From Decision Trees to Boolean Logic: A Fast and Unified SHAP Algorithm

Alexander Nadel[1], **Ron Wettenstein**[2]

[1]Faculty of Data and Decision Sciences, Technion, Haifa, Israel
[2]Reichman University, Herzliya, Israel
alexandernad@technion.ac.il, ron.wettenstein@post.runi.ac.il

Abstract

SHapley Additive exPlanations (SHAP) is a key tool for interpreting decision tree ensembles by assigning contribution values to features. It is widely used in finance, advertising, medicine, and other domains. Two main approaches to SHAP calculation exist: *Path-Dependent SHAP*, which leverages the tree structure for efficiency, and *Background SHAP*, which uses a background dataset to estimate feature distributions.

We introduce WOODELF, a SHAP algorithm that integrates decision trees, game theory, and Boolean logic into a unified framework. For each consumer, WOODELF constructs a pseudo-Boolean formula that captures their feature values, the structure of the decision tree ensemble, and the entire background dataset. It then leverages this representation to compute Background SHAP in linear time. WOODELF can also compute Path-Dependent SHAP, Shapley interaction values, Banzhaf values, and Banzhaf interaction values.

WOODELF is designed to run efficiently on CPU and GPU hardware alike. Available via the WOODELF Python package, it is implemented using NumPy, SciPy, and CuPy without relying on custom C++ or CUDA code. This design enables fast performance and seamless integration into existing frameworks, supporting large-scale computation of SHAP and other game-theoretic values in practice.

For example, on a dataset with $3\,000\,000$ rows, $5\,000\,000$ background samples, and 127 features, WOODELF computed all Background Shapley values in 162 seconds on CPU and 16 seconds on GPU—compared to 44 minutes required by the best method on any hardware platform, representing $16\times$ and $165\times$ speedups, respectively.

Full Version — https://arxiv.org/abs/2511.09376
The WOODELF Python Package —
https://github.com/ron-wettenstein/woodelf
Experiment Notebooks —
https://github.com/ron-wettenstein/WoodelfExperiments
IEEE-CIS Dataset —
https://www.kaggle.com/c/ieee-fraud-detection
KDD-Cup Dataset —
https://kdd.ics.uci.edu/databases/kddcup99/kddcup99

1 Introduction

Decision trees are widely used predictive models for classification and regression. To improve predictive accuracy, ensemble methods such as XGBoost (Chen and Guestrin 2016), Random Forest (Breiman 2001), and CatBoost (Dorogush et al. 2017), train multiple decision trees and average their predictions.

Recent efforts focus on explaining models using feature attributions. Local attribution shows how each feature affects a single prediction, which is often crucial for regulatory compliance (Knight 2019; Selbst and Powles 2017). Global attribution evaluates which features matter most overall, often by combining many local attributions (Covert, Lundberg, and Lee 2020). This is essential for model comprehension and feature selection.

SHAP (SHapley Additive exPlanations) (Lundberg et al. 2020) is a widely preferred method for both local and global feature attribution (Gill et al. 2020; Hall and Gill 2019). It assigns Shapley values to features, providing a unified (Lundberg and Lee 2017) and consistent (Lundberg, Erion, and Lee 2019) approach grounded in game theory.

1.1 Shapley Values

Originating from cooperative game theory, Shapley values offer a fair method for distributing profits among players based on their individual contributions. Players whose contributions are crucial to the group's success receive a larger share of the profit, while those with smaller contributions receive less. Players who negatively impact the group's performance may receive negative payments.

Shapley values constitute the unique solution satisfying four key properties: efficiency, null player, symmetry, and linearity (Shapley 1953). The Shapley value formula uses the game's characteristic function to evaluate the player's impact across all possible coalitions.

Definition 1 (Characteristic Function)**.** *For a group of players N, the* characteristic function *M is a function of the form $M : 2^{[N]} \to \mathbb{R}$ mapping any subset $S \subseteq N$ to the profit when only players in S participate.*

The Shapley value formula for player i, shown below, considers all subsets excluding i and compares the profit with and without the player. These contributions are then normalized by a factor that depends on the coalition size.

$$\phi_i(M) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} (M(S \cup \{i\}) - M(S)) \tag{1}$$

A naive Shapley values calculation takes exponential time, as the formula considers all possible subsets of $N \setminus \{i\}$. However, efficient computation is possible for certain scenarios, including decision tree ensembles (Lundberg et al. 2020), certain Boolean circuits classes (Arenas, Bertossi, and Monet 2021), and tuples in query answering (Livshits et al. 2021). For other scenarios like neural networks where the computation is #P-Hard (den Broeck et al. 2021; Huang and Marques-Silva 2024), approximation methods exist (Chen et al. 2023).

1.2 Feature Importance Using Shapley Values

A predictive model, like a decision tree ensemble, can be viewed as a "game", where feature values act as "players" and the model's prediction represents the "game's profit". In this context, Shapley values quantify how each feature ("player") affected the prediction (the "game's profit").

Our goal is to define the characteristic function of this "game" and compute its Shapley values. This requires specifying the model's output when only a subset of features is present, while the rest are considered missing. Several definitions exist for handling missing features; see (Sundararajan and Najmi 2020) for a survey. Of these, three are most commonly used today:

- *Baseline SHAP*: Assigns each feature a fixed baseline value used whenever the feature is missing.
- *Path-Dependent SHAP*: Leverages the tree structure and the cover property (i.e., how many rows reached each node during training) to infer the effect of missing features.
- *Background SHAP*: Replaces fixed baselines with a background dataset. When features are missing, their values are taken from this dataset, predictions are computed, and results are averaged. This method is the most accurate, see the full version for an example.

(Lundberg et al. 2020) were the first to present a polynomial-time algorithm for the three SHAP approaches discussed above. Their method efficiently tracks the number of feature subsets that reach each node, avoiding the need to enumerate them explicitly. These algorithms are implemented in the widely used SHAP Python package.

Since then, several works have improved these methods. FastTreeShap v2 (Yang 2022) accelerates Path-Dependent SHAP by extracting key information from the decision tree in advance. PLTreeShap (Zern, Broelemann, and Kasneci 2023), which preprocesses both the decision tree and the background dataset, reduces the complexity of Background SHAP from $O(mn)$—where n is the number of consumers and m is the background size—to $O(m + n)$.

GPU-based implementations include GPUTreeSHAP (Mitchell, Frank, and Holmes 2022), which computes Shapley values for each path in parallel and uses optimized bin-packing techniques to distribute work across GPU warps, and FourierSHAP (Gorji, Amrollahi, and Krause 2025), which performs well on models with a small number of features.

1.3 Paper Outline and Our Contribution

Sects. 2 and 3 introduce a novel linear-time algorithm for computing Shapley values, Shapley interaction values, Banzhaf values, and Banzhaf interaction values (Grabisch and Roubens 1999) for formulas in Weighted Disjunctive Normal Form (WDNF) (Zhang and Jang 2005). Our main contribution, WOODELF, is introduced in Sect. 9, with several preliminary steps presented in Sects. 4, 5, 6, 7 and 8.

WOODELF is a **unified**, **generic**, **GPU-friendly** and **efficient** approach for SHAP calculation:

- **Unified** across Path-Dependent and Background SHAP, demonstrating that a single algorithm can handle the two problems previously thought to require distinct approaches.
- **Metric-Generic**: WOODELF can compute Shapley values, Shapley interaction values, Banzhaf values, Banzhaf interaction values, and any other value over the Path-Dependent or Background characteristic functions that satisfies the linearity property. It is the first algorithm that supports such a broad range of metrics.
- **GPU-friendly** and **pure Python**: Unlike existing approaches, in WOODELF, all the major algorithmic steps can be expressed as standard vectorized operations, making the algorithm inherently Single Instruction, Multiple Data (SIMD)- and GPU-friendly. Implementation-wise WOODELF is written entirely in Python, with the bottleneck operations implemented in NumPy and SciPy. By using CuPy, these operations can seamlessly run on GPUs without any custom CUDA code. In contrast, SHAP and other state-of-the-art (SOTA) implementations rely heavily on custom C++ and CUDA. WOODELF achieves high efficiency while maintaining a pure Python design, simplifying integration and extensibility.
- **Efficient**: By utilizing vectorized operations alongside efficient algorithmics (Sect. 9.1), WOODELF significantly advances SOTA performance. In Sect. 10, we demonstrate WOODELF's effectiveness on two large industrial datasets, achieving 24× to 333× speed-ups on GPU and 16× to 31× speed-ups on CPU, compared to the SOTA Background SHAP on any hardware platform.

2 Linear-Time SHAP Calculation for WDNF

Towards defining WDNF, we need additional notations. A *literal* is a Boolean variable x_i or its negation $\neg x_i$. A *cube* is a conjunction (set) of literals.

Definition 2 (Pseudo-Boolean (PB) Function and Weighted Disjunctive Normal Form (WDNF))**.** *A* pseudo-Boolean (PB) *function is a function of the form* $F(x_1, \ldots, x_h) : \{0,1\}^h \to \mathbb{R}$. *A* Weighted Disjunctive Normal Form (WDNF) *formula (Zhang and Jang 2005) is a PB function expressed as:*

	Game	Decision Tree	Pseudo-Boolean Function
The Game		The model: T	The function: F
The Players		The features: f_1=v_1, f_2=v_2, f_3=v_3	The variables: x_1, x_2, x_3
The Profit		The prediction: T(v_1, v_2, v_3)	F(x_1=1, x_2=1, x_3=1)
Missing Feature Definition	Participating:  Missing: 	Participating: f_1=v_1, f_2=v_2, f_3=v_3 Missing: f_1=b_1, f_2=b_2, f_3=b_3	Participating: x_1=1, x_2=1, x_3=1 Missing: x_1=0, x_2=0, x_3=0
Profit when Player 2 is Missing		The prediction: T(v_1, b_2, v_3)	F(x_1=1, x_2=0, x_3=1)

Figure 1: An illustration how both PB functions and decision trees relate to well-established concepts in game theory. In decision trees, the model represents a game, features serve as players, and the prediction corresponds to profit. Under the baseline characteristic function definition, a missing player (e.g., player 2) is set to its baseline value (b_2) before making a prediction. In PB functions, the function itself represents a game, and the variables serve as players. Each variable is `True` when it participates and `False` when it is missing.

$$F(x_1, \ldots, x_h) = \sum_{k=1}^{m} w_k \cdot c_k(x_1, \ldots, x_h)$$

Where each c_k is a cube and $w_k \in \mathbb{R}$ is its weight.

For instance, the PB formula $F(x_1, x_2, x_3) = 3(\neg x_1) + 1(\neg x_1 \wedge x_2) + 5(x_1 \wedge \neg x_2 \wedge x_3)$ is in WDNF. Assigning $x_1 = 0, x_2 = 1, x_3 = 1$ results in $F(0, 1, 1) = 3 + 1 = 4$.

For a cube c_k, we denote by S_k the set of variables in c_k, partitioned into positive variables S_k^+ and negated variables S_k^-. For example, in the cube $c_k \equiv x_1 \wedge \neg x_2 \wedge x_3$, we have $S_k^+ = \{x_1, x_3\}$ and $S_k^- = \{x_2\}$.

A WDNF formula, and any other PB function, can be interpreted as a game where variables are players and the formula's output is the profit (Grabisch and Roubens 1999). The characteristic function (recall Def. 1) of this game is defined by Def. 3. To find the profit of a coalition S, we set $x_i = 1$ for all $i \in S$, $x_i = 0$ for all $i \notin S$, and evaluate the WDNF formula. See Fig. 1 for an illustration.

Definition 3 (PB function's Characteristic Function)**.** *Given a PB function $F(V) : \{0,1\}^h \to \mathbb{R}$ over $V = \{x_1, x_2, \ldots, x_h\}$, its characteristic function $\mathcal{M}_{F(V)}(S)$ is defined for any subset of variables $S \subseteq V$ as follows:*

$$\mathcal{M}_{F(V)}(S) = F\left(\mathbf{x} = \begin{cases} 1 & \text{if } x_i \in S \\ 0 & \text{if } x_i \notin S \end{cases}\right)$$

This means that for each $x_i \in S$, we set $x_i = 1$ and for each $x_i \notin S$, we set $x_i = 0$. The characteristic function then evaluates F under this assignment.

Having defined the characteristic function for a WDNF formula F, we can now compute its Shapley values via Formula 1. In general, computing Shapley values for pseudo-Boolean functions is #P-Hard (see full version and (Arenas, Bertossi, and Monet 2021)). A key insight of this paper is that, for WDNF, Shapley values can be computed in linear time using the following formula:

$$\phi_i(F) = \sum_{k=1}^{m} w_k \times \begin{cases} \frac{1}{|S_k^+| \binom{|S_k|}{|S_k^+|}} & \text{if } i \in S_k^+ \\ \frac{-1}{|S_k^-| \binom{|S_k|}{|S_k^-|}} & \text{if } i \in S_k^- \\ 0 & \text{if } i \notin S_k \end{cases} \quad (2)$$

Prior to evaluating Formula 2 and the formulas presented in Sect 3, we remove all cubes where $S_k^+ \cap S_k^- \neq \emptyset$ as such cubes are unsatisfiable. In the full version, we prove that Formula 2 correctly computes the Shapley values. The proof leverages the linearity and the null player out (NPO) properties of Shapley values. Additionally, we show how Formula 2 can be computed in linear time and applied to Weighted Conjunctive Normal Form (WCNF) formulas (da Silva 2021).

3 Beyond SHAP

In this section, we present formulas that efficiently compute the Shapley interaction values $\phi_{i,j}$ (Table 1), Banzhaf values β_i, and Banzhaf interaction values $\beta_{i,j}$ over WDNFs. Proofs of correctness appear in the full version.

Shapley interaction values measure how the interaction between two features affects the prediction (Grabisch and Roubens 1999). One possible definition is:

Definition 4 (Shapley Interaction Values)**.** *Shapley interaction value of features f_i and f_j is the difference between the Shapley values of f_j when f_i always participates and when f_i is always missing: $\phi_{i,j\ i \neq j} = \phi_{j|i=1} - \phi_{j|i=0}$*

By combining Formula 2 with Def. 4, we can derive simple formulas for Shapley interaction values, see Table 1 for further details:

	$i \in S_k^+$	$i \in S_k^-$	$i \notin S_k$
$j \in S_k^+$	$\frac{w}{(\lvert S_k^+\rvert -1)\binom{\lvert S_k\rvert -1}{\lvert S_k^+\rvert -1}}$	$\frac{-w}{\lvert S_k^+\rvert \binom{\lvert S_k\rvert -1}{\lvert S_k^+\rvert}}$	0
$j \in S_k^-$	$\frac{-w}{\lvert S_k^-\rvert \binom{\lvert S_k\rvert -1}{\lvert S_k^-\rvert}}$	$\frac{w}{(\lvert S_k^-\rvert -1)\binom{\lvert S_k\rvert -1}{\lvert S_k^-\rvert -1}}$	0
$j \notin S_k$	0	0	0

Table 1: To calculate $\phi_{i,j}$ iterate through all the cubes of the WDNF formula. For each cube c_k and pair of variables $i, j \in S_k$ select the appropriate cell from the table above and apply its formula.

Banzhaf values satisfy three of Shapley's four properties: null player, symmetry, and linearity. However, they do not satisfy efficiency (Banzhaf 1965). They also possess another useful property: the Banzhaf value of player i, equals the difference in the expected game's profit, under a uniform distribution over all subsets, with and without i:

$$\beta_i(M) = \mathbb{E}[M(S)|i \in S] - \mathbb{E}[M(S)|i \notin S] \tag{3}$$

Previous work has shown how to calculate Banzhaf values on decision trees (Karczmarz et al. 2022; Muschalik et al. 2024), facts in query answering (Abramovich et al. 2023), and on other tasks. The formula below enables their linear-time computation on a WDNF formula:

$$\beta_i(F) = \sum_{k=1}^{m} \frac{w_k}{2^{|S_k|-1}} \times \begin{cases} 1 & \text{if } i \in S_k^+ \\ -1 & \text{if } i \in S_k^- \\ 0 & \text{if } i \notin S_k \end{cases} \tag{4}$$

Banzhaf interaction values examine the difference in expectations when feature i and j are both missing/participating versus when only one participates (Fujimoto, Kojadinovic, and Marichal 2006). Given a WDNF they be computed using the formula below:

$$\beta_{i,j\ i\neq j}(F) = \sum_{k=1}^{m} \frac{w_k}{2^{|S_k|-2}} \times \begin{cases} 1 & \text{if } i,j \in S_k^+ \\ 1 & \text{if } i,j \in S_k^- \\ -1 & \text{if } i \in S_k^+ \wedge j \in S_k^- \\ -1 & \text{if } i \in S_k^- \wedge j \in S_k^+ \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

4 Decision Pattern

This section introduces the concept of a *decision pattern*, central to WOODELF. We begin by defining decision trees and root-to-leaf paths, and then build on these foundations to define decision patterns and present the efficient *CalcDecisionPatterns* algorithm for computing them.

Definition 5 (Decision Tree)**.** *A* decision tree *is a rooted binary tree $T = (N_T = \{L_T \cup I_T\}, E_T, r_T)$, where:*

- *$r_T \in N_T$ is the* root node.
- *Each node $n \in N_T$ is either a childless* leaf *$l \in L_T$ or an* inner node *$n \in I_T$ with two children: $n.left$ and $n.right$.*

Algorithm 1: Mapping Each Leaf to Its Decision Patterns

```
function CalcDecisionPatterns(T, C)
    P_leaves ← {}
    P_all ← {r_T : (0)_{∀c∈C}}
    for n in BFS(T) do:
        if n is a leaf then
            P_leaves[n] = P_all[n]
        else
            P_all[n.left] = (P_all[n] << 1) + n.split(C)
            P_all[n.right] = (P_all[n] << 1)+¬n.split(C)
    return P_leaves
```

- *A leaf $l \in L_T$ stores an output value $w_l \in \mathbb{R}$.*
- *An inner node $n \in I_T$ is associated with a feature $n.feature \in \{1, \ldots, h\}$ and a threshold value $\theta_n \in \mathbb{R}$.*

For a node n and consumer feature values $c = (c_1, c_2, \ldots, c_h) \in \mathbb{R}^h$, we define the function $n.split(c)$:

$$n.split(c) = \begin{cases} True & \text{if } c_{n.feature} < \theta_n \\ False & \text{otherwise} \end{cases}$$

Definition 6 (Root-to-Leaf Path)**.** *Given a decision tree T and its leaf $l \in L_T$, the* root-to-leaf path *of l is the unique simple path from the root r_T to l: $(n_1 \equiv r_T, n_2, \ldots, n_{D-1}, n_D \equiv l)$.*

Definition 7 (Decision Pattern)**.** *Given a decision tree T, its leaf $l \in L_T$, their root-to-leaf path $(n_1 \equiv r_T, n_2, \ldots, n_{D-1}, l)$, and consumer feature values $c = (c_1, c_2, \ldots, c_h) \in \mathbb{R}^h$, the* decision pattern *p is a binary sequence of length $D-1$. The i'th bit in the sequence is:*

$$p[i] = \begin{cases} 1 & \text{if } (n_i.split(c) = True) \wedge (n_i.left = n_{i+1}) \\ 1 & \text{if } (n_i.split(c) = False) \wedge (n_i.right = n_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$

This bit indicates whether, at node n_i, the consumer c would follow the root-to-leaf path. A value of 1 means the consumer continues along the path to n_{i+1}, while a value of 0 means it would branch off in a different direction.

In Fig. 2, the 'Consumer Pattern' and 'Baseline Pattern' columns illustrate how the decision patterns are computed.

Let C be the consumer data matrix with rows $c \in C$, and define $n.\text{split}(C) = (n.\text{split}(c))_{\forall c \in C}$.

Given a decision tree T with L leaves and consumer data C of size n, *CalcDecisionPatterns* (Alg. 1) traverses T using Breadth-First Search (BFS), applying Def. 7 at each node. Running in $O(nL)$ time, it returns a dictionary P mapping each leaf $l_j \in L_T$ to its consumer decision patterns, where $P[l_j][c_i]$ stores the pattern for consumer $c_i \in C$ at leaf l_j.

5 Constructing a WDNF Representation

Our baseline SHAP algorithm constructs a WDNF formula F representing the model's characteristic function. An illustration of this construction is provided in Fig. 2. The variables in F correspond to model features, and each cube captures the contribution of a single leaf. A variable f_i set to

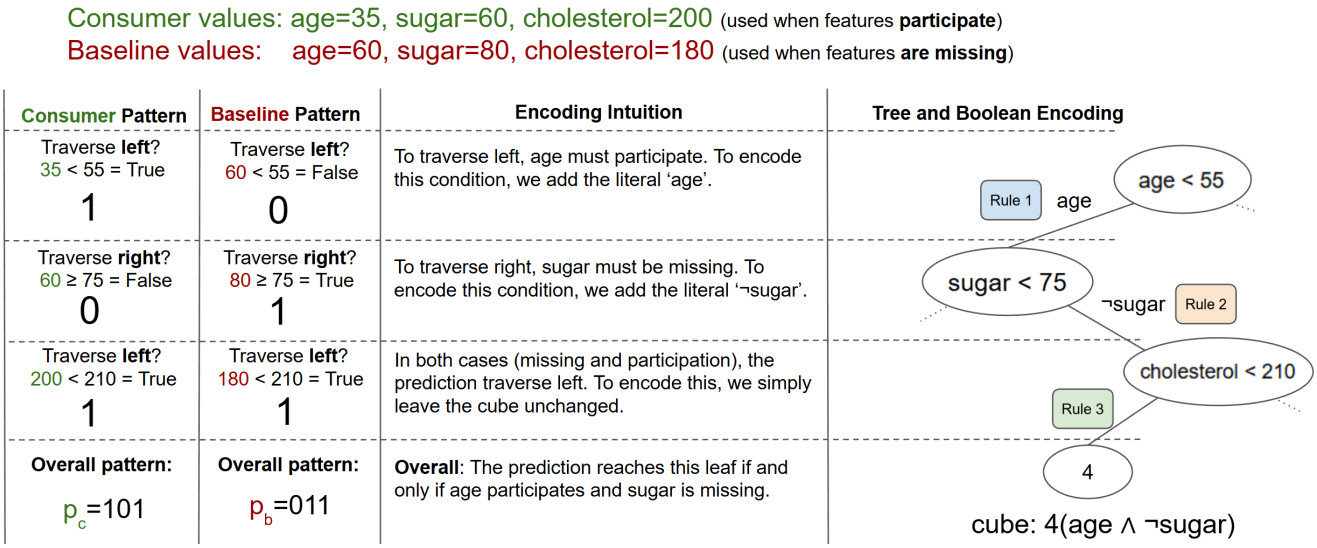

Figure 2: An illustration of the WDNF construction process on a small example. The consumer and baseline values are shown alongside a root-to-leaf path. The table explains how a weighted cube is iteratively constructed from these inputs. To compute the Shapley value contribution of the shown leaf, apply Formula 2 to the constructed weighted cube: $4(age \wedge \neg sugar)$. For Banzhaf values, use Formula 4; for Banzhaf interaction values, use Formula 5; and for Shapley interaction values, use Table 1.

1 indicates the feature is present (i.e., set to the consumer's value $c[f_i]$), while 0 denotes a missing feature (i.e., set to the baseline value $b[f_i]$). A cube is satisfied if and only if the prediction reaches its corresponding leaf:

$$Model\left(\mathbf{f} = \begin{cases} c[f_i] & \text{if } f_i \in S \\ b[f_i] & \text{if } f_i \notin S \end{cases}\right) = F\left(\mathbf{x} = \begin{cases} 1 & \text{if } f_i \in S \\ 0 & \text{if } f_i \notin S \end{cases}\right) \tag{6}$$

We present four simple rules for constructing the WDNF formula. Our goal is to construct a cube that represents a leaf l, a consumer c, and a baseline b. We first run *CalcDecisionPatterns* to obtain the consumer decision pattern p_c and baseline decision pattern p_b. Then, using the path, p_c, p_b, and the four rules below, we construct the cube. The rules are applied from the tree root ($i = 1$) down to the leaf's parent node ($i = D - 1$):

1. If $p_c[i] = 1$ and $p_b[i] = 0$: The prediction reaches n_{i+1} only when the consumer value is used—i.e., when f_i (that is, $n_i.feature$) participates. Add f_i to the cube.
2. If $p_c[i] = 0$ and $p_b[i] = 1$: The prediction reaches n_{i+1} only when the baseline value is used—i.e., when f_i is missing. Add the literal $\neg f_i$ to the cube.
3. If $p_c[i] = 1$ and $p_b[i] = 1$: The prediction always reaches n_{i+1}. Leave the cube unchanged.
4. If $p_c[i] = 0$ and $p_b[i] = 0$: The prediction never reaches n_{i+1}. Set the cube to $\perp$ (unsatisfiable).

Since the number of decision patterns is limited, we can precompute the cube for every possible input. The *MapPatternsToCube* function (Alg. 2) takes the list of features along a root-to-leaf path and applies the four rules above to map each pair of consumer and baseline patterns (p_c and p_b) to the corresponding cube.

Algorithm 2: Decision Patterns to Cube Mapping

function MAPPATTERNSTOCUBE(*features*)
  $d \leftarrow \{0 \mapsto \{0 \mapsto (\emptyset, \emptyset)\}\}$
  for $f \in$ *features* **do**
    $d_{old} \leftarrow d$
    $d \leftarrow \{\}$
    for p_c in d_{old} **do**
      for p_b in $d_{old}[p_c]$ **do**
        $(S^+, S^-) \leftarrow d_{old}[p_c][p_b]$
        $d[2p_c + 1][2p_b + 0] \leftarrow (S^+ \cup \{f\}, S^-)$
        $d[2p_c + 0][2p_b + 1] \leftarrow (S^+, S^- \cup \{f\})$
        $d[2p_c + 1][2p_b + 1] \leftarrow (S^+, S^-)$
  return d

6 A Generic Baseline SHAP Implementation

Let $P[l][x]$ denote the decision pattern of consumer or baseline values x on leaf l, computed using *CalcDecisionPatterns*. Let d_l be the mapping for leaf l built using *MapPatternsToCube*. Using P and d_l, one can now compute the *Baseline SHAP*. The WDNF of a decision tree T, a consumer c, and baseline values b can be calculated using Formula 7, which constructs a WDNF by aggregating the cubes of all leaves, each weighted by its corresponding leaf weight.

$$\sum_{l \in L_T} w_l \cdot d_l[\, P[l][c]\,][\, P[l][b]\,] \tag{7}$$

We apply the linear-time Shapley values formula (Formula 2) to the resulting WDNF to compute the desired baseline SHAP. Similarly, this WDNF can be used to compute Shapley interaction values, Banzhaf values, or Banzhaf interaction values by leveraging Table 1, Formula 4, or Formula 5, respectively.

7 Efficient Background SHAP Equation

Background SHAP takes three inputs: a decision tree T with depth D and L leaves; consumer data C with n rows; and background data B, a matrix with m rows of baseline feature values. Unlike Baseline SHAP, which uses one fixed baseline for missing features, Background SHAP averages the Shapley values across all baselines in the background data, providing more accurate results.

Formula 8 computes Background SHAP for a single consumer $c \in C$ and a single feature i in $O(m)$ (assuming L is constant). It uses the baseline WDNF formula (Formula 7) and the linear-time Shapley values formula (Formula 2).

$$\phi_i(\frac{1}{|B|}\sum_{b_k \in B}\sum_{l \in L_T} w_l \cdot d_l[\,P[l][c]\,][\,P[l][b_k]\,]) \tag{8}$$

Using Formula 8, computing Background SHAP for all n consumers in C takes $O(nm)$ time. We derive a new $O(n+m)$ formula that leverages GPU-friendly matrix multiplication. The derivation is detailed below:

$$\begin{aligned}
&\phi_i(\frac{1}{|B|}\sum_{b_k \in B}\sum_{l \in L_T} w_l \cdot d_l[\,P[l][c]\,][\,P[l][b_k]\,]) \overset{(a)}{=} \\
&\frac{1}{|B|}\sum_{l \in L_T}\sum_{b_k \in B} w_l \cdot \phi_i(d_l[\,P[l][c]\,][\,P[l][b_k]\,]) \overset{(b)}{=} \\
&\frac{1}{|B|}\sum_{l \in L_T}\sum_{p_b \in \{F,T\}^{D-1}} vc_{l,p_b} \cdot w_l \cdot \phi_i(d_l[\,P[l][c]\,][p_b]) \overset{(c)}{=} \\
&\frac{1}{|B|}\sum_{l \in L_T}\sum_{p_b \in \{F,T\}^{D-1}} \mathit{VC}[l][p_b] \cdot w_l \cdot \phi_i(d_l[\,P[l][c]\,][p_b]) \overset{(d)}{=} \\
&\sum_{l \in L_T} w_l \cdot (\sum_{p_b \in \{F,T\}^{D-1}} \frac{\mathit{VC}[l][p_b]}{|B|} \cdot \phi_i(d_l[\,P[l][c]\,][p_b])) \overset{(e)}{=} \\
&\sum_{l \in L_T} (w_l \cdot \mathbf{M_{l,i}} \cdot \mathbf{f_l})[\,P[l][c]\,] \overset{(f)}{=} \\
&\sum_{l \in L_T} \mathbf{s_{l,i}}[\,P[l][c]\,] \overset{(g)}{=}
\end{aligned} \tag{9}$$

We explain each transformation step below:

a) This is Formula 8.

b) Follows from the linearity property of the Shapley value: Functions of the form $f_1 = f_2 + w \cdot f_3$ satisfy $\phi_i(f_1) = \phi_i(f_2) + w \cdot \phi_i(f_3)$. We also reorder the summations.

c) Mark $vc_{l,p_b} = |\{b_k \in B \mid P[l][b_k] = p_b\}|$. Since the summands depend only on the decision patterns, we can rewrite the summation by looping over all possible baseline decision patterns. For each pattern, we multiply by the number of background instances that match it.

d) At the start of the algorithm, we precompute the number of baselines matching each pattern in $O(mL)$ time:
$\mathit{VC} = \mathit{CalcDecisionPatterns}(T, B).\mathit{value\_counts}()$
where *value_counts* is a standard function from the `pandas` Python package. For each leaf l and pattern p_b, we have $vc_{l,p_b} = \mathit{VC}[l][p_b] = |\{b_k \in B \mid P[l][b_k] = p_b\}|$.
This precomputation reduces the summation complexity of the formula from $O(nm)$ to $O(n+m)$.

e) Simple arithmetic: We push $\frac{1}{|B|}$ into the inner summation and pull w_l out.

f) The inner summation becomes a matrix-vector product:
Let $\mathbf{f_l}$ be the frequency vector of the background patterns, where $\mathbf{f_l}[p_b] = \frac{\mathit{VC}[l][p_b]}{|B|}$, i.e., the relative frequency of baseline pattern p_b at leaf l.
Let $\mathbf{M_{l,i}}$ be the Shapley matrix, where $\mathbf{M_{l,i}}[p_c][p_b] = \phi_i(d_l[p_c][p_b])$, representing the contribution of leaf l to feature i's Shapley value for the pair (p_c, p_b), assuming the leaf weight is 1.
The multiplication $w_l \cdot \mathbf{M_{l,i}} \cdot \mathbf{f_l}$ sums all the effects that leaf l has on the Shapley values of player i across all baseline decision patterns, weighted by their frequency. The result is a vector representing the Shapley value effects for all consumer patterns.
Extracting $(w_l \cdot \mathbf{M_{l,i}} \cdot \mathbf{f_l})[P[l][c]]$ yields the Shapley value effect for consumer c at leaf l.

g) Since the vector $w_l \cdot \mathbf{M_{l,i}} \cdot \mathbf{f_l}$ is independent of the consumer c, we can precompute it for every leaf l and feature i. We denote this vector by $\mathbf{s_{l,i}}$. After these vectors are built, computing SHAP values per consumer only requires fetching, for each leaf l, the element in $\mathbf{s_{l,i}}$ indexed by the consumer's decision pattern at l. Computing SHAP values for all features takes $O(nLD)$ time, since each root-to-leaf path involves at most D features.

The derivation above uses only the linearity property of Shapley values (see step b). Therefore, it holds for any metric that satisfies linearity, including Shapley interaction values, Banzhaf values, and Banzhaf interaction values.

8 Efficient Path-Dependent SHAP Equation

Instead of computing the frequencies using a background dataset, Path-Dependent SHAP estimates them using the nodes cover property (i.e. the number of training samples that reached the node during training).

Path-Dependent SHAP can be computed by simply replacing the Background frequency vector f_l with the Path-Dependent frequency vector $f_{l_{pd}}$ in Formula 9(f). Given a decision tree T, a leaf l with its root-to-leaf path $(n_1 \equiv r_T, n_2, \ldots, n_{D-1}, n_D \equiv l)$, and a baseline decision pattern p_b, the vector $f_{l_{pd}}$ is computed as follows:

$$\mathbf{f_{l_{pd}}}[p_b] = \prod_{i=1}^{D-1} \begin{cases} \frac{n_{i+1}.\mathit{cover}}{n_i.\mathit{cover}} & \text{if } p_b[i] = 1 \\ 1 - \frac{n_{i+1}.\mathit{cover}}{n_i.\mathit{cover}} & \text{if } p_b[i] = 0 \end{cases} \tag{10}$$

For example, the frequency of the pattern 5 (binary 101) along the root-to-leaf path $(n_1 \equiv r_T, n_2, n_3, n_4 \equiv l)$ is:

$$\mathbf{f_{l_{pd}}}[5] = \frac{n_2.\mathit{cover}}{n_1.\mathit{cover}} \cdot (1 - \frac{n_3.\mathit{cover}}{n_2.\mathit{cover}}) \cdot \frac{n_4.\mathit{cover}}{n_3.\mathit{cover}}$$

9 WOODELF Algorithm

We are now ready to present our main algorithm, WOODELF, shown in Alg. 3. WOODELF takes as input a decision tree T, consumer data C, background data B (empty B means Path-Dependent SHAP), and a function v.

Algorithm 3: An Efficient SHAP and Banzhaf algorithm

1: **function** WOODELF(T, C, B, v)
▷ Step 1, compute f
2: **if** $|B| > 0$ **then** ▷ Background
3: $P_b = \textit{CalcDecisionPatterns}(T, B)$
4: $f = P_b.\textit{value\_counts}(\textit{normalize} = True)$
5: **else** ▷ Path Dependent
6: Compute f using T and Formula 10
▷ Step 2, compute M
7: $M = \{\}$
8: **for** $l \in L_T$ **do**
9: $path = root\_to\_leaf\_path(T, l)$
10: $path\_features = (n.\textit{feature})_{\forall n \in path}$
11: $d_l = \textit{MapPatternsToCube}(path\_features)$
12: **for** p_c in d_l **do**
13: **for** p_b in $d_l[p_c]$ **do**
14: $cube = d_l[p_c][p_b]$
15: **for** $feature, value$ in $v(cube)$ **do**
16: $M[l][feature][p_c][p_b] = value$
▷ Step 3, compute s
17: $s = \{\}$
18: **for** $l \in L_T$ **do**
19: **for** $feature$ in $M[l]$ **do**
20: $s[l][feature] = w_l \cdot M[l][feature] \cdot f[l]$
▷ Step 4, compute the actual values
21: $P_c = \textit{CalcDecisionPatterns}(T, C)$
22: $values = \{\}$
23: **for** $l \in L_T$ **do**
24: **for** $feature$ in $s[l]$ **do**
25: $values[feature]$ += $s[l][feature][\, P_c[l]\,]$
26: **return** $values$

The function v takes a cube where each variable represents a feature, e.g. (age $\wedge \neg$sugar), and returns a mapping from feature subsets (of size one or more) to real numbers. For instance, v can compute Shapley values for individual features, as well as interaction values for feature pairs.

WOODELF outputs Path-Dependent or Background (depending on whether B is empty) Shapley/Banzhaf values or interaction values (depending on v) on the decision tree T for the given consumers. To compute values for a decision tree ensemble, one simply runs WOODELF on each tree and sums the results. Correctness follows from the linearity property of both Shapley and Banzhaf values.

The algorithm uses the equations from Sect. 7 and 8. Lines 2–6 compute the frequency vector f for either Background or Path-Dependent SHAP. Lines 7–16 compute the contribution matrix M. Lines 17–20 use f and M to compute s, the vector mapping consumer patterns to contributions. Finally, lines 21–25 use s to compute the desired Shapley/Banzhaf values or interaction values.

9.1 Algorithmic Improvements and Complexity

Our actual implementation is more advanced than the version shown in Alg. 3. It incorporates several key optimizations that substantially reduce the algorithm's runtime — with the first even improving its theoretical complexity:

1. Each matrix $M[l]$[feature] has size 4^D (recall that D is the depth of the tree), since both p_c and p_b can take any value between 0 and 2^D. Furthermore, the dictionary returned by *MapPatternsToCube* has size 3^D, as the number of cubes triples at each step. This means the matrix $M[l]$[feature] is sparse, with at most 3^D non-zero entries. By using sparse matrix multiplications, we reduce the complexity of line 20 from $O(4^D)$ to $O(3^D)$, thereby improving WOODELF 's overall complexity (see Table 2).
2. The function *MapPatternsToCube* and the matrix $M[l]$[feature] depend solely on the features repeated along the root-to-leaf path and the path's length. For example, all leaves at depth 6 with unique features share the same matrices. We exploit this by using a caching mechanism, which significantly reduces the computations in lines 7–16.
3. For every consumer/baseline x, the decision pattern of neighboring leaves l_i and l_{i+1} ($\exists n$ s.t. $n.left = l_i$, $n.right = l_{i+1}$) differ only in the last bit (see Def. 7). We leverage this property to accelerate lines 4 and 25.
4. We only need to compute half of the Shapley/Banzhaf interaction values because, for all i, j, $\phi_{i,j} = \phi_{j,i}$.
5. The length of each decision pattern is limited by the tree's depth. In the *CalcDecisionPatterns* algorithm, we select the appropriate unsigned integer type (e.g., `uint8`, `uint16`, `uint32`) based on the tree's maximum depth. This reduces compute time by enabling more efficient use of SIMD.
6. Line 25 utilizes vectorized NumPy indexing. It treats $P_c[l]$ as a series of indices and returns a series of the corresponding elements from $s[l][feature]$.

Table 2 summarizes the complexity of WOODELF, with detailed analysis in the full version. The state-of-the-art Path-Dependent SHAP algorithm is FastTreeShap, while PLTreeShap is the state-of-the-art for Background SHAP; both outperform the SHAP Python package.

WOODELF improves on PLTreeShap's complexity when $L < n$ by leveraging a core step (lines 7–20) whose cost is independent of dataset size—a key factor behind the empirical gains shown in the next section. However, this step might become a bottleneck for very deep trees or small datasets, where PLTreeShap and the SHAP Python package may outperform WOODELF.

Task	WOODELF	State-of-the-art
PD	$O(nTLD+TL3^DD)$	$O(nTLD+TL2^DD)$
BG	$O(mTL+nTLD+TL3^DD)$	$O(mTL+nT3^DD)$
PDIV	$O(nTLD^2+TL3^DD^2)$	$O(nTLD^2)$
BGIV	$O(mTL+nTLD^2+TL3^DD^2)$	$O(mTL+nT3^DD^2)$

Table 2: Complexity results. Legend: PD = Path-Dependent SHAP, BG = Background SHAP, BGIV/PDIV = Calculation of all Shapley interaction values, $n = |C|$, $m = |B|$, T = number of trees, L = leaves per tree, and D = tree depth.

IEEE-CIS

Task	SOTA CPU Algorithm	SHAP package	SOTA		WOODELF	
			CPU	GPU	CPU	GPU
Path-Dependent SHAP	FastTreeShap v2	151 sec	16 sec	0.9 sec	6 sec	3.3 sec
Background SHAP	PLTreeShap	10 days*	245 sec	14 hours*	12 sec	10 sec
Path-Dependent SHAP IV	FastTreeShap v1	33 hours*	350 sec	105 sec*	11 sec	8 sec
Background SHAP IV	PLTreeShap	X	597 sec*	X	19 sec	12 sec

KDD Cup 1999

Task	SOTA CPU Algorithm	SHAP package	SOTA		WOODELF	
			CPU	GPU	CPU	GPU
Path-Dependent SHAP	FastTreeShap v2	51 min	373 sec	7.9 sec	96 sec	3.3 sec
Background SHAP	PLTreeShap	8 years*	44 min	3 months*	162 sec	16 sec
Path-Dependent SHAP IV	FastTreeShap v1	8 days*	221 min*	229 sec*	193 sec	6 sec
Background SHAP IV	PLTreeShap	X	105 min*	X	262 sec	19 sec

Table 3: Performance comparison between the SHAP Python package, the state-of-the-art (SOTA) methods and WOODELF. The 'SOTA CPU Algorithm' column lists the best known CPU algorithm for each task, and the 'SOTA, CPU' column shows its runtime. The SOTA GPU algorithm for all tasks is GPUTreeSHAP. "SHAP" refers to computing the Shapley values of all features, while "SHAP IV" refers to computing all Shapley interaction values. Values marked with * are estimates. Estimation was necessary due to RAM limitations, long runtimes, and implementation constraints. Notably, the SHAP Python package currently supports background datasets of up to 100 rows, implicitly using only the first 100 rows of larger datasets. See the full version for details on the estimation method. 'X' means there is no available implementation for this task.

10 Experimental Results

We implemented the WOODELF Python package, which includes our algorithm. The notebooks used in the experiments are provided in the WOODELF Experiments repository. Detailed setup and empirical validation of the algorithm's correctness appear in the full version.

We compared WOODELF performance with that of the SHAP package and the relevant SOTA algorithms. To evaluate WOODELF at scale, we selected two of the largest and well-known tabular datasets.

The IEEE-CIS fraud detection dataset from the Kaggle competition is widely recognized, with related studies including (Jiang et al. 2023; Xiao 2024; Chen et al. 2021; B et al. 2024; Deng et al. 2021). In IEEE-CIS, $|B| = 118\,108$, $|C| = 472\,432$, and $F = 397$ (after applying one-hot encoding to categorical features in both datasets, where F denotes the total number of features after preprocessing).

The KDD Cup 1999 dataset serves as a well-established benchmark for network intrusion detection research, with related studies including (Pfahringer 2000; Tavallaee et al. 2009; Agalit and Khamlichi 2024). In KDD Cup: $|B| = 4\,898\,431,\ |C| = 2\,984\,154,\ F = 127$.

In both cases, we trained an XGBoost regressor with 100 trees of depth 6 (XGBoost's default *max_depth*). All algorithms were run sequentially without parallelization.

Our experiments were conducted in Google Colab's CPU environment with the additional RAM option enabled, utilizing 50GB of RAM instead of the standard 12GB. The GPU execution used the A100 GPU Colab runtime type.

Table 3 shows that WOODELF outperforms the state-of-the-art in all tasks—except GPU Path-Dependent SHAP on IEEE, where runtimes are already short. For Background SHAP, WOODELF achieves speed-ups of 24×, 50×, 165×, and 333× on GPU, and 16×, 20×, 31×, and 24× on CPU, relative to the best method on any hardware platform.

A striking example of historical improvement is Background SHAP on the KDD dataset. In 2020, (Lundberg et al. 2020) introduced the first polynomial-time algorithm for this task, but its quadratic complexity would still require an estimated 8 years on this dataset. Two years later, (Mitchell, Frank, and Holmes 2022) proposed a GPU-based implementation, reducing runtime to 3 months. In 2023, (Zern, Broelemann, and Kasneci 2023) achieved a breakthrough with a linear-time method, cutting runtime to 44 minutes. Our WOODELF algorithm completes the task in just 162 seconds on CPU and 16 seconds on GPU. Over just five years, the runtime has been reduced from 8 years to mere seconds!

11 Conclusion

We introduced WOODELF, a fast, unified, and GPU-friendly SHAP algorithm leveraging a novel connection between decision trees and Boolean logic. On the evaluated datasets, it outperformed state-of-the-art Background SHAP methods by 16–31× on CPU and 24–333× on GPU.

WOODELF provides a unified framework for model interpretability, supporting a range of attribution metrics (e.g., Banzhaf values) across different characteristic function definitions (e.g., Path-Dependent). With its efficiency and flexibility, WOODELF lays a solid foundation for future research into more advanced and precise interpretability methods.

References

Abramovich, O.; Deutch, D.; Frost, N.; Kara, A.; and Olteanu, D. 2023. Banzhaf Values for Facts in Query Answering. arXiv:2308.05588.

Agalit, M.; and Khamlichi, Y. 2024. Optimization of Intrusion Detection with Deep Learning: A Study Based on the KDD Cup 99 Database. *International Journal of Safety and Security Engineering*, 14: 1029–1038.

Arenas, M.; Bertossi, P. B. L.; and Monet, M. 2021. The Tractability of SHAP-Score-Based Explanations over Deterministic and Decomposable Boolean Circuits. arXiv:2007.14045.

B, S. P.; N, A. B.; Reddy, H.; Singh, R. P.; and Kanchan, S. 2024. A Machine Learning Approach for Credit Card Fraud Detection in Massive Datasets Using SMOTE and Random Sampling. In *2024 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, 1–8.

Banzhaf, J. F. 1965. Weighted Voting Doesn't Work: A Mathematical Analysis. *Rutgers Law Review*, 19: 317–343.

Breiman, L. 2001. Random Forests. *Machine Learning*, 45(1): 5–32.

Chen, H.; Covert, I. C.; Lundberg, S. M.; and Lee, S.-I. 2023. Algorithms to estimate Shapley value feature attributions. *Nature Machine Intelligence*, 5(6): 590–601.

Chen, L.; Guan, Q.; Chen, N.; and YiHang, Z. 2021. A StackNet Based Model for Fraud Detection. In *2021 2nd International Conference on Education, Knowledge and Information Management (ICEKIM)*, 328–331.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 785–794. New York, NY, USA: Association for Computing Machinery.

Covert, I.; Lundberg, S.; and Lee, S.-I. 2020. Understanding Global Feature Contributions With Additive Importance Measures. arXiv:2004.00668.

da Silva, P. F. M. 2021. *Max-SAT Algorithms For Real World Instances*. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa.

den Broeck, G. V.; Lykov, A.; Schleich, M.; and Suciu, D. 2021. On the Tractability of SHAP Explanations. arXiv:2009.08634.

Deng, W.; Huang, Z.; Zhang, J.; and Xu, J. 2021. A Data Mining Based System For Transaction Fraud Detection. In *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, 542–545.

Dorogush, A. V.; Gulin, A.; Gusev, G.; Kazeev, N.; Prokhorenkova, L. O.; and Vorobev, A. 2017. Fighting biases with dynamic boosting. *CoRR*, abs/1706.09516.

Fujimoto, K.; Kojadinovic, I.; and Marichal, J.-L. 2006. Axiomatic characterizations of probabilistic and cardinal-probabilistic interaction indices. *Games and Economic Behavior*, 55(1): 72–99.

Gill, N.; Hall, P.; Montgomery, K.; and Schmidt, N. 2020. A Responsible Machine Learning Workflow with Focus on Interpretable Models, Post-hoc Explanation, and Discrimination Testing. *Information*, 11(3).

Gorji, A.; Amrollahi, A.; and Krause, A. 2025. SHAP values via sparse Fourier representation. arXiv:2410.06300.

Grabisch, M.; and Roubens, M. 1999. An axiomatic approach to the concept of interaction among players in cooperative games. *International Journal of Game Theory*, 28(4): 547–565.

Hall, P.; and Gill, N. 2019. *An introduction to machine learning interpretability*. O'Reilly Media, Incorporated.

Huang, X.; and Marques-Silva, J. 2024. Updates on the complexity of SHAP scores. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24.

Jiang, S.; Dong, R.; Wang, J.; and Xia, M. 2023. Credit Card Fraud Detection Based on Unsupervised Attentional Anomaly Detection Network. *Systems*, 11: 305.

Karczmarz, A.; Michalak, T.; Mukherjee, A.; Sankowski, P.; and Wygocki, P. 2022. Improved feature importance computation for tree models based on the Banzhaf value. In Cussens, J.; and Zhang, K., eds., *Proceedings of UAI 2022*, volume 180 of *Proceedings of Machine Learning Research*, 969–979. PMLR.

Knight, E. 2019. AI and machine learning-based credit underwriting and adverse action under the ECOA. *Bus. & Fin. L. Rev.*, 3: 236.

Livshits, E.; Bertossi, L.; Kimelfeld, B.; and Sebag, M. 2021. The Shapley Value of Tuples in Query Answering. *Logical Methods in Computer Science*, Volume 17, Issue 3.

Lundberg, S. M.; Erion, G.; Chen, H.; DeGrave, A.; Prutkin, J. M.; Nair, B.; Katz, R.; Himmelfarb, J.; Bansal, N.; and Lee, S.-I. 2020. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2: 56–67.

Lundberg, S. M.; Erion, G. G.; and Lee, S.-I. 2019. Consistent Individualized Feature Attribution for Tree Ensembles. arXiv:1802.03888.

Lundberg, S. M.; and Lee, S.-I. 2017. A Unified Approach to Interpreting Model Predictions. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Mitchell, R.; Frank, E.; and Holmes, G. 2022. GPUTreeShap: Massively Parallel Exact Calculation of SHAP Scores for Tree Ensembles. arXiv:2010.13972.

Muschalik, M.; Fumagalli, F.; Hammer, B.; and Hüllermeier, E. 2024. Beyond TreeSHAP: Efficient Computation of Any-Order Shapley Interactions for Tree Ensembles. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(13): 14388–14396.

Pfahringer, B. 2000. Winning the KDD99 classification cup: bagged boosting. *SIGKDD Explor. Newsl.*, 1(2): 65–66.

Selbst, A. D.; and Powles, J. 2017. Meaningful information and the right to explanation. *International Data Privacy Law*, 7(4): 233–242.

Shapley, L. S. 1953. A value of n-person games. *Contributions to the Theory of Games*, 307–317.

Sundararajan, M.; and Najmi, A. 2020. The many Shapley values for model explanation. arXiv:1908.08474.

Tavallaee, M.; Bagheri, E.; Lu, W.; and Ghorbani, A. A. 2009. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 1–6.

Xiao, Z. 2024. IEEE-CIS Fraud Detection Based on XGB. In Li, X.; Yuan, C.; and Kent, J., eds., *Proceedings of the 7th International Conference on Economic Management and Green Development*, 1785–1796. Singapore: Springer Nature Singapore.

Yang, J. 2022. Fast TreeSHAP: Accelerating SHAP Value Computation for Trees. arXiv:2109.09847.

Zern, A.; Broelemann, K.; and Kasneci, G. 2023. Interventional SHAP Values and Interaction Values for Piecewise Linear Regression Trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9): 11164–11173.

Zhang, B.; and Jang, H. 2005. Molecular Learning of wDNF Formulae. In Carbone, A.; and Pierce, N. A., eds., *DNA Computing, 11th International Workshop on DNA Computing, DNA11, Revised Selected Papers*, volume 3892 of *Lecture Notes in Computer Science*, 427–437. Springer.