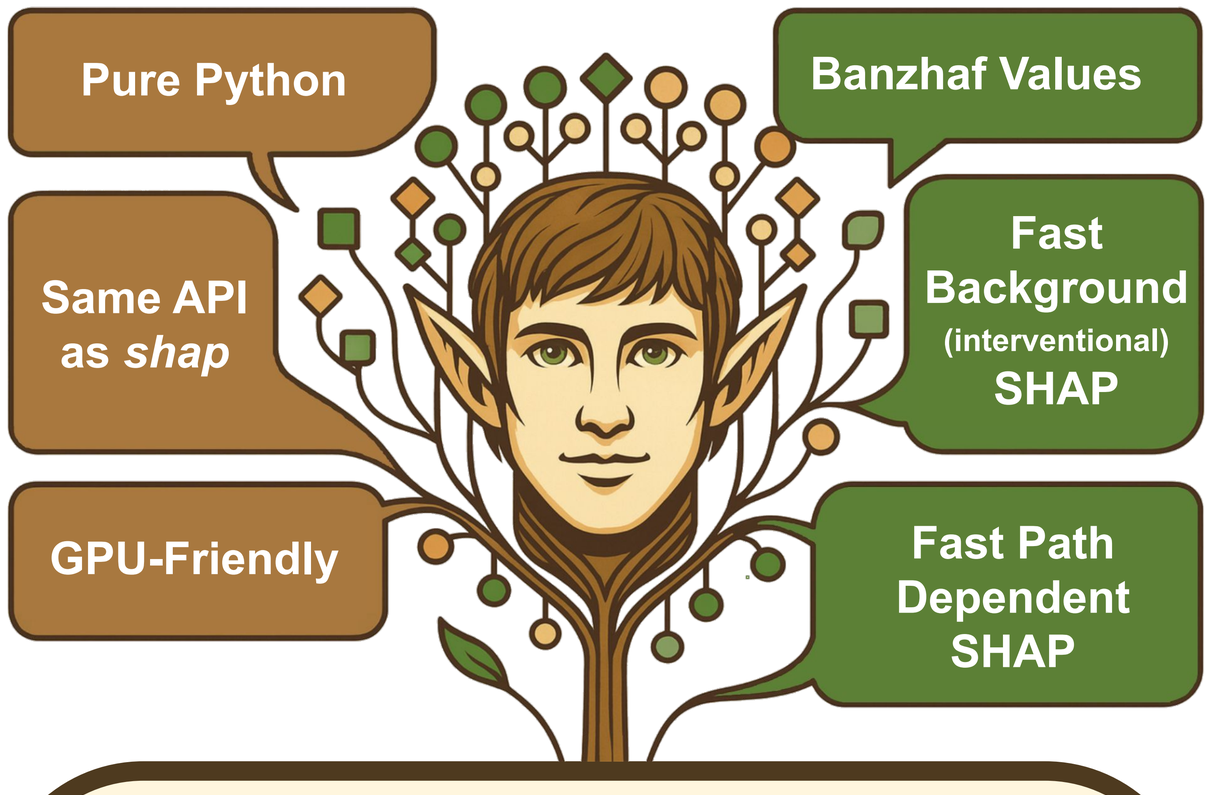
From Decision Trees to Boolean Logic:
A Fast and Unified SHAP Algorithm

Alexander Nadel, Ron Wettenstein
alexandernad@technion.ac.il, ron.wettenstein@post.runi.ac.il



The WOODELF Algorithm

Origin: Shapley values, rooted in game theory, are widely used for feature importance in machine learning.

WOODELF is a unified and efficient algorithm for computing attribution values on decision trees, supporting:

- **Execution:** GPU and CPU.
- **Explanation types:** Path Dependent and Background.
- **Values:** Shapley values & Shapley interaction values, Banzhaf values & Banzhaf interaction values.
- **All in one algorithm**.

Performance:

- Optimized for data-scaling using NumPy and SciPy.
- Extensive **SIMD utilization** for speed.

Installation: *pip install woodelf_explainer*

API (same as shap):

```
from woodelf import WoodelfExplainer
explainer = WoodelfExplainer(model, bg_data)
values = explainer.shap_values(data)
```

Results

Explains **3M predictions** using **5M background samples** for an XGBoost model (100 trees, depth 6) on KDD-Cup 99, comparing WOODELF to SHAP and the state of the art.

PD = Path Dependent, BG = Background, IV = interactions

Task	shap package	SOTA		WOODELF	
		CPU	GPU	CPU	GPU
PD SHAP	51 min	6.2 min	7.9 sec	1.6 min	**3.3 sec**
BG SHAP	8 years	44 min	3 months	2.7 min	**16 sec**
PD SHAP IV	8 days	221 min	3.8 min	3.2 min	**6 sec**
BG SHAP IV	Not available	105 min	Not available	4.4 min	**19 sec**

* Long running times (8 years, 3 months, 8 days, 221 min, 105 min) are estimated. 3.8 min is also estimated due to GPU RAM limitations.
* All models were trained on 127 features.
* State-of-the-art algorithms:
Path Dependent (CPU): FastTreeShap
Background (CPU): PLTreeShap
Path Dependent + Background (GPU): GPUTreeSHAP.

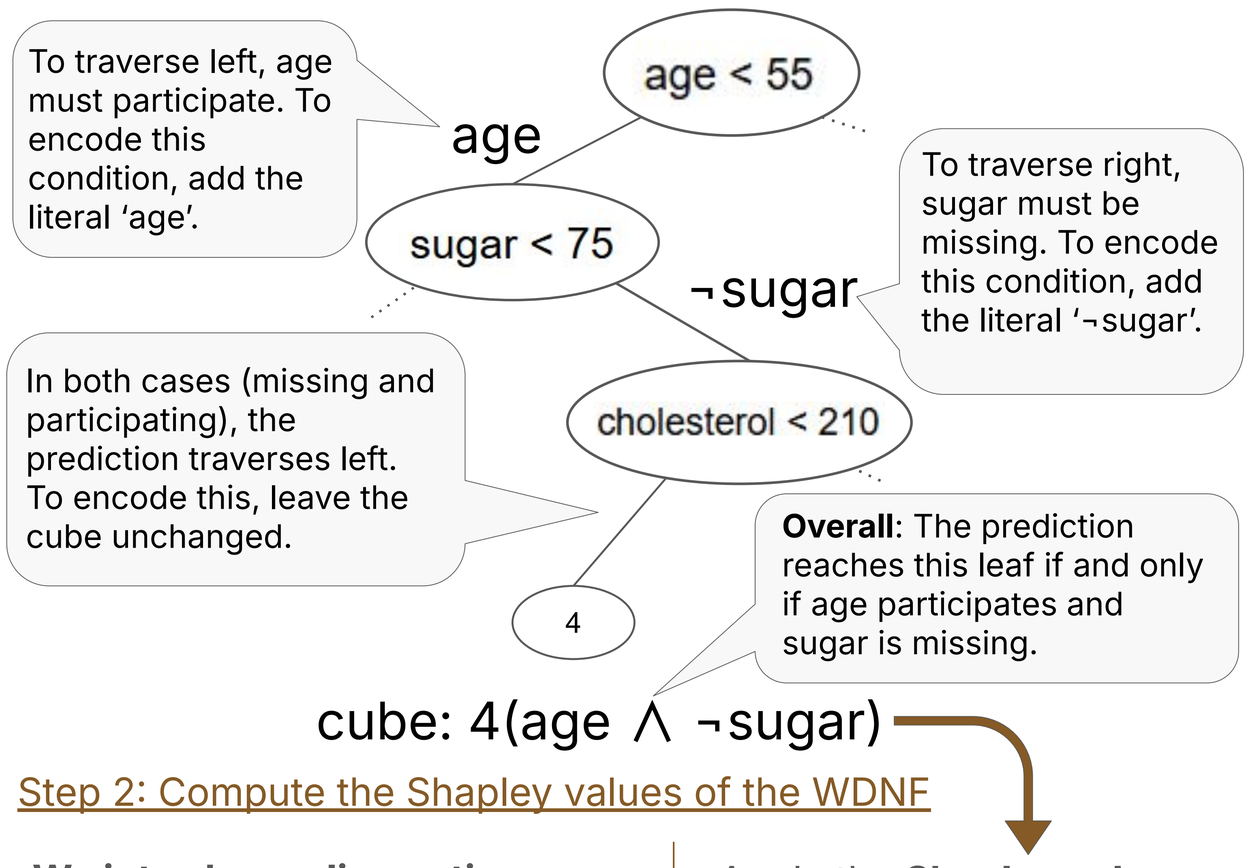
Baseline SHAP Approach

Baseline SHAP: In the baseline SHAP game, participating features retain their original values, while missing features are replaced with baseline values. The **payoff** is the resulting model prediction. We present WOODELF's approach to Baseline SHAP:

Step 1: Encode the Baseline SHAP game as a WDNF

WDNF is a weighted sum of cubes, e.g. $3(x_1 \wedge x_2)+5(\neg x_1 \wedge \neg x_3 \wedge x_4)$
In our encoding, each leaf gets a weighted cube. Example:

Patient values (participating): age=35, sugar=60, cholesterol=200
Baseline values (missing): age=60, sugar=80, cholesterol=180



Step 2: Compute the Shapley values of the WDNF

We introduce a linear-time Shapley values formula for WDNF:

For cube c_k mark: $S(c_k)$ its literals $S^+(c_k)$ its positive literals $S^-(c_k)$ its negated literals w_k its weight. Drop all $|S^-(c_k) \cap S^+(c_k)| > 0$.

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

We also introduce similar formulas for Banzhaf values, Banzhaf interaction values and Shapley interaction values.

Apply the **Shapley value formula for WDNF** on the **leaf's weighted cube**.

$$\phi(\ 4(age \wedge \neg sugar)\ ):$$

1. $\phi_{age} = \frac{4}{1\binom{2}{1}} = 2$
2. $\phi_{sugar} = \frac{-4}{1\binom{2}{1}} = -2$
3. $\phi_{cholesterol} = 0$

Do this for **all leaves** and **sum up the results**.
The outcome: **Baseline SHAP values**.

Background SHAP Approach

Background SHAP averages Shapley values over a background dataset of size *m*, treating each background sample as a baseline. **Naively, explaining *n* samples requires *O(nm)* time**. **WOODELF achieves *O(n+m)* complexity**.

- In the **example above,** we only care whether the **patient's age passes the threshold**. The cube is **identical for all patient ages < 55**.
- This sufficient information is encoded in a **Decision Pattern**: a binary sequence indicating which nodes along a root-to-leaf path are satisfied.
- For trees of depth D, there are only **2^D possible patterns**—far fewer than the number of samples or baselines.
- WOODELF **precomputes the cube and its Shapley contribution** for each (sample pattern, baseline pattern) pair.
- These precomputations are reused across all data points, reducing complexity from **O(nm) to O(n + m)**.
- **Further boost**: **All operations that scale with the data size are fully vectorized using NumPy/SciPy.** These operations are: conditioning ($x < \theta$), bit shifts, additions, bincount, matrix-vector multiplication and fancy indexing. Many of these operations support **SIMD execution** and all support **GPU acceleration**.