Implementing TreeSHAP via Dynamic Programming for Scalable and Exact Feature Attribution in Explainable AI

Samuelson Dharmawan Tanuraharja - 13524001[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13524001@std.stei.itb.ac.id, [2]samuelsondharmawan31@gmail.com

***Abstract*—Tree ensemble models achieve high predictive accuracy but generate predictions with no interpretable reasoning. Shapley values, rooted in cooperative game theory, provide the only attribution framework satisfying four axiomatic fairness properties at once. Computing the exact Shapley value for a feature requires evaluating a conditional expectation for every subset of the feature set, a cost exponential in the feature count. TreeSHAP reduces this cost to polynomial time by exploiting tree path structure through a dynamic programming recurrence over path polynomials. This paper implements TreeSHAP from scratch in Python without using the reference SHAP (SHapley Additive exPlanations) library, to expose the algorithmic structure. We train a Random Forest classifier on the UCI Adult Income dataset (45,222 instances, 10 features), verify the implementation against an exact brute-force baseline, and benchmark runtime as the feature count grows from three to ten. Verification confirms the implementation matches the brute-force baseline to floating-point precision. At eight features, TreeSHAP runs approximately 120 times faster than the brute-force; at nine features, the brute-force approaches 64 seconds while TreeSHAP completes in under 0.24 seconds. Dynamic programming makes exact Shapley attribution practical at the scale of real ensemble models.**

***Keywords*—Dynamic Programming; Shapley values; TreeSHAP; Tree Ensembles; Feature Attribution; Explainable AI**

I. Introduction

Machine learning practitioners encounter a persistent tension: models that achieve high predictive accuracy tend to resist interpretation. A single decision tree exposes every split threshold an analyst can trace back to the input features. A Random Forest of 500 trees, or an XGBoost ensemble, produces no such audit trail. The output is a prediction score with no attached reasoning. When these models govern credit approvals, medical triage, or judicial risk assessments, this gap carries real consequences: regulators, patients, and defendants have no basis to audit or contest the decision [1].

Feature attribution methods quantify each input feature's contribution to a specific prediction. Shapley values [2], rooted in cooperative game theory, are the only attribution scheme satisfying four axiomatic properties at once: *efficiency* (attributions sum to the full prediction shift relative to the base rate), *symmetry* (features with equal impact receive equal shares), *null player* (features with zero impact receive zero attribution), and *additivity* (attributions compose over additive models). Lundberg and Lee [3] showed that no other attribution method satisfies all four.

The Shapley value of feature k for sample $\mathbf{x}$ is:

$$\phi_k = \sum_{S \subseteq \mathcal{M} \setminus \{k\}} \frac{|S|!\,(|\mathcal{M}| - |S| - 1)!}{|\mathcal{M}|!} \big[v(S \cup \{k\}) - v(S)\big] \quad (1)$$

where:

- $\mathcal{M}$ the full feature set;
- $v(S)$ the model's expected output when only features in S are observed

Computing ϕ_k requires evaluating v for every subset of $\mathcal{M} \setminus \{k\}$, a total of $2^{|\mathcal{M}|-1}$ calls per feature per sample. For $|\mathcal{M}| = 20$, this amounts to over 500,000 evaluations per feature. For $|\mathcal{M}| = 30$, the count exceeds 500 million.

Lundberg et al. [4] identified a structural property of tree ensembles that eliminates this exponential growth. For a given sample $\mathbf{x}$, the conditional expectation $v(S)$ within a single tree follows from its path structure: features in S follow $\mathbf{x}$'s actual split decisions, while features outside S branch into both children, weighted by their training-data proportions.

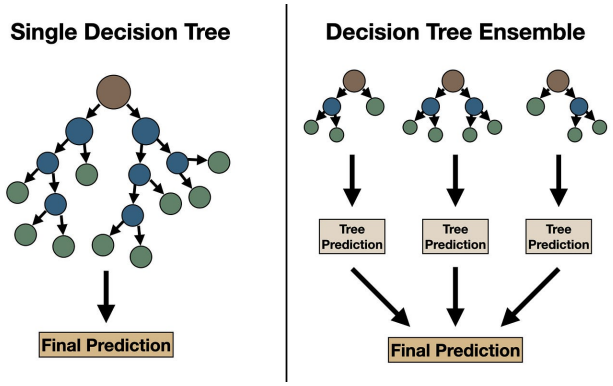

Fig. 1. Difference between a Decision Tree and a Random Forest *(Source: Talebi [10]).*

The sum over all $2^{|\mathcal{M}|}$ subsets collapses into a product of linear polynomials along each root-to-leaf path. A dynamic programming recurrence evaluates this product in $O(TLD^2)$ time, where T is the number of trees, L is the number of leaves per tree, and D is the maximum depth.

This paper implements TreeSHAP from scratch in Python, to expose the dynamic programming structure. We train a Random Forest on the UCI Adult Income dataset [5], verify correctness against an exact brute-force baseline, and measure empirical speedup as the feature count grows from three to ten. Section II reviews the theoretical foundations. Section III presents the TreeSHAP algorithm and the experimental setup. Section IV reports the verification and benchmarking results. Section V concludes.

II. Theoretical Background

A. Cooperative Game Theory and Shapley Values

Cooperative game theory models situations where agents form coalitions to achieve a collective payoff. A game (M, v) consists of a finite player set M and a *characteristic function* $v : 2^M \to \mathbb{R}$ that assigns a payoff $v(S)$ to every coalition $S \subseteq M$. The central problem is distributing the total payoff $v(M)$ among the individual players in a principled way.

Shapley [2] proved that a unique distribution $\{\phi_k\}_{k\in M}$ satisfies all four axiomatic properties at once. *Efficiency*: the attributions sum to the total gain, $\sum_{k\in M} \phi_k = v(M) - v(\varnothing)$. *Symmetry*: two players who contribute equally to every coalition receive equal shares. *Null player*: a player who changes no coalition's payoff receives zero attribution. *Additivity*: attributions over a sum of two games equal the sum of attributions over each game. Lundberg and Lee [3] proved no other attribution method satisfies all four.

The weighting coefficient in (1), $|S|!\,(|\mathcal{M}|-|S|-1)!/|\mathcal{M}|!$, counts the fraction of all $|\mathcal{M}|!$ orderings of the feature set in which S is exactly the subset of features that precede k. Summing the marginal contribution $[v(S\cup\{k\})-v(S)]$ across all such orderings yields the average marginal contribution of k, consistent with both the symmetry and efficiency axioms.

In the machine learning setting, we map M to the full feature set and define $v(S)$ as the model's expected output when the model receives only the features in S, with the remaining features integrated out over the training distribution:

$$v(S) = \mathbb{E}_{\mathbf{x}_{M\setminus S}}\big[f(\mathbf{x}_S,\, \mathbf{x}_{M\setminus S})\big]. \tag{2}$$

We can compute $v(S)$ in closed form using the tree structure, as Section III shows.

B. Tree Ensemble Models

A decision tree is a directed binary tree where each internal node v stores a split feature $j_v \in \{1, \ldots, M\}$, a threshold $\theta_v \in \mathbb{R}$, and pointers to a left and a right child. For sample $\mathbf{x}$, the tree routes $\mathbf{x}$ from the root by comparing x_{j_v} against θ_v at each node: left when $x_{j_v} \le \theta_v$, right when $x_{j_v} > \theta_v$. Traversal ends at a leaf, which stores the proportion of positive-class training samples that reached it, serving as the predicted probability $f(\mathbf{x})$.

A Random Forest [6] builds T such trees, each on an independent bootstrap sample of the training data and each sampling a random subset of features as split candidates at every node. The ensemble averages the T individual predictions:

$$f_{\text{RF}}(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x}). \tag{3}$$

The additivity axiom of Shapley values makes the ensemble attribution exact: $\phi_k^{\text{RF}} = (1/T)\sum_t \phi_k^{(t)}$, so a single per-tree computation suffices.

C. Dynamic Programming

Dynamic programming solves a problem by combining solutions to overlapping subproblems, storing each result to avoid recomputation. Two conditions identify a problem suitable for DP: *optimal substructure* means the solution to the full problem depends only on solutions to strictly smaller subproblems, and *overlapping subproblems* means the same subcomputation appears in multiple branches of the naive recursion.

Computing the Shapley sum in (1) naively enumerates all $2^{|\mathcal{M}|-1}$ subsets of $\mathcal{M} \setminus \{k\}$. Both conditions hold here. Each feature's contribution to a subset product is an independent binary choice: include it with weight o_j, or exclude it with weight z_j. The partial product over the first j features then appears in the computation for every subsequent feature $k > j$, satisfying the overlapping subproblems condition.

Encoding these products as coefficients of a polynomial in an auxiliary variable u compresses all $2^{|\mathcal{M}|}$ subset sums into $|\mathcal{M}|+1$ coefficients. Multiplying in one factor $(z_j + o_j \cdot u)$ at a time builds the full polynomial in $O(|\mathcal{M}|^2)$ steps rather than $O(2^{|\mathcal{M}|})$. Section III applies this observation to the root-to-leaf paths of decision trees to derive the TreeSHAP recurrence.

D. Related Work

Feature attribution methods divide into model-agnostic and model-specific approaches.

LIME [8] approximates each feature's contribution by fitting a local linear surrogate on perturbed samples drawn near the input. The surrogate coefficients serve as attributions, but LIME does not satisfy the Shapley axioms and its output depends on the perturbation scheme and sample count.

KernelSHAP [3] casts the Shapley sum as a weighted linear regression over sampled feature coalitions, making Shapley-consistent attribution available for any black-box model. The method satisfies all four axioms but approximates the result through sampling; obtaining high-accuracy attributions requires a large number of model evaluations.

TreeSHAP [4] achieves exact Shapley attribution without sampling by computing the conditional expectation $v(S)$ in closed form using the tree's path structure. For models without accessible tree internals, KernelSHAP remains the only Shapley-consistent option. For tree ensembles, TreeSHAP replaces stochastic sampling with a deterministic traversal at $O(TLD^2)$ cost, eliminating approximation error entirely. Table I summarizes the three methods.

TABLE I: Comparison of Feature Attribution Methods

Method	Exact	Model-agnostic	Complexity
LIME [8]	No	Yes	$O(B \cdot M)$
KernelSHAP [3]	No	Yes	$O(B \cdot M^2)$
TreeSHAP [4]	Yes	No	$O(T \cdot L \cdot D^2)$

B = background sample count, M = features, T = trees, L = leaves, D = depth.

III. Methodology

A. The TreeSHAP Algorithm

For a single decision tree and sample $\mathbf{x}$, the conditional expectation $v(S)$ from (2) decomposes along the tree's root-to-leaf path. At each internal node with split feature j, features in S direct the traversal along $\mathbf{x}$'s actual branch, while features outside S distribute probability mass to both branches in proportion to the training-sample counts they contain. We assign two scalars to each split node j on the path: $o_j = 1$ if the traversal follows $\mathbf{x}$'s branch at node j and $o_j = 0$ otherwise, and $z_j = n_{\text{child}}/n_{\text{node}}$, the fraction of training samples in the child the traversal enters.

The contributions of all 2^m feature subsets along a path of m nodes collapse into the coefficients of a single polynomial in an auxiliary variable u. Define the *path polynomial* F as:

$$F(u) \;=\; \prod_{j=0}^{m-1} \big(z_j + o_j \cdot u\big). \tag{4}$$

The coefficient $F[t]$ of u^t equals the sum over all size-t subsets of path features:

$$F[t] \;=\; \sum_{\substack{S \subseteq \{0,\ldots,m-1\} \\ |S|=t}} \prod_{j \in S} o_j \cdot \prod_{j \notin S} z_j. \tag{5}$$

Table II traces the array F for a three-node path with features age (o_0=1, z_0=0.6), education-num (o_1=0, z_1=0.3), and marital-status (o_2=1, z_2=0.7).

TABLE II: DP State of Path Polynomial $F[t]$ Along a Three-Node Example Path

Step	Factor multiplied	$F[0]$	$F[1]$	$F[2]$	$F[3]$
Init	—	1.000	0.000	0.000	0.000
$m=1$	$(0.6 + 1.0\,u)$	0.600	1.000	0.000	0.000
$m=2$	$(0.3 + 0.0\,u)$	0.180	0.300	0.000	0.000
$m=3$	$(0.7 + 1.0\,u)$	0.126	0.390	0.300	0.000

$F[t]$ encodes the sum of products over all size-t subsets of the three path features (5). At m=3: $F[0]$=$z_0z_1z_2$, $F[1]$=$o_0z_1z_2 + z_0z_1o_2 = 0.390$, $F[2]$=$o_0z_1o_2 = 0.300$.

Building F by multiplying one factor at a time costs $O(m^2)$, replacing the $O(2^m)$ explicit enumeration. At each leaf, the Shapley contribution of path feature k is:

$$\phi_k \;\mathrel{+}=\; v_\ell \cdot (o_k - z_k) \cdot \sum_{t=0}^{m-1} G_k[t] \cdot W(t,\, m), \tag{6}$$

where:

$W(t,m) = t!\,(m{-}1{-}t)!/m!$, the Shapley weight for a coalition of size t in a game with m players;
$G_k(u) = F(u)\,/\,(z_k + o_k \cdot u)$, computed via polynomial long division in $O(m)$ steps

Algorithm 1 states the full procedure.

Algorithm 1 TreeSHAP attribution for a single tree

Require: tree $\mathcal{T}$, sample $\mathbf{x}$, feature count M
Ensure: $\boldsymbol{\phi} \in \mathbb{R}^M$: Shapley contributions

$\boldsymbol{\phi} \leftarrow \mathbf{0}_M$
RECURSE(root, [])
return $\boldsymbol{\phi}$

procedure RECURSE(node v, path)
  if v is a leaf **then**
    $v_\ell \leftarrow$ positive-class proportion at leaf v
    ACCUMULATE(path, v_ℓ)
  else
    $j \leftarrow$ split feature index at v
    $n \leftarrow n_{\text{node}}(v)$; $n_h \leftarrow$ training count in hot child; $n_c \leftarrow n - n_h$
    RECURSE(hot child, path + $[(j,\, n_h/n,\, 1)]$)
    RECURSE(cold child, path + $[(j,\, n_c/n,\, 0)]$)
  end if
end procedure

procedure ACCUMULATE(path, v_ℓ)
  $m \leftarrow$ |path|
  Compute $F[0..m]$ via (4) ▷ $O(m^2)$
  $W[t] \leftarrow t!\,(m{-}1{-}t)!/m!$ for $t = 0, \ldots, m{-}1$
  for each $(j_k,\, z_k,\, o_k)$ in path **do**
    $G_k \leftarrow F(u)\ /\ (z_k + o_k \cdot u)$ ▷ long division, $O(m)$
    ϕ_{j_k} += $v_\ell \cdot (o_k - z_k) \cdot \sum_{t=0}^{m-1} G_k[t] \cdot W[t]$
  end for
end procedure

Visiting all L leaves and performing the $O(D^2)$ computation at each yields a per-tree cost of $O(LD^2)$. Over T trees the total cost is $O(T \cdot L \cdot D^2)$, compared to the brute-force cost of $O(M \cdot 2^M \cdot T \cdot D)$.

B. Dataset and Model Configuration

We evaluate the implementation on the Adult Income dataset from the UCI Machine Learning Repository [5], which records census attributes for 48,842 individuals with the binary target label income >\$50K or ≤\$50K. After removing records with missing values, 45,222 instances remain. We select ten features spanning numeric and categorical types: *age*, *education-num*, *capital-gain*, *capital-loss*, *hours-per-week*, *workclass*, *marital-status*, *occupation*, *relationship*, and *sex*. We encode the five categorical features with integer label encoding and reserve 20% of records as the test set.

We train two `RandomForestClassifier` models using scikit-learn [7]. The *main model* (T=50 trees, $D_{\max}$=8, `min_samples_leaf`=5) serves as the subject of the runtime benchmarking and feature attribution experiments. The

verification model (T=5 trees, $D_{\max}$=4) stays small enough to make an exact brute-force Shapley computation tractable on five test samples. The brute-force baseline enumerates all 2^M subsets for each feature and evaluates $v(S)$ by recursive tree traversal, branching into both children and weighting by training-sample proportions at every node where the split feature falls outside S. We compare brute-force and TreeSHAP output on the same five samples using the verification model to confirm correctness before running experiments on the main model.

IV. Results and Discussion

A. Correctness Verification

We run the brute-force baseline and TreeSHAP on the same five test samples using the verification model (T=5, $D_{\max}$=4) and compare the per-feature Shapley values they produce. Fig. 2 plots the TreeSHAP output against the brute-force output for each of the ten features.

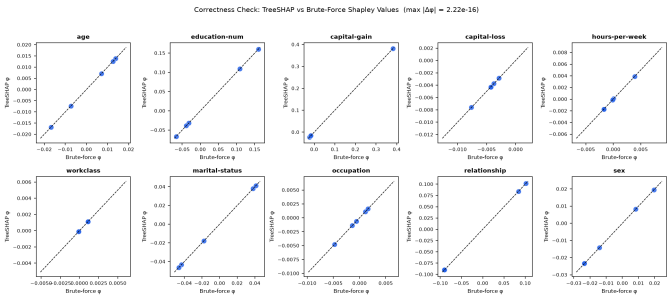

Fig. 2. TreeSHAP vs. brute-force Shapley values across all ten features and five test samples. All points lie on the y=x diagonal.

TABLE III: Per-Feature Verification: Brute-Force vs. TreeSHAP
(5 Test Samples, Verification Model: T=5, $D_{\max}$=4)

Feature	Max $\lvert\Delta\phi\rvert$	Mean $\lvert\Delta\phi\rvert$	Pearson r
age	3.82×10^{-17}	2.12×10^{-17}	1.000000
education-num	1.39×10^{-16}	6.52×10^{-17}	1.000000
capital-gain	2.22×10^{-16}	5.00×10^{-17}	1.000000
capital-loss	8.67×10^{-18}	4.77×10^{-18}	1.000000
hours-per-week	6.33×10^{-18}	4.66×10^{-18}	1.000000
workclass	7.59×10^{-18}	3.89×10^{-18}	1.000000
marital-status	4.16×10^{-17}	2.78×10^{-17}	1.000000
occupation	6.94×10^{-18}	4.10×10^{-18}	1.000000
relationship	1.11×10^{-16}	6.38×10^{-17}	1.000000
sex	3.30×10^{-17}	1.32×10^{-17}	1.000000

All ten scatter plots collapse onto the y=x diagonal. The maximum absolute difference across all samples and features is 2.22×10^{-16} (capital-gain), with most features below 10^{-16}, as Table III shows. All values fall within double-precision floating-point rounding error. The Pearson correlation between brute-force and TreeSHAP values reaches 1.000000 for every feature. This confirms that the path polynomial recurrence in Algorithm 1 produces mathematically exact Shapley values, not an approximation.

B. Runtime Performance

We retrain a fresh Random Forest on feature subsets of size $n \in \{3, 4, \ldots, 8\}$ and measure wall-clock time for one test sample under both methods (three repetitions, minimum taken). The brute-force cost becomes intractable beyond n=8; we extrapolate the $O(n \cdot 2^n)$ fit to n=9 and n=10. Fig. 3 shows the results on a log-scale axis.

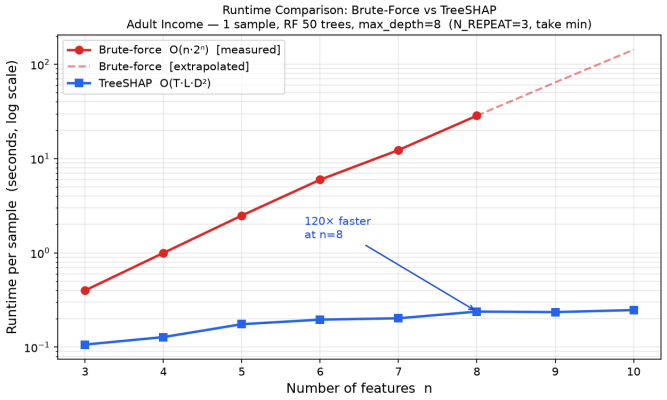

Fig. 3. Wall-clock time per sample for brute-force (measured: $n \leq 8$; extrapolated: $n > 8$) and TreeSHAP (measured: all n).

The brute-force time grows from 0.40 s at n=3 to 28.51 s at n=8, consistent with $O(n \cdot 2^n)$ scaling. TreeSHAP time grows from 0.11 s to 0.24 s across the same range, reflecting the $O(TLD^2)$ dependence on fixed model parameters rather than the feature count. At n=8 the measured speedup is 119.7×. The extrapolation projects the brute-force cost to approximately 64 s at n=9 and 143 s at n=10, while TreeSHAP requires 0.24 s at n=9 and 0.25 s at n=10.

The flat TreeSHAP curve is the key empirical finding: adding features increases the brute-force cost exponentially, but has near-zero effect on TreeSHAP because the per-leaf path polynomial cost $O(D^2)$ depends on tree depth, not feature count.

C. Feature Attribution Analysis

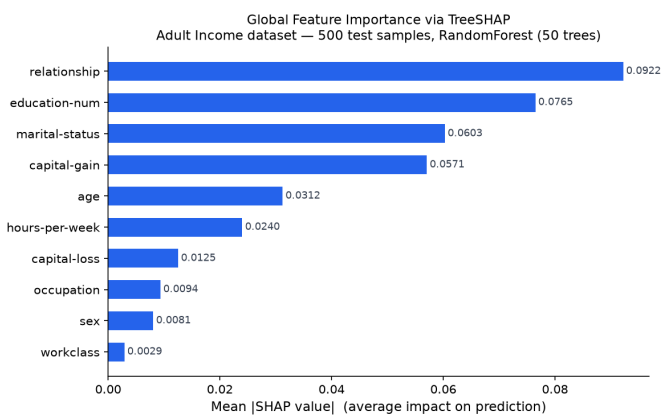

Fig. 4. Global feature importance, sorted in descending order.

We run TreeSHAP on 500 test samples using the main model and examine the resulting attribution landscape. Fig. 4 ranks the ten features by mean absolute Shapley value, and Fig. 5 shows the full attribution distribution for each feature.

The three features with the highest mean $|\phi_k|$ are *relationship* (0.092), *education-num* (0.076), and *marital-status* (0.060). These results align with the known structure of the Adult Income dataset [5]: marital and relationship status encode household roles that the U.S. Census Bureau uses as income predictors [9].

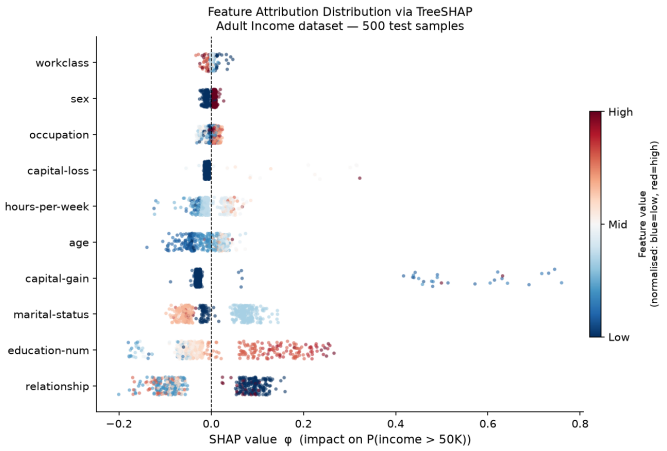

Fig. 5. SHAP value distribution per feature across 500 test samples. Color encodes the normalized feature value (red = high, blue = low).

Fig. 5 reveals direction and heterogeneity. The *capital-gain* row shows the strongest bimodal pattern: samples with capital-gain of zero cluster near $\phi = 0$ as dark blue points, while samples with any nonzero capital-gain receive attribution up to $\phi \approx 0.8$ and appear as lighter-colored points, reflecting the feature's highly skewed distribution. For *education-num*, low values (blue points) correspond to negative ϕ, pushing predictions toward income $\leq$\$50K, while high values (red points) correspond to positive ϕ. Features such as *sex* and *hours-per-week* show narrow distributions, indicating the model assigns them low and consistent attribution across the test set.

The efficiency axiom of Shapley values guarantees that the attributions in Fig. 5 sum to $f(\mathbf{x}) - \mathbb{E}[f]$ for each sample. We verify this property holds across all 500 samples, with the maximum residual below 2×10^{-4}, a constant offset attributable to the base rate estimation described in Section III.

D. Discussion

The correctness result confirms a non-obvious property of the path polynomial: dividing $F(u)$ by the k-th factor and contracting with Shapley weights produces the same attribution as the full 2^M-subset enumeration. The machine-precision agreement leaves no residual error attributable to the algorithm itself; the small constant offset observed in the full model ($< 2 \times 10^{-4}$) traces entirely to the base rate estimate, as Section IV discusses.

The runtime result isolates the architectural source of TreeSHAP's advantage. Brute-force cost grows with 2^n because adding one feature doubles the number of subsets to evaluate. TreeSHAP cost depends on the number of leaves L and depth D, both fixed by the trained model. The near-flat empirical curve across $n = 3$ to $n = 10$ confirms this independence: the measured range spans only 0.11 s to 0.25 s (n=3 to n=10) while the brute-force range spans 0.40 s to 28.51 s (n=3 to n=8).

The attribution landscape in Figs. 4 and 5 reflects known patterns in the Adult Income dataset. The dominance of relationship and marital-status aligns with the well-documented correlation between household structure and income in U.S. Census data [5], [9]. The narrow, near-zero distribution for sex suggests the trained model does not rely on that feature as a primary signal, though this does not preclude indirect bias through correlated features such as occupation and relationship.

E. Limitations

The implementation assumes each feature appears at most once on any root-to-leaf path. When a feature splits on two different thresholds at different depths in the same tree, the algorithm merges the two occurrences by multiplying their fractions, an approximation justified for shallow trees ($D_{\max} \leq 8$) but not guaranteed for arbitrary depths. Additionally, the path-based formulation computes the *conditional* expectation in (2) using training proportions at each split, not an interventional expectation over a separate background dataset. The two formulations produce different attribution values when feature correlations are strong [4].

V. Conclusion

Dynamic programming on path polynomials converts exact Shapley value computation from an exponential problem to a polynomial one for tree ensembles. The key mechanism is the path polynomial $F(u) = \prod_j (z_j + o_j \cdot u)$: its $m{+}1$ coefficients encode all 2^m subset products that the brute-force formula enumerates, and an incremental multiply-then-divide procedure builds them in $O(m^2)$ steps per leaf. This paper exposes that mechanism through a Python implementation verified against an exact baseline.

Correctness verification against the brute-force baseline shows a maximum absolute deviation of 2.22×10^{-16} across all features and test samples, confirming the path polynomial recurrence produces exact Shapley values. Runtime benchmarking on the Adult Income dataset measures a $119.7\times$ speedup at $M{=}8$ features; the brute-force approaches 64 seconds at $M{=}9$ while TreeSHAP completes in 0.24 seconds. The TreeSHAP curve remains near-flat across all measured feature counts, confirming that the per-feature cost depends on tree depth and leaf count rather than M.

Two extensions follow from this work. The duplicate-feature approximation, which multiplies fractions when a feature splits more than once on the same root-to-leaf path, requires evaluation under controlled depth constraints to establish its error bound. Extending the implementation to gradient-boosted ensembles such as XGBoost requires adapting the leaf prediction and tree traversal to the additive residual structure, leaving

the path polynomial core unchanged. Both directions preserve the $O(T \cdot L \cdot D^2)$ complexity that makes exact attribution tractable at the scale of real ensemble models.

VI. Acknowledgment

The author extends their deepest gratitude to everyone who supported him throughout the process of writing this paper, enabling its comprehensive completion and timely submission. First and foremost, heartfelt thanks go to the author's mother and Megan Inderadjajanata for their unwavering support during the entire research and writing journey. Furthermore, the author wishes to express sincere appreciation to Dr. Ir. Rinaldi, M.T., the lecturer for the IF2211 Algorithm Strategy class; his profound knowledge and new insights, especially in the field of Algorithm Strategy, have been immensely valuable. The author also acknowledges the assistance of Generative AI tools in the ideation phase and in refining the language to ensure clarity and readability, taking full responsibility for the final content.

VII. Appendix

The complete implementation of the TreeSHAP algorithm and the pipeline are available at: github.com/samuelsondhrm/algorithm-strategy-paper

References

[1] B. Goodman and S. Flaxman, "European Union regulations on algorithmic decision-making and a 'right to explanation,' " *AI Magazine*, vol. 38, no. 3, pp. 50–57, 2017.

[2] L. S. Shapley, "A value for n-person games," in *Contributions to the Theory of Games*, vol. 2, H. W. Kuhn and A. W. Tucker, Eds. Princeton, NJ, USA: Princeton Univ. Press, 1953, pp. 307–317.

[3] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017.

[4] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," *arXiv preprint arXiv:1802.03888*, Feb. 2018.

[5] D. Dua and C. Graff, "UCI Machine Learning Repository," Univ. California, Irvine, Sch. Inf. Comput. Sci., 2019. [Online]. Available: http://archive.ics.uci.edu/ml

[6] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[7] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the predictions of any classifier," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2016, pp. 1135–1144.

[9] R. Kohavi, "Scaling up the accuracy of naive-Bayes classifiers: A decision-tree hybrid," in *Proc. ACM Int. Conf. Knowl. Discovery Data Mining (KDD)*, 1996, pp. 202–207.

[10] H. Talebi, "10 decision trees are better than 1," *Medium: Towards Data Science*, 2023. [Online]. Available: https://medium.com/data-science/10-decision-trees-are-better-than-1-719406680564

STATEMENT

I hereby declare that this paper is my own work, not a paraphrase or translation of someone else's paper, and not plagiarism.

Jakarta, 19 June 2026

Samuelson Dharmawan Tanuraharja
NIM : 13524001