

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

Alexander Nadel¹, Ron Wettenstein²

¹Faculty of Data and Decision Sciences, Technion, Haifa, Israel, alexandernad@technion.ac.il

²Reichman University, Herzliya, Israel, 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.

This is the full version of our paper to appear in AAAI-26.

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:](https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html)

[//kdd.ics.uci.edu/databases/kddcup99/kddcup99.html](https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html)

1 Introduction

Decision trees are widely used predictive models for classification and regression. To improve predictive ac-

curacy, 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 $M : 2^{[N]} \rightarrow \mathbb{R}$ 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.

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

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 Appendix A 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 GPUShap (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 $24\times$ to $333\times$ speed-ups on GPU and $16\times$ to $31\times$ 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 $F(x_1, \dots, x_h) : \{0, 1\}^h \rightarrow \mathbb{R}$. 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.

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

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 $F(V) : \{0, 1\}^h \rightarrow \mathbb{R}$ over $V = \{x_1, x_2, \dots, x_h\}$, 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 Appendix B.5). A key

insight of this paper is that, for WDNF, Shapley values can be computed in linear time using the following formula:

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

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 Appendix B, 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 Appendix C.

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: $\phi_{i,j} \text{ } i \neq j = \phi_{j|i=1} - \phi_{j|i=0}$*

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

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

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

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:

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

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:

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

4 Decision Pattern

This section introduces the concept of a *decision pattern*, central to WOODSELF. 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: $n.\text{left}$ and $n.\text{right}$.

Algorithm 1: Mapping Each Leaf to Its Decision Patterns

```

1: function CALCDECISIONPATTERNS( $T, C$ )
2:    $P_{\text{leaves}} \leftarrow \{\}$ 
3:    $P_{\text{all}} \leftarrow \{r_T : (0)_{\forall c \in C}\}$ 
4:   for  $n$  in BFS( $T$ ) do:
5:     if  $n$  is a leaf then
6:        $P_{\text{leaves}}[n] = P_{\text{all}}[n]$ 
7:     else
8:        $P_{\text{all}}[n.\text{left}] = (P_{\text{all}}[n] << 1) + n.\text{split}(C)$ 
9:        $P_{\text{all}}[n.\text{right}] = (P_{\text{all}}[n] << 1) + \neg n.\text{split}(C)$ 
10:  return  $P_{\text{leaves}}$ 

```

- 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 $n.\text{feature} \in \{1, \dots, h\}$ and a threshold value $\theta_n \in \mathbb{R}$.
For a node n and consumer feature values $c = (c_1, c_2, \dots, c_h) \in \mathbb{R}^h$, we define the function $n.\text{split}(c)$:

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

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 : $(n_1 \equiv r_T, n_2, \dots, n_{D-1}, n_D \equiv l)$.

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

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

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

Consumer values: age=35, sugar=60, cholesterol=200 (used when features **participate** in the ‘game’)
 Baseline values: age=60, sugar=80, cholesterol=180 (used when features **are missing** from the ‘game’)

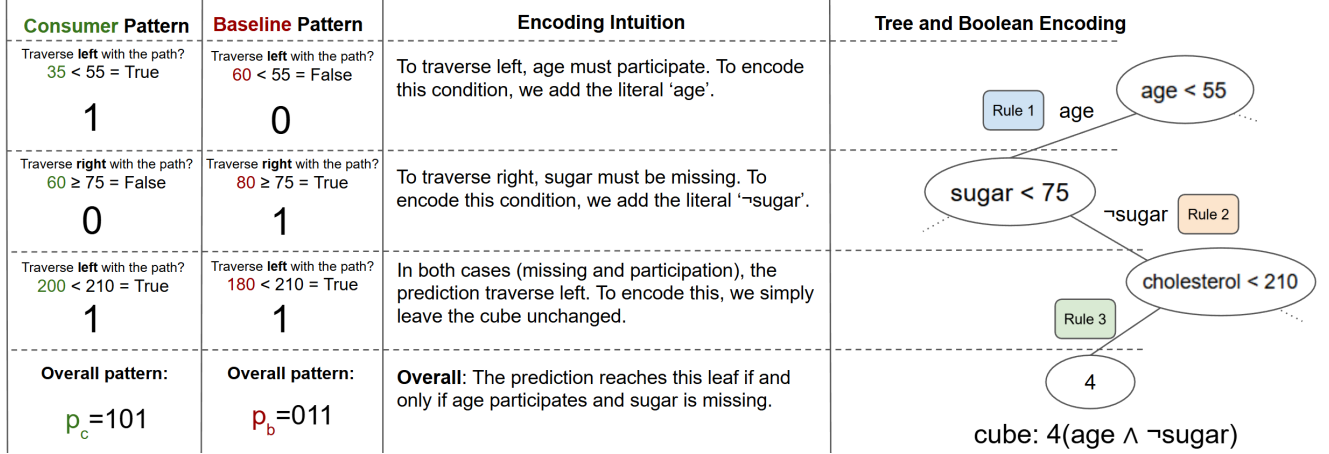


Figure 2: An illustration of the WDNF construction process on a small example. The consumer and baseline values are shown alongside a root-to-leaf path. The table explains how a weighted cube is iteratively constructed from these inputs. To compute the Shapley value contribution of the shown leaf, apply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

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

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, $n_i.\text{feature}$) 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

```

1: function MAPPATTERNSTOCUBE(features)
2:    $d \leftarrow \{0 \mapsto \{0 \mapsto (\emptyset, \emptyset)\}\}$ 
3:   for  $f \in \text{features}$  do
4:      $d_{old} \leftarrow d$ 
5:      $d \leftarrow \{\}$ 
6:     for  $p_c$  in  $d_{old}$  do
7:       for  $p_b$  in  $d_{old}[p_c]$  do
8:          $(S^+, S^-) \leftarrow d_{old}[p_c][p_b]$ 
9:          $d[2p_c + 1][2p_b + 0] \leftarrow (S^+ \cup \{f\}, S^-)$ 
10:         $d[2p_c + 0][2p_b + 1] \leftarrow (S^+, S^- \cup \{f\})$ 
11:         $d[2p_c + 1][2p_b + 1] \leftarrow (S^+, S^-)$ 
12:   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.

$$\sum_{l \in L_T} w_l \cdot d_l[P[l][c]][P[l][b]] \quad (7)$$

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).

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

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:

$$\begin{aligned} & \phi_i\left(\frac{1}{|B|} \sum_{b_k \in B} \sum_{l \in L_T} w_l \cdot d_l[P[l][c][P[l][b_k]]]\right) \stackrel{(a)}{=} \\ & \frac{1}{|B|} \sum_{l \in L_T} \sum_{b_k \in B} w_l \cdot \phi_i(d_l[P[l][c][P[l][b_k]]]) \stackrel{(b)}{=} \\ & \frac{1}{|B|} \sum_{l \in L_T} \sum_{p_b \in \{F, T\}^{D-1}} v_{c_l, p_b} \cdot w_l \cdot \phi_i(d_l[P[l][c][p_b]]) \stackrel{(c)}{=} \\ & \frac{1}{|B|} \sum_{l \in L_T} \sum_{p_b \in \{F, T\}^{D-1}} VC[l][p_b] \cdot w_l \cdot \phi_i(d_l[P[l][c][p_b]]) \stackrel{(d)}{=} \\ & \sum_{l \in L_T} w_l \cdot \left(\sum_{p_b \in \{F, T\}^{D-1}} \frac{VC[l][p_b]}{|B|} \cdot \phi_i(d_l[P[l][c][p_b]]) \right) \stackrel{(e)}{=} \\ & \sum_{l \in L_T} (w_l \cdot \mathbf{M}_{l,i} \cdot \mathbf{f}_l)[P[l][c]] \stackrel{(f)}{=} \\ & \sum_{l \in L_T} \mathbf{s}_{l,i}[P[l][c]] \stackrel{(g)}{=} \end{aligned} \quad (9)$$

We explain each transformation step below:

- This is Formula 8.
- 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.
- Mark $v_{c_l, p_b} = |\{b_k \in B \mid P[l][b_k] = p_b\}|$. 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.
- At the start of the algorithm, we precompute the number of baselines matching each pattern in $O(mL)$ time: $VC = \text{CalcDecisionPatterns}(T, B).value_counts()$ where *value_counts* is a standard function from the pandas Python package. For each leaf l and pattern p_b , we have $v_{c_l, p_b} = VC[l][p_b] = |\{b_k \in B \mid P[l][b_k] = p_b\}|$. This precomputation reduces the summation complexity of the formula from $O(nm)$ to $O(n+m)$.

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

- The inner summation becomes a matrix-vector product: Let $\mathbf{f}_l$ be the frequency vector of the background patterns, where $\mathbf{f}_l[p_b] = \frac{VC[l][p_b]}{|B|}$, 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 .

- 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 $\mathbf{f}_l$ with the Path-Dependent frequency vector $\mathbf{f}_{l_{pd}}$ in Formula 9(f). Given a decision tree T , a leaf l with its root-to-leaf path $(n_1 \equiv r_T, n_2, \dots, n_{D-1}, n_D \equiv l)$, and a baseline decision pattern p_b , the vector $\mathbf{f}_{l_{pd}}$ is computed as follows:

$$\mathbf{f}_{l_{pd}}[p_b] = \prod_{i=1}^{D-1} \begin{cases} \frac{n_{i+1}.cover}{n_i.cover} & \text{if } p_b[i] = 1 \\ 1 - \frac{n_{i+1}.cover}{n_i.cover} & \text{if } p_b[i] = 0 \end{cases} \quad (10)$$

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:

$$\mathbf{f}_{l_{pd}}[5] = \frac{n_2.cover}{n_1.cover} \cdot \left(1 - \frac{n_3.cover}{n_2.cover}\right) \cdot \frac{n_4.cover}{n_3.cover}$$

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:      $P_b = \text{CalcDecisionPatterns}(T, B)$ 
4:      $f = P_b.\text{value\_counts}(\text{normalize} = \text{True})$ 
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:      $\text{path} = \text{root\_to\_leaf\_path}(T, l)$ 
10:     $\text{path\_features} = (n.\text{feature})_{n \in \text{path}}$ 
11:     $d_l = \text{MapPatternsToCube}(\text{path\_features})$ 
12:    for  $p_c$  in  $d_l$  do
13:      for  $p_b$  in  $d_l[p_c]$  do
14:         $\text{cube} = d_l[p_c][p_b]$ 
15:        for  $\text{feature}, \text{value}$  in  $v(\text{cube})$  do
16:           $M[l][\text{feature}][p_c][p_b] = \text{value}$ 
    ▷ Step 3, compute  $s$ 
17:    $s = \{\}$ 
18:   for  $l \in L_T$  do
19:     for  $\text{feature}$  in  $M[l]$  do
20:        $s[l][\text{feature}] = w_l \cdot M[l][\text{feature}] \cdot f[l]$ 
    ▷ Step 4, compute the actual values
21:    $P_c = \text{CalcDecisionPatterns}(T, C)$ 
22:    $\text{values} = \{\}$ 
23:   for  $l \in L_T$  do
24:     for  $\text{feature}$  in  $s[l]$  do
25:        $\text{values}[\text{feature}] += s[l][\text{feature}][P_c[l]]$ 
26:   return  $\text{values}$ 

```

The function v takes a cube where each variable represents a feature, e.g. ($\text{age} \wedge \neg \text{sugar}$), 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 $M[l][\text{feature}]$ 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 $M[l][\text{feature}]$ 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 $M[l][\text{feature}]$ 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. $n.\text{left} = l_i$, $n.\text{right} = l_{i+1}$) 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 $\text{CalcDecisionPatterns}$ 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 $s[l][\text{feature}]$.

Table 2 summarizes the complexity of WOODELF, with detailed analysis in Appendix D. 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	$O(nTLD + TL3^D D)$	$O(nTLD + TL2^D D)$
BG	$O(mTL + nTLD + TL3^D D)$	$O(mTL + nT3^D D)$
PDIV	$O(nTLD^2 + TL3^D D^2)$	$O(nTLD^2)$
BGIV	$O(mTL + nTLD^2 + TL3^D D^2)$	$O(mTL + nT3^D D^2)$

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 Appendix E.2 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 Appendix E.

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 (Pfahring 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 $24\times$, $50\times$, $165\times$, and $333\times$ on GPU, and $16\times$, $20\times$, $31\times$, and $24\times$ 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 $16\text{--}31\times$ on CPU and $24\text{--}333\times$ 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.
- Bracci, A. 2021. background data size limited to 100, xgboost. <https://github.com/shap/shap/issues/1991>. GitHub Issue #1991.
- 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, H.; Janizek, J. D.; Lundberg, S.; and Lee, S.-I. 2020. True to the Model or True to the Data? *arXiv:2006.16234*.
- 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.
- Derks, J.; and Haller, H. 1999. Null players out? Linear values for games with variable supports. *International Game Theory Review*, 1: 301–314.
- 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.
- Laberge, G.; and Pequignot, Y. 2022. Understanding Interventional TreeSHAP : How and Why it Works. *arXiv:2209.15123*.
- 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.; and contributors, S. 2024. SHAP GPU Tree Explainer Example. https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/explainers/GPUTree.html.
- 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. GPUtree-Shap: Massively Parallel Exact Calculation of SHAP Scores for Tree Ensembles. arXiv:2010.13972.

Muschalik, M.; Baniecki, H.; Fumagalli, F.; Kolpaczki, P.; Hammer, B.; and Hüllermeier, E. 2024. shapiq: Shapley Interactions for Machine Learning. arXiv:2410.01649.

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

Porwiz, M. 2019. What would be needed to implement interaction values for other Explainers? For which one would it be most straightforward? <https://github.com/shap/shap/issues/777>. GitHub Issue #777.

Provan, J. S.; and Ball, M. O. 1983. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM Journal on Computing*, 12(4): 777–788.

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.

Tavallae, 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.

A Background SHAP is the Most Accurate

We now examine the example in Fig. 3 to illustrate why Background SHAP is the most accurate approach. Details on the three common approaches analyzed in this appendix are provided in Sect. 1.2. Suppose the decision tree M in Fig. 3 was trained on the dataset X .

As part of a characteristic function definition, we seek to determine the model’s prediction when $f_1 = 2$, $f_3 = 5$ and f_2 is missing. Ideally, this prediction should be the marginal

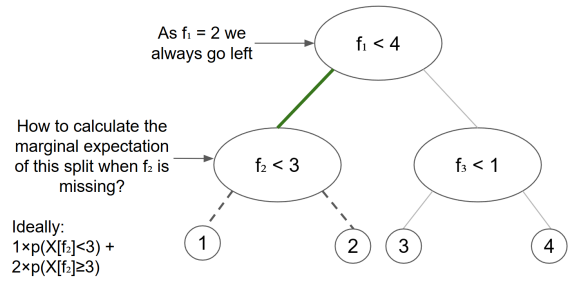


Figure 3: A simple decision tree illustrating the expected prediction computation, given $f_1 = 2$ and f_2 is missing.

expectation $\mathbb{E}[M(f_1, f_2, f_3) \mid do(f_1 = 2, f_3 = 5)]$ (For more on marginal vs. conditional SHAP, see (Chen et al. 2020)).

At the split $f_1 < 4$, we go left since $f_1 = 2$. The question is how to handle the subsequent split $f_2 < 3$ when f_2 is missing.

To proceed, we must estimate the probability that the condition holds, denoted $p(f_2 < 3)$. We then use this probability to compute the expected prediction at this node: $\mathbb{E}[M(f_1, f_2, f_3) \mid do(f_1 = 2, f_3 = 5)] = 1 \cdot p(f_2 < 3) + 2 \cdot (1 - p(f_2 < 3))$.

Baseline SHAP can not estimate the full probability as it only has a single baseline value for f_2 . *Path-Dependent SHAP* estimates this probability using the cover property, which is based on the training data that reached the split node in the training process. Since only rows with $X[f_1] < 4$ reached this node, the estimate is conditioned on $X[f_1] < 4$, leading to potential bias when f_1 and f_2 are correlated. *Background SHAP* can use the full training set to compute this probability accurately.

B Shapley and Banzhaf values on WDNF and WCNF - With Correctness Proofs

This appendix provides a deeper analysis of the Shapley and Banzhaf definitions for pseudo-Boolean functions and their computation over WDNF and WCNF formulas. We formally prove the correctness of Formulas 2 and 4, and derive their asymptotic computational complexity.

B.1 Shapley and Banzhaf values

Shapley values are defined by Formula 1. They are known to be the unique method that satisfies four desirable properties (Shapley 1953):

1. **Efficiency:** Summing the Shapley values for all players equals the characteristic function M output when all the features exist ($M(N)$) minus the output when all are missing ($M(\{\})$). Given a group of N features: $\sum_{i=1}^N \phi_i(M) = M(N) - M(\{\})$
2. **Null Player:** A player who does not influence any coalition gets a Shapley value of zero.
3. **Symmetry** (or equal treatment of equals): If players i and j contribute equally to any subset, their Shapley values are equal.

4. **Linearity:** The Shapley value is linear, meaning that for any two characteristic functions M_1 and M_2 , the Shapley value of their sum equals the sum of their Shapley values. That is, for a player i and $M = M_1 + M_2$, we have: $\phi_i(M) = \phi_i(M_1) + \phi_i(M_2)$

Banzhaf values (Banzhaf 1965) originate from voting theory. Their formula is similar to the Shapley values formula but gives the same weight to all subsets:

$$\beta_i(M) = \sum_{S \subseteq N \setminus \{i\}} \frac{1}{2^{(|N|-1)}} (M(S \cup \{i\}) - M(S)) \quad (11)$$

See Sects. 1.1 and 3 for additional background on the Shapley and Banzhaf values. Both Banzhaf and Shapley are computed over the game's characteristic function M . We have demonstrated how a pseudo-Boolean (PB) function F can be treated as a game's characteristic function (see Def. 3). Below are the Shapley and Banzhaf values formulas for a variable (player) in a PB function, derived by applying our characteristic function definition to the standard Shapley and Banzhaf value formulas. Here, $x_{-i} = \{x_1, \dots, x_h\} \setminus \{x_i\}$ (the set of all variables excluding x_i), h is the total number of variables, and $\sum_{x_j \in x_{-i}} x_j$ is the number of participating players in x_{-i} .

The Shapley values formula is:

$$\phi_i(F) = \sum_{x_{-i} \in \{0,1\}^{h-1}} \frac{(\sum_{x_j \in x_{-i}} x_j)!(h - (\sum_{x_j \in x_{-i}} x_j) - 1)!}{h!} \times \Delta F, \quad (12)$$

$$\Delta F = F(x_{-i}, x_i = 1) - F(x_{-i}, x_i = 0).$$

The Banzhaf values formula is:

$$\beta_i(F) = \sum_{x_{-i} \in \{0,1\}^{h-1}} \frac{1}{2^{h-1}} \times \Delta F, \quad (13)$$

$$\Delta F = F(x_{-i}, x_i = 1) - F(x_{-i}, x_i = 0).$$

Direct calculation of these formulas is computationally expensive due to the exponential size of $\{0,1\}^{h-1}$. In this section, we show that this problem is #P-hard in general. A key insight of our research is that Shapley and Banzhaf values for PB functions, represented in the well-known normal forms WDNF and WCNF, can be computed in linear time.

B.2 WCNF

The definition of Weighted Conjunctive Normal (WCNF) Form is very similar to the WDNF definition (see Def 2). A *clause* is a disjunction of literals. For a clause 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 clause $c_k \equiv x_1 \vee \neg x_2 \vee x_3$, we have $S_k^+ = \{x_1, x_3\}$, $S_k^- = \{x_2\}$ and $S_k = S_k^+ \cup S_k^-$.

Definition 8 (Weighted Conjunctive Normal Form (WCNF) (da Silva 2021)). WCNF formulas are pseudo-Boolean functions of the form:

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

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

For example, the PB function $F(x_1, x_2, x_3) = 3(\neg x_1) + 1(\neg x_1 \vee x_2) + 5(x_1 \vee \neg x_2 \vee x_3)$ is in WCNF. Assigning $x_1 = 0$, $x_2 = 1$, and $x_3 = 1$ results in $F(0, 1, 1) = 3 + 1 + 5 = 9$.

B.3 Linear Time Shapley and Banzhaf Formulas

Shapley and Banzhaf values can be computed in linear time on WDNF and WCNF representations using Formulas 2 and 4, respectively. Before applying these formulas, remove any cubes or clauses that contain both a variable and its negation — that is, remove any cube or clause where $|S_k^+ \cap S_k^-| \geq 1$. This trivial preprocessing step is correct because such cubes are always false, and such clauses are always true. These cubes/clauses are also removed before the interaction values computation.

See tables 4 and 5 for Banzhaf and Shapley calculation examples:

	$3(\neg x_1)$	$5(x_1 \wedge \neg x_3)$	$2(x_2 \wedge x_3 \wedge \neg x_1)$	total value
β_1 (Banzhaf for x_1)	$\frac{-3}{2^0} = -3$	$\frac{5}{2^1} = 2.5$	$\frac{-2}{2^2} = -0.5$	-1
β_2 (Banzhaf for x_2)	0	0	$\frac{2}{2^2} = 0.5$	0.5
β_3 (Banzhaf for x_3)	0	$\frac{-5}{2^2} = -2.5$	$\frac{2}{2^2} = 0.5$	-2

Table 4: Detailed Banzhaf values calculation on $\psi = 3(\neg x_1) + 5(x_1 \wedge \neg x_3) + 2(x_2 \wedge x_3 \wedge \neg x_1)$.

	$3(\neg x_1)$	$5(x_1 \wedge \neg x_3)$	$2(x_2 \wedge x_3 \wedge \neg x_1)$	total value
ϕ_1 (Shapley for x_1)	$\frac{-3}{1 \binom{1}{1}} = -3$	$\frac{5}{1 \binom{1}{1}} = 2.5$	$\frac{-2}{1 \binom{2}{2}} = -0.667$	-1.167
ϕ_2 (Shapley for x_2)	0	0	$\frac{2}{2 \binom{2}{2}} = 0.333$	0.333
ϕ_3 (Shapley for x_3)	0	$\frac{-5}{1 \binom{2}{2}} = -2.5$	$\frac{2}{2 \binom{2}{2}} = 0.333$	-2.167

Table 5: Detailed Shapley values calculation on ψ (as defined in Table 4). Observe that the efficiency property holds as the sum of the Shapley values is -3 and: $\psi(x_1 = x_2 = x_3 = 1) - \psi(x_1 = x_2 = x_3 = 0) = -3$

In the following subsections, we will prove that these formulas correctly compute the Shapley and Banzhaf values, and demonstrate how to use them to calculate the Shapley/Banzhaf value for all variables in linear time.

B.4 Correctness Proofs

We will mark the linear-time formulas as “simplified” and the exponential time formulas as “original”. Mark Formula 12 as ϕ_i^{original} , Formula 13 as $\beta_i^{\text{original}}$, Formula 2 as $\phi_i^{\text{simplified}}$, and Formula 4 as $\beta_i^{\text{simplified}}$.

Theorem 1. For any player i , the simplified Banzhaf formula $\beta_i^{\text{simplified}}$ is equal to the original formula $\beta_i^{\text{original}}$ when the input is in WDNF.

Proof. Due to the linearity property, Banzhaf values can be calculated by computing them for each cube independently and summing the results. Given a weighted cube c_k and a player i we will show $\beta_i^{\text{simplified}}(c_k) = \beta_i^{\text{original}}(c_k)$.

We will consider three scenarios:

1. $i \notin S_k$: From null player property: $\beta_i^{original}(c_k) = 0$, and due to the third case in Formula 4: $\beta_i^{simplified}(c_k) = 0$.
2. $i \in S_k^+$: As the literal x_i is included in the cube, any assignment with $x_i = 0$ will leave the cube unsatisfied: $\forall x_{-i} : c_k(x_{-i}, x_i = 0) = 0$.
This property allows us to simplify $\beta_i^{original}$ (Formula 13) by directly substituting $c_k(x_{-i}, x_i = 0) = 0$:

$$\beta_i^{original}(c_k) = \frac{1}{2^{h-1}} \sum_{x_{-i} \in \{0,1\}^{h-1}} c_k(x_{-i}, x_i = 1) - 0$$

Next, we will compute the summation by counting the number of satisfying assignments. The cube is satisfiable because we have already removed all cubes c_j s.t. $|S^+(c_j) \cap S^-(c_j)| \geq 1$. There are $h-1$ variables (ignoring x_i as we already assigned it to 1, see Defs. 2 and 8), $|S_k| - 1$ of them appear in c_k . To satisfy c_k , we must correctly assign all its variables. The remaining variables can have any assignment. Therefore, $c_k(x_{-i}, x_i = 1)$ has $2^{h-1-(|S_k|-1)}$ satisfying assignments, each contributes a weight of w_k .

$$\beta_i^{original}(c_k) = \frac{1}{2^{h-1}} \cdot w_k \cdot 2^{h-1-(|S_k|-1)} = \frac{w_k}{2^{|S_k|-1}}$$

3. $i \in S_k^-$: Now any assignment with $x_i = 1$ will leave the cube unsatisfied. We will use the same technique as above, simplifying the original formula and counting the number of satisfying assignments.

$$\beta_i^{original}(c_k) = \frac{1}{2^{h-1}} \sum_{x_{-i} \in \{0,1\}^{h-1}} 0 - c_k(x_{-i}, x_i = 0) = \frac{1}{2^{h-1}} \cdot (-w_k) \cdot 2^{h-1-(|S_k|-1)} = \frac{-w_k}{2^{|S_k|-1}}$$

□

Theorem 2. For any player i , the simplified Shapley formula $\phi_i^{simplified}$ is equal to the original formula $\phi_i^{original}$ when the input is in WDNF.

Proof. Using the linearity property, it is sufficient to show that for a weighted cube c_k and a player i , we have $\phi_i^{simplified}(c_k) = \phi_i^{original}(c_k)$.

This proof utilizes the Shapley value's *null player out* (NPO) property. The *null player* property guarantees that players that have zero contribution will get Shapley value of zero. *Null player out* considers the case when null players are removed completely from the game.

By combining the *null player* property with the *efficiency* property, we can deduce that removing null players from the game does not affect the sum of the Shapley values of the remaining players. The question then becomes whether the removal of a null player can redistribute the Shapley values, maintaining their sum but altering how they are distributed among the remaining players. The *null player out* (NPO) property assures that this will not occur. The NPO property guarantees that the Shapley values of the other players remain unchanged when null players are removed or added.

For a simple proof of this property, see Lemma 2 in (Laberge and Pequignot 2022), and for an in-depth review, refer to (Derks and Haller 1999).

In our proof, we will remove all null players, observe that after removal, only one assignment satisfies c_k , and use this observation to derive $\phi_i^{simplified}$ from $\phi_i^{original}$. We consider three separate cases:

1. $i \notin S_k$: From null player property and applying the $\phi_i^{simplified}$ formula: $\phi_i^{original}(c_k) = 0 = \phi_i^{simplified}(c_k)$.
2. $i \in S_k^+$: As the literal x_i is included in the cube, any assignment with $x_i = 0$ will leave the cube unsatisfied. We can use this to simplify the formula $\phi_i^{original}$ by directly substituting $c_k(x_{-i}, x_i = 0) = 0$:

$$\begin{aligned} \phi_i^{original}(w_k \cdot c_k) &= \sum_{x_{-i} \in \{0,1\}^{h-1}} \frac{(\sum_{x_j \in x_{-i}} x_j)!(h - (\sum_{x_j \in x_{-i}} x_j) - 1)!}{h!} \cdot \Delta c, \\ \Delta c &= (w_k \cdot c_k(x_{-i}, x_i = 1) - 0) \end{aligned} \quad (14)$$

The cube is satisfiable as we already removed all cubes where $|S^+(c_j) \cap S^-(c_j)| \geq 1$. Since all variables $x_j \notin S_k$ are null players, the NPO property allows us to remove them without changing $\phi_i^{original}(w_k \cdot c_k)$. After removing them, we are left with only a single assignment that satisfies the cube. We will mark this assignment, excluding x_i , as a . The assignment is:

$$a = \begin{cases} 1 & \text{if } x_j \in S_k^+ \\ 0 & \text{if } x_j \in S_k^- \end{cases} \quad | \quad \forall x_j \in S_k, j \neq i$$

Observe that $|a| = |S_k| - 1$ as a includes all the cube's variables excluding x_i . We will mark:

$$a = \{a_1, a_2, \dots, a_{|S_k|-1}\}.$$

Considering this, we can simplify Formula 14 above. Instead of going over all assignments, we only need to consider the assignment $a \cup \{x_i = 1\}$, as this is the only one that satisfies the cube:

$$\begin{aligned} \phi_i^{original}(c_k) &= \frac{(\sum_{j=1}^{|S_k|-1} a_j)! (|S_k| - (\sum_{j=1}^{|S_k|-1} a_j) - 1)!}{|S_k|!} w_k \stackrel{(a)}{=} \\ &= \frac{(|S_k^+| - 1)! (|S_k| - |S_k^+|)!}{|S_k|!} w_k \stackrel{(b)}{=} \\ &= \frac{w_k}{|S_k^+| \binom{|S_k|}{|S_k^+|}} \stackrel{(c)}{=} \phi_i^{simplified}(c_k) \end{aligned}$$

- (a) Equation 14 when only considering the single satisfying assignment:

- i. We remove the $\sum_{x_{-i} \in \{0,1\}^{h-1}}$ as there is only one satisfying assignment.

- ii. Replace h with $|S_k|$ (the size of $a \cup \{x_i = 1\}$, as we removed all null players).
- iii. Replace $\sum_{x_j \in x_{-i}} x_j$ with $\sum_{j=1}^{|S_k|-1} a_j$ as we consider the case where $a = \{a_1, \dots, a_{|S_k|-1}\} = x_{-i}$.
- iv. Replace $c_k(x_{-i}, x_i = 1)$ with w_k as the assignment satisfies the cube.
- (b) $\sum_{j=1}^{|S_k|-1} a_j = |S_k^+| - 1$ as a includes $|S_k^+| - 1$ variables that are set to 1. See the definition of a above.
- (c) Simple arithmetic.
- 3. $i \in S_k^-$: Now, any assignment with $x_i = 1$ will leave the cube unsatisfied. We will use the same technique as above, simplifying the original formula, removing null players, observing that after the removal there is only one satisfying assignment, and simplifying the formula even further using this observation:

$$\begin{aligned} \phi_i^{\text{original}}(c_k) &= \sum_{x_{-i} \in \{0,1\}^{h-1}} \frac{(\sum_{x_j \in x_{-i}} x_j)!(h - (\sum_{x_j \in x_{-i}} x_j) - 1)!}{h!} (0 - c_k(x_{-i}, x_i = 0)) \stackrel{(a)}{=} \\ &= \frac{(\sum_{j=1}^{|S_k|-1} a_j)! (|S_k| - (\sum_{j=1}^{|S_k|-1} a_j) - 1)!}{|S_k|!} (-w_k) \stackrel{(b)}{=} \\ &= \frac{(|S_k^+|)! (|S_k| - |S_k^+| - 1)!}{|S_k|!} (-w_k) \stackrel{(c)}{=} \\ &= \frac{(|S_k| - |S_k^-|)! (|S_k^-| - 1)!}{|S_k|!} (-w_k) \stackrel{(d)}{=} \\ &= \frac{-w_k}{|S_k^-|!} \stackrel{(e)}{=} \phi_i^{\text{simplified}}(c_k) \end{aligned}$$

- (a) As $x_i \in S_k^-$, any assignment with $x_i = 1$ will leave the cube unsatisfied.
- (b) Only considering the single satisfying assignment (with a defined as in the $x_i \in S_k^+$ case).
- (c) Now when $i \notin S_k^+$, the sum $(\sum_{j=1}^{|S_k|-1} a_j)$ for the satisfying assignment is $|S_k^+|$ and not $|S_k^+| - 1$.
- (d) We apply the identity $|S_k| = |S_k^-| + |S_k^+|$, since all cubes with $|S_k^+ \cap S_k^-| \geq 1$ have been excluded.
- (e) Simple arithmetic.

□

A direct corollary of Theorems 1 and 2 is that the simplified formulas for Shapley and Banzhaf values are robust w.r.t WDNF representations: equivalent WDNF will get the same Shapley and Banzhaf values.

Corollary 3. *The simplified formulas are robust. For two equal WDNF formulas $\forall x : F_1(x) = F_2(x)$, even if they are represented by different cubes, $\beta_i^{\text{simplified}}(F_1) = \beta_i^{\text{simplified}}(F_2)$ and $\phi_i^{\text{simplified}}(F_1) = \phi_i^{\text{simplified}}(F_2)$ for any player i .*

Proof. For any player i , ϕ_i^{original} and $\beta_i^{\text{original}}$ are robust as they are functions of assignments: they use F by calling it $(F(x))$ and not by considering its cube structure. Using Theorems 1 and 2:

$$\beta_i^{\text{simplified}}(F_1) = \beta_i^{\text{original}}(F_1) = \beta_i^{\text{original}}(F_2) = \beta_i^{\text{simplified}}(F_2)$$

$$\phi_i^{\text{simplified}}(F_1) = \phi_i^{\text{original}}(F_1) = \phi_i^{\text{original}}(F_2) = \phi_i^{\text{simplified}}(F_2) \quad \square$$

While the proof of Corollary 3 is straightforward, based on Theorems 1 and 2, the statement itself is not trivial, as the simplified formulas only consider the structure of the WDNF. For example, the WDNF:

$$5(x_1) - 5(x_3) + 3(\neg x_1 \wedge \neg x_3) + 10(\neg x_1 \wedge x_3) - 2(\neg x_1 \wedge \neg x_2 \wedge x_3)$$

is equal to the WDNF in Tables 4 and 5. Calculating its Banzhaf and Shapley values using the simplified formulas will return the same values as in the tables. More trivial functions over cubes, like the simple weight difference of positive versus negated literals, presented in Formula 15, are not robust.

$$\Delta W_i = \sum_{k=1, i \in S_k^+}^m w_k - \sum_{k=1, i \in S_k^-}^m w_k \quad (15)$$

Next, we show that $\beta_i^{\text{simplified}}$ and $\phi_i^{\text{simplified}}$ also apply to WCNF (see Def. 8).

Theorem 4. *For any player i and a Weighted Conjunctive Normal Form (WCNF) function F the simplified formulas are equal to the original ones: $\phi_i^{\text{simplified}}(F) = \phi_i^{\text{original}}(F)$, $\beta_i^{\text{simplified}}(F) = \beta_i^{\text{original}}(F)$.*

Proof. One can carefully apply the proofs of Theorems 1 and 2 above to WCNF, changing them slightly to consider the new form. We present another approach that utilizes the robustness property.

Given a WCNF formula, we will transform it to WDNF, run $\beta_i^{\text{simplified}}$ and $\phi_i^{\text{simplified}}$ on this WDNF, and show that it is equivalent to running them directly on the WCNF. As $\beta_i^{\text{simplified}}$, $\phi_i^{\text{simplified}}$, $\beta_i^{\text{original}}$, and ϕ_i^{original} are robust, transforming the WCNF formula to an equivalent WDNF formula does not change the Shapley/Banzhaf values.

$$w\psi = w - w(\neg\psi)$$

For example:

$$3(x_1 \vee \neg x_3) = 3 - 3(\neg x_1 \wedge x_3)$$

Applying the transformation on F we get the WDNF formula $F' = \sum_{k=1}^m w_k - w_k(\neg c_k)$. Calculating Shapley/Banzhaf on F' :

1. $w_k - w_k(\neg c_k)$ has a number w_k and a weighted cube. We can treat the number as a weighted cube with zero literals and observe that constant numbers have no effect on Shapley/Banzhaf values.
2. The original weight of the clause was multiplied by -1 .
3. Using de-Morgan's laws:

$$|S^+(c_k)| = |S^-(\neg c_k)|, \quad |S^-(c_k)| = |S^+(\neg c_k)|$$

When applying 2 and 3 to $\beta_i^{\text{simplified}}$ and $\phi_i^{\text{simplified}}$, both changes cancel out and we are left with the same equations. □

A direct corollary of Theorem 4 is the robustness of $\beta_i^{\text{simplified}}$ and $\phi_i^{\text{simplified}}$ also w.r.t WCNF formulas.

Corollary 5. *For any player i , both $\beta_i^{\text{simplified}}$ and $\phi_i^{\text{simplified}}$ are robust for both WDNF and WCNF.*

Proof. Identical to Corollary 3 proof. The robustness holds even if one formula is a WCNF and the other is a WDNF. $\square$

To summarize all of the above:

Theorem 6. *For any player i , $\beta_i^{\text{simplified}} = \beta_i^{\text{original}}$ and $\phi_i^{\text{simplified}} = \phi_i^{\text{original}}$ for both WDNF and WCNF. These simplified formulas are robust.*

B.5 Complexity Overview

First, we demonstrate how to use $\beta_i^{\text{simplified}}$ and $\phi_i^{\text{simplified}}$ to calculate the values of all the variables in the WDNF/WCNF formula in linear-time.

Proposition 7. Linear Complexity: *$\phi_i^{\text{simplified}}$ and $\beta_i^{\text{simplified}}$ can calculate the Shapley/Banzhaf values of all variables in linear-time with respect to the length of the WDNF/WCNF formula. In a single linear-time calculation, the Shapley/Banzhaf values of all players can be obtained.*

Proof. The computation of Shapley and Banzhaf values begins by initializing a dictionary that maps each variable to zero. Throughout the process, this dictionary is updated to store the current Shapley or Banzhaf value of each variable. The algorithm then traverses the formula, visiting each cube once, and incrementally updates the mapping with the partial values accumulated up to that cube. For each cube, it iterates over its variables and computes their contributions using the simplified formulas.

This process is illustrated in Tables 4 and 5. Conceptually, the algorithm “fills” these tables column by column - updating only the cells corresponding to variables present in the current cube, while the remaining entries remain zero.

For a cube c_k , with $|S_k|$ variables, a naive implementation might require $O(|S_k|^2)$, as calculating $\frac{1}{2^{|S_k|-1}}$, $\frac{1}{|S_k^-| \binom{|S_k|}{|S_k^-|}}$, and $\frac{1}{|S_k^+| \binom{|S_k|}{|S_k^+|}}$ takes $O(|S_k|)$ time for each of the $|S_k|$ variables.

However, we can optimize this by computing these fractions once per cube and reusing them. This approach requires $O(|S_k|)$ operations per cube for the fraction calculations and another $O(|S_k|)$ for the Shapley/Banzhaf value computations. Consequently, the overall complexity becomes linear with respect to the WDNF/WCNF length. $\square$

Let us formally define Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF):

Definition 9 (Conjunctive Normal Form (CNF)). *Conjunctive Normal Form (CNF) is a set of clauses, where the function returns 1 if all clauses are satisfied and 0 otherwise.*

Definition 10 (Disjunctive Normal Form (DNF)). *Disjunctive Normal Form (DNF) is a set of cubes, where the function returns 1 if at least one cube is satisfied and 0 otherwise.*

While Banzhaf and Shapley values on WCNF and WDNF can be calculated in linear time, calculating Shapley values on CNF and DNF formulas is #P-Hard.

Model counting is the task of counting the number of satisfying assignments; for CNF and DNF formulas, this is known to be #P-Hard (Provan and Ball 1983). A polynomial reduction between Shapley values computation and the model counting problem, demonstrating that computing Shapley values on CNF and DNF formulas is #P-Hard, is presented in (Arenas, Bertossi, and Monet 2021). Below, we provide a simple reduction between Banzhaf value calculation and model counting on CNF and DNF formulas, proving that Banzhaf value calculation on these formulas is also #P-Hard.

Theorem 8. *Banzhaf calculation on CNF and DNF is #P-Hard.*

Proof. Given a CNF/DNF formula ψ with n variables, we build the formula ψ' by adding a variable x_{n+1} and a cube/clause (x_{n+1}) — a cube/clause of size 1 that includes only the positive literal x_{n+1} .

- In CNF, when $x_{n+1} = 0$, the new formula ψ' is never satisfied, as (x_{n+1}) is false. When $x_{n+1} = 1$, the new formula is satisfied if and only if the old formula is satisfied:

$$\beta_{n+1}(\psi') = \mathbb{E}[\psi'(x)|x_{n+1} = 1] - \mathbb{E}[\psi'(x)|x_{n+1} = 0] =$$

$$\mathbb{E}[\psi(x)] = \frac{1}{2^{|N|}} \sum_{x \in \{0,1\}^n} \psi(x)$$

- In DNF, when $x_{n+1} = 0$, the new formula ψ' is satisfied if and only if the old formula is satisfied. When $x_{n+1} = 1$, the new formula is always satisfied, as (x_{n+1}) is true:

$$\beta_{n+1}(\psi') = \mathbb{E}[\psi'(x)|x_{n+1} = 1] - \mathbb{E}[\psi'(x)|x_{n+1} = 0] =$$

$$1 - \mathbb{E}[\psi(x)] = 1 - \frac{1}{2^{|N|}} \sum_{x \in \{0,1\}^n} \psi(x)$$

The model counting of ψ is $\sum_{x \in \{0,1\}^n} \psi(x)$. This can be easily extracted from β_{n+1} in both cases above. We have shown a simple reduction between Banzhaf value computation and model counting, thus proving that Banzhaf value computation is #P-Hard. $\square$

See a summary of the complexity of Shapley and Banzhaf values computation in Table 6.

Formula type	Shapley Complexity	Banzhaf Complexity
CNF	#P-Hard	#P-Hard
DNF	#P-Hard	#P-Hard
Weighted CNF	Linear time	Linear time
Weighted DNF	Linear time	Linear time

Table 6: Shapley and Banzhaf values calculation complexity on the different Boolean formulas.

C Shapley and Banzhaf Interaction Values on WDNF and WCNF - With Correctness Proofs

Shapley interaction values are defined in Def. 4, and their efficient computation for WDNF/WCNF formulas is presented in Table 1. Banzhaf interaction values are defined in Def. 11 below, and can be efficiently computed on WDNF/WCNF formulas using Formula 5.

Definition 11 (Banzhaf Interaction Values). *Banzhaf Interaction values examine the difference in expectations when features i and j are both missing or participating, versus when only one participates (Fujimoto, Kojadinovic, and Marichal 2006):*

$$\beta_{i,j \text{ } i \neq j}(M) = \mathbb{E}[M(S) \mid i, j \in S] + \mathbb{E}[M(S) \mid i, j \notin S] - \mathbb{E}[M(S) \mid i \in S \wedge j \notin S] - \mathbb{E}[M(S) \mid i \notin S \wedge j \in S] \quad (16)$$

Interaction values on WCNF: To compute Shapley and Banzhaf interaction values on WCNF, use Table 1 and Formula 5, and multiply their outputs by -1 . The correctness of this approach follows from proofs analogous to those of Table 1 (Theorem 10) and Formula 5 (Theorem 12).

Complexity of the Efficient Approach: Mark L as the number of cubes and D as the number of literals in each cube. While the calculation of Shapley and Banzhaf values has linear complexity $O(LD)$, computing all interaction values requires examining all pairs of literals in each cube, resulting in a complexity of $O(LD^2)$.

C.1 Correctness Proofs

Lemma 9. [Linearity Property of Shapley Interaction Values (folklore)] *Given a WDNF/WCNF formula $F = \sum_{k=1}^m w_k \cdot c_k$, for any players i, j such that $i \neq j$, the following holds:*

$$\phi_{i,j \text{ } i \neq j}(F) = \sum_{k=1}^m \phi_{i,j \text{ } i \neq j}(c_k)$$

Proof. The linearity property holds for Shapley interaction values because they are computed as a linear combination of standard Shapley values:

$$\begin{aligned} \phi_{i,j \text{ } i \neq j}(F) &= \phi_{j|i=1}(F) - \phi_{j|i=0}(F) = \\ &= \sum_{k=1}^m \phi_{j|i=1}(c_k) - \sum_{k=1}^m \phi_{j|i=0}(c_k) = \\ &= \sum_{k=1}^m [\phi_{j|i=1}(c_k) - \phi_{j|i=0}(c_k)] = \sum_{k=1}^m \phi_{i,j \text{ } i \neq j}(c_k) \end{aligned}$$

□

Theorem 10. *The formulas in Table 1 correctly calculate Shapley interaction values (as defined in Def. 4) on WDNF formulas.*

Proof. Given a WDNF formula F , from the linearity property (Lemma 9), it is enough to show that the formula holds for each cube c_k separately. Let us consider all possible cases and summarize them in Table 7:

1. $i \notin S_k$: Setting i to 0 or 1 will never change the satisfied weight, and $\phi_{j|i=1} = \phi_{j|i=0} = \phi_j$.
2. $j \notin S_k$: By the null player property, $\phi_{j|i=1} = \phi_{j|i=0} = 0$.
3. $i, j \in S_k^+$: When $i = 0$, the cube c_k is always falsified, so $\phi_{j|i=0} = 0$ (the Shapley values of the variables participating in a constant function are always zero, as the variables have no effect on the function output. We used this fact in the proof of Theorem 4). When we set $i = 1$, we are left with a cube c'_k such that $|S(c'_k)| = |S(c_k)| - 1$, $|S^+(c'_k)| = |S^+(c_k)| - 1$, and $|S^-(c'_k)| = |S^-(c_k)|$. Let $V = \{x_1, x_2, \dots, x_h\}$ be the Boolean variables of F (see Def. 2). Using Def. 4 and assignment in Formula 2, we get:

$$\phi_{i,j \text{ } i \neq j}(c_k) = \phi_j(c_k(V|x_i = 1)) - \phi_j(c_k(V|x_i = 0)) = \frac{w_k}{(|S^+(c_k)| - 1)(|S^+(c_k)| - 1)} - 0$$

4. $i, j \in S_k^-$: Symmetrically, when $i = 1$, the cube is always falsified, and $\phi_{j|i=1} = 0$. When we set $i = 0$, we are left with a cube c'_k such that $|S(c'_k)| = |S(c_k)| - 1$, $|S^+(c'_k)| = |S^+(c_k)|$, and $|S^-(c'_k)| = |S^-(c_k)| - 1$. Using Def. 4 and assignment in Formula 2, we get:

$$\phi_{i,j \text{ } i \neq j}(c_k) = 0 - \frac{-w_k}{(|S^-(c_k)| - 1)(|S^-(c_k)| - 1)}$$

5. $i \in S_k^+$ and $j \in S_k^-$: We know $\phi_{j|i=0} = 0$, and assigning $i = 1$ results in c'_k such that $|S(c'_k)| = |S(c_k)| - 1$ and $|S^-(c'_k)| = |S^-(c_k)|$:

$$\phi_{i,j \text{ } i \neq j}(c_k) = \frac{-w_k}{(|S^-(c_k)|)(|S^-(c_k)| - 1)} - 0$$

6. $i \in S_k^-$ and $j \in S_k^+$: Symmetrically:

$$\phi_{i,j \text{ } i \neq j}(c_k) = 0 - \frac{w_k}{(|S^+(c_k)|)(|S^+(c_k)| - 1)}$$

We have shown the formulas hold for all cases. Table 7 summarizes all cases. □

Lemma 11. *Banzhaf interaction values can also be defined as follows:*

$$\beta_{i,j \text{ } i \neq j} = \beta_{j|i=1} - \beta_{j|i=0} \quad (17)$$

Proof. Combining Def. 11 with Formula 3:

$$\begin{aligned} \beta_{i,j \text{ } i \neq j}(M) &= \mathbb{E}[M(S)|i, j \in S] + \mathbb{E}[M(S)|i, j \notin S] \\ &- \mathbb{E}[M(S)|i \in S \wedge j \notin S] - \mathbb{E}[M(S)|i \notin S \wedge j \in S] = \\ &= (\mathbb{E}[M(S)|j \in S \wedge i \in S] - \mathbb{E}[M(S)|j \notin S \wedge i \in S]) - \\ &= (\mathbb{E}[M(S)|j \in S \wedge i \notin S] - \mathbb{E}[M(S)|j \notin S \wedge i \notin S]) = \\ &= \beta_{j|i=1}(M) - \beta_{j|i=0}(M) \end{aligned}$$

Table 7: Summarize the Shapley interaction values proof. For each case we calculate both $\phi_{j|i=1}$ and $\phi_{j|i=0}$ to find the interaction value. We denote $S = S(c_k) = S_k$, $S^+ = S^+(c_k) = S_k^+$, $S^- = S^-(c_k) = S_k^-$, $w = w_k$.

Case	$\phi_{j i=1}$	$\phi_{j i=0}$	$\phi_{i,j} \ i \neq j$
$(i \in S^+) \wedge (j \in S^-)$	$\frac{-w}{(S^-)(\binom{ S -1}{ S^+ })}$	0	$\frac{-w}{(S^-)(\binom{ S -1}{ S^+ })}$
$(i \in S^-) \wedge (j \in S^+)$	0	$\frac{w}{(S^+)(\binom{ S -1}{ S^- })}$	$\frac{-w}{(S^+)(\binom{ S -1}{ S^- })}$
$i, j \in S^+$	$\frac{w}{(S^+ -1)(\binom{ S -1}{ S^+ -1})}$	0	$\frac{w}{(S^+ -1)(\binom{ S -1}{ S^+ -1})}$
$i, j \in S^-$	0	$\frac{-w}{(S^- -1)(\binom{ S -1}{ S^- -1})}$	$\frac{w}{(S^- -1)(\binom{ S -1}{ S^- -1})}$
$(i \notin S) \wedge (j \in S^+)$	$\frac{w}{(S^+)(\binom{ S }{ S^+ })}$	$\frac{w}{(S^+)(\binom{ S }{ S^+ })}$	0
$(i \notin S) \wedge (j \in S^-)$	$\frac{-w}{(S^-)(\binom{ S }{ S^- })}$	$\frac{-w}{(S^-)(\binom{ S }{ S^- })}$	0
$j \notin S$	0	0	0

□

Theorem 12. Formula 5 correctly calculates Banzhaf interaction values on WDNF formulas.

Proof. The linearity property holds for the Banzhaf interaction values due to the linearity of expectations and Definition 11. Utilizing the definition presented in Lemma 11, we can apply the proof structure of Shapley interaction values here and derive Formula 5. Table 8 summarizes the cases in the proof. □

Table 8: Summary of the Banzhaf interaction values proof. For each case, we calculate both $\beta_{j|i=1}$ and $\beta_{j|i=0}$ to find the interaction value. We denote $S = S(c_k) = S_k$, $S^+ = S^+(c_k) = S_k^+$, $S^- = S^-(c_k) = S_k^-$, $w = w_k$.

Case	$\beta_{j i=1}$	$\beta_{j i=0}$	$\beta_{i,j} \ i \neq j$
$(i \in S^+) \wedge (j \in S^-)$	$\frac{-w}{2^{ S -2}}$	0	$\frac{-w}{2^{ S -2}}$
$(i \in S^-) \wedge (j \in S^+)$	0	$\frac{w}{2^{ S -2}}$	$\frac{-w}{2^{ S -2}}$
$i, j \in S^+$	$\frac{w}{2^{ S -2}}$	0	$\frac{w}{2^{ S -2}}$
$i, j \in S^-$	0	$\frac{-w}{2^{ S -2}}$	$\frac{w}{2^{ S -2}}$
$(i \notin S) \wedge (j \in S^+)$	$\frac{w}{2^{ S -1}}$	$\frac{w}{2^{ S -1}}$	0
$(i \notin S) \wedge (j \in S^-)$	$\frac{-w}{2^{ S -1}}$	$\frac{-w}{2^{ S -1}}$	0
$j \notin S$	0	0	0

D Complexity Analysis

This section presents a complexity analysis of our algorithms. We begin by analyzing the complexity of *MapPatternsToCube* and *CalcDecisionPatterns*, followed by an analysis of our main algorithm WOODELF. The input for our algorithms is:

- A decision tree ensemble with T trees, each tree with depth D , K nodes and L leaves. In the complexity analysis we assume the trees are more or less balanced, i.e. $O(K) = O(2^D) = O(L)$.

- Consumer data C with n rows
- Background data B with m rows
- A function v that takes a cube and returns a mapping from feature subsets (of size one or more) to real numbers. This function can compute Shapley/Banzhaf values, interaction values and any other metric that satisfies the linearity property.

CalcDecisionPatterns complexity: The algorithm's input is a single decision tree and the consumer data C . The algorithm performs a Breadth-First Search (BFS), applying the split function of each node to all consumers. This results in a time complexity of $O(nK) = O(nL)$. The output size is also $O(nL)$, indicating that the algorithm is asymptotically optimal.

MapPatternsToCube complexity: The algorithm's input is the list of features along a root-to-leaf path, which has length D . The algorithm iterates over this list, tripling the number of entries in the dictionary d at each step. As a result, both the runtime and output size are $O(3^D)$, indicating that the algorithm is asymptotically optimal.

Function v complexity: WOODELF complexity depends on the function v . In this section we use amortized complexity analysis and consider the average-case complexity and output size of v over all possible cubes defined on $\{x_1, \dots, x_D\}$.

The average case complexity for cubes of length up to D is defined as the total worst-case complexity of running v on all possible cubes over $\{x_1, \dots, x_D\}$ (or any subset of these variables), divided by the number of cubes, 3^D . Similarly the average case output size is the total worst-case output size on these cubes divided by 3^D . Here, the $\{x_1, \dots, x_D\}$ variables represents features along a root-to-leaf path. We consider the worst-case where all paths in the tree include unique features (no path with repeating features). We denote the average output size by $O(v_o^{avg})$ and the average complexity by $O(v_c^{avg})$. Running v on all cubes then costs $O(3^D v_c^{avg})$ and produces $O(3^D v_o^{avg})$ outputs. Clearly, $O(v_o^{avg}) \leq O(v_c^{avg})$.

For Shapley/Banzhaf values, $O(v_c^{avg}) = O(v_o^{avg}) = O(D)$, as for each cube the function v computes and returns one value per each variable. For Shapley/Banzhaf interaction values, $O(v_c^{avg}) = O(v_o^{avg}) = O(D^2)$, as for each cube the function v computes and returns one value per each pair of variables.

For both Shapley/Banzhaf values and their interaction values, the average-case complexity and output size are asymptotically the same as the complexity and output size on the worst case cube/clause. Thus, using average-case instead of worst-case measures does not change the overall complexity for these values. We still provide the tightest bounds we can, hoping they will be useful for future research.

Another property of v needed in the analysis is $O(v_s)$, the number of unique feature subsets returned by all possible cubes over $\{x_1, \dots, x_D\}$. For Shapley/Banzhaf values $O(v_s) = D$, since v returns all subsets of size one. For Shapley/Banzhaf interaction values $O(v_s) = D^2$, since v returns all subsets of size two.

WOODELF complexity: The algorithm has several steps: computing f , M , s , and finally the Shapley/Banzhaf values. We analyze the complexity of each step individually.

1. Computing f :

In Background: Calling *CalcDecisionPatterns*(T, B) takes $O(mL)$ per tree and returns an output of size $O(mL)$. The *value_counts* function and *bincount* function, a more efficient Numpy alternative we used in practice, processes this output in linear time, resulting in an overall complexity of $O(mTL)$.

In Path-Dependent: Formula 10 can be computed for all leaves in a single traversal of the tree, requiring $O(L2^D) = O(L^2)$ time per tree and $O(TL^2)$ overall.

2. **Computing M :** For each leaf in each tree we iterate over all $O(3^D)$ cubes generated by *MapPatternsToCube*, apply v to each cube, and iterate over its output. These cubes are all the possible cubes over the D variables that represents the features along the root-to-leaf path. Thus, the complexity of the step is $O(TL3^D v_c^{avg})$. Note: for this analysis to be correct we assume $1 \leq v_c^{avg}$, even if most cubes are skipped the average complexity can not be less than $O(1)$.

This step constructs $O(TLv_s)$ matrices, and these matrices together contain $O(TL3^D v_o^{avg})$ non-zero entries. Each matrix is sparse: its size is 4^D (since both p_c and p_b can range between 0 and 2^D), but it holds at most 3^D non-zero entries.

3. **Computing s :** The bottleneck here is the matrix-vector multiplication $M_{l,i} \cdot f_i$. By performing sparse matrix multiplications, we reduce the complexity of the matrix-vector multiplication from $O(4^D)$ to $O(3^D)$.

This step constructs $O(TLv_s)$ vectors, one for each matrix. Sparse matrix-vector and matrix-scalar multiplications run in time proportional to the number of non-zero entries. Therefore, this step has a complexity of $O(TL3^D v_o^{avg})$.

4. **Computing the final values:** Computing P_c requires $O(nTL)$ time. The final Shapley/Banzhaf values or interaction values computed are via NumPy indexing over all s vectors, each requiring $O(n)$ time. This yields an overall complexity of $O(nTLv_s)$.

By summing the complexities above, we obtain the total runtime of WOODELF for each case, as summarized in Table 9.

E Expanded Experimental Section

This section expands on the experimental results presented in Sec. 10 and provides additional details on the empirical correctness verification employed in our study.

E.1 Additional Experimental Details

1. **Estimated Runtime:** the SHAP Python package supports Baseline and Path-Dependent SHAP, whereas Background SHAP is limited to 100 background samples (Bracci 2021), and Background Shapley interaction values are not supported (Porwisz 2019). Therefore, the

Task	WOODELF Complexity
Path Dependent on function v	$O(nTLv_s + TL3^D v_c^{avg})$
Background on function v	$O(mTL + nTLv_s + TL3^D v_c^{avg})$
Path Dependent SHAP	$O(nTLD + TL3^D D)$
Path Dependent Banzhaf	$O(nTLD + TL3^D D)$
Background SHAP	$O(mTL + nTLD + TL3^D D)$
Background Banzhaf	$O(mTL + nTLD + TL3^D D)$
Path Dependent SHAP IV	$O(nTLD^2 + TL3^D D^2)$
Path Dependent Banzhaf IV	$O(nTLD^2 + TL3^D D^2)$
Background SHAP IV	$O(mTL + nTLD^2 + TL3^D D^2)$
Background Banzhaf IV	$O(mTL + nTLD^2 + TL3^D D^2)$

Table 9: WOODELF complexity. SHAP/Banzhaf IV refers to the task of calculating all Shapley/Banzhaf interaction values. Legend: n = consumer data size, m = background data size, T = number of trees, L = number of leaves per tree, D = tree depth, $O(v_c^{avg})$ is function v average complexity on all the cubes generated by a single root-to-leaf path and $O(v_s)$ is the number of unique feature subsets that v returns from these cubes.

running times of these Background SHAP are estimated. The running times of Shapley interaction values are also estimated due to RAM limitations. See the estimation methodology in Appendix. E.2.

2. **Modeling Framework:** Several state-of-the-art SHAP packages do not support XGBoost. To enable comparison on similar models, we used known alternatives: PLTreeShap (Zern, Broelemann, and Kasneci 2023) was run on a LightGBM model, and FastTreeSHAP (Yang 2022) on a scikit-learn RandomForest. All models consisted of 100 trees with a maximum depth of 6.
3. **GPU SOTA:** The state-of-the-art (SOTA) GPU algorithm is GPUPTreeSHAP. It is available in the SHAP package, but requires cloning the repository and installing it locally (Lundberg and contributors 2024). The Path-Dependent variant of the same algorithm is also integrated into the XGBoost Python package and can be used without additional setup. Although both implementations are based on the same algorithm, the XGBoost version is significantly faster in practice. In our experiments, we used the XGBoost implementation as the SOTA for Path-Dependent SHAP and the SHAP package as the SOTA for Background SHAP.

See the results and performance comparison in Sect. 10.

E.2 SOTA Running Time Estimation

Path-Dependent SHAP times represent actual measurements. We run the SHAP Python package Background SHAP on both CPU and GPU using a background dataset of size 100 (the largest size possible (Bracci 2021)) and the full consumer dataset. Even with this small background dataset, the SHAP took 3 minutes on IEEE-CIS and 90 minutes on KDD—already significantly slower than WOODELF, which

processed the entire background dataset in just 12 seconds on IEEE-CIS and 162 seconds on KDD. To estimate SHAP running time on the entire background dataset we scaled the sample running time by $0.01 \times Actual_Background_Size$. Additional experiments with background sizes of 10, 20, and 50 confirmed that SHAP running time scales linearly with background size.

Due to RAM limitations we estimate the running time of Shapley interaction values tasks using the same technic. See table 10 for estimation details.

Task	Framework	Dataset	$ B $	$ C $	Runtime
BG SHAP	SHAP	IEEE-CIS	100	all	177 sec
BG SHAP	SHAP	KDD	100	all	5428 sec
BG SHAP	SHAP (GPU)	IEEE-CIS	100	all	11 sec
BG SHAP	SHAP (GPU)	KDD	100	all	160 sec
PD SHAP IV	SHAP	IEEE-CIS	-	100	101 sec
PD SHAP IV	SHAP	KDD	-	100	24 sec
PD SHAP IV	SHAP (GPU)	IEEE-CIS	-	10000	9 sec
PD SHAP IV	SHAP (GPU)	KDD	-	10000	0.8 sec
PD SHAP IV	FastTreeShap	KDD	-	100000	445 sec
BG SHAP IV	PLTreeSHAP	IEEE-CIS	all	10000	70 sec
BG SHAP IV	PLTreeSHAP	KDD	all	10000	195 sec

Table 10: Estimation approach for all estimated tasks. Legend: $|B|$: background data size, $|C|$ consumer data size, BG: Background, PD: Path Dependent (on Path Dependent SHAP B is not used and thus $|B| = 0$), SHAP is the shap Python package. The "Runtime" column shows the time taken on the sampled data, before extrapolation.

E.3 Empirical Correctness Verification

To validate the correctness of WOODELF, we compared the Shapley values and interaction values it computed with those generated by the SHAP Python package. This comparison was carried out on the first 1000 consumers of the IEEE-CIS dataset, using the first 80 rows from the corresponding background dataset. The results showed strong agreement, with all values differing by at most 0.00001. Similar consistency was observed with Path-Dependent SHAP and Path-Dependent SHAP IV.

To validate the correctness of WOODELF's Banzhaf values, we compared its output on a small synthetic dataset to a direct exponential-time implementation. On 100 consumers and background size 3, all values differed by at most 0.00001.