemble models in the Secure Multiparty Computation (SMPC) setting. More specifically, we propose a privacy-preserving algorithm for computing SHAP values, `XorSHAP`, based on the `Manticore` framework [BCD+23] for secure multiparty computation in the semi-honest setting. Our algorithm is inspired by the `TreeSHAP` algorithm [LEL18] but relies on several data-independent (oblivious) operations such as additions, multiplications, divisions, comparisons as well as sorting permutations.

Our algorithm is applicable to the most generic data distribution setup where input data comes from two or more private data sources and is either horizontally split among the owners (i.e., every owner has different samples/rows sharing the same features/columns), vertically split (every owner has complete set of features for all the samples/rows) or any combination of the two. The goal is to compute SHAP values for a given decision tree ensemble model in the MPC setting using the fully stacked dataset and without revealing private information. Although the private data owners may play the role of the compute parties in some setting, this is not a strict requirement. In fact, the *Manticore* framework supports scenarios where private data owners secret share data among a set of compute parties that is not necessarily the same as the set of data owners. The currently used security model is semi-honest, that is, the players execute the exact steps of the algorithm with full-threshold security meaning that if all players except one decide to collude, the data of the non-colluding player is still protected.

Just as in `TreeSHAP`, the major complexity gain comes from the fact that, instead of summing over all subsets of $\{1, \ldots, M\}$ (thus, introducing a factor 2^M in the complexity), one sums over all 2^D possible paths in the tree and then over all subsets of the set of feature indices of the split nodes in the path (a set of size at most D). Our main contribution is in the way we make that part oblivious, thus, resulting in complexity that no longer has the large exponential factor 2^M where M is the number of features, thus, resulting in total complexity of $O(T\widetilde{M}D2^D)$, where T is the number of trees, D is the depth of each binary decision tree in the ensemble model and $\widetilde{M} = \max(M, 2^D)$.

The efficiency of our privacy-preserving approach uses several key ideas and techniques: 1) efficient oblivious permutations and oblivious sorting algorithms; 2) computing the path conditional probabilities $W_{\ell,S}$ in data-independent manner using the set of distinct feature indices for a given path (a set of cardinality at most D), a slightly extended set of cardinality exactly D and a special ordering on the power set of the extended set from Section 3.2.2. The latter is described precisely in Lemma 6.

Finally, we demonstrate the efficiency and scalability of `XorSHAP` in practice by implementing it on the [BCD+23] framework for secure multiparty computation and by presenting some preliminary benchmarks. The data presented in Section 5 shows that the run-time scales linearly with both the number of trees T as well as the number of samples M. For a decision tree ensemble model with $T = 60$ trees of depth $D = 4$ and $M = 100$ features, the computation of the SHAP values for 100 samples takes a little more than 7 minutes on Intel Xeon E5-2666 v3 @ 2.90GHz CPU with 32GB RAM and network that has a throughput of 120Mbps and a latency of 0.3ms.

1.3 Prior art

While initial attempts have been made to address the question of privacy-preserving explainable AI, this area of research on the intersection of machine learning and data privacy is relatively nascent and will likely grow with the expanding interest on Trustworthy and Ethical AI.

Prior work have been made to introduce differential privacy in XAI and more specifically, in the computation of SHAP values to preserve the privacy of the input data [WAYS22].

Another important research direction has been in the area of federated machine learning. In [Wan19], the author proposes a method for computing feature importance via SHAP values in the setting of vertical federated learning with two parties, host and guest, where the host owns part of the features, the guest owns the remaining features and the host wants to interpret a specific prediction the model makes. In order for the host achieve the latter, the guest *à priori* needs to send the feature importance for its entire feature space to the host which may reveal too much information to the host. The idea of [Wan19] is to replace the feature space with a single unified federated feature, letting the host compute a metric for its own feature space together with that single federated feature. While this is not computing exactly the SHAP model explainability metrics, experiments show that it gives some approximate information matching the feature importance one would get by using the entire feature space of the guest.

In addition, the recent work [BIS+22] attempts to compute SHAP values for both horizontally or vertically partitioned data via an explainable data collaboration framework based on the model-agnostic additive feature attribution algorithm `KernelSHAP` in the privacy-preserving distributed machine learning setting. The starting point is that, in a horizontal federated learning scenario where each party owns part of the samples, a given party may have a biased view of the samples. Similarly, in a vertical federated learning scenario, the individual parties have a partial view of the feature space. The first proposed algorithm, `Horizontal DC-SHAP`, addresses the challenge with the discrepancies among the contributing parties by using a sharable anchor dataset to produce consistent explanation across all collaborators. In the vertical scenario, the authors propose two different algorithms `Vertical DC-SHAP (i)` and `(ii)` depending on the use case: 1) feature attribution is requested at a third party for the entire set of features; 2) feature attribution is requested by one of the users for the partial set of features.

There have been some attempts to compute SHAP values in the secure multiparty computation setting in the context of data valuation and data marketplaces [TLL^{+}22]. Recall that data valuation refers to the task of fairly compensating data owners for their contributions in the data marketplace and a suitable profit-sharing and fair scheme comes from Shapley values and cooperative game theory. The proposed secure multiparty computation approach relies on active learning, that is, a machine learning algorithm that is allowed to query the user or an external data source to label data points. While interesting as a use case and an approach, this method is not quite applicable to capture the very generic setting where no active learning is assumed.

As far as we are aware, our work is the first attempt to compute feature importance metrics such as SHAP values for decision tree ensemble models in the most generic SMPC setting with additive secret sharing and no active learning.

1.4 Organization of the paper

The paper is organized as follows: in Section 2 we recall some background on the general theory of Shapley values from cooperative game theory (see Section 2.1). We then specialize this theory to the case of binary decision trees in Section 2.2. We also recall some basic background on Secure Multiparty Computation with additive secret sharing in Section 2.3 as well as the `Manticore` framework [BCD^{+}23] and its security model. The main algorithm `XorSHAP` is presented in Section 3. We complete the paper by proper complexity analysis (presented in Section 4) as well as implementation and benchmarks (in Section 5).

2 Background and Preliminaries

2.1 Background on Shapley values and SHAP

Shapley values are a concept from cooperative game theory originally introduced by Lloyd Shapley [Sha53] and used to allocate the total contribution or payoff of a group to its individual members.

2.1.1 Marginal contributions.

Shapley values are formally defined in terms of expected total payoffs of subsets of players in cooperative game theory, that is, in terms of a function v that assigns to each coalition S of players the worth $v(S)$ of that coalition. In machine learning, the analogue of players are model features and the worth function is obtained using the prediction function of a model as follows: given a subset S of features, a model f with M features as well as a sample $x = (x_1, \dots, x_M)$, we define the worth function

$$f_S(x) := \mathbb{E}[f(\xi)|\xi_S = x_S], \tag{1}$$

where the conditional expectation is interpreted as follows: we fix the value for feature j to x_j for all $j \in S$ and use random values for the features $j \notin S$. In practice, given a training set of samples, one computes the expectation as the average over these samples.

The conditional expectation $f_S(x)$ represents the total value of the coalition S of players. One uses that to define the marginal contribution of a player $i \notin S$ simply as $f_{S\cup\{i\}}(x) - f_S(x)$.

2.1.2 Definition of SHAP values.

Given the model f and the sample $x = (x_1, \dots, x_M)$, for each feature index $i = 1, \dots, M$, we define the SHAP value as

$$\phi_i(x) = \sum_{S \subset \{1,\dots,M\}\setminus\{i\}} \frac{|S|!(M-1-|S|)!}{M!} \left(f_{S\cup\{i\}}(x) - f_S(x)\right), \tag{2}$$

that is, the weighted average of all marginal contributions of feature i (the subsets $S = \emptyset$ and $S = \{1, \dots, M\}\setminus\{i\}$ have the largest weight, whereas subsets of size $\lfloor (M-1)/2 \rfloor$ and $\lceil (M-1)/2 \rceil$ respectively have the smallest weight). Computing this naïvely is infeasible as the sum is over an exponential number of subsets and for each subset, one needs to estimate the model prediction over the entire training set over which we estimate the expectation.

2.1.3 Computing SHAP values in the generic case

In general (for arbitrary models), an exact computation of SHAP values is often infeasible as the definition requires summing over exponentially many coalitions of features - in our case, 2^M. In practice, SHAP values are usually approximated by sampling random coalition of features and summing over the marginal contributions for these subsets.

One way to approximate SHAP values is via Monte–Carlo simulations [SK14]. More specifically, this approximation method samples random feature coalitions representing random subsets of features by generating random permutations or random combinations of features and then computes the marginal contribution of each feature not included in the coalition, thus computing the chain in prediction when the feature joins the coalition. It then averages the computed marginal contributions across the sampled coalitions to provide an approximation for the SHAP value for each feature. This method has a trade-off between the number of random coalitions sampled and the accuracy of the SHAP values.

2.2 SHAP values for decision trees

Even if computing SHAP values in the model-agnostic case is quite expensive, specializing to decision tree models provides some significant performance advantages which we now explain.

2.2.1 Binary decision trees. Let M be a fixed positive integer (the number of features). A *binary decision tree* of depth D on the feature space $\mathbb{R}^M$ consists of $2^D - 1$ inner nodes (referred to as non-leaf nodes or split nodes) and 2^D outer nodes (referred to as leaf nodes). We denote by $\mathcal{N}$ the set of inner nodes and by L_D (or simply L) the set of leaf nodes. Associated to each inner node $n \in \mathcal{N}$ is a pair (j_n, t_n) of a *feature index* $j_n \in \{1, \ldots, M\}$ and a *threshold* t_n (a real number). Associated to each leaf node $\ell \in L$ is a *weight* w_ℓ (a real number). We thus represent a tree as

$$\texttt{Tree} = (\texttt{TreeStructure},\ \texttt{TreeWeights}),$$

where

$$\texttt{TreeStructure} = ((j_1, t_1), \ldots, (j_{2^D-1}, t_{2^D-1}))$$

is the list of pairs associated to the (list) of inner nodes and

$$\texttt{TreeWeights} = (w_1, \ldots, w_{2^D})$$

is the list of weights.

As described in [DDG$^+$22, §2], given a sample $x = (x_1, \ldots, x_M)$ and a tree $\texttt{Tree}$, there is a recursive evaluation function $\texttt{eval}(x, \texttt{Tree})$ which, for a fixed tree $\texttt{Tree}$, yields a piecewise constant function $\texttt{Tree}\colon \mathbb{R}^M \to \mathbb{R}$. We let $\hat{y}$ be the estimate for the response variable y. Given a tree ensemble $\{\texttt{Tree}^{(1)}, \ldots, \texttt{Tree}^{(T)}\}$ and a learning rate parameter η, one defines the predictions on x recursively as

$$\widehat{y}^{(t)} = \widehat{y}^{(t-1)} + \eta\, \texttt{Tree}^{(t)}(x) \in \mathbb{R}. \tag{3}$$

The reason for the learning rate η is to dampen the contribution of the new tree added to the current model. Often, one takes $\eta = 1$ to obtain the total prediction on x as

$$\widehat{y}^{(T)} = \sum_{t=1}^{T} \texttt{Tree}^{(t)}(x) \in \mathbb{R}. \tag{4}$$

2.2.2 SHAP values for decision trees. Since Shapley values are linear in the worth function v and since the prediction of a tree ensemble model is a linear combination of the evaluation of its trees, by linearity of the mathematical expectation it suffices to explain how to compute the SHAP values for a single tree.

We now explain how to explicitly compute the conditional expectation for a decision tree via a recurrence formula. Let $\texttt{Tree}$ be a binary decision tree of depth D on M features, and let $x = (x_1, \dots, x_M)$ be a sample of M feature values to be interpreted (or explained). For a subset $S \subset \{1, \dots, M\}$ of feature indices, consider the conditional expectation $\mathbb{E}[\mathtt{Tree}(\xi) \mid \xi_S = x_S]$ as in (1). It can be computed as

$$\mathbb{E}[\mathtt{Tree}(\xi) \mid \xi_S = x_S] = G_S(x; n_{\mathrm{root}}),$$

where, for a given node n of the tree, $G_S(x; n)$ is recursively defined as:

$$G_S(x;n) = \begin{cases} w_n & \text{if } n \text{ is a leaf node,} \\ (x_{j_n} < t_n) \;\; ? \;\; G_S(x; n_{\mathrm{left}}) \;\; : \;\; G_S(x; n_{\mathrm{right}}) & \text{else if } j_n \in S, \\ \frac{c_{n_{\mathrm{left}}}}{c_n} G_S\left(x; n_{\mathrm{left}}\right) + \frac{c_{n_{\mathrm{right}}}}{c_n} G_S\left(x; n_{\mathrm{right}}\right) & \text{else.} \end{cases} \tag{5}$$

Here, c_n is the *cover* of node n, i.e., the number of samples that are visiting node n upon evaluation. It is computed as the Hamming weight of the instance vector IV_n of that node (the characteristic vector of the subset of samples visiting n).

Remark 1. For now we assume that $c_n > 0$ for all node n of the tree. We will explain in Section 3.3 how to remedy this assumption.

Given a leaf node $\ell \in L_D$, we use $\mathcal{P}_\ell = \{n_0^{(\ell)}, \dots, n_{D-1}^{(\ell)}\}$ for the (ordered) set of D split nodes in the path leading from the root $n_0^{(\ell)}$ to leaf ℓ. We also define $\mathcal{J}_\ell = \{j_0^{(\ell)}, \dots, j_{D-1}^{(\ell)}\}$ to be the (ordered) multi-set of D feature indices for the split nodes in $\mathcal{P}_\ell$. Finally, let $\mathcal{F}_\ell$ be the set of distinct feature indices for the split nodes of the path leading to leaf ℓ (we thus have $|\mathcal{F}_\ell| \leq D$).

Definition 1. *We call a path $\mathcal{P}_\ell$* consistent *with respect to the pair (x, S) of a sample and a feature subset S if for every node $n \in \mathcal{P}_\ell$ such that $j_n \in S$, the child node determined by the condition $x_{j_n} < t_n$ coincides with the node of $\mathcal{P}_\ell \cup \{\ell\}$ that succeeds n in the path. Otherwise, the path is called* inconsistent*.*

In what follows, it is important to distinguish the following metric

$$W_{\ell,S}(x) := \begin{cases} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \notin S}} \frac{c_{n_{\mathrm{child}}}}{c_n} & \text{if } \mathcal{P}_\ell \text{ is consistent with } (x, S), \\ 0 & \text{else.} \end{cases} \tag{6}$$

Given a subset S and a leaf node ℓ, $W_{\ell,S}$ computes the proportion of paths (conditional on x_S, the subset S of fixed feature values for S) that flows down to leaf node ℓ. The conditional expectation is then simply given by

$$\mathbb{E}[\mathtt{Tree}(\xi) \mid \xi_S = x_S] = \sum_{\ell \in L} W_{\ell,S}(x) \cdot w_\ell. \tag{7}$$

Substituting (7) into the definition (2) of SHAP value yields

$$\begin{aligned}\phi_i(x) &= \sum_{S\subset\{1,\ldots,M\}\setminus\{i\}} \frac{|S|!(M-1-|S|)!}{M!} \sum_{\ell\in L} \left(W_{\ell,S\cup\{i\}}(x) - W_{\ell,S}(x)\right)\cdot w_\ell \\ &= \sum_{\ell\in L}\left(\sum_{S\subset\{1,\ldots,M\}\setminus\{i\}} \frac{|S|!(M-1-|S|)!}{M!}(W_{\ell,S\cup\{i\}}(x) - W_{\ell,S}(x))\right)\cdot w_\ell \end{aligned} \quad (8)$$

2.2.3 `KernelSHAP` and `TreeSHAP`. The framework SHAP for interpreting predictions was first introduced in [LL17] and was based on Shapley values. Since the latter are very expensive to compute, several approximations have been proposed.

The `KernelSHAP` method originally described in [LL17, §4] is model agnostic and is based on approximating SHAP values by sampling subsets of features containing the given feature and averaging.

In the specific case of decision trees, however, the complexity of `KernelSHAP` is $O(T2^{D+M})$ where T is the number of trees in the ensemble, D is the tree depth and M is the total number of features.

An alternative algorithm for computing SHAP values in the case of decision trees is `TreeSHAP` [LEL18, LEC+20]. The algorithm has complexity $O(T2^D D^2)$ and is asymptotically faster than `KernelSHAP` (we are assuming that the number of features M is asymptotically much larger than D^2). There are further refinements to the algorithm [Yan21] resulting in `FastTreeSHAP v.1` (a 1.5x speedup and the same memory overhead as `TreeSHAP`) and `FastTreeSHAP v.2` (a 3x speedup with via some preprocessing and a slightly larger memory overhead). Furthermore, `GPUTreeSHAP` proposed in [MFH22] provides a more scalable version of `TreeSHAP` resulting in a 20x speedup.

2.3 Secure Multiparty Computation

Multiparty computation (MPC) is a method for cryptographic computing allowing several parties holding private data to evaluate a public function on their aggregate data while revealing only the output of the function and nothing else. Recent advances in the area make these protocols practical and suitable for real-world applications such as machine and statistical learning [BLW08, BCG+17, BCD+23, DPSZ12, EGK+20, Kel20, KOS16, KPR18, MR18, MZ17, WGC19].

In general, our `XorSHAP` algorithm is agnostic to underlying MPC method or framework. The choice of the `Manticore` MPC framework in our implementation is favorable for several reasons: 1) it provides access to boolean arithmetic as well as arithmetic with real numbers represented using modular integers [BCD+23] or the prior floating-point numbers framework [BCG+17]; 2) the real number representation of `Manticore` via modular integers guarantees information-theoretically security; 3) `Manticore` provides oblivious sorting and oblivious permutations functionality. There are, however, other MPC libraries that support Boolean and real number arithmetic such as `SCALE-MAMBA` [aM23], SecureML [MZ17], ABY [PSSY21,MR18] as well as SPDZ-2K [CDE+18]. In Manticore, computations are split into offline and online phases. The offline phase generates random precomputed data without accessing the private data and can be performed either interactively by techniques such as oblivious transfer or by an independent party, different from the compute parties, known as trusted dealer. In the former case, the MPC protocol is slower but the security model is stronger. In the latter case, the offline phase is significantly accelerated but in order to protect the privacy of the input data, the trusted dealer is assumed not to collude with any of the compute parties. In addition, to ensure security against malicious external adversaries, all communications between the trusted dealer as well as all communication between the players during the online phase is end-to-end encrypted.

2.4 Secret sharing of binary decision trees

In the setting that we will apply our algorithm, there are different possibilities for the decision tree ensemble model. It can be proprietary model for a given party, or it can already be computed with a privacy-preserving algorithm such as XorBoost [DDG+22] or any other privacy-preserving algorithm. We refer to [DDG+22, §2.4] for details on how one can secret share a decision tree model. Note that, depending on the training algorithm, the column index j can be represented either in arithmetic shares or in Boolean shares. This is the reason why in the discussions in Section 3, we make no assumption on the secret sharing method (arithmetic or Boolean) and present a generic arithmetic-to-Boolean conversion method in Lemma 1.

Note that the actual representation of arithmetic secret shares varies depending on the SMPC implementation or framework used. For example, for most of the applications Manticore relies on a particular representation of real numbers with modular integers that can differ from other schemes such as SPDZ. Our algorithms are agnostic to the particular instantiation of arithmetic secret sharing.

3 XorSHAP: Privacy-preserving TreeSHAP

In this section, we present the main contribution of this paper – an algorithm, XorSHAP that is a privacy-preserving variant of the TreeSHAP method [LEL18].

Our main mathematical result that yields our privacy-preserving algorithm is Theorem 2 - a data-independent formula for the SHAP values. Applying that formula in the secure multiparty computation setting requires the following operations:

- Additions of secret shared numbers (over both $\mathbb{R}$ and $\mathbb{Z}/2\mathbb{Z}$),
- Multiplications of secret shared numbers (over $\mathbb{R}$ and $\mathbb{F}_2$),
- Private divisions of secret shared numbers (over $\mathbb{R}$)
- Comparisons of secret shared real numbers,

– Sorting and `argsort` of secret shared real-valued vectors,
– Applying secret shared sorting permutations.

Unless otherwise stated, all subsequent variables are secret and all operations are data-independent.

3.1 Notations.

In order to present the algorithm, we use the notations introduced in Section 2.2.2, particularly, $\mathcal{P}_\ell$, $\mathcal{J}_\ell$ and $\mathcal{F}_\ell$. We also introduce some extra notation that is relevant for data-independent representation of the various structures and hence, for an SMPC-friendly algorithm.

Given a binary decision tree of depth D, recall that $L = L_D$ denotes the set of leaf nodes of the tree. We assume that sample x is drawn from a dataset with M features. For a given leaf node $\ell \in L$, define

– P_ℓ: a binary matrix of size $M \times D$ whose columns are the binary feature selector vectors for the split nodes of the path leading to leaf ℓ (here, a feature with index $1 \leq j \leq M$ is encoded with the binary column vector

$$\mathbf{b}_j := (\underbrace{0, \ldots, 0}_{j-1}, 1, \underbrace{0, \ldots, 0}_{M-j})^t.$$

Thus, $P_\ell = \texttt{concat}(\mathbf{b}_j : j \in \mathcal{J}_\ell)$. The matrix P_ℓ is helpful for a secret shared representation of the set $\mathcal{J}_\ell$.
– F_ℓ: the characteristic vector of the set $\mathcal{F}_\ell$, i.e., a binary (column) vector of size M whose jth entry is 1 if $j \in \mathcal{F}_\ell$ (i.e., feature j appears as a splitting feature in the path leading to leaf ℓ) and 0 otherwise.
– $\mathbf{o}_\ell = (o_n^{(\ell)} : n \in \mathcal{P}_\ell)$: a binary (column) vector of size D, where $o_n^{(\ell)} = 1$ if evaluating node n on the sample x yields the child node that is in the path $\mathcal{P}_\ell \cup \{\ell\}$; otherwise $o_n^{(\ell)} = 0$. For simplicity, we also write $o_{\ell,d} = o^{(\ell)}_{n_d^{(\ell)}}$ for the dth component of $\mathbf{o}_\ell$.
– $\mathbf{z}_\ell = (z_n^{(\ell)} : n \in \mathcal{P}_\ell)$: a (column) vector of real numbers of size D corresponding to the cover ratios of the nodes in the path $\mathcal{P}_\ell \cup \{\ell\}$; i.e., for node n,

$$z_n^{(\ell)} = \frac{c_{n_{\texttt{child}}}}{c_n},$$

where n_{child} is the successor node of n in $\mathcal{P}_\ell \cup \{\ell\}$. For simplicity, we also write $z_{\ell,d} = z^{(\ell)}_{n_d^{(\ell)}}$ for the dth component of $\mathbf{z}_\ell$. We compute $z_n^{(\ell)}$ using the private division algorithm from [BCD+23].

The following lemma is straightforward so we omit the proof:

Lemma 1. *For secret shared index $j \in \{1, \ldots, M\}$ (in arithmetic secret shares), one can compute Boolean shares for $\mathbf{b}_j$ as follows:*

1. *Arithmetic-to-Boolean conversion of secret shares for j;*

2. One-hot binary decoder circuit applied to the binary expansion of j.

For a basic reference on one-hot binary decoder circuits, see [DH12, §8.1].

Next, the lemma below computes the characteristic vector F_ℓ in a less efficient, but data independent manner:

Lemma 2. *F_ℓ is obtained from P_ℓ by* `OR`*-ing the columns, i.e.,*

$$F_\ell = \bigvee_{j \in \mathcal{J}_\ell} \mathbf{b}_j.$$

Remark 2. Note that $F_\ell = \bigvee_{j \in \mathcal{F}_\ell} \mathbf{b}_j$, however the right-hand side cannot be computed in a data-independent way since $\mathcal{F}_\ell$ is not known.

Finally, we will need to compute the vector $\mathbf{o}_\ell$ in a data-independent manner. This is achieved by the following:

Lemma 3. *We compute $o_{\ell,d}$ as follows: let (j,t) be the feature index and threshold associated to node $n_d^{(\ell)}$, and let $\beta = (x_j < t) \in \{0,1\}$. Let n_{child} be the successor node of $n_d^{(\ell)}$ in $\mathcal{P}_\ell \cup \{\ell\}$ and let n_{left} and n_{right} be the left and right children nodes of $n_d^{(\ell)}$ respectively (when looking at the whole tree). Then:*

$$o_{\ell,d} = \begin{cases} \beta & \text{if } n_{child} = n_{left}, \\ \neg\beta & \text{if } n_{child} = n_{right}. \end{cases}$$

3.2 `XorSHAP`.

Let x be a fixed sample. For what follows we will simply write $W_{\ell,S}$ instead of $W_{\ell,S}(x)$. With the notation introduced in Section 3.1 we can express $W_{\ell,S}$ as:

Lemma 4.

$$W_{\ell,S} = \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \in S}} o_n^{(\ell)} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \notin S}} z_n^{(\ell)}.$$

Proof. $\mathcal{P}_\ell$ is consistent with (x,S) if and only if $\prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \in S}} o_n^{(\ell)} = 1$.

Remark 3. Lemma 4 does not provide a way to obliviously compute $W_{\ell,S}$. Instead of the costly (in a data-independent approach) evaluation of the criteria $j_n \in S$ (respectively $j_n \notin S$), our approach is based on a specific linear ordering of the subsets of $\mathcal{F}_\ell$ (or even a slightly larger set), see Lemma 6.

The algorithm `XorSHAP` is based on the following theorem:

Theorem 1. *The SHAP values for a binary decision tree can be computed as follows:*

$$\phi_i(x) = \sum_{\substack{\ell \in L \\ i \in \mathcal{F}_\ell}} \left(\sum_{S \subset \mathcal{F}_\ell \setminus \{i\}} \frac{|S|!(|\mathcal{F}_\ell| - 1 - |S|)!}{|\mathcal{F}_\ell|!} \cdot W_{\ell,S} \right) \cdot t_i^{(\ell)} \cdot w_\ell, \tag{9}$$

where $t_i^{(\ell)} = \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n = i}} o_n^{(\ell)} / z_n^{(\ell)} - 1.$

Proof. Since $W_{\ell,S\cup\{i\}} = W_{\ell,S}$ whenever $i \notin \mathcal{F}_\ell$, summation in (8) simplifies to

$$\phi_i(x) = \sum_{\substack{\ell \in L \\ i \in \mathcal{F}_\ell}} \left(\sum_{S \subset \{1,\ldots,M\} \setminus \{i\}} \frac{|S|!(M - 1 - |S|)!}{M!} (W_{\ell,S\cup\{i\}} - W_{\ell,S}) \right) \cdot w_\ell. \tag{10}$$

Following the proof of [Yan21, Thm. 1], (10) reduces to

$$\phi_i(x) = \sum_{\substack{\ell \in L \\ i \in \mathcal{F}_\ell}} \left(\sum_{S \subset \mathcal{F}_\ell \setminus \{i\}} \frac{|S|!(|\mathcal{F}_\ell| - 1 - |S|)!}{|\mathcal{F}_\ell|!} (W_{\ell,S\cup\{i\}} - W_{\ell,S}) \right) \cdot w_\ell. \tag{11}$$

Note that

$$\begin{aligned}
W_{\ell,S\cup\{i\}} &= \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \in S\cup\{i\}}} o_n^{(\ell)} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \notin S\cup\{i\}}} z_n^{(\ell)} \\
&= \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \in S}} o_n^{(\ell)} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \notin S}} z_n^{(\ell)} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n = i}} o_n^{(\ell)}/z_n^{(\ell)} \\
&= W_{\ell,S} \cdot \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n = i}} o_n^{(\ell)}/z_n^{(\ell)}
\end{aligned}$$

and hence,

$$W_{\ell,S\cup\{i\}} - W_{\ell,S} = W_{\ell,S} \cdot t_i^{(\ell)},$$

which concludes the proof. For implementation aspects it is worth noting that $W_{\ell,S}$ depends on S but not on i, while $t_i^{(\ell)}$ depends on i but not on S.

Remark 4. The problem with the above version of the theorem in the privacy-preserving setting is that it is data dependent:

i) The definition of $o_n^{(\ell)}$ depends on the value of the predicate; Lemma 3 shows how to obliviously compute $o_n^{(\ell)}$.

ii) The condition $i \in \mathcal{F}_\ell$ depends on knowing $\mathcal{F}_\ell$, which we do not. We will therefore define a vector $\mathbf{t}_\ell$ of size M with $t_{\ell,i} = t_i^{(\ell)}$ if $i \in \mathcal{F}_\ell$ and $t_{\ell,i} = 0$ otherwise. Lemma 8 shows how to obliviously compute $\mathbf{t}_\ell$. We can then rewrite (9) as

$$\phi_i(x) = \sum_{\ell \in L} \left(\sum_{S \subset \mathcal{F}_\ell \setminus \{i\}} \frac{|S|!(|\mathcal{F}_\ell| - 1 - |S|)!}{|\mathcal{F}_\ell|!} \cdot W_{\ell,S} \right) \cdot t_{\ell,i} \cdot w_\ell. \tag{12}$$

Note that if a leaf ℓ is such that $i \notin \mathcal{F}_\ell$, then the inner sum needs to exclude the summand where $S = \mathcal{F}_\ell$ because $(|\mathcal{F}_\ell| - 1 - |S|)!$ is not well-defined. This is addressed in Section 3.2.4.

iii) To compute the inner sum, one needs to consider a (potentially) larger set $\mathcal{R}_\ell \supset \mathcal{F}_\ell$ of size D, see Section 3.2.1, and sum over all 2^D subsets of $\mathcal{R}_\ell$. We do not know the set $\mathcal{R}_\ell$, all we know is it contains $\mathcal{F}_\ell$. We will then make use of obliviously computed characteristic vectors to annihilate the contribution of subsets that violate the condition $S \subset \mathcal{F}_\ell \setminus \{i\}$ (assuming $i \in \mathcal{F}_\ell$ - otherwise $t_{\ell,i}$ annihilates the inner sum algothether, see ii)).

The main contribution of this work is Theorem 2, a data-independent version of Theorem 1.

3.2.1 Size reduction. SHAP values are originally defined as sums over subsets of $\{1, \ldots, M\}$ (see Definition 2) which introduces an overhead of 2^M in the complexity of computing them. The main observation of `TreeSHAP`, Theorem 1, is that by changing the index of summation in (8) (i.e., summing over the leaves $\ell \in L$ first), for each leaf ℓ, it suffices to sum only over the subsets of the set $\mathcal{F}_\ell$ (which has size at most D). This essentially yields a plaintext version of the `TreeSHAP` algorithm. The challenge of designing a data-independent privacy-preserving algorithm is that we do not know the exact size of $\mathcal{F}_\ell$. Therefore, the best we can do in a privacy-preserving setting is to reduce the overhead to 2^D for each leaf ℓ (instead of $2^{|\mathcal{F}_\ell|}$) by considering a possibly larger subset $\mathcal{R}_\ell \supset \mathcal{F}_\ell$ of $\{1, \ldots, M\}$ (see below for the exact definitions) of cardinality exactly equal to D (i.e., a cardinality that is data independent). As $D \ll M$ in most of the practical applications, this is still a major gain in complexity.

Our approach is based on sorting permutations and reduces the characteristic vector F_ℓ of $\mathcal{F}_\ell$ (a binary vector of size M) to a vector R_ℓ of size D that encodes the same information as F_ℓ. Let $\mathfrak{S}_M$ be the set of permutations in M letters and let $\sigma_{F_\ell} \in \mathfrak{S}_M$ be a permutation that sorts (in ascending order) the binary vector F_ℓ. Note that $\sigma_{F_\ell}(F_\ell) \in \{0,1\}^M$ consists of $M - |\mathcal{F}_\ell|$ 0's followed by $|\mathcal{F}_\ell|$ 1's. We denote by $R_\ell \in \{0,1\}^D$ the vector formed by the last D entries of $\sigma_{F_\ell}(F_\ell)$.

We call a *valid index* any of the $|\mathcal{F}_\ell|$ indices in $\mathcal{F}_\ell$ (equivalently, the position of the 1 in F_ℓ respectively R_ℓ it corresponds to). Otherwise, we call an index *invalid*; there are $D - |\mathcal{F}_\ell|$ invalid indices in R_ℓ, and we denote by $\mathcal{I}_\ell = \mathcal{R}_\ell \setminus \mathcal{F}_\ell$ the set of invalid indices of $\mathcal{R}_\ell$. Note that the order of the valid indices in R_ℓ depends on the choice of σ_{F_ℓ} and hence, does not yield useful information about any natural order of the indices in the path (e.g., the top-to-bottom order of the nodes in the path).

The concept of valid indices will be used to encode the subsets $S \subset \mathcal{F}_\ell$ in the inner sum of (9) in a way that does not require knowing $\mathcal{F}_\ell$. One way to get a data-independent algorithm for computing (9) is to rewrite the inner sum as a sum over all 2^D subsets of $\mathcal{R}_\ell$ and make sure the contribution to the sum of any subset containing invalid indices is 0. We formalize this intuition in Theorem 2 below after introducing the required notions.

3.2.2 Ordering the power set $P(\mathcal{R}_\ell)$.

We now introduce a convenient linear ordering on the power set $P(\mathcal{R}_\ell)$ of $\mathcal{R}_\ell$ that is induced by the choice of σ_{F_ℓ} (i.e., is non-canonical) satisfying the following properties:

i) The first subset in the ordering is the entire set $\mathcal{R}_\ell$ and the last subset is the empty set.
ii) The power set $P(\mathcal{F}_\ell)$ corresponds to the last $2^{|\mathcal{F}_\ell|}$ subsets whereas the first $2^D - 2^{|\mathcal{F}_\ell|}$ subsets contain invalid indices.
iii) The order of indexing of $P(\mathcal{R}_\ell)$ depends on σ_{F_ℓ} and is therefore non-canonical; yet, ii) always holds.
iv) The cardinality of a subset $S \subset \mathcal{R}_\ell$ is recovered explicitly from the binary expansion of its index in the linear ordering.

Let $\{\tau_1, \ldots, \tau_D\}$ be the elements of $\mathcal{R}_\ell$ with ordering determined by the permutation σ_{F_ℓ}. Note that the first $D - |\mathcal{F}_\ell|$ indices are invalid and the last $|\mathcal{F}_\ell|$ are valid. The subset corresponding to position $s = 1, \ldots, 2^D$ in the linear ordering of $P(\mathcal{R}_\ell)$ is the one whose characteristic vector is equal to the binary expansion of $2^D - s$ (msb-to-lsb binary representation). The cardinality of the subset can thus be obtained as the sum of the bits of the binary expansion. For instance, if $D = 3$ then the ordering is the following:

$$\{\tau_1, \tau_2, \tau_3\}, \{\tau_1, \tau_2\}, \{\tau_1, \tau_3\}, \{\tau_1\}, \{\tau_2, \tau_3\}, \{\tau_2\}, \{\tau_3\}, \varnothing.$$

E.g., for $s = 2$, the binary expansion of $8 - 2$ is 110 and hence the 2nd subset in the ordering is $\{\tau_1, \tau_2\}$. Note that this ordering is agnostic to the presence of invalid indices (which is secret information).

3.2.3 Computing $W_{\ell,S}$.

Let $O_\ell \in \{0,1\}^D$ be the binary vector given by

$$O_{\ell,d} = \begin{cases} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n = \tau_d}} o_n^{(\ell)} & \text{if } \tau_d \text{ is a valid index,} \\ 1 & \text{otherwise.} \end{cases}$$

and let $Z_\ell \in [0,1]^D$ be the vector given by

$$Z_{\ell,d} = \begin{cases} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n = \tau_d}} z_n^{(\ell)} & \text{if } \tau_d \text{ is a valid index,} \\ 1 & \text{otherwise.} \end{cases}$$

Lemma 5. *We obliviously compute O_ℓ and Z_ℓ as:*

$$O_\ell = \pi_D \circ \sigma_{F_\ell}\left(\prod_{k=1}^{D}\left(o_{\ell,k-1}\cdot P_\ell^{(k)} + (1 - P_\ell^{(k)})\right)\right)$$

and

$$Z_\ell = \pi_D \circ \sigma_{F_\ell}\left(\prod_{k=1}^{D}\left(z_{\ell,k-1}\cdot P_\ell^{(k)} + (1 - P_\ell^{(k)})\right)\right),$$

respectively, where $P_\ell^{(k)}$ is the kth column of P_ℓ and $\pi_D\colon \mathbb{R}^M \to \mathbb{R}^D$ is the projection onto the last D coordinates.

Proof. Let τ_d be a valid index, that is, $T_d := \{1 \le k \le D\colon j_{n_{k-1}^{(\ell)}} = \tau_d\}$ is non-empty. Moreover, for all $k \in T_d$ we have $P_\ell^{(k)} = \mathbf{b}_{\tau_d}$ (the only non-zero entry is at position τ_d), and for all $k \in \{1,\dots,D\}\setminus T_d$ the τ_dth entry of $P_\ell^{(k)}$, which we denote by $P_{\ell,\tau_d}^{(k)}$, is 0. Hence,

$$\begin{aligned}\prod_{\substack{n\in\mathcal{P}_\ell\\ j_n=\tau_d}} o_n^{(\ell)} &= \prod_{k\in T_d} o_{\ell,k-1}\\ &= \prod_{k=1}^{D}\left(o_{\ell,k-1}\cdot P_{\ell,\tau_d}^{(k)} + (1-P_{\ell,\tau_d}^{(k)})\right),\end{aligned}$$

and $\pi_D \circ \sigma_{F_\ell}$ sends the τ_dth entry of a vector of size M to the dth entry of a vector of size D.

For an invalid index τ_d one observes that $P_{\ell,\tau_d}^{(k)} = 0$ for all $k = 1,\dots,D$ and hence,

$$\prod_{k=1}^{D}\left(o_{\ell,k-1}\cdot P_{\ell,\tau_d}^{(k)} + (1-P_{\ell,\tau_d}^{(k)})\right) = 1,$$

which concludes the proof for O_ℓ. The proof for Z_ℓ is identical.

Remark 5. The (plaintext) definition (6) of $W_{\ell,S}$ applies to subsets $S \subset \mathcal{F}_\ell$. We can naturally extend this definition to $S \subset \mathcal{R}_\ell$ by setting $W_{\ell,S} := W_{\ell,S\cap\mathcal{F}_\ell}$ and note that Lemma 4 still holds.

Let $W_\ell^{P(\mathcal{R}_\ell)} \in [0,1]^{2^D}$ be the vector whose Sth entry is $W_{\ell,S}$ (when indexing the power set $P(\mathcal{R}_\ell)$ by the subsets of $\mathcal{R}_\ell$ as described in Section 3.2.2).

Lemma 6. *We obliviously compute $W_\ell^{P(\mathcal{R}_\ell)}$ as:*

$$W_\ell^{P(\mathcal{R}_\ell)} = \prod_{d=1}^{D} \mathtt{concat}_d(O_{\ell,d}, Z_{\ell,d}),$$

where $\mathtt{concat}_d(a,b) = (\underbrace{a,\dots,a}_{2^{D-d}},\underbrace{b,\dots,b}_{2^{D-d}},\dots,\underbrace{a,\dots,a}_{2^{D-d}},\underbrace{b,\dots,b}_{2^{D-d}}) \in \mathbb{R}^{2^D}$.

Proof. According to Lemma 4 and Remark 5, for any subset $S \subset \mathcal{R}_\ell$,

$$W_{\ell,S} = \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \in S}} o_n^{(\ell)} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \notin S}} z_n^{(\ell)}.$$

Fix a subset $S \subset \mathcal{R}_\ell$ with corresponding position $1 \leq s \leq 2^D$, and let $b_{D-1} \cdots b_0$ be the binary expansion of $2^D - s$. Recall that $\tau_1 \in S \Leftrightarrow b_{D-1} = 1$, $\tau_2 \in S \Leftrightarrow b_{D-2} = 1$, etc. Since $O_{\ell,d} = \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n = \tau_d}} o_n^{(\ell)}$ and $Z_{\ell,d} = \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n = \tau_d}} z_n^{(\ell)}$ if τ_d is a valid index (and $O_{\ell,d} = Z_{\ell,d} = 1$ if τ_d is an invalid index), one deduces that

$$W_{\ell,S} = \prod_{d=1}^{D} (b_{D-d} \ ? \ O_{\ell,d} \ : \ Z_{\ell,d}).$$

Moreover, the sth entry of $\texttt{concat}_d(a, b)$ equals a if and only if $(2^D - s) \ / \ 2^{D-d}$ is odd (Euclidean division) if and only if $b_{D-d} = 1$, which concludes the proof.

3.2.4 Computing Shapley weights.

To filter subsets that contain valid indices only, we need to make the following definition:

Definition 2. *Let $S \subset \mathcal{R}_\ell$. The* Shapley weight *of S with respect to $\mathcal{F}_\ell$ is*

$$C_S = \begin{cases} \frac{|S|!(|\mathcal{F}_\ell|-1-|S|)!}{|\mathcal{F}_\ell|!} & \text{if } S \subsetneq \mathcal{F}_\ell \\ 0 & \text{otherwise.} \end{cases}$$

Remark 6. The reason why we exclude $S = \mathcal{F}_\ell$ is the inner summation of (12). If $i \in \mathcal{F}_\ell$ then $S \subset \mathcal{F}_\ell \setminus \{i\}$ implies $S \subsetneq \mathcal{F}_\ell$, and if $i \notin \mathcal{F}_\ell$ then the case $S = \mathcal{F}_\ell$ would yield factorial of a negative number.

Since $\mathcal{F}_\ell$ and $|\mathcal{F}_\ell|$ are not known a priori, we first compute a larger public matrix C of size $2^D \times D$ whose dth column $C^{(d)}$ is the vector of the 2^D Shapley weights (one for each subset $S \subset \mathcal{R}_\ell$) for the case $|\mathcal{F}_\ell| = d$. In a second step we compute $\delta_{|\mathcal{F}_\ell|}$, the secret characteristic vector of $|\mathcal{F}_\ell|$, and obliviously select the correct column of C.

We compute the sth entry $C_s^{(d)}$ of the dth column of C as

$$C_s^{(d)} = \begin{cases} \frac{|S|!(d-1-|S|)!}{d!} & \text{if } s > 2^D - 2^d + 1 \\ 0 & \text{otherwise,} \end{cases}$$

for all $d = 1, \ldots, D$ and all $s = 1, \ldots, 2^D$, where the cardinality of the corresponding subset S at position s is given by property iv) from Section 3.2.2. Note that the matrix C is pre-computed (it is the same for all $i \in \{1, \ldots, M\}$ and for all $\ell \in L$).

We are now interested in obliviously computing the secret-shared, one-hot encoding (indicator) vector $\delta_{|\mathcal{F}_\ell|}$ of the secret cardinality $|\mathcal{F}_\ell|$, that is, the binary vector of size D that has a single 1 at the $|\mathcal{F}_\ell|$th position. This is done via the following simple lemma:

Lemma 7. *If $R_{\ell,d}$ denotes the dth entry of the vector $R_\ell = (\underbrace{0,\dots,0}_{D-|\mathcal{F}_\ell|},\underbrace{1,\dots,1}_{|\mathcal{F}_\ell|})$ then*

$$\delta_{|\mathcal{F}_\ell|} = (R_{\ell,D} \oplus R_{\ell,D-1}, \dots, R_{\ell,2} \oplus R_{\ell,1}, R_{\ell,1})^t \in \{0,1\}^D.$$

Since we have R_ℓ available in secret shares, the above lemma yields secret shares for $\delta_{|\mathcal{F}_\ell|}$. Finally, we compute $C^{(|\mathcal{F}_\ell|)}$, the $|\mathcal{F}_\ell|$th column of C, as

$$C^{(|\mathcal{F}_\ell|)} = \sum_{d=1}^{D} \delta_{|\mathcal{F}_\ell|,d} \cdot C^{(d)} \in [0,1]^{2^D}. \tag{13}$$

3.2.5 Computing $\mathbf{t}_\ell$. Let $\mathbf{t}_\ell \in [-1, N-1]^M$ be the vector with ith entry given by

$$t_{\ell,i} = \begin{cases} \prod_{\substack{n\in\mathcal{P}_\ell \\ j_n=i}} o_n^{(\ell)}/z_n^{(\ell)} - 1 & \text{if } i \in \mathcal{F}_\ell \\ 0 & \text{otherwise,} \end{cases}$$

where N is the number of samples the tree model was trained with.

Lemma 8. *We obliviously compute $\mathbf{t}_\ell$ as:*

$$\mathbf{t}_\ell = \prod_{k=1}^{D} \left(o_{\ell,k-1} \cdot z_{\ell,k-1}^{-1} \cdot P_\ell^{(k)} + (1 - P_\ell^{(k)}) \right) - 1,$$

where $P_\ell^{(k)}$ is the kth column of P_ℓ.

Proof. For all $i = 1, \dots, M$, let $T_i := \{1 \le k \le D \colon j_{n_{k-1}^{(\ell)}} = i\}$. For all $k \in T_i$ we have $P_\ell^{(k)} = \mathbf{b}_i$ (the only non-zero entry is at position i), and for all $k \in \{1,\dots,D\} \setminus T_i$ the ith entry of $P_\ell^{(k)}$, which we denote by $P_{\ell,i}^{(k)}$, is 0. Hence, if $i \in \mathcal{F}_\ell$ then

$$\begin{aligned}\prod_{\substack{n\in\mathcal{P}_\ell \\ j_n=i}} o_n^{(\ell)}/z_n^{(\ell)} - 1 &= \prod_{k\in T_i} o_{\ell,k-1} \cdot z_{\ell,k-1}^{-1} - 1 \\ &= \prod_{k=1}^{D} \left(o_{\ell,k-1} \cdot z_{\ell,k-1}^{-1} \cdot P_{\ell,i}^{(k)} + (1 - P_{\ell,i}^{(k)}) \right) - 1.\end{aligned}$$

And if $i \notin \mathcal{F}_\ell$ then $T_i = \varnothing$ and hence,

$$\prod_{k=1}^{D} \left(o_{\ell,k-1} \cdot z_{\ell,k-1}^{-1} \cdot P_{\ell,i}^{(k)} + (1 - P_{\ell,i}^{(k)}) \right) - 1 = \prod_{k=1}^{D} 1 - 1 = 0.$$

3.2.6 Putting everything together. For $i \in \{1, \dots, M\}$, let $\mathtt{I}_\ell^{(i)} \in \{0,1\}^{2^D}$ be the indicator vector of $\{S \subset \mathcal{R}_\ell \colon i \notin S\}$, i.e., $\mathtt{I}_{\ell,s}^{(i)} = 1$ if and only if $i \notin S$, where $S \subset \mathcal{R}_\ell$ is the subset corresponding to position s.

Lemma 9. *We have*

$$I_\ell^{(i)} = \neg \left(\bigoplus_d^D \mathtt{concat}_d(\eta_{\ell,d}^{(i)}, 0) \right),$$

where $\eta_\ell^{(i)} = \pi_D \circ \sigma_{F_\ell} \left((\underbrace{0, \dots, 0}_{i-1}, F_{\ell,i}, \underbrace{0, \dots, 0}_{M-i}) \right) \in \{0,1\}^D.$

Theorem 2. *We obliviously compute (in SMPC) secret shares for $\phi_i(x)$ as follows:*

$$\phi_i(x) = \sum_{\ell \in L} \left(\sum_{s=1}^{2^D} I_{\ell,s}^{(i)} \cdot C_s^{(|\mathcal{F}_\ell|)} \cdot W_{\ell,s}^{P(\mathcal{R}_\ell)} \right) \cdot t_{\ell,i} \cdot w_\ell. \tag{14}$$

Proof. We want to show that (14) computes (9). For all leaves $\ell \in L$ we have $t_{\ell,i} = 0$ whenever $i \in \{1, \dots, M\} \setminus \mathcal{F}_\ell$ and hence, (9) reduces to

$$\sum_{\ell \in L} \left(\sum_{S \subset \mathcal{F}_\ell \setminus \{i\}} \frac{|S|!(|\mathcal{F}_\ell| - 1 - |S|)!}{|\mathcal{F}_\ell|!} \cdot W_{\ell,S} \right) \cdot t_{\ell,i} \cdot w_\ell.$$

Regarding the inner sum, for each leaf $\ell \in L$, the subsets $S \subsetneq \mathcal{F}_\ell$ correspond to the positions $s = 2^D - 2^{|\mathcal{F}_\ell|} + 2, \dots, 2^D$ with respect to the ordering defined in Section 3.2.2. Moreover, $C_s^{(|\mathcal{F}_\ell|)} = 0$ for $s = 1, \dots, 2^D - 2^{|\mathcal{F}_\ell|} + 1$ and $C_s^{(|\mathcal{F}_\ell|)} = \frac{|S|!(|\mathcal{F}_\ell|-1-|S|)!}{|\mathcal{F}_\ell|!}$ for $s = 2^D - 2^{|\mathcal{F}_\ell|} + 2, \dots, 2^D$. Note that the Shapley weight depends on the cardinality of S only, which by iv) of Section 3.2.2 is uniquely determined by the position s in the ordering (and not the ordering itself). Since $W_\ell^{P(\mathcal{R}_\ell)}$ is ordered in a way that respects iv) of Section 3.2.2, for each subset $S \subsetneq \mathcal{F}_\ell$ the quantity $W_{\ell,S}$ is multiplied with the right Shapley weight $\frac{|S|!(|\mathcal{F}_\ell|-1-|S|)!}{|\mathcal{F}_\ell|!}$.

Finally, the characteristic vector $\mathtt{I}_\ell^{(i)}$ annihilates the contribution of all subsets S that contain i, which concludes the proof.

3.3 `XorSHAP` for the 0-cover case.

As pointed out in Remark 1, the definition of $W_{\ell,S}(x)$, see (6), is only valid if all split nodes n have positive cover (i.e., $c_n > 0$). In a plaintext training algorithm, any split node with 0-cover will be removed during pruning. This, however, modifies the tree structure. A *full binary tree* of depth D is a tree with $2^D - 1$ split nodes and 2^D leaf nodes. In order for `XorSHAP` to work correctly in the presence of 0-cover nodes, the full binary trees of the model are supposed to be pruned during training in a way that preserves the tree structure.

By *pruning a node* of a decision tree we mean the process of replacing the subtree rooted at that node by a leaf node with some specified weight.

Definition 3. *Let `Pruning` be any plaintext pruning algorithm. Its tree structure-preserving counterpart `PruningFBT` is the following algorithm: suppose that `Pruning` replaces the subtree rooted at a node n by a single leaf node with some weight w; then, `PruningFBT` is the algorithm that keeps the binary subtree rooted at n and only replaces all leaf weights of that subtree by w.*

A pruned full binary tree by a `PruningFBT` is thus still a full binary tree, however, as soon as a sample evaluates through a split node n that got pruned the predicted value no longer depends on the taken sub-path starting from n. While evaluating a `FBT`-pruned full binary tree `Tree` that contains 0-cover nodes is straightforward, computing the conditional expectation $\mathbb{E}[\texttt{Tree}(\xi) \mid \xi_S = x_S]$ is not.

Let `Tree` be a pruned full binary tree (it may contain 0-cover nodes). We define $G'_S(x;n)$, a generalization of $G_S(x;n)$, see (5), as

$$G'_S(x;n) = \begin{cases} w_n & \text{if } n \text{ is a leaf node,} \\ G'_S(x;n_{\text{right}}) & \text{else if } c_{n_{\text{left}}} = 0, \\ G'_S(x;n_{\text{left}}) & \text{else if } c_{n_{\text{right}}} = 0, \\ (x_{j_n} < t_n) \ ? \ G'_S(x;n_{\text{left}}) \ : \ G'_S(x;n_{\text{right}}) & \text{else if } j_n \in S, \\ \frac{c_{n_{\text{left}}}}{c_n} G'_S(x;n_{\text{left}}) + \frac{c_{n_{\text{right}}}}{c_n} G'_S(x;n_{\text{right}}) & \text{else.} \end{cases} \tag{15}$$

Lemma 10. *Let `Tree` be a full binary tree containing 0-cover nodes and let `Pruning` be a plaintext pruning algorithm, with `PruningFBT` its tree structure preserving counterpart. If $G'_S(x;n_{root})$ is computed with `PruningFBT(Tree)` then*

$$\mathbb{E}[\texttt{Pruning(Tree)}(\xi) \mid \xi_S = x_S] = G'_S(x;n_{root}).$$

Proof. Suppose that sample x evaluates through a split node n with non-zero cover, however a child node of n has zero cover. Without loss of generality assume that $c_{n_{\text{left}}} = 0$. Since `Pruning` replaces n by a leaf node (say with weight w), all leaf nodes of the full subtree of `PruningFBT(Tree)` with root n will have weight w. A simple induction then shows that $G'_S(x;n_{\text{right}}) = w$.

We can extend Definition 1 to a notion of consistency between a path $\mathcal{P}_\ell$ and a pair (x,S) of a sample and a feature subset to the case where a full binary tree contains 0-cover nodes.

Definition 4. *We call a path $\mathcal{P}_\ell$* consistent *with respect to the pair (x,S) if every node in $\mathcal{P}_\ell \cup \{\ell\}$ has non-zero cover and if for every node $n \in \mathcal{P}_\ell$ such that $j_n \in S$, either the child node determined by the condition $x_{j_n} < t_n$ coincides with the node of $\mathcal{P}_\ell \cup \{\ell\}$ that succeeds n in the path or the child node determined by the condition $\neg(x_{j_n} < t_n)$ has zero cover. Otherwise, the path is called* inconsistent.

The quantity $W_{\ell,S}$, see (6), can naturally be extended to the case of full binary trees containing 0-cover nodes, with the notion of consistency introduced in Definition 4.

It remains to show how to obliviously evaluate $G'_S(x;n)$ for a pruned oblivious (i.e., full binary) tree `Tree`. Note that only the weights of leaves whose path is consistent with (x,S) contribute to $G'_S(x;n)$. Unlike `eval`(x, `Tree`), which can be obliviously computed from the secret bits $\beta = (x_j < t)$, see [DDG+22], one needs to consider more than the bits β to evaluate $G'_S(x;n)$. Recall that for the evaluation of $G_S(x;n)$, one computes $\mathbf{o}_\ell$ from the bits β only, see Lemma 3.

Given a leaf node $\ell \in L$, let $\mathcal{P}_\ell$ be the path of split nodes leading to ℓ. We define $\tilde{\mathbf{o}}_\ell = (\tilde{o}_n^{(\ell)} : n \in \mathcal{P}_\ell)$, similar to $\mathbf{o}_\ell$, where $\tilde{o}_n^{(\ell)} = 1$ if the successor node n_{child} of n in $\mathcal{P}_\ell \cup \{\ell\}$ has non-zero cover and either evaluating node n on the sample x yields n_{child} or the sibling node of n_{child} (when looking at the whole tree) has zero cover; otherwise $\tilde{o}_n^{(\ell)} = 0$.

Lemma 11.

$$W_{\ell,S}(x) = \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \in S}} \tilde{o}_n^{(\ell)} \prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \notin S}} z_n^{(\ell)}. \tag{16}$$

Proof. If $\mathcal{P}_\ell \cup \{\ell\}$ contains a 0-cover node then the path is inconsistent with (x,S). In this case, let n be the unique node in the path with non-zero cover and whose successor node n_{child} has zero cover (a child of a 0-cover node has zero cover, so n is unique in the path). If $j_n \in S$ then $\tilde{o}_n^{(\ell)} = 0$, else $z_n^{(\ell)} = \frac{c_{n_{\text{child}}}}{c_n} = 0$.

If $\mathcal{P}_\ell \cup \{\ell\}$ does not contain any 0-cover node then $\mathcal{P}_\ell$ is consistent with (x,S) if and only if $\prod_{\substack{n \in \mathcal{P}_\ell \\ j_n \in S}} \tilde{o}_n^{(\ell)} = 1$.

Note that for any 0-cover split node n the ratio $z_n^{(\ell)}$ is not well-defined. A data-oblivious division algorithm is expected to return arbitrary output if the denominator is 0 and hence, the presence of 0-cover split nodes in the path does not invalidate Lemma 11.

It remains to show how to obliviously compute $\tilde{\mathbf{o}}_\ell$. One then applies Lemma 5 and Lemma 6 with $\tilde{\mathbf{o}}_\ell$ to obliviously compute $\phi_i(x)$ as in Theorem 2. Fix an index d. As in Lemma 3, let (j,t) be the feature index and threshold associated to node $n_d^{(\ell)}$, and let $\beta = (x_j < t) \in \{0,1\}$. Let n_{left} and n_{right} be the left and right children nodes of $n_d^{(\ell)}$ respectively (when looking at the whole tree). Consider the following bits

$$\gamma_{\text{left}} = ((c_{n_{\text{left}}} == 0)\ ?\ 0 : ((c_{n_{\text{right}}} == 0)\ ?\ 1 : \beta))$$

and

$$\gamma_{\text{right}} = \neg\gamma_{\text{left}} \oplus (c_{n_{\text{left}}} == 0) \wedge (c_{n_{\text{right}}} == 0),$$

which can be computed obliviously.

Lemma 12.

$$\tilde{o}_{\ell,d} = \begin{cases} \gamma_{left} & \text{if } n_{child} = n_{left} \\ \gamma_{right} & \text{if } n_{child} = n_{right} \end{cases}$$

Proof. We have:

- if $c_{n_{\text{left}}} > 0$ and $c_{n_{\text{right}}} > 0$ then $\gamma_{\text{left}} = \beta$ and $\gamma_{\text{right}} = \neg\beta$,
- if $c_{n_{\text{left}}} > 0$ and $c_{n_{\text{right}}} = 0$ then $\gamma_{\text{left}} = 1$ and $\gamma_{\text{right}} = 0$,
- if $c_{n_{\text{left}}} = 0$ and $c_{n_{\text{right}}} > 0$ then $\gamma_{\text{left}} = 0$ and $\gamma_{\text{right}} = 1$,
- if $c_{n_{\text{left}}} = 0$ and $c_{n_{\text{right}}} = 0$ then $\gamma_{\text{left}} = 0$ and $\gamma_{\text{right}} = 0$.

4 Complexity Analysis

For analyzing the complexity of `XorSHAP`, consider a model consisting of a single tree. For an arbitrary tree ensemble model, one can simply multiply the complexity by the number T of decision trees in the ensemble.

Below, we describe in detail the complexities of the various computations needed in the different sections of Section 3.

§3.1
- There are 2^{D-1} binary matrices P_ℓ of size $M \times D$ and 2^{D-1} binary matrices F_ℓ of size $M \times 1$ (obtained by OR-ing the columns of P_ℓ). The number is 2^{D-1} since sibling leaves share the same path of split nodes.
- Computing all 2^D $\mathbf{o}_\ell$'s requires a total of $2^D - 1$ comparisons of the form $t_n < x_{j_n}$.
- Computing all 2^D $\mathbf{z}_\ell$'s requires one private division between two vectors of size $2^D - 1 \times 1$ (there is a relation between the cover ratio of a left child node and the cover ratio of its right sibling node, so it suffices to compute the left ratios only).

§3.2.1
- The 2^{D-1} permutations $\sigma_{\mathcal{F}_\ell}$ are obtained by sorting the 2^{D-1} binary vectors F_ℓ. Note that $\sigma_{\mathcal{F}_\ell}(F_\ell)$ is a by-product of the sorting algorithm from [BCD+23], and R_ℓ can therefore be computed locally.

§3.2.3
- There are 2^D binary vectors O_ℓ of size $D \times 1$, requiring D ANDs between a bit and a binary vector of size $M \times 1$, $D - 1$ ANDs between binary vectors of size $M \times 1$ and one call to σ_{F_ℓ} each.
- There are 2^D vectors Z_ℓ of size $D \times 1$, requiring D multiplications between a scalar and a binary vector of size $M \times 1$, $D - 1$ multiplications between vectors of size $M \times 1$ and one call to σ_{F_ℓ} each.
- There are 2^D vectors $W_\ell^{P(\mathcal{R}_\ell)}$ of size $2^D \times 1$, requiring $D - 1$ multiplications between vectors of size $2^D \times 1$ each.

§3.2.4
- C is a publicly computed matrix of size $2^D \times D$.
- There are 2^D binary vectors $\delta_{|\mathcal{F}_\ell|}$ of size $D \times 1$, which can be computed locally from the binary vector R_ℓ. There are 2^D vectors $C^{|\mathcal{F}_\ell|}$ of size $2^D \times 1$ which are computed as the matrix multiplication between the public matrix C and the secret vector $\delta_{|\mathcal{F}_\ell|}$.

§3.2.5

– First, we need to compute all inverse cover ratios $1/z_n^{(\ell)}$, which requires two private divisions between two vectors of size $2^D - 1 \times 1$ (unlike for the computation of $\mathbf{z}_\ell$, there is no easily computable relation between the inverse cover ratio of a left child node and the inverse cover ratio of its right sibling node).
– There are 2^D vectors $\mathbf{t}_\ell$ of size $M \times 1$, requiring D multiplications between two scalars, D multiplications between a scalar and a binary vector of size $M \times 1$ and $D-1$ multiplications between vectors of size $M \times 1$ each.

§3.2.6
– Computing the $M2^{D-1}$ binary vectors $\eta_\ell^{(i)}$ respectively $I_\ell^{(i)}$ requires $M2^{D-1}$ permutations of binary vectors of size $M \times 1$, the remaining computations can be performed locally.
– Finally, (14) is computed with: for each of the 2^D leaves ℓ the inner sum requires $2 \cdot 2^D$ scalar multiplications, and the outer sum requires $2 \cdot 2^D$ scalar multiplications. That is, a total of $2^{D+1}(2^D+1)$ scalar multiplications.

Comparison of the above-listed complexities shows that the bottlenecks are:

– Computing all Z_ℓ's from Section 3.2.3 is $O(MD2^D)$,
– Computing $W_\ell^{P(\mathcal{R}_\ell)}$ from Section 3.2.3 is $O(D2^{2D})$,
– Computing $C^{|\mathcal{F}_\ell|}$ from Section 3.2.4 is $O(D2^{2D})$,
– Computing all $\mathbf{t}_\ell$'s is $O(MD2^D)$,
– Computing (14) is $O(2^{2D+1})$.

Hence, if $\tilde{M} = \max(M, 2^D)$, then `XorSHAP` runs in

$$O(T\tilde{M}D2^D)$$

multiplications, where T is the number of trees.

5 Implementation and Benchmarks

We have implemented the above algorithm on the `Manticore` framework [BCD+23]. The offline phase (generation of random precomputed data) is performed by the trusted dealer. The benchmarks were run in a 2-party scenario with input data already provided in secret shares across the two parties (i.e., simulating any kind of partition). We vary the number of trees T, the tree depth D, the number of features M as well as the number of testing samples (simultaneous computation of the SHAP values). The benchmarks include offline and online phase and were run on an Intel Xeon E5-2666 v3 @ 2.90GHz CPU with 32GB RAM. Over all runs, the L_∞-distance between the SHAP values computed in MPC and the baseline plaintext computation with the classical SHAP implementaiton[2] is around 10^{-13}.

[2] See `https://shap.readthedocs.io/en/latest/` as well as the function `shap.TreeExplainer.shap_values(x)`.

In Table 1 and Table 2, we provide benchmarks (offline/online runtime and number of outgoing connections) and (network transfer per party and memory overhead) respectively, across different values of T for simultaneous computations of 20 samples/predictions, for $M = 20$ features and binary trees of depth $D = 6$. Similarly, in Table 3 and Table 4, we provide the same type of data for varying number of features M.

Figure 1 and Figure 2 show the total runtime for varying number of trees and tree depth, and varying number of features and number of simultaneous evaluations respectively. The time on the y-axis consists of the offline compute time plus the online compute time plus and estimate of the network transfer time. The latter is computed by using the network transfer (offline triplet size and online network size), number of outgoing connections, the throughput and the latency of the network. The estimation uses a network whose throughput is 120Mbps and whose latency is 0.3ms.

T	offline compute time (in s)	online compute time (in s)	number of outgoing connections
100	97	95	334
80	75	74	334
60	57	55	334
40	37	36	334
20	17	14	334

Table 1. #samples = 20, $M = 20$, $D = 6$

T	network transfer (in MB, per party, offline + online)	memory offline (in MB)	memory online (in MB)
100	5329	3867	11679
80	4263	3112	9379
60	3196	2346	7056
40	2131	1572	4744
20	1065	806	2430

Table 2. #samples = 20, $M = 20$, $D = 6$

6 Conclusion

The proposed XorSHAP is one of the first attempt to compute Shapley values in an efficient and scalable fashion for decision tree ensemble models in the generic

M	offline compute time (in s)	online compute time (in s)	number of outgoing connections
100	64	60	399
80	50	49	379
60	38	38	356
40	26	25	336
20	15	14	313

Table 3. $D = 4$, $T = 60$, #samples = 100

M	network transfer (in MB, per party, offline + online)	memory offline (in MB)	memory online (in MB)
100	4606	2834	5836
80	3751	2277	4691
60	2897	1739	3554
40	2043	1191	2444
20	1189	632	1295

Table 4. $D = 4$, $T = 60$, #samples = 100

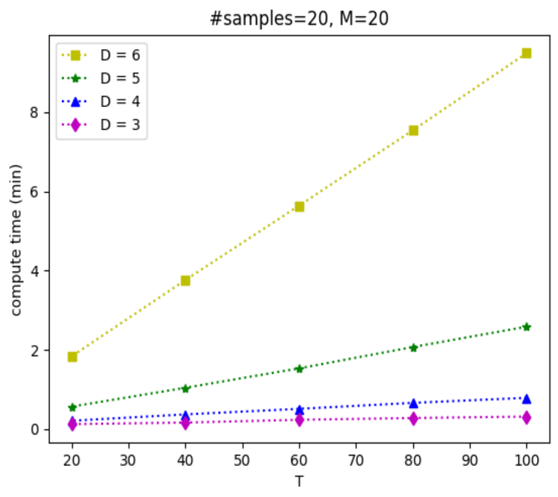

Fig. 1.

secure multiparty computation setting. On a decision tree ensemble model with T binary decision trees, with M features in depth D, our algorithm has a runtime $O(T\widetilde{M}D2^D)$ where $\widetilde{M} = \max(M, 2^D)$, thus, being linear in the number of features.

We implemented `XorSHAP` using the `Manticore` framework for Secure Multiparty Computation (SMPC) with additive secret sharing using modular real secret shares and Boolean secret shares. The implementation scales to real-world

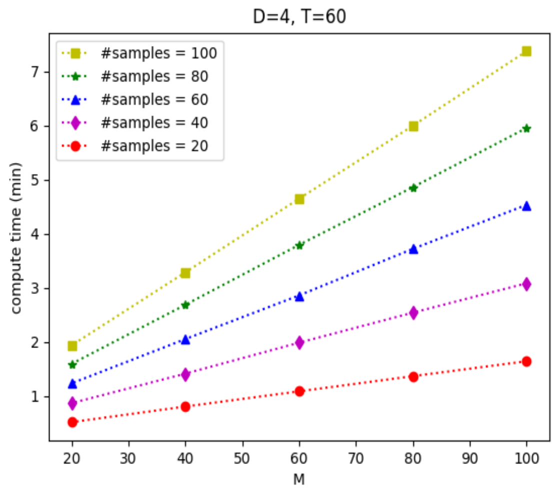

Fig. 2.

datasets and is already part of Inpher XOR Platform for privacy-preserving computations.

Note that the current implementation of `XorSHAP` takes the input in the most general form, namely, in secret shares. A promising future direction of research would be to understand how one can scale the algorithm in the cases when the data is either horizontally or vertically split across multiple parties by using local plaintext computations at the distinct parties, thus minimizing the overhead from the more expensive SMPC operations. This is closer to a typical federated machine learning scenario and is often expected to be much easier to scale than scenarios where the inputs are secret shared.

Finally, the parallelization-friendly nature of our method opens another future research direction on massively scaling the computation using hardware acceleration with GPUs. One specific line of research along these lines is finding privacy-preserving variants of `GPUTreeSHAP` proposed in [MFH22].

References

Act23. EU AI Act. European Union Artificial Intelligence Act. *https://artificialintelligenceact.eu/*, 2023.

aM23. Scale and Mamba. Scale and mamba. `https://github.com/KULeuven-COSIC/SCALE-MAMBA`, 2023.

BCD$^+$23. M. Georgieva Belorgey, S. Carpov, K. Deforth, D. Jetchev, A. Sae-Tang, M. Vuille, N. Gama, J. Katz, I. Leontiadis, and M. Mohammadi. Manticore: A framework for efficient multiparty computation supporting real number and boolean arithmetic. *J. Cryptol.*, 36(3):31, 2023.

BCG$^+$17. C. Boura, I. Chillotti, N. Gama, D. Jetchev, S. Peceny, and A. Petric. High-precision privacy-preserving real-valued function evaluation. *IACR Cryptology ePrint Archive*, 2017.

BIS$^+$22. A. Bogdanova, A. Imakura, T. Sakurai, T. Fujii, T. Sakamoto, and H. Abe. Achieving transparency in distributed machine learning with explainable data collaboration. *CoRR*, abs/2212.03373, 2022.

BLW08. D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

CDE$^+$18. R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. SPD$\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority. In *Advances in Cryptology – CRYPTO 2018*, pages 769–798, 2018.

DDG$^+$22. K. Deforth, M. Desgroseilliers, N. Gama, M. Georgieva, D. Jetchev, and M. Vuille. XORBoost: Tree boosting in the multiparty computation setting. *Proc. Priv. Enhancing Technol.*, 2022(4):66–85, 2022.

DH12. W.J. Dally and R.C. Harting. *Digital Design: A Systems Approach*. Digital Design: A Systems Approach. Cambridge University Press, 2012.

DPSZ12. I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

EGK$^+$20. D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *40th Annual International Cryptology Conference, CRYPTO*, volume 12171 of *Lecture Notes in Computer Science*, pages 823–852, 2020.

Kel20. M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1575–1590, 2020.

KOS16. M. Keller, E. Orsini, and P. Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

KPR18. M. Keller, V. Pastro, and D. Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT 2018*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189, 2018.

LEC$^+$20. S. Lundberg, G. Erion, H. Chen, A. DeGrave, J. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. From local explanations to global understanding with explainable AI for trees. *Nat. Mach. Intell.*, 2(1):56–67, 2020.

LEL18. S.. Lundberg, G. Erion, and S. Lee. Consistent individualized feature attribution for tree ensembles. *CoRR*, abs/1802.03888, 2018.

LL17. S. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

MFH22. R. Mitchell, E. Frank, and G. Holmes. Gputreeshap: massively parallel exact calculation of SHAP scores for tree ensembles. *PeerJ Comput. Sci.*, 8:e880, 2022.

Mol22. C. Molnar. *Interpretable Machine Learning*. 2 edition, 2022.

MR18. P. Mohassel and P. Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and

XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018.

MZ17. P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017.

PSSY21. A. Patra, T. Schneider, A. Suresh, and H. Yalame. Aby2. 0: Improved mixed-protocol secure two-party computation. In *30th USENIX Security Symposium*, 2021.

RSG16. M. Túlio Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the Demonstrations Session, NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 97–101. The Association for Computational Linguistics, 2016.

Sha53. L. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

SK14. E. Strumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowl. Inf. Syst.*, 41(3):647–665, 2014.

TLL$^+$22. Z. Tian, J. Liu, J. Li, X. Cao, R. Jia, and K. Ren. Private data valuation and fair payment in data marketplaces. *CoRR*, abs/2210.08723, 2022.

Wan19. G. Wang. Interpret federated learning with shapley values. *CoRR*, abs/1905.04519, 2019.

WAYS22. L. Watson, R. Andreeva, H.-T. Yang, and R. Sarkar. Differentially private shapley values for data evaluation. *CoRR*, abs/2206.00511, 2022.

WGC19. S. Wagh, D. Gupta, and N. Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, 2019.

Yan21. Jilei Yang. Fast TreeSHAP: Accelerating SHAP value computation for trees. *arXiv preprint arXiv:2109.09847*, 2021.