QUADRATURE-TREESHAP: FAST AND STABLE SHAPLEY VALUES AND INTERACTIONS AT ANY DEPTH

Anonymous authors
Paper under double-blind review

ABSTRACT

Shapley values are a standard tool for explaining predictions of tree ensembles, with Path-Dependent SHAP being the most widely used variant. Despite substantial progress, existing methods still exhibit trade-offs between depth-dependent runtime, numerical stability, and support for higher-order interactions. To address these challenges, we introduce Quadrature-TreeSHAP, a quadrature-based reformulation of Path-Dependent TreeSHAP that is numerically stable, naturally extends to any-order Shapley interaction values and is practically insensitive to tree depth. Our implementation supports both CPU and GPU and is integrated into XGBoost. Our method is based on a weighted-Banzhaf interaction polynomial, which expresses Banzhaf interaction values as expectations under a feature participation probability p. Shapley values and any-order interaction values are then recovered by integrating these polynomials over p from 0 to 1. We evaluate these integrals using Gauss–Legendre quadrature, and show that, in practice, only 8 fixed quadrature points are sufficient to reach machine precision. In fact, Quadrature-TreeSHAP with 8 fixed points achieves greater numerical stability than TreeSHAP. This fixed-point formulation removes depth dependence from the inner computation and enables efficient SIMD execution. We confirm these advantages empirically. On 12 XGBoost benchmarks, Quadrature-TreeSHAP computes Shapley values $1.06\times$–$10.59\times$ faster than TreeSHAP on CPU and $1.84\times$–$6.95\times$ faster than GPUTreeSHAP on GPU. Shapley pairwise interactions are $3.80\times$–$58.11\times$ faster on CPU, with higher-order interactions achieving speedups of up to $1200\times$ compared to TreeSHAP-IQ.

1 INTRODUCTION

Decision tree ensembles such as XGBoost (Chen & Guestrin, 2016), Random Forests (Breiman, 2001), and CatBoost (Dorogush et al., 2017) are widely used for classification and regression. As their use has grown, so has the need to explain their predictions. SHAP (Lundberg & Lee, 2017) addresses this by framing explanation as a cooperative game and assigning each feature a Shapley value (Shapley, 1953). While computing Shapley values on certain games and models is #P-hard (Matsui & Matsui, 2001; Huang & Marques-Silva, 2024; Van den Broeck et al., 2022), the additivity structure of decision trees allows their efficient computation (Lundberg et al., 2020). Beyond individual attributions, SHAP also extends to Shapley interaction values, which capture pairwise feature interactions (Grabisch & Roubens, 1999; Dhamdhere et al., 2020; Fumagalli et al., 2023).

SHAP on decision trees has two commonly used variants. Background SHAP (Laberge & Pequignot, 2022) integrates over an explicit background dataset and can be computationally expensive, while Path-Dependent SHAP (Lundberg et al., 2020) efficiently estimates the effect of missing features using the training statistics stored along the tree. Although Path-Dependent SHAP can be inconsistent, providing different explanations for equivalent decision trees (as shown in (Liu et al., 2025a; Filom et al., 2024)), its efficient computation makes it the most widely used approach in practice.

TreeSHAP (Lundberg et al., 2020) is the first proposed algorithm for Path-Dependent SHAP and GPUTreeSHAP (Mitchell et al., 2022) provides a GPU implementation of the approach. Both run in $O(MTLD^2)$ time, where M is the number of samples, T is the number of trees, L the number of leaves, and D the maximum tree depth. FastTreeSHAP v2 (Yang, 2022) and Woodelf (Nadel & Wettenstein, 2026) reduce the complexity to $O(MTLD)$ time, but introduced a preprocessing

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

step that is independent of M but exponential in D, making them impractical for high depths trees. FastTreeSHAP v1 (Yang, 2022) has the same complexity as TreeSHAP and reduces constant factors.

Linear TreeSHAP (Yu et al., 2022) recast the computation as polynomial arithmetic, achieving $O(MTND)$ time (N = nodes), but suffers from numerical instability (see (Li et al., 2026), Section 5 and (Jiang et al., 2025), Section 3.3). This polynomial view is further examined in (Karczmarz et al., 2022), which compares Shapley and Banzhaf attributions and studies their computational and theoretical trade-offs for tree models. TreeGrad-Shap (Li et al., 2026) derives a gradient-based formulation via the multilinear extension, achieving $O(MTN\lceil d/2\rceil)$, where d is the maximal number of unique features along a root-to-leaf path ($d \leq D$), and having no numerical instability. QuadraSHAP (Mohammadi et al., 2026) applies this gradient-based formulation to product-kernel methods (e.g., SVMs and kernel ridge regressors with RBF or polynomial kernels), and provides parallelization with $O(\log d)$ parallel time.

TreeGrad handles only first-order feature attributions. TreeSHAP-IQ (Muschalik et al., 2024b;a) extends Linear TreeSHAP polynomial arithmetic to any order interactions. Its complexity, when computing order-s interaction on a dataset with F features is $O(MTLD\binom{F-1}{s-1})$. In this paper, we provide an any-order Shapley interaction algorithm based on a gradient formulation that significantly improves the complexity of TreeSHAP-IQ.

Our contributions are as follows:

1. We present an efficient method for computing any-order Shapley interaction values, improving the state of the art complexity of TreeSHAP-IQ from $O(MTLD\binom{F-1}{s-1})$ to $O(MTNd^s)$. This yields substantial gains in the highly common case when the number of features exceeds the tree depth ($F > D \geq d$).
2. Our approach computes Shapley values as integrals of a weighted Banzhaf polynomial, evaluated efficiently using Gauss-Legendre quadrature. We provide both theoretical and empirical analysis of the number of quadrature points required. We show that in practice, even in very high depths, 8 Gauss–Legendre points are enough to reach the machine precision. Using a constant number of quadrature points removes the dependence on d in practice, reducing the effective complexity from $O(MTN\lceil d/2\rceil)$ to $O(MTN)$. More generally, for order-s interactions, it reduces from $O(MTNd^s)$ to $O(MTNd^{s-1})$.
3. We show how to exploit SIMD efficiently in this fixed-point formulation, and implement the resulting method in C++. Additionally, we give the first GPU implementation of this quadrature-based idea in XGBoost. Our implementation (CPU and GPU) achieves $1.06\times$–$10.59\times$ speedups over TreeSHAP and $1.84\times$–$6.95\times$ speedups over GPUTreeSHAP.
4. The method is open-sourced, replacing the TreeSHAP/GPUTreeSHAP backend in the XGBoost library.

Together, these contributions yield a numerically stable, hardware-efficient, GPU-friendly, and interaction-aware reformulation of Path-Dependent SHAP, improving both theory and practice.

2 Polynomial Construction for TreeSHAP

We first recall the polynomial view of TreeSHAP introduced by Linear TreeSHAP.

Consider a decision tree T with internal nodes and leaves. For a sample x, the root-to-leaf path to leaf v, marked as $path(v)$, passes through edges $e_1, \ldots, e_k$. Each edge e has a *split feature* $f(e)$ and an *edge weight* $w_e = n_{\text{child}}/n_{\text{parent}}$, the proportion of training samples flowing through that edge.

Define the *empty prediction* for leaf v with *leaf value* $val(v)$ as:

$$R^v_\emptyset = \text{val}(v) \cdot \prod_{e \in \text{path}(v)} w_e \tag{1}$$

For each feature i on the path, the *marginal multiplier* is:

$$q^v_i = \begin{cases} \prod_{e:\, f(e)=i} 1/w_e & \text{if } x \text{ satisfies all splits on } i, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

A feature not on the path has $q_i = 1$ and is called a *null player*. Including it in any coalition does not change the prediction. Using the above notation, the Path-Dependent game $f_v : 2^{[F]} \to \mathbb{R}$ associated with a leaf v, defined over all feature subsets $S \subseteq [F]$, can be written as (Yu et al., 2022):

$$f_v(S) = R_S^v = R_\emptyset^v \cdot \prod_{i \in S} q_i^v \tag{3}$$

The combinatorial information along a decision path can be encoded by the following summary polynomial (where $M(v)$ is the set of distinct features on the root-to-leaf path of v):

$$G_v(y) := R_\emptyset^v \prod_{j \in M(v)} (q_j^v + y) \tag{4}$$

Linear TreeSHAP (Yu et al., 2022) computes the Shapley contribution ϕ_i^v of feature $i \in M(v)$ by removing its factor and applying a weighted linear function ψ to the resulting polynomial. The ψ function is a weighted inner product over polynomial coefficients with a_s the coefficient of y^s:

$$\phi_i^v = (q_i^v - 1)\, \psi\left(\frac{G_v(y)}{q_i^v + y}, |M(v)| \right), \quad \psi(A, m) = \frac{1}{m} \sum_{s=0}^{\deg(A)} \frac{a_s}{\binom{m-1}{s}} \tag{5}$$

In this formulation, the Shapley value reduces to a weighted sum of polynomial coefficients, with weights given by the binomial terms in ψ. Linear TreeSHAP represents the polynomial in interpolation form and recovers ψ via a Vandermonde transform, while (Karczmarz et al., 2022) operates directly on coefficient vectors. TreeSHAP-IQ extends Formula 5 to higher-order interactions and derives a corresponding algorithm.

3 The Weighted Banzhaf Interaction Values Polynomial

Banzhaf values (Banzhaf, 1965), provide a classical alternative to Shapley values. Prior work has studied Banzhaf-based explanations for decision trees (Karczmarz et al., 2022; Muschalik et al., 2024b; Li et al., 2026; Nadel & Wettenstein, 2026), as well as other settings (Wang & Jia, 2023; Abramovich et al., 2023; Liu et al., 2025b). Formally, for a game $f_v : 2^{[F]} \to \mathbb{R}$, 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:

$$\beta_i(f_v) = \mathbb{E}[f_v(S) \mid i \in S] - \mathbb{E}[f_v(S) \mid i \notin S] \tag{6}$$

More generally, the uniform distribution can be replaced with a participation probability p, yielding the weighted Banzhaf value (Radzik et al., 1997), which admits several axiomatic characterizations (Nowak & Radzik, 2000). Extending this to feature subsets leads to any-order weighted Banzhaf interactions (Marichal & Mathonet, 2011):

Definition 1 (Any Order Weighted Banzhaf interaction values)**.** *For a set function $f_v : 2^{[F]} \to \mathbb{R}$ and a subset $S \subseteq [F]$, define the discrete derivative*

$$\Delta_S f_v(T) := \sum_{L \subseteq S} (-1)^{|S|-|L|} f_v(T \cup L)$$

The order-$|S|$ weighted Banzhaf interaction at inclusion probability $p \in [0, 1]$ is then

$$B_p^{(S)}(v) := \mathbb{E}_{T \sim p}[\Delta_S f_v(T)]$$

where each feature in $[F] \setminus S$ is included independently in T with probability p.

The case $|S| = 1$ recovers the standard weighted Banzhaf value, $|S| = 2$ corresponds to pairwise interactions, $|S| = 3$ captures interactions among three players, and so on. These higher-order objects are the natural Banzhaf-side analogue of Shapley interaction indices (Grabisch & Roubens, 1999; Fujimoto et al., 2006). Our key observation is that, for decision trees, these quantities also admit a simple polynomial form.

Theorem 1 (Leaf-level weighted Banzhaf polynomial)**.** *Let v be a leaf, let $M(v)$ be the set of features appearing on the path to v. For $S \subseteq [F]$, the Path-Dependent weighted Banzhaf interaction value of S, where each feature in $[F] \setminus S$ is independently included in T with probability p, is:*

$$B_{p,v}^{(S)} = \begin{cases} R_{\emptyset}^{v} \left(\prod_{j \in S} (q_j^v - 1) \right) \prod_{j \in M(v) \setminus S} \left((1-p) + p q_j^v \right) & S \subseteq M(v), \\ 0 & \textit{otherwise}. \end{cases} \tag{7}$$

Equivalently, defining $\alpha_j^v := q_j^v - 1$ and $H_v(p) := \prod_{j \in M(v)} (1 + \alpha_j^v p)$, we obtain:

$$B_{p,v}^{(S)} = \begin{cases} R_{\emptyset}^{v} \left(\prod_{j \in S} \alpha_j^v \right) \dfrac{H_v(p)}{\prod_{j \in S} (1 + \alpha_j^v p)} & S \subseteq M(v), \\ 0 & \textit{otherwise}. \end{cases} \tag{8}$$

Proof sketch. First, we derive a closed-form expression for the discrete derivative $\Delta_S f_v(T)$ by expanding the alternating sum and using the product structure of the leaf rule. We then take expectation over T under participation probability p, and use linearity of expectations and independence of feature inclusion to obtain the stated polynomial form. See full proof in Appendix C.

4 From Weighted Banzhaf Interactions to Shapley Interactions

The weighted Banzhaf family is especially useful because Shapley quantities can be recovered by integrating over the participation probability $p \in [0, 1]$. For first-order values, this follows from classical results on multilinear extensions and probabilistic values (Owen, 1972). More generally, this relation extends to higher-order interactions as part of the probabilistic interaction framework (Marichal & Mathonet, 2011). Let $\mathrm{SII}^{(S)}(f_v)$ denote the order-$|S|$ Shapley interaction value:

$$\mathrm{SII}^{(S)}(f_v) = \int_0^1 B_p^{(S)}(f_v)\, dp \tag{9}$$

For completeness, we prove equation 9 in Appendix E.

This observation is algorithmically important because the integral over $[0, 1]$ can be computed very efficiently with Gauss–Legendre quadrature. If $B_p^{(S)}$ is a polynomial in p of degree at most D, then the corresponding Shapley quantity can be computed exactly as a weighted sum of weighted Banzhaf interaction values evaluated at fixed quadrature nodes. Let $\{(t_m, w_m)\}_{m=1}^n$ be the Gauss–Legendre nodes and weights on $[0, 1]$ (exact when $2n - 1 \geq D$). Applying this at the leaf level using the weighted Banzhaf polynomial (equation 8) yields:

$$\mathrm{SII}_v^{(S)} = \int_0^1 B_{p,v}^{(S)}\, dp \approx \sum_{m=1}^{n} w_m\, B_{t_m,v}^{(S)} = R_{\emptyset}^{v} \left(\prod_{j \in S} \alpha_j^v \right) \sum_{m=1}^{n} w_m \frac{H_v(t_m)}{\prod_{j \in S} (1 + \alpha_j^v t_m)} \tag{10}$$

TreeGrad (Li et al., 2026) was the first tree-explanation method to adopt this viewpoint. It operates on the gradients of the multilinear extension, which correspond to first-order weighted Banzhaf values. In fact, Lemma 1 in TreeGrad is the first-order special case of our weighted Banzahf polynomial in Theorem 1.

Our weighted-Banzhaf polynomial extends directly to arbitrary interaction orders, enabling the same quadrature framework to compute both Shapley values and higher-order interactions. Moreover, this representation shows that the tree-induced integrand is a low-degree polynomial (see equation 7), explaining why quadrature achieves high accuracy in both our method and TreeGrad.

5 Our algorithm

Our algorithm evaluates equation 10 for all order-s interactions and all leaves simultaneously by a single depth-first traversal and sums the resulting leaf contributions on the fly. The DFS propagates,

Algorithm 1: The Quadrature-TreeSHAP algorithm for order-s interactions

Input: Tree T, sample x, interaction order s, fixed Gauss–Legendre nodes $\{t_m, w_m\}_{m=1}^{n}$
Output: All order-s Shapley interaction values $\Phi[S]$ for $S \subseteq [F]$ with $|S| = s$
$\Phi[\cdot] \leftarrow 0$; $p[\cdot] \leftarrow$ UNSEEN; $A \leftarrow \emptyset$;
Function DFS (v, $\mathbf{c}$, w_{prod}) **:**
  if v *is a leaf* **then**
    return $\mathbf{c} \cdot \text{val}(v) \cdot w_{\text{prod}}$;
  Let $f \leftarrow \text{feature}(v)$, children l, r;
  foreach *child* $u \in \{l, r\}$ **do**
    $w_e \leftarrow n_u / n_v$; $s_e \leftarrow [x$ satisfies edge $v \to u]$;
    if $p[f] =$ *UNSEEN* **then**
      $p_e \leftarrow 1/w_e$ if s_e, else 0; $p_\uparrow \leftarrow 1$;
    else if $p[f] = 0$ **then**
      $p_e \leftarrow 0$; $p_\uparrow \leftarrow 0$;
    else
      $p_e \leftarrow p[f]/w_e$ if s_e, else 0; $p_\uparrow \leftarrow p[f]$;
    $\alpha_e \leftarrow p_e - 1$;
    $\mathbf{c}' \leftarrow \mathbf{c} \odot (1 + \alpha_e \, \mathbf{t})$;
    if $p[f] \neq$ *UNSEEN* ***and*** $|p[f] - 1| > \epsilon$ **then**
      $\mathbf{c}' \leftarrow \mathbf{c}' \oslash (1 + (p[f] - 1) \, \mathbf{t})$;
    Save $p[f]$ and whether $f \in A$; set $p[f] \leftarrow p_e$; add f to A;
    $\mathbf{H}_u \leftarrow$ DFS (u, $\mathbf{c}'$, $w_{\text{prod}} \cdot w_e$) ;
    $\boldsymbol{\delta}_e \leftarrow \frac{\alpha_e}{1+\alpha_e \mathbf{t}} - \frac{p_\uparrow - 1}{1+(p_\uparrow - 1)\mathbf{t}}$;
    foreach $(s-1)$-*subset* $P \subseteq A \setminus \{f\}$ **do**
      $\boldsymbol{\gamma}_P \leftarrow \prod_{j \in P} \frac{p[j] - 1}{1 + (p[j] - 1)\mathbf{t}}$;
      $\Phi[P \cup \{f\}] \mathrel{+}= \sum_{m=1}^{n} w_m \, \mathbf{H}_u[m] \, \boldsymbol{\delta}_e[m] \, \boldsymbol{\gamma}_P[m]$;
    Restore $p[f]$; if f was not previously active, remove f from A;
  return $\mathbf{H}_l + \mathbf{H}_r$;
DFS (*root*, $\mathbf{1}_n$, 1.0) ;

for each node, the values of the path polynomial at a fixed set of Gauss–Legendre nodes and merges child subtrees by pointwise addition. In practice we use a fixed set of 8 Gauss–Legendre nodes on $[0, 1]$ and propagate all eight traces during the traversal. Sect. 5.1 and 6 explain why 8 points suffice.

For repeated features, we use the telescoping idea originally proposed by Linear TreeSHAP, but apply it to arbitrary interaction order. Let e be a split on feature i with head $h(e)$, let p_e be the accumulated multiplier at edge e, let $p_{e\uparrow}$ be the value at the closest ancestor edge on the same feature, and let p_j denote the current live multiplier of feature j on the active path. Then for any set S of size $s - 1$ drawn from the active path and not containing i, the child subtree contributes:

$$SII^{(S \cup \{i\})} \mathrel{+}= \sum_{m=1}^{n} w_m \, H_{h(e)}(t_m) \left(\frac{p_e - 1}{1 + (p_e - 1)t_m} - \frac{p_{e\uparrow} - 1}{1 + (p_{e\uparrow} - 1)t_m} \right) \prod_{j \in S} \frac{p_j - 1}{1 + (p_j - 1)t_m} \tag{11}$$

When $S = \emptyset$, equation 11 reduces to the first-order update. The pseudocode for the general order-s algorithm is provided in Alg 1.

In the first-order case, our algorithm is algebraically equivalent to TreeGrad-Shap (Li et al., 2026) (specifically, TreeGrad's vectorized Algorithm 2). The difference lies not in the computation itself, but in the derivation and scope: TreeGrad derives the method via gradients of the multilinear extension and is limited to first-order values, whereas our approach is based on the weighted-Banzhaf polynomial and reveals that this is simply the first-order case of a more general interaction-values framework.

5.1 Quadrature Point Requirements

We established that Shapley values and interactions can be computed using Gauss–Legendre quadrature. We now ask how many quadrature points are needed, both theoretically and in practice.

Theoretical analysis For first-order values, TreeGrad shows that Gauss-Legendre quadrature with $\lceil D/2 \rceil$ points suffices. Their Shapley algorithm uses the tighter bound $\lceil \min(D, F)/2 \rceil$, where F is the number of features. This can be further tightened to $\lceil d/2 \rceil$, where d is the maximum number of unique features along a root-to-leaf path.

We extend this bound to arbitrary interaction orders, showing that the order-s Shapley interaction integrand has lower degree than the first-order one. For example, if $d = 10$, then first-order values require 5 points, while third-order interactions require only $\lceil (10 - 3 + 1)/2 \rceil = 4$ points.

Theorem 2 (Decreasing degree for order-s interactions)**.** *Let $S \subseteq [F]$ with $|S| = s$. The integrand for the order-s Shapley interaction value $\mathrm{SII}(S)$ is a polynomial of degree at most $d - s$. Therefore, Gauss–Legendre quadrature is exact whenever the number of points satisfies $n \geq \lceil (d - s + 1)/2 \rceil$.*

Proof sketch. For a fixed interaction set, each leaf contribution is obtained by canceling the linear factors associated with the interacting features, so every requested interaction lowers the degree by one (see equation 8). Since a root-to-leaf path can involve at most d distinct features, after cancellation each leaf term has degree at most $d - s$, and summing over leaves does not increase that bound. Gauss–Legendre exactness then yields the result. See full proof in Appendix D.

Appendix B bounds the error when fewer quadrature points are used than required for exactness. For a single leaf of depth at most 50 and value 1, the first-order SHAP error is at most 4.73165×10^{-4} with 8 points and 2.14006×10^{-11} with 16 points. Table 8 reports the corresponding bounds for additional depths and number of quadrature points.

Practical analysis In practice, we do not need accuracy beyond the floating-point noise floor. For IEEE 754 single precision (float32), which is the output format returned here and in TreeSHAP, the unit roundoff is $u = 2^{-24} \approx 5.96 \times 10^{-8}$, so improvements far below roughly 10^{-7} are no longer reliably observable in the final `float32` result (Goldberg, 1991; Higham, 2002). Once the quadrature error is below this scale, using more quadrature points brings no practical benefit.

Gauss–Legendre quadrature converges rapidly for smooth integrands and is exact for polynomials of degree at most $2n - 1$ (Davis & Rabinowitz, 1984; Golub & Welsch, 1969). Our integrands inherit this favorable polynomial structure from the tree paths. Empirically, we find that 8 quadrature points are sufficient. Our experiments consistently reach `float32`'s numerical noise floor with 8 points. This is somewhat surprising, since Theorem 2 gives a higher worst-case requirement for exactness and Appendix B gives error bounds above this noise floor. Appendix A.2 shows similar behavior at higher precision, with 16 points approaching the noise floor of `float64`.

5.2 SIMD and GPU Friendly approach

SIMD (Single Instruction, Multiple Data) (Hughes, 2015) accelerates computation by applying a single instruction to multiple values packed into a wide register. Prior work has leveraged SIMD to accelerate decision tree inference and related tasks (Lucchese et al., 2016; Ye et al., 2018). GPUs extend this idea through SIMT (Single Instruction, Multiple Threads), where lanes are grouped into warps and execute the same instruction in parallel (NVIDIA, 2026). GPUTreeSHAP (Mitchell et al., 2022) is designed around this execution model, decomposing computation into root-to-leaf path subproblems and bin-packing similarly sized tasks into warps to improve lane utilization, at the cost of additional preprocessing and scheduling overhead.

Our quadrature formulation is well suited to SIMD and SIMT. Because the computation reduces to evaluating the same fixed set of 8 quadrature nodes for every path, the arithmetic is regular and fully utilizes vector lanes. In our implementation, we vectorize 4 points in parallel on CPU (using SSE instructions[1]) and 32 points in parallel on GPU (the size of a warp). This fixed-width structure also simplifies the GPU algorithm, eliminating the binning strategy used by GPUTreeSHAP and yielding significant speedups, as shown in Sect. 6.

[1]Longer 8 word vector instructions are not used in XGBoost due to portability.

Table 1: Benchmark XGBoost ensembles. Small and large models use breadth-first growth, while sparse models use depth-first growth with a 512-leaf cap. Here D denotes realized maximum depth and d denotes realized maximum unique-feature depth.

Dataset	Setting	Trees	Growth	Target	D	d	Mean leaves/tree
Adult	small	10	breadth-first	max depth 6	6	6	40.1
Adult	large	1000	breadth-first	max depth 16	16	14	613.3
Adult	sparse	100	depth-first	512 leaves	28	13	512.0
CalHousing	small	10	breadth-first	max depth 6	6	5	64.0
CalHousing	large	1000	breadth-first	max depth 16	16	8	3702.8
CalHousing	sparse	100	depth-first	512 leaves	20	8	512.0
CovType	small	80	breadth-first	max depth 6	6	6	4.9
CovType	large	8000	breadth-first	max depth 16	16	14	2.4
CovType	sparse	800	depth-first	512 leaves	24	18	4.0
Fashion-MNIST	small	100	breadth-first	max depth 6	6	6	57.5
Fashion-MNIST	large	10000	breadth-first	max depth 16	16	16	295.7
Fashion-MNIST	sparse	1000	depth-first	512 leaves	52	48	412.4

6 Experimental Results

We implemented Quadrature-TreeSHAP in C++ and on GPU for first- and second-order interactions, and in Python for higher-order cases. We evaluate Quadrature-TreeSHAP on 12 XGBoost benchmark ensembles spanning four datasets and three regimes per dataset (small, large, and sparse). The datasets are Adult (UCI Machine Learning Repository, 1996) (48,842 rows, 14 features), CalHousing (Pace & Barry, 1997) (20,640 rows, 8 features), CovType (Blackard & Dean, 1999) (581,012 rows, 54 features), and Fashion-MNIST (Xiao et al., 2017) (70,000 rows, 784 features). Table 1 summarizes the benchmark models.

The first-order benchmark explains 1000 rows and reports standard Shapley values, while the second-order benchmark explains 100 rows and reports pairwise Shapley interaction values. These two benchmarks compare the XGBoost implementations of TreeSHAP, GPUTreeSHAP, and Quadrature-TreeSHAP. The higher-order Python experiments reported below instead use a single explained row and compare Python implementations of TreeSHAP-IQ and Quadrature-TreeSHAP: Table 4 reports third-order interaction runtimes, and Figure 2 shows CovType-sparse scaling in orders 2 to 6.

Appendix A.1 additionally reports a direct first-order Python comparison against TreeGrad-Shap, but we compare primarily against TreeSHAP in the main text because the XGBoost/`shap` implementation is, in practice, the fastest available. All runtimes below are wall-clock times; missing entries are marked as "–" and denote either runs that exceeded the 600-second timeout or, for GPUTreeSHAP, models with depth greater than 32. The benchmarks were run on a machine with an Intel Xeon E5-2698 v4 @ 2.20GHz CPU and an NVIDIA Tesla V100-SXM2-32GB GPU.

First-order TreeSHAP Table 2 reports first-order runtimes. On CPU, Quadrature-TreeSHAP is faster than TreeSHAP on all 12 benchmarks, with speedups ranging from $1.06\times$ to $10.59\times$ and a median speedup of $3.84\times$. The largest first-order CPU gains appear on the Fashion-MNIST large and sparse ensembles, while even the smallest gain on CalHousing-small remains slightly favorable. On GPU, Quadrature-TreeSHAP outperforms GPUTreeSHAP on every benchmark with a reported baseline, with speedups between $1.84\times$ and $6.95\times$.

Second-order TreeSHAP Table 3 reports pairwise interaction runtimes. The CPU advantage widens substantially for second-order explanations: among the 10 benchmarks with a reported TreeSHAP baseline, speedups range from $3.80\times$ to $58.11\times$, with a median speedup of $12.86\times$. The largest CPU gains occur on the CovType large and sparse models, with consistently strong gains across the Adult benchmarks, where the fixed-size quadrature computation avoids the rapid growth in interaction overhead. On GPU, Quadrature-TreeSHAP is faster on 8 of the 11 benchmarks

Table 2: First-order runtimes on 1000 explained rows. Times are wall-clock seconds; speedups are relative to TreeSHAP or GPUTreeSHAP.

		CPU			GPU		
Dataset	Setting	TreeSHAP	Q-TreeSHAP	Speedup	GPUTreeSHAP	Q-TreeSHAP	Speedup
Adult	small	0.004	0.002	2.26	0.009	0.001	6.82
Adult	large	7.842	1.308	6.00	1.615	0.602	2.68
Adult	sparse	0.609	0.108	5.64	0.128	0.051	2.51
CalHousing	small	0.003	0.003	1.06	0.008	0.001	6.95
CalHousing	large	29.034	7.681	3.78	8.390	2.705	3.10
CalHousing	sparse	0.341	0.103	3.33	0.085	0.035	2.47
CovType	small	0.004	0.002	1.87	0.008	0.001	6.00
CovType	large	0.178	0.031	5.73	0.036	0.019	1.84
CovType	sparse	0.036	0.009	3.90	0.019	0.004	5.55
Fashion-MNIST	small	0.046	0.021	2.16	0.023	0.012	2.01
Fashion-MNIST	large	51.800	5.639	9.19	14.140	2.431	5.82
Fashion-MNIST	sparse	8.299	0.783	10.59	–	0.372	–

Table 3: Second-order runtimes on 100 explained rows. Times are wall-clock seconds; speedups are relative to TreeSHAP or GPUTreeSHAP.

		CPU			GPU		
Dataset	Setting	TreeSHAP	Q-TreeSHAP	Speedup	GPUTreeSHAP	Q-TreeSHAP	Speedup
Adult	small	0.190	0.010	19.68	0.008	0.001	6.29
Adult	large	32.542	2.415	13.48	3.057	0.460	6.65
Adult	sparse	2.643	0.207	12.77	0.216	0.038	5.71
CalHousing	small	0.018	0.004	4.67	0.007	0.001	6.51
CalHousing	large	59.608	15.707	3.80	9.611	2.609	3.68
CalHousing	sparse	0.903	0.211	4.28	0.096	0.033	2.93
CovType	small	0.059	0.005	12.95	0.012	0.013	0.89
CovType	large	2.838	0.049	58.11	0.068	0.159	0.43
CovType	sparse	0.686	0.013	51.21	0.041	0.027	1.48
Fashion-MNIST	small	15.549	2.052	7.58	1.764	3.975	0.44
Fashion-MNIST	large	–	12.923	–	35.208	7.161	4.92
Fashion-MNIST	sparse	–	3.355	–	–	4.514	–

with reported measurements and reaches $6.65\times$ speedup, although several workloads still favor GPUTreeSHAP, notably CovType-small, CovType-large, and Fashion-MNIST-small.

Numerical stability Figure 1 examines the most numerically challenging benchmark in Table 1, Fashion-MNIST-sparse. We gradually increase the maximum depth of the learned trees and measure the resulting maximum absolute SHAP error for first-order Shapley values, as compared to the $\lceil D/2 \rceil$ point reference computed in float64. TreeSHAP becomes unstable beyond depth 32, whereas Quadrature-TreeSHAP remains remarkably stable. Even 6 quadrature points provide good accuracy, with no practical improvement beyond 8, underscoring the effectiveness of Gauss–Legendre quadrature for these smooth integrals. Although these experiments focus on first-order attributions, higher-order interactions are numerically easier in our framework: Theorem 2 shows that they require no more quadrature points than first-order values, and often fewer. Additional numerical stability results are provided in Appendices A.2 and A.3.

The instability of TreeSHAP is structural. Per leaf, TreeSHAP works in the monomial basis, whose coefficients scale binomially, and recovers each Shapley value as a Bernstein-weighted alternating sum of those coefficients, so any rounding in the inflated monomial terms surfaces as catastrophic cancellation, and these errors accumulate across the leaves of a deep ensemble. Quadrature-TreeSHAP

Table 4: Third-order interaction runtimes on a single explained row. CalHousing-large is excluded as both methods reached the 600s timeout.

Benchmark	F	D	d	TreeSHAP-IQ	Q-TreeSHAP	Speedup
CalHousing-small	8	6	5	0.253	0.014	18.73
CalHousing-sparse	8	20	8	30.027	1.362	22.04
CovType-small	54	6	6	0.182	0.009	20.90
CovType-large	54	16	14	25.489	0.962	26.49
CovType-sparse	54	24	18	3.685	0.186	19.84

sidesteps the basis entirely: it evaluates the summary polynomial at a handful of Gauss–Legendre nodes in $[0, 1]$, where every factor is bounded and positive so the product is forward-stable and no cancellation occurs, which is why 6–8 quadrature points already achieves high accuracy.

Higher-order interactions To evaluate interaction orders beyond the pairwise XGBoost benchmark, we compare Python implementations of TreeSHAP-IQ and Quadrature-TreeSHAP on the benchmark models (excluding Fashion-MNIST which has too many features and Adult as its models include categorical splits which are unsupported by TreeSHAP-IQ). Table 4 reports third-order interaction runtimes on a single explained row. Quadrature-TreeSHAP is consistently faster, with speedups ranging from $18.73\times$ to $26.49\times$.

Figure 2 shows the CovType-sparse runtime trend on a single explained row as the interaction order increases from 2 to 6: the runtime gap widens rapidly, and the speedup grows from $12.02\times$ at order 2 to $1209.37\times$ at order 6, reflecting both the stronger order dependence of TreeSHAP-IQ and the practical benefit of our SIMD-friendly implementation.

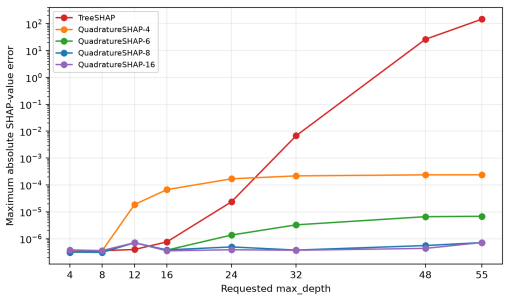


Figure 1: First-order numerical error on Fashion-MNIST-sparse vs. `max_depth`.

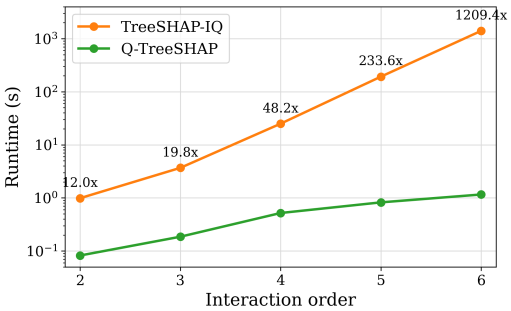


Figure 2: CovType-sparse runtime versus interaction order on a single explained row for TreeSHAP-IQ and Quadrature-TreeSHAP.

7 Conclusion

We introduced Quadrature-TreeSHAP, a quadrature-based reformulation of Path-Dependent TreeSHAP that removes practical depth dependence, remains numerically stable, and extends naturally to higher-order Shapley interaction values. Because the computation reduces to a small fixed set of evaluation points, it is also well suited to efficient CPU and GPU implementations.

In experiments on 12 XGBoost benchmarks, this structure translated into substantial gains: Quadrature-TreeSHAP achieved CPU speedups of $1.06\times$ to $10.59\times$ for first-order attributions and $3.80\times$ to $58.11\times$ for second-order interactions, while on GPU it reached speedups of up to $6.95\times$ and $6.65\times$, respectively. In higher-order Python benchmarks on CovType-sparse, the advantage over TreeSHAP-IQ grew from $12.02\times$ at order 2 to $1209.37\times$ at order 6. These results show that our quadrature-based formulation yields a practical, stable, and hardware-efficient approach to fast tree-based attribution across interaction orders.

AI USE STATEMENT

In this work, we used generative AI tools for research ideation, implementation, manuscript phrasing, and literature review. The Python implementation was developed with the assistance of Claude Code, and the code migrated into XGBoost was developed with the assistance of Codex. The initial proofs related to Theorem 1 and equation 9 were drafted with the assistance of OpenAI LLMs. These proofs were subsequently validated and elaborated by us.

The main research ideas were developed by the authors and were not suggested by LLMs. These include the tree-specific weighted-Banzhaf polynomial formulation, its use for computing Shapley values, the extension to arbitrary-order interaction values, the empirical finding that eight quadrature points are sufficient for numerical stability, and the use of a fixed eight-point formulation for efficient SIMD and SIMT execution.

We reviewed and validated all AI-assisted work. The code migrated into XGBoost is extensively tested and was validated both through automated tests and manual inspection. The proofs were carefully checked and elaborated by us, and the manuscript was reviewed several times. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Our main algorithm is now the default TreeSHAP algorithm in XGBoost. All datasets used in our experiments are publicly available, and the experimental setup is described in detail in the paper. Most of our experimental results can be recreated without any provided code, by just comparing the newest version of XGBoost with an older version on the datasets we used.

We provide all code used for the experiments in the supplementary material, including both the C++ XGBoost implementation and the Python implementation for arbitrary-order Shapley interaction values. Complete proofs of our theoretical results are provided in the appendices.

REFERENCES

Omer Abramovich, Daniel Deutch, Nave Frost, Ahmet Kara, and Dan Olteanu. Banzhaf values for facts in query answering, 2023. URL https://arxiv.org/abs/2308.05588.

John F. Banzhaf. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19: 317–343, 1965.

Jock A. Blackard and Denis J. Dean. Covertype data set. https://archive.ics.uci.edu/ml/datasets/covertype, 1999.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. doi: 10.1023/A:1010933404324.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. Association for Computing Machinery, 2016.

Philip J Davis and Philip Rabinowitz. *Methods of Numerical Integration*. Academic Press, 2 edition, 1984.

Kedar Dhamdhere, Ashish Agarwal, and Mukund Sundararajan. The shapley taylor interaction index. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Anna Veronika Dorogush, Andrey Gulin, Gleb Gusev, Nikita Kazeev, Liudmila Ostroumova Prokhorenkova, and Aleksandr Vorobev. Fighting biases with dynamic boosting. *CoRR*, 2017. URL http://arxiv.org/abs/1706.09516.

Khashayar Filom, Alexey Miroshnikov, Konstandinos Kotsiopoulos, and Arjun Ravi Kannan. On marginal feature attributions of tree-based models. *Foundations of Data Science*, 6(4):395–467,

2024. doi: 10.3934/fods.2024021. URL https://www.aimsciences.org/article/id/6640081f475da12c51d5e2f8.

Katsushige Fujimoto, Ivan Kojadinovic, and Jean-Luc Marichal. Axiomatic characterizations of probabilistic and cardinal-probabilistic interaction indices. *Games and Economic Behavior*, 55(1): 72–99, 2006. doi: 10.1016/j.geb.2005.03.002.

Fabian Fumagalli, Maximilian Muschalik, Patrick Kolpaczki, Eyke Hüllermeier, and Barbara Hammer. Shap-iq: unified approximation of any-order shapley interactions. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991. ISSN 0360-0300. URL https://doi.org/10.1145/103162.103163.

Gene H Golub and John H Welsch. Calculation of gauss quadrature rules. *Mathematics of Computation*, 23(106):221–230, 1969.

Michel Grabisch and Marc Roubens. An axiomatic approach to the concept of interaction among players in cooperative games. *International Journal of Game Theory*, 28(4):547–565, Nov 1999. doi: 10.1007/s001820050125.

Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2 edition, 2002.

Xuanxiang Huang and Joao Marques-Silva. Updates on the complexity of shap scores. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024. doi: 10.24963/ijcai.2024/45.

Christopher Hughes. Single-instruction multiple-data execution. *Synthesis Lectures on Computer Architecture*, 10:1–121, 05 2015. doi: 10.2200/S00647ED1V01Y201505CAC032.

Zhongli Jiang, Min Zhang, and Dabao Zhang. Fast calculation of feature contributions in boosting trees. In Silvia Chiappa and Sara Magliacane (eds.), *Proceedings of the Forty-first Conference on Uncertainty in Artificial Intelligence*, volume 286 of *Proceedings of Machine Learning Research*, pp. 1859–1875. PMLR, 21–25 Jul 2025. URL https://proceedings.mlr.press/v286/jiang25a.html.

Adam Karczmarz, Tomasz Michalak, Anish Mukherjee, Piotr Sankowski, and Piotr Wygocki. Improved feature importance computation for tree models based on the banzhaf value. In James Cussens and Kun Zhang (eds.), *Proceedings of UAI 2022*, volume 180 of *Proceedings of Machine Learning Research*, pp. 969–979. PMLR, 01–05 Aug 2022.

Gabriel Laberge and Yann Pequignot. Understanding interventional treeshap : How and why it works, 2022. URL https://arxiv.org/abs/2209.15123.

Weida Li, Yaoliang Yu, and Bryan Kian Hsiang Low. Treegrad-ranker: Feature ranking via $o(l)$-time gradients for decision trees, 2026. URL https://arxiv.org/abs/2602.11623.

Jinyang Liu, Tessa Steensgaard, Marvin N. Wright, Niklas Pfister, and Munir Hiabu. Fast estimation of partial dependence functions using trees. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 39496–39534. PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/liu25bm.html.

Yurong Liu, R. Teal Witter, Flip Korn, Tarfah Alrashed, Dimitris Paparas, Christopher Musco, and Juliana Freire. Kernel banzhaf: A fast and robust estimator for banzhaf values, 2025b. URL https://arxiv.org/abs/2410.08336.

Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Exploiting cpu simd extensions to speed-up document scoring with tree ensembles. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pp. 833–836, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340694. doi: 10.1145/2911451.2914758. URL https://doi.org/10.1145/2911451.2914758.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1):56–67, 2020. doi: 10.1038/s42256-019-0138-9.

Jean-Luc Marichal and Pierre Mathonet. Weighted banzhaf power and interaction indexes through weighted approximations of games. *European Journal of Operational Research*, 211(2):352–358, june 2011. ISSN 0377-2217. doi: 10.1016/j.ejor.2010.11.027. URL http://dx.doi.org/10.1016/j.ejor.2010.11.027.

Yasuko Matsui and Tomomi Matsui. Np-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263(1):305–310, 2001. URL https://www.sciencedirect.com/science/article/pii/S0304397500002516. Combinatorics and Computer Science.

Rory Mitchell, Eibe Frank, and Geoffrey Holmes. Gputreeshap: massively parallel exact calculation of shap scores for tree ensembles. *PeerJ. Computer science*, 8:e880, 2022. ISSN 2167-9843. doi: 10.7717/peerj-cs.880. URL https://europepmc.org/articles/PMC9044362.

Majid Mohammadi, Grigory Reznikov, Pavel Sinitcyn, Krikamol Muandet, and Siu Lun Chau. Quadrashap: Stable and scalable shapley values for product games via gauss-legendre quadrature, 2026. URL https://arxiv.org/abs/2605.05870.

Maximilian Muschalik, Hubert Baniecki, Fabian Fumagalli, Patrick Kolpaczki, Barbara Hammer, and Eyke Hüllermeier. shapiq: Shapley interactions for machine learning. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 130324–130357. Curran Associates, Inc., 2024a. doi: 10.52202/079017-4141. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/eb3a9313405e2d4175a5a3cfcd49999b-Paper-Datasets_and_Benchmarks_Track.pdf.

Maximilian Muschalik, Fabian Fumagalli, Barbara Hammer, and Eyke Hüllermeier. Beyond treeshap: Efficient computation of any-order shapley interactions for tree ensembles. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(13):14388–14396, Mar. 2024b. doi: 10.1609/aaai.v38i13.29352. URL https://ojs.aaai.org/index.php/AAAI/article/view/29352.

Alexander Nadel and Ron Wettenstein. From decision trees to boolean logic: A fast and unified shap algorithm. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(29):24476–24485, Mar. 2026. doi: 10.1609/aaai.v40i29.39630. URL https://ojs.aaai.org/index.php/AAAI/article/view/39630.

Andrzej S. Nowak and Tadeusz Radzik. An alternative characterization of the weighted banzhaf value. *International Journal of Game Theory*, 29(1):127–132, Feb 2000. ISSN 1432-1270. doi: 10.1007/s001829900027. URL https://doi.org/10.1007/s001829900027.

NVIDIA. CUDA C++ Programming Guide. NVIDIA Documentation, 2026. See SIMT execution, warps, and divergence behavior.

Guillermo Owen. Multilinear extensions of games. *Management Science*, 18(5-part-2):64–79, 1972.

R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33 (3):291–297, 1997. doi: 10.1016/S0167-7152(96)00140-X.

Tadeusz Radzik, Andrzej S. Nowak, and Theo S. H. Driessen. Weighted banzhaf values. *Mathematical Methods of Operations Research*, 45(1):109–118, Feb 1997. ISSN 1432-5217. doi: 10.1007/BF01194250. URL https://doi.org/10.1007/BF01194250.

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

UCI Machine Learning Repository. Adult data set. https://archive.ics.uci.edu/ml/datasets/adult, 1996.

Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of shap explanations. *J. Artif. Int. Res.*, 74, September 2022. ISSN 1076-9757. doi: 10.1613/jair.1.13283.

Jiachen T. Wang and Ruoxi Jia. Data banzhaf: A robust data valuation framework for machine learning. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 206 of *Proceedings of Machine Learning Research*, pp. 6388–6421, 2023.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Jilei Yang. Fast treeshap: Accelerating shap value computation for trees, 2022. URL https://arxiv.org/abs/2109.09847.

Ting Ye, Hucheng Zhou, Will Y. Zou, Bin Gao, and Ruofei Zhang. Rapidscorer: Fast tree ensemble evaluation by maximizing compactness in data level parallelization. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pp. 941–950, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219857. URL https://doi.org/10.1145/3219819.3219857.

Peng Yu, Albert Bifet, Jesse Read, and Chao Xu. Linear treeshap. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 25818–25828. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/a5a3b1ef79520b7cd122d888673a3ebc-Paper-Conference.pdf.

A Additional Experimental Results

A.1 First-Order Python Comparison with TreeGrad

To complement the XGBoost first-order benchmark in Sect. 6, Table 5 compares Python implementations of TreeGrad-Shap and Quadrature-TreeSHAP on 10 explained rows. TreeGrad-Shap uses its model-dependent exact quadrature count $n = \lceil d/2 \rceil$, which ranges from 3 to 26 across these models, whereas Q-TreeSHAP uses 8 fixed points. On the nine supported noncategorical benchmarks, Q-TreeSHAP is $2.15\times$–$2.67\times$ faster, with a median speedup of $2.28\times$, while agreeing with TreeGrad-Shap up to a maximum absolute difference of 1.86×10^{-11}. Adult models are shown without a TreeGrad-Shap baseline because the available adapter does not support categorical XGBoost splits.

A.2 Float64 Numerical Accuracy

To evaluate numerical errors below the float32 floor and measure errors directly at the feature level, we repeat the accuracy experiment in float64 using Fashion-MNIST models with depths 20, 30, 40, and 50. We explain 100 instances using 8, 16, and $\lceil d/2 \rceil$ Gauss–Legendre quadrature points. The $\lceil d/2 \rceil$ rule exactly integrates the polynomial integrand up to floating-point roundoff and is therefore used as the grand truth.

We report two error metrics. The *maximum per-feature error* is the largest absolute deviation of any attribution from the reference. The *efficiency error* is the absolute difference between the sum of the Shapley values and the model output minus its background mean.

Table 5: First-order Python runtimes on 10 explained rows. TreeGrad-Shap uses its model-specific exact quadrature count $n = \lceil \min(D, F)/2 \rceil$, whereas Q-TreeSHAP uses 8 fixed points. Times are total wall-clock seconds; speedups are relative to TreeGrad-Shap. Missing TreeGrad-Shap entries indicate models with categorical XGBoost splits, which are unsupported by the available TreeGrad adapter.

Benchmark	F	D	d	n_{TG}	TreeGrad-Shap	Q-TreeSHAP	Speedup
Adult-small	14	6	6	–	–	0.050	–
Adult-large	14	16	14	–	–	69.498	–
Adult-sparse	14	28	13	–	–	5.663	–
CalHousing-small	8	6	5	3	0.186	0.074	2.52
CalHousing-large	8	16	8	4	1079.476	410.543	2.63
CalHousing-sparse	8	20	8	4	14.772	5.682	2.60
CovType-small	54	6	6	3	0.091	0.040	2.28
CovType-large	54	16	14	8	3.881	1.806	2.15
CovType-sparse	54	24	18	12	0.679	0.306	2.22
Fashion-MNIST-small	784	6	6	3	1.970	0.737	2.67
Fashion-MNIST-large	784	16	16	8	738.263	325.558	2.27
Fashion-MNIST-sparse	784	52	48	26	104.485	45.954	2.27

Table 6: Float64 per-feature (max) and efficiency (eff.) errors on Fashion-MNIST with 8 and 16 quadrature points, 100 explained rows.

Depth	8-pt max	8-pt eff.	16-pt max	16-pt eff.	Ref. eff.
20	5.1×10^{-12}	6.9×10^{-11}	2.2×10^{-15}	6.3×10^{-15}	4.8×10^{-15}
30	4.5×10^{-10}	2.2×10^{-9}	1.4×10^{-15}	7.7×10^{-15}	7.4×10^{-15}
40	1.2×10^{-9}	1.6×10^{-8}	2.2×10^{-15}	7.6×10^{-15}	1.0×10^{-14}
50	9.8×10^{-9}	4.0×10^{-8}	1.9×10^{-15}	7.7×10^{-15}	9.1×10^{-15}

With 8 quadrature points, the maximum per-feature error ranges from 5.1×10^{-12} at depth 20 to 9.8×10^{-9} at depth 50, remaining below float32 unit roundoff (5.96×10^{-8}) at every tested depth. Thus, the float32 results in Fig. 1 are limited primarily by floating-point precision rather than quadrature error. With 16 points, both the per-feature and efficiency errors reach the same 10^{-15} numerical floor as the reference. Hence, 16 points are empirically indistinguishable from exact integration in float64 over the tested models. Moreover, both 8 and 16 points map naturally to SIMD lanes and GPU warps, allowing increased numerical accuracy without sacrificing hardware utilization.

A.3 Conservative Attribution Error

To evaluate numerical accuracy as conservatively as possible, we also measure the total absolute attribution error across all 784 Fashion-MNIST features in the experiments discussed in Appendix A.2. We call this metric `sum_abs_errs` or `sae` in short. For each explained row, we compute:

$$sae = \sum_{i=1}^{784} \left| \phi_i - \phi_i^{\mathrm{ref}} \right| \tag{12}$$

Unlike the efficiency error, which sums signed attribution errors and may therefore benefit from cancellation, this metric accumulates every feature-wise deviation in absolute value. To obtain a conservative dataset-level measure, we then report the maximum of this sum over all 100 explained rows, together with its average.

Table 7: Maximum and average `sae` over 100 explained rows on Fashion-MNIST with 8 and 16 quadrature points, in float64.

Depth	Actual depth	8 points		16 points	
		`sae` max	`sae` avg	`sae` max	`sae` avg
20	20	1.3×10^{-10}	1.5×10^{-11}	5.8×10^{-15}	3.7×10^{-15}
30	30	3.1×10^{-9}	3.4×10^{-10}	6.5×10^{-15}	3.8×10^{-15}
40	40	2.5×10^{-8}	1.7×10^{-9}	1.3×10^{-14}	7.5×10^{-15}
50	50	6.6×10^{-8}	4.7×10^{-9}	1.0×10^{-14}	5.9×10^{-15}

Even under this conservative measure, the errors remain extremely small. With 8 quadrature points, the maximum total absolute error remains near the float32 precision scale even at depth 50, while with 16 points it stays at the float64 numerical noise floor across all tested depths.

B Quadrature Error Bounds

In this appendix, we derive an upper bound on the quadrature error. We consider an individual leaf's contribution to a first-order SHAP value. If the root-to-leaf path contains at most 50 distinct features, then for a leaf with value of 1, the error is at most 4.73165×10^{-4} with 8 quadrature points and 2.14006×10^{-11} with 16 points. Table 8 shows the upper bound for more depths and quadrature points. These bounds scale linearly with the absolute leaf value. Summing the bounds over leaves gives an upper bound on the full attribution error. All statements concern quadrature error in exact arithmetic.

B.1 Definitions and Reformulation of the Weighted Banzhaf Polynomial

Fix an explained input x, a leaf v, and a target feature $i \in M(v)$. Write $V = \mathrm{val}(v)$ and $d_v = |M(v)| \le d$, where d is the maximum number of distinct features on a root-to-leaf path. Set $m = d - 1$, so there are at most m non-target path features. If $i \notin M(v)$, the leaf contribution is zero by Theorem 1. Empty products equal one.

Using the edge weights $w_e \in (0, 1]$ from Section 2, combine all occurrences of each feature $j \in M(v)$:

$$\rho_j = \prod_{\substack{e \in \mathrm{path}(v) \\ f(e)=j}} w_e, \qquad \delta_j = \begin{cases} 1, & x \text{ satisfies every path condition on } j, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

Thus $0 < \rho_j \le 1$. We call j matched when $\delta_j = 1$ and unmatched when $\delta_j = 0$. The definitions in Section 2 give $R^v_\emptyset = V \prod_{j \in M(v)} \rho_j$ and $q^v_j = \delta_j / \rho_j$. Specializing Theorem 1 to $S = \{i\}$ gives

$$\begin{aligned} B(p) := B^{(\{i\})}_{p,v} &= V \left(\prod_{j \in M(v)} \rho_j \right) \left(\frac{\delta_i}{\rho_i} - 1 \right) \prod_{j \in M(v) \setminus \{i\}} \left((1-p) + p \frac{\delta_j}{\rho_j} \right) \\ &= V(\delta_i - \rho_i) \prod_{j \in M(v) \setminus \{i\}} [\rho_j (1-p) + \delta_j p]. \end{aligned} \tag{14}$$

Here $p \in [0, 1]$ is the participation probability.

Define

$$c = V(\delta_i - \rho_i) = \begin{cases} V(1 - \rho_i), & \delta_i = 1, \\ -V\rho_i, & \delta_i = 0, \end{cases} \qquad |c| \le |V|. \tag{15}$$

Partition the non-target features into the matched and unmatched sets

$$A = \{j \in M(v) \setminus \{i\} : \delta_j = 1\}, \qquad U = \{j \in M(v) \setminus \{i\} : \delta_j = 0\}.$$

Let $a = |A|$, $u = |U|$, so $a + u = d_v - 1 \le m$, and define $R_A = \prod_{j \in A} \rho_j$ and $R_U = \prod_{j \in U} \rho_j$. Equation equation 14 becomes

$$B(p) = cR_U(1-p)^u P_A(p), \qquad P_A(p) = \prod_{j \in A} [\rho_j + (1-\rho_j)p]. \tag{16}$$

B.2 Bounding the Error

For $n \ge 1$, let $t_{\ell,n} \in (0,1)$ and $w_{\ell,n} > 0$, $\ell = 1, \dots, n$, be the Gauss–Legendre nodes and weights on $[0,1]$, with $\sum_{\ell=1}^{n} w_{\ell,n} = 1$. Define

$$Q_n[f] = \sum_{\ell=1}^{n} w_{\ell,n} f(t_{\ell,n}), \qquad \mathcal{E}_n[f] = \int_0^1 f(p)\,dp - Q_n[f]. \tag{17}$$

Thus $\widehat{\phi}_i^{v,(n)} = Q_n[B]$ and $|\phi_i^v - \widehat{\phi}_i^{v,(n)}| = |\mathcal{E}_n[B]|$. As used in the quadrature-point theorem, the rule is exact for polynomials of degree at most $2n-1$.

Lemma 3 (Quadrature error of a beta monomial)**.** *For every $n \ge 1$ and nonnegative integers k, u,*

$$\mathcal{E}_n[p^k(1-p)^u] = \frac{k!\,u!}{(k+u+1)!} - \sum_{\ell=1}^{n} w_{\ell,n} t_{\ell,n}^k (1-t_{\ell,n})^u. \tag{18}$$

Proof. The beta–gamma identity gives

$$\int_0^1 p^k(1-p)^u\,dp = \mathrm{B}(k+1, u+1) = \frac{\Gamma(k+1)\Gamma(u+1)}{\Gamma(k+u+2)} = \frac{k!\,u!}{(k+u+1)!},$$

where $\Gamma(r+1) = r!$ for every nonnegative integer r.

Subtracting $Q_n[p^k(1-p)^u]$ proves the claim. □

Write $e_{k,u,n} := \mathcal{E}_n[p^k(1-p)^u]$. Polynomial exactness gives $e_{k,u,n} = 0$ whenever $k + u \le 2n - 1$.

Lemma 4 (Matched-product coefficients)**.** *The matched product in equation 16 has an expansion*

$$P_A(p) = \sum_{k=0}^{a} \lambda_k p^k, \qquad \lambda_k \ge 0, \qquad \sum_{k=0}^{a} \lambda_k = 1, \qquad \lambda_0 = R_A. \tag{19}$$

Proof. Every factor has nonnegative coefficients, so $\lambda_k \ge 0$. Their sum is $P_A(1) = 1$, and the constant coefficient is $P_A(0) = R_A$. □

Lemma 5 (General leaf bound)**.** *For every $n \ge 1$ and every match pattern,*

$$\mathcal{E}_n[B] = cR_U \sum_{k=0}^{a} \lambda_k e_{k,u,n}, \qquad |\mathcal{E}_n[B]| \le |c| R_U \sum_{k=0}^{a} \lambda_k |e_{k,u,n}|. \tag{20}$$

Proof. Substituting equation 19 into equation 16 gives $B(p) = cR_U \sum_{k=0}^{a} \lambda_k p^k (1-p)^u$.

Linearity gives the equality in equation 20.

The triangle inequality and $\lambda_k \ge 0$ give the bound. □

For a nonnegative integer m, define

$$E_{m,n} = \max_{\substack{k,u \in \mathbb{Z}_{\ge 0} \\ k+u \le m}} |e_{k,u,n}|. \tag{21}$$

The maximum includes all degrees up to m and all match patterns.

Corollary 6 (Uniform leaf bound). *For every $n \geq 1$ and every match pattern with $a + u \leq m$,*

$$\boxed{|\phi_i^v - \widehat{\phi}_i^{v,(n)}| = |\mathcal{E}_n[B]| \leq |V| R_U E_{m,n} \leq |V| E_{m,n}.} \tag{22}$$

In particular, $E_{m,n} = 0$ when $m \leq 2n - 1$, recovering exactness for first-order values whenever $d \leq 2n$.

Proof. Since $k + u \leq a + u \leq m$ and $\sum_k \lambda_k = 1$, $\sum_k \lambda_k |e_{k,u,n}| \leq E_{m,n}$. Apply Lemma 5, $|c| \leq |V|$, and $R_U \leq 1$.

The exactness claim follows because every $e_{k,u,n}$ in equation 21 vanishes when $m \leq 2n - 1$. □

B.3 Numerical bounds

Table 8 gives $E_{d-1,n}$ for $n = 4, 8, 16, 32$ and d up to 100. Multiply each entry by $|\operatorname{val}(v)|$ to obtain the absolute-error bound for one leaf's first-order contribution. A zero denotes exact quadrature.

For example, $n = 8$ and $d \leq 50$ give

$$|\phi_i^v - \widehat{\phi}_i^{v,(8)}| \leq 4.73165 \times 10^{-4} |\operatorname{val}(v)|.$$

For $n = 4$, the finite search through degree 99 is maximized by p^{46} and $(1 - p)^{46}$, explaining the repeated entries in that column. When the path covers are known, equation 20 using c, R_U, and the coefficients λ_k, can give a smaller bound.

Computing the bounds. Treat $Q_n[f]$ as a Gauss–Legendre quadrature routine on $[0, 1]$; for example, SciPy provides `fixed_quad(f, 0, 1, n=n)[0]`. Enumerate $k, u \geq 0$ with $k + u \leq m$, evaluate $e_{k,u,n} = \mathrm{B}(k + 1, u + 1) - Q_n[p^k (1 - p)^u]$, and take the largest absolute value. Set terms with $k + u \leq 2n - 1$ to zero by exactness.

d	$n = 4$	$n = 8$	$n = 16$	$n = 32$
10	1.02041×10^{-4}	0	0	0
16	2.59721×10^{-3}	0	0	0
17	3.28898×10^{-3}	3.55136×10^{-10}	0	0
20	5.51999×10^{-3}	4.65879×10^{-8}	0	0
25	9.05212×10^{-3}	1.97437×10^{-6}	0	0
30	1.17497×10^{-2}	1.54223×10^{-5}	0	0
35	1.35117×10^{-2}	5.77714×10^{-5}	1.19360×10^{-17}	0
40	1.44913×10^{-2}	1.44868×10^{-4}	1.34131×10^{-14}	0
45	1.48892×10^{-2}	2.83990×10^{-4}	1.00345×10^{-12}	0
50	1.49265×10^{-2}	4.73165×10^{-4}	2.14006×10^{-11}	0
55	1.49265×10^{-2}	7.04022×10^{-4}	2.17496×10^{-10}	0
60	1.49265×10^{-2}	9.65182×10^{-4}	1.35627×10^{-9}	0
65	1.49265×10^{-2}	1.24483×10^{-3}	5.99672×10^{-9}	4.58079×10^{-39}
70	1.49265×10^{-2}	1.53222×10^{-3}	2.05829×10^{-8}	1.73201×10^{-33}
75	1.49265×10^{-2}	1.81843×10^{-3}	5.82576×10^{-8}	4.42774×10^{-30}
80	1.49265×10^{-2}	2.09647×10^{-3}	1.41824×10^{-7}	1.55264×10^{-27}
85	1.49265×10^{-2}	2.36124×10^{-3}	3.06136×10^{-7}	1.66297×10^{-25}
90	1.49265×10^{-2}	2.60919×10^{-3}	5.99342×10^{-7}	7.94340×10^{-24}
95	1.49265×10^{-2}	2.83812×10^{-3}	1.08269×10^{-6}	2.10314×10^{-22}
100	1.49265×10^{-2}	3.04683×10^{-3}	1.82893×10^{-6}	3.54642×10^{-21}

Table 8: The bound $E_{d-1,n}$ per unit leaf value, evaluated at high precision and rounded upward to six significant figures. It applies to every match pattern with at most d distinct path features.

C Weighted Banzhaf Polynomial Proof

This section includes the proof of Theorem 1. We start by proving two required lemmas, and then continue to the main proof.

C.1 Required Lemma Proofs

Lemma 7. *Let $f_v : 2^{[F]} \to \mathbb{R}$ denote the Path-Dependent game associated with a leaf v, as defined in equation 3. For $S \subseteq [F]$ and $T \subseteq [F] \setminus S$, define (same as in Def. 1):*

$$\Delta_S f_v(T) := \sum_{L \subseteq S} (-1)^{|S|-|L|} f_v(T \cup L),$$

Then:

$$\Delta_S f_v(T) = \begin{cases} R^v_\emptyset \left(\prod_{j \in S} (q^v_j - 1) \right) \left(\prod_{j \in T \cap (M(v) \setminus S)} q^v_j \right), & S \subseteq M(v), \\ 0 & \textit{otherwise}. \end{cases}$$

Proof. We start with the definition:

$$\Delta_S f_v(T) = \sum_{L \subseteq S} (-1)^{|S|-|L|} f_v(T \cup L).$$

Substituting f_v and splitting the product using the identities below:

1. By the distributive property: $(T \cup L) \cap M(v) = (T \cap M(v)) \cup (L \cap M(v))$.
2. Since T, S are disjoint: $T \cap M(v) = T \cap (M(v) \setminus S)$
3. Since T, S are disjoint and $L \subseteq S$, it follows that T and L are disjoint.
4. Since T, L are disjoint, $(T \cap (M(v) \setminus S))$ and $(L \cap M(v))$ are also disjoint.

$$f_v(T \cup L) = R^v_\emptyset \prod_{j \in (T \cup L) \cap M(v)} q^v_j = R^v_\emptyset \left(\prod_{j \in T \cap (M(v) \setminus S)} q^v_j \right) \left(\prod_{j \in L \cap M(v)} q^v_j \right).$$

Thus, (the set $T \cap (M(v) \setminus S)$ is independent of L):

$$\Delta_S f_v(T) = R^v_\emptyset \left(\prod_{j \in T \cap (M(v) \setminus S)} q^v_j \right) \sum_{L \subseteq S} (-1)^{|S|-|L|} \prod_{j \in L \cap M(v)} q^v_j.$$

By Lemma 8 (see below):

$$\sum_{L \subseteq S} (-1)^{|S|-|L|} \prod_{j \in L \cap M(v)} q^v_j = \begin{cases} \prod_{j \in S} (q^v_j - 1), & S \subseteq M(v), \\ 0, & \text{otherwise.} \end{cases}$$

which completes the proof. □

Lemma 8 (Subset Expansion and Null-Player Cancellation)**.** *Let $S \subseteq [F]$ and let $M \subseteq [F]$. For any coefficients $\{q_j\}_{j \in [F]}$, define*

$$\Phi(S, M) := \sum_{L \subseteq S} (-1)^{|S|-|L|} \prod_{j \in L \cap M} q_j.$$

Then:

$$\Phi(S, M) = \begin{cases} \prod_{j \in S} (q_j - 1), & S \subseteq M, \\ 0, & \textit{otherwise}. \end{cases}$$

Proof. **Case 1:** $S \not\subseteq M$**.**

Let $i \in S \setminus M$. Since $i \notin M$, the factor q_i never appears in any product. Thus, for every $L \subseteq S \setminus \{i\}$,

$$\prod_{j \in L \cap M} q_j = \prod_{j \in (L \cup \{i\}) \cap M} q_j.$$

Consider the two terms indexed by L and $L \cup \{i\}$. Their coefficients differ by a sign:

$$(-1)^{|S|-|L|} \quad \text{and} \quad (-1)^{|S|-|L|-1}.$$

Hence,

$$(-1)^{|S|-|L|} \prod_{j \in L \cap M} q_j + (-1)^{|S|-|L|-1} \prod_{j \in (L \cup \{i\}) \cap M} q_j = 0.$$

Pairing all subsets in this way shows that every term cancels, and therefore

$$\Phi(S, M) = 0.$$

Case 2: $S \subseteq M$**.**

In this case, since $S \subseteq M$, we have $L \cap M = L$ for all $L \subseteq S$. Hence

$$\Phi(S, M) = \sum_{L \subseteq S} (-1)^{|S|-|L|} \prod_{j \in L} q_j.$$

We now show that this sum equals $\prod_{j \in S}(q_j - 1)$.

First, observe that

$$\prod_{j \in S}(q_j - 1) = \prod_{j \in S}\big(-(1 - q_j)\big) = (-1)^{|S|} \prod_{j \in S}(1 - q_j).$$

Next, we expand the product $\prod_{j \in S}(1 - q_j)$ using the standard subset expansion:

$$\prod_{j \in S}(1 - q_j) = \sum_{L \subseteq S} \prod_{j \in L}(-q_j) = \sum_{L \subseteq S} (-1)^{|L|} \prod_{j \in L} q_j.$$

Substituting this back, we obtain

$$\prod_{j \in S}(q_j - 1) = (-1)^{|S|} \sum_{L \subseteq S} (-1)^{|L|} \prod_{j \in L} q_j = \sum_{L \subseteq S} (-1)^{|S|+|L|} \prod_{j \in L} q_j.$$

Considering the parity of both $|S|$ and $|L|$, we observe that $(-1)^{|S|+|L|} = (-1)^{|S|-|L|}$. Thus:

$$\prod_{j \in S}(q_j - 1) = \sum_{L \subseteq S} (-1)^{|S|-|L|} \prod_{j \in L} q_j.$$

Comparing with the expression for $\Phi(S, M)$, we conclude that

$$\Phi(S, M) = \prod_{j \in S}(q_j - 1),$$

as claimed.

□

C.2 THEOREM 1 PROOF

Having established the required lemmas, we now provide the proof for Theorem 1. We take expectation over $T \sim p$ and assume $S \subseteq M(v)$ (otherwise the result is zero). Using the Lemma 7:

$$\Delta_S f_v(T) = R^v_\emptyset \left(\prod_{j \in S} (q_j^v - 1) \right) \left(\prod_{j \in T \cap (M(v) \setminus S)} q_j^v \right).$$

Taking expectation and extracting factors that do not depend on p and T:

$$B^{(S)}_{p,v} = R^v_\emptyset \left(\prod_{j \in S} (q_j^v - 1) \right) \mathbb{E}_{T \sim p} \left[\prod_{j \in T \cap (M(v) \setminus S)} q_j^v \right].$$

We rewrite the product using indicator variables, where $\mathbf{1}\{j \in T\}$ denotes whether feature j is included in the random subset T. Then we use the independence of feature inclusion:

$$\mathbb{E}_{T \sim p} \left[\prod_{j \in T \cap (M(v) \setminus S)} q_j^v \right] = \mathbb{E}_{T \sim p} \left[\prod_{j \in M(v) \setminus S} (q_j^v)^{\mathbf{1}\{j \in T\}} \right] = \prod_{j \in M(v) \setminus S} \mathbb{E}_{T \sim p} \left[(q_j^v)^{\mathbf{1}\{j \in T\}} \right],$$

For each $j \in M(v) \setminus S$, we have

$$\mathbb{E}_{T \sim p} \left[(q_j^v)^{\mathbf{1}\{j \in T\}} \right] = (1-p) \cdot (q_j^v)^0 + p \cdot (q_j^v)^1 = (1-p) + p q_j^v.$$

Therefore,

$$\mathbb{E}_{T \sim p} \left[\prod_{j \in T \cap (M(v) \setminus S)} q_j^v \right] = \prod_{j \in M(v) \setminus S} \big((1-p) + p q_j^v \big).$$

Define:

$$\alpha_j^v := q_j^v - 1.$$

Then:

$$(1-p) + p q_j^v = 1 + \alpha_j^v p.$$

And:

$$B^{(S)}_{p,v} = R^v_\emptyset \left(\prod_{j \in S} (q_j^v - 1) \right) \mathbb{E}_{T \sim p} \left[\prod_{j \in T \cap (M(v) \setminus S)} q_j^v \right] = R^v_\emptyset \left(\prod_{j \in S} \alpha_j^v \right) \prod_{j \in M(v) \setminus S} \big(1 + \alpha_j^v p \big).$$

Note that by assigning $\alpha_j^v = q_j^v - 1$ in the polynomial above, we obtain equation 7.

Define the polynomial:

$$H_v(p) := \prod_{j \in M(v)} (1 + \alpha_j^v p).$$

Thus:

$$\prod_{j \in M(v) \setminus S} (1 + \alpha_j^v p) = \frac{H_v(p)}{\prod_{j \in S} (1 + \alpha_j^v p)}.$$

Recall that at the start of this section we assumed $S \subseteq M(v)$; otherwise, $\Delta_S f_v(T) = 0$. We obtain:

$$\boxed{B^{(S)}_{p,v} = \begin{cases} R^v_\emptyset \left(\prod_{j \in S} \alpha_j^v \right) \dfrac{H_v(p)}{\prod_{j \in S} (1 + \alpha_j^v p)} & S \subseteq M(v), \\ 0 & \text{otherwise.} \end{cases}}$$

This proves Theorem 1.

D PROOF OF THE QUADRATURE-POINT THEOREM

In this section we prove Theorem 2.

Proof of Theorem 2. Fix a set $S \subseteq [F]$ with $|S| = s$. By the weighted-Banzhaf polynomial formula, the leaf-level contribution can be written as

$$\mathrm{SII}_v(S) = R^v_\emptyset \left(\prod_{j \in S} \alpha^v_j \right) \int_0^1 \frac{H_v(t)}{\prod_{j \in S}(1 + \alpha^v_j t)}\, dt.$$

Since

$$H_v(t) = \prod_{j \in M(v)} (1 + \alpha^v_j t),$$

the denominator cancels the factors indexed by S, and therefore

$$\frac{H_v(t)}{\prod_{j \in S}(1 + \alpha^v_j t)} = H_v^{\setminus S}(t) := \prod_{j \in M(v) \setminus S} (1 + \alpha^v_j t).$$

Hence

$$\mathrm{SII}(S) = \int_0^1 \underbrace{\sum_{v:\, S \subseteq M(v)} R^v_\emptyset \left(\prod_{j \in S} \alpha^v_j \right) H_v^{\setminus S}(t)}_{I_S(t)}\, dt.$$

For each leaf v, the polynomial $H_v^{\setminus S}(t)$ has degree $|M(v)| - s$. By the definition of d (the maximum number of unique features along a root-to-leaf path), $|M(v)| \le d$. Therefore,

$$\deg H_v^{\setminus S}(t) \le d - s.$$

The total integrand $I_S(t)$ is a finite weighted sum of such polynomials, so it is itself a polynomial of degree at most $d - s$.

Finally, Gauss–Legendre quadrature with n points is exact for all polynomials of degree at most $2n - 1$ (Davis & Rabinowitz, 1984; Golub & Welsch, 1969). Thus it is enough to require

$$2n - 1 \ge d - s,$$

which is equivalent to

$$n \ge \left\lceil \frac{d - s + 1}{2} \right\rceil.$$

This proves Theorem 2. □

E RECOVERING SHAPLEY INTERACTION VALUES FROM WEIGHTED BANZHAF INTERACTIONS POLYNOMIAL

In this section we show that Shapley interaction values can be recovered by integrating the weighted Banzhaf interaction values over the participation probability $p \in [0, 1]$.

Lemma 9 (Beta-function identity)**.** *For any integers $k \ge 0$ and $n \ge k$, we have*

$$\int_0^1 p^k (1 - p)^{n-k}\, dp = \frac{k!(n - k)!}{(n + 1)!}.$$

Proof. This is the standard Beta-function identity:

$$\int_0^1 p^{a-1} (1 - p)^{b-1}\, dp = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a + b)}.$$

Setting $a = k + 1$ and $b = n - k + 1$, and using $\Gamma(m + 1) = m!$ for integers m, gives the result. □

Theorem 10 (Shapley interaction values as an integral of weighted Banzhaf interactions)**.** *Let $f_v : 2^{[F]} \to \mathbb{R}$ be a set function and let $S \subseteq [F]$. Then*

$$\mathrm{SII}(S) = \int_0^1 B_p^{(S)}(f_v)\, dp.$$

Proof. By definition of the weighted Banzhaf interaction, where T is sampled by including each feature in $[F] \setminus S$ independently with probability p (i.e., $T \sim \mathrm{Bernoulli}(p)$):

$$B_p^{(S)}(f_v) = \mathbb{E}_{T\sim p}[\Delta_S f_v(T)] = \sum_{T\subseteq[F]\setminus S} p^{|T|}(1-p)^{F-|S|-|T|}\Delta_S f_v(T),$$

where

$$\Delta_S f_v(T) := \sum_{L\subseteq S}(-1)^{|S|-|L|} f_v(T\cup L).$$

Integrating over p and exchanging sum and integral,

$$\int_0^1 B_p^{(S)}(v)\, dp = \sum_{T\subseteq[F]\setminus S}\left(\int_0^1 p^{|T|}(1-p)^{F-|S|-|T|}\, dp\right)\Delta_S f_v(T).$$

Applying Lemma 9 with $k = |T|$ and $n = F - |S|$, we obtain

$$\int_0^1 B_p^{(S)}(v)\, dp = \sum_{T\subseteq[F]\setminus S}\frac{|T|!(F-|S|-|T|)!}{(F-|S|+1)!}\Delta_S f_v(T).$$

This is exactly the definition of the Shapley interaction index (Grabisch & Roubens, 1999):

$$\mathrm{SII}(S) = \sum_{T\subseteq[F]\setminus S}\frac{|T|!(F-|S|-|T|)!}{(F-|S|+1)!}\Delta_S f_v(T).$$

Therefore,

$$\mathrm{SII}(S) = \int_0^1 B_p^{(S)}(f_v)\, dp.$$

□

Corollary 11 (First-order case)**.** *For $S = \{i\}$, the theorem reduces to*

$$\phi_i(f_v) = \int_0^1 B_p^{(\{i\})}(f_v)\, dp.$$