Amortized Linear-time Exact Shapley Value for Product-Kernel Methods

Majid Mohammadi[‡]
Department of Biomedical Data Science
Leiden University Medical Center
The Netherlands

Siu Lun Chau[†]
College of Computing & Data Science
Nanyang Technological University
Singapore

Krikamol Muandet
Rational Intelligence Lab
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany

Abstract

Kernel methods are widely used in machine learning and statistics for their flexibility and expressive power, yet their black-box nature limits adoption in high-stakes applications. Shapley value–based attribution methods such as SHAP, and kernel-specific adaptations including RKHS-SHAP, provide a principled framework for explainability—but exact computation of Shapley values is generally intractable, forcing existing approaches to rely on approximations that incur unavoidable estimation error. We introduce PKeX-Shapley, an algorithm that exploits the multiplicative structure of product kernels to compute exact Shapley values for all d features in quadratic time in d. The method rests on a distribution-free removal operator intrinsic to the product-kernel structure: removing a feature replaces its kernel factor with the multiplicative identity. This yields a parameter-free value function—requiring no sampling and no density estimation—and uniquely determines a functional decomposition of the model, from which the exact Shapley value can be computed. Building on this value function, we develop shared recursive formulations that evaluate all feature attributions jointly, achieving amortized linear time per feature with numerical stability. Beyond predictive modeling, the framework extends to widely used kernel-based discrepancies such as the Maximum Mean Discrepancy (MMD) and the Hilbert–Schmidt Independence Criterion (HSIC), providing new tools for interpretable statistical analysis.

1 Introduction

Shapley values [Shapley, 1953], a solution concept from cooperative game theory, provide a principled axiomatic framework for feature attribution in machine learning (ML) [Lundberg and Lee, 2017, Covert et al., 2020]. Their rigorous foundation has driven widespread adoption in the explainable AI community [Sundararajan and Najmi, 2020]: a model's output—predictions, losses, or other quantities of interest—can be *fairly* distributed across input features according to their individual and joint contributions. A range of algorithms has been developed to estimate Shapley values under different modelling assumptions, from model-agnostic methods such as Kernel SHAP [Lundberg and Lee, 2017] to model-specific methods that leverage structural properties for statistical or computational gains: TreeSHAP for tree-based models [Lundberg et al., 2020], GPSHAP for Gaussian

[*]Corresponding author: majid.mohammadi690@gmail.com
[†]These authors contributed equally.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

processes [Chau et al., 2023], Deep SHAP for deep neural networks [Lundberg and Lee, 2017], and, most relevant to our work, RKHS-SHAP for kernel methods [Chau et al., 2022]. Kernel methods are notable well beyond prediction: they underpin a broad range of statistical inference tasks, including measuring distributional closeness [Gretton et al., 2006, Naslidnyk et al., 2025], two-sample testing [Schrab, 2025, Chau et al., 2025], goodness-of-fit testing [Chwialkowski et al., 2016, Liu et al., 2016], independence testing [Albert et al., 2022], and causal inference and discovery [Mitrovic et al., 2018, Zhang et al., 2026]. As kernel methods see growing use in high-stakes applications for their flexibility and expressive power, interpretability has become essential.

Despite their rigorous game-theoretic foundation, Shapley values face two major challenges when applied to ML. The first is to define and estimate a value function that quantifies the contribution of a feature coalition. Ideally, this function should capture the model's behavior when the complementary features are absent. A natural approach retrains the model on each subset and uses the resulting prediction as the value [Lipovetsky and Conklin, 2001], but the exponential number of retrainings makes this infeasible. A more practical alternative simulates feature absence via marginal or conditional expectations of the model's output given the retained subset; see Sundararajan and Najmi [2020] for further options. Chau et al. [2022] pursue this route in the context of kernel methods, exploiting the structure of reproducing kernel Hilbert spaces (RKHS) and using kernel distributional embeddings [Muandet et al., 2017] to estimate value functions nonparametrically, thereby avoiding density estimation. An increasingly popular alternative defines the value function through functional decomposition [Gevaert and Saeys, 2024, Fumagalli et al., 2025, Mohammadi et al., 2026b], identifying the contribution of a feature subset with the component of the model that depends on those features.

Once a value function is chosen, the second challenge is computing the Shapley values themselves. Exact evaluation requires the value function on all 2^d feature subsets, which quickly becomes prohibitive. Approximation methods—Monte Carlo sampling, regression-based estimators—sidestep the exponential cost by evaluating only a subset of coalitions, but introduce estimation errors that grow with dimensionality [Kumar et al., 2020]. Model-specific structure can sometimes be exploited to recover exactness: TreeSHAP, for instance, leverages tree decompositions for exact polynomial-time computation. RKHS-SHAP improves the statistical estimation of the value function but still relies on regression-based approximation for the Shapley values, inheriting the same computational limitations.

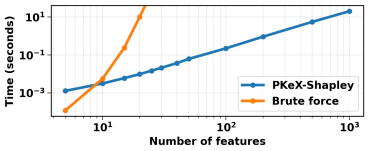

Figure 1: Execution time vs. number of features (log–log). Brute-force reaches at most 20 features in 20 s; PKeX-Shapley scales to 1,000.

To design an efficient Shapley value–based attribution for kernel methods, we focus on the subclass employing product kernels—which we call *product-kernel methods*—and introduce *PKeX-Shapley* (**P**roduct-**K**ernel-based **eX**act **Shapley** attribution). Product kernels are simple to implement yet retain strong theoretical guarantees: the product of universal kernels is itself universal, so the resulting RKHS can approximate any continuous function whenever each component kernel is sufficiently expressive [Szabó and Sriperumbudur, 2018]. This narrows the scope relative to Chau et al. [2022], which considers generic kernels, but the additional structure lets us address both challenges raised above directly. We first identify the *removal operator* intrinsic to product-kernel models: removing a feature j amounts to replacing its kernel factor by the multiplicative identity, $k_j \mapsto 1$. This choice preserves the product structure, requires no background distribution, and induces a parameter-free value function—no sampling, no density estimation. By Gevaert and Saeys [2024], every removal operator determines a unique *functional decomposition*; ours admits a simple closed form indexed by feature subsets. Building on this value function, we derive a recursive formulation that computes the Shapley value of a single feature in quadratic time. Organizing these recursions across features via a prefix–suffix construction of elementary symmetric polynomials, we then show that all d Shapley values can be computed jointly in $O(d^2 n)$ time—an *amortized linear-time* cost per feature in d [Cormen et al., 2022, Chapter 17]; see Figure 1 for comparison against brute-force computation.

As an additional, independent contribution, we demonstrate that our framework extends beyond kernel-based predictive models. In particular, we show how our method applies to popular kernel-based statistical discrepancies, including the Maximum Mean Discrepancy (MMD) and Hilbert–Schmidt Independence Criterion (HSIC), thereby enabling interpretable statistical inference. A Python implementation of our method is publicly available [Mohammadi et al., 2026a].

2 Background

Notation. Let $\mathcal{D}$ denote the set of d features, and $2^{\mathcal{D}}$ its power set. The training set $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ consists of n samples, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$ (or a discrete label set for classification). Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the feature matrix, and $\mathbf{X}_{\mathcal{S}} \in \mathbb{R}^{n \times s}$ the submatrix restricted to features in subset $\mathcal{S} \subseteq \mathcal{D}$, and we write $\mathbf{X}_j := \mathbf{X}_{\{j\}}$. We use capital letters for random variables, bold capital letters for matrices, calligraphic letters for sets, and bold lowercase letters for vectors. The restriction of a vector $\boldsymbol{x}$ to features in $\mathcal{S}$ is denoted by $\boldsymbol{x}_{\mathcal{S}}$. The element-wise product and division are respectively denoted by $\odot$ and $\oslash$, and expectation by $\mathbb{E}$. A symmetric positive (semi-)definite kernel function over $\mathcal{D}$ is denoted by k, and its restriction to a subset $\mathcal{S}$ by $k_{\mathcal{S}}$. The corresponding kernel matrices are denoted by $\mathbf{K}$ and $\mathbf{K}_{\mathcal{S}}$, respectively. All proofs are provided in Appendix D.

Shapley value. The Shapley value [Shapley, 1953] is a widely used game-theoretic solution concept for feature attribution in predictive models. It assigns importance to each input feature by averaging its marginal contribution across all feature subsets, weighted by coalition size. Given a value function $v : 2^{\mathcal{D}} \to \mathbb{R}$ that quantifies the contribution of a feature subset, the Shapley value for feature j is

$$\phi_j = \sum_{\mathcal{S} \subseteq \mathcal{D} \setminus \{j\}} \mu(|\mathcal{S}|)\big(v(\mathcal{S} \cup \{j\}) - v(\mathcal{S})\big), \tag{1}$$

where $\mu(s) = {}^{s!\,(d-s-1)!}/_{d!}$ and $v(\mathcal{S} \cup \{j\}) - v(\mathcal{S})$ is the marginal contribution of j to $\mathcal{S}$. For any fixed value function, the Shapley value is the unique attribution satisfying efficiency, null player, symmetry, and linearity [Shapley, 1953]. In explainability, however, the choice of v is itself a non-trivial design decision. Several formulations have been proposed [Sundararajan and Najmi, 2020]; two are dominant. The *interventional* value function [Janzing et al., 2020], $v_{\boldsymbol{x}}(\mathcal{S}) = \mathbb{E}_{X_{\mathcal{D}\setminus\mathcal{S}}}\left[f(\boldsymbol{x}_{\mathcal{S}}, X_{\mathcal{D}\setminus\mathcal{S}})\right]$, replaces missing features with samples from their marginal distributions, while the *observational* value function [Chen et al., 2020], $v_{\boldsymbol{x}}(\mathcal{S}) = \mathbb{E}[f(X) \mid X_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}]$, conditions on the observed features. Both require sampling from marginal or conditional distributions, incurring repeated model evaluations and a dependence on background samples that can substantially shape the resulting explanations [Molnar, 2025, Chapter 21].

Functional decomposition. An increasingly popular alternative defines the value function through a functional decomposition of the learned model [Gevaert and Saeys, 2024, Fumagalli et al., 2025, Mohammadi et al., 2026b], expressing f as a sum of components indexed by feature subsets, $f(\boldsymbol{x}) = \sum_{\mathcal{S} \subseteq \mathcal{D}} f_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}})$, where each $f_{\mathcal{S}}$ depends only on the variables in $\mathcal{S}$. The value function can then be defined as the Möbius transform of the functional components $\{f_S\}$ and thus $\nu_{\boldsymbol{x}}(S) = \sum_{\mathcal{T} \subseteq S} f_{\mathcal{T}}(\boldsymbol{x}_{\mathcal{T}})$, aggregating all sub-coalition effects of S.

This framing connects Shapley-based attribution to a century of statistical theory: ANOVA [Ståhle and Wold, 1989], Sobol indices [Owen, 2013, 2014], partial dependence [Berzal et al., 2002], and generalized additive models [Hastie, 2017] are all functional decomposition methods that long predate modern XAI. Writing f as $\sum_{\mathcal{S}} f_{\mathcal{S}}$ makes a *structural* claim about the model that can be inspected componentwise, and many desirable properties of attribution methods, analogues of Shapley's axioms stated directly in terms of f rather than an induced cooperative game, admit clean characterizations at the decomposition level [Gevaert and Saeys, 2024, Section 6.3].

Furthermore, Gevaert and Saeys [2024] sharpen this picture by establishing a correspondence between functional decompositions and *removal operators*—mappings that render the model independent of a given feature subset. Every removal operator induces a unique decomposition, and every decomposition arises this way. Designing a value function therefore reduces to a single modelling choice: pick a sensible removal operator and read off the decomposition. We adopt the removal-operator perspective. In the next section, we show that the natural removal operator for product-kernel models—replacing each absent kernel factor with the multiplicative identity—yields a parameter-free value function and induces a functional decomposition with a simple closed form.

3 Exact Shapley Value for Product-Kernel Learning Methods

This section develops an algorithm that computes all d Shapley values for product-kernel methods (e.g., support vector machines and kernel ridge regression) in $O(d^2 n)$ time, i.e., amortized linear time per feature, using a value function induced by a natural, distribution-free removal operator that requires neither sampling nor density estimation.

3.1 Value Function for Product-Kernel Methods

Product-kernel methods rely on kernel functions to capture complex relationships between input features and output. A kernel-based decision function is generally expressed as $f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \boldsymbol{\alpha}^\top k(\mathbf{X}, \boldsymbol{x})$ where k is a kernel function and α_i are the model-specific coefficients. When k is a product kernel, it can be expressed as: $k(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \prod_{j \in \mathcal{D}} k_j(x_j, x_j^{(i)})$. Product-kernel methods are widely used in ML for their simplicity and effectiveness in modelling similarities in high-dimensional data [Gardner et al., 2018]. They also come with strong theoretical guarantees: if the base kernels are universal—capable of approximating any continuous function on the marginal input—then the product kernel inherits this property [Szabó and Sriperumbudur, 2018]. Several widely used kernels are of product form, including the radial basis function (RBF) with isotropic or anisotropic bandwidth, $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\big(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/2\sigma^2\big) = \prod_{j=1}^{d} \exp\big(-(x_j - x_j')^2/2\sigma^2\big)$; further examples, as well as commonly used kernels that are *not* of product form, are given in Appendix A. We index the factors by individual features for notational simplicity, but nothing in what follows requires the factors to be univariate: each k_j may act on a single real-valued feature, on a group of features, or on structured inputs such as categorical variables or sequences. PKeX-Shapley attributes at the level of the factors, yielding feature-level attributions when each factor corresponds to one feature and group-level attributions otherwise. For product-kernel methods, we utilize the canonical neutral-factor removal: removing a feature subset $\mathcal{T}$ corresponds to replacing each kernel factor k_j, $j \in \mathcal{T}$, with the multiplicative identity 1, giving

$$P_{\mathcal{T}}(f)(\boldsymbol{x}) = \boldsymbol{\alpha}^\top k_{\mathcal{D}\setminus\mathcal{T}}(\mathbf{X}_{\mathcal{D}\setminus\mathcal{T}}, \boldsymbol{x}_{\mathcal{D}\setminus\mathcal{T}}).$$

This removal preserves the product-kernel structure and requires no background distribution, sampling, or density estimation—absent features contribute a neutral factor. By the canonical additive decomposition framework of Gevaert and Saeys [2024], once this removal family is chosen, it uniquely determines the corresponding functional decomposition via Möbius inversion on the subset lattice. The resulting components are given in Proposition 1.

Proposition 1. *The removal operator above induces the functional decomposition* $f = \sum_{\mathcal{S} \subseteq \mathcal{D}} f_{\mathcal{S}}$ *with components* $f_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \sum_{i=1}^{n} \alpha_i \prod_{j \in \mathcal{S}} \big(k_j(x_j, x_j^{(i)}) - 1\big)$, *where* $f_\emptyset = \sum_{i=1}^{n} \alpha_i$.

The value function for coalition $\mathcal{S}$ is obtained by applying the removal operator to the absent features.

Proposition 2. *The value function induced by the canonical neutral-factor removal is*

$$v_{\boldsymbol{x}}(\mathcal{S}) \;=\; \boldsymbol{\alpha}^\top k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}}) - f_\emptyset, \tag{2}$$

where $f_\emptyset = \sum_{i=1}^{n} \alpha_i$, *satisfying* $v_{\boldsymbol{x}}(\emptyset) = 0$ *and* $v_{\boldsymbol{x}}(\mathcal{D}) = f(\boldsymbol{x}) - f_\emptyset$.

Since $f_\emptyset$ is constant, it cancels in every marginal contribution $v_{\boldsymbol{x}}(\mathcal{S} \cup \{j\}) - v_{\boldsymbol{x}}(\mathcal{S})$ and has no effect on the Shapley values. We therefore drop it henceforth and work with $v_{\boldsymbol{x}}(\mathcal{S}) = \boldsymbol{\alpha}^\top k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}})$, which is exact and parameter-free: it requires no sampling and no density estimation, and can be evaluated in $O(|\mathcal{S}| \cdot n)$ time.

Why the neutral factor, and what the attributions measure. Three remarks delineate the scope of this value function (details in Appendix B). *(i) Functional Null.* Beyond algebraic convenience, $k_j \mapsto 1$ is the unique constant replacement satisfying the Functional Null axiom of Gevaert and Saeys [2024, Def. 10]: a feature the model does not depend on has a factor identically one, so removing it is a no-op and its Shapley value is zero, whereas any constant $c \neq 1$ can credit a feature the model ignores (Appendix B.1). *(ii) Factorization convention.* The attributions are relative to the factorization specified by the trained model: rewriting $k_0 k_1$ as $(\beta k_0)(\beta^{-1} k_1)$ changes them, since a constant moved between factors is no feature's contribution. We therefore absorb any constant scale into $\boldsymbol{\alpha}$ rather than into a kernel factor; this fixes the factorization uniquely, leaves the predictions unchanged, and affects neither the recursions nor their complexity that come next. *(iii) Correlated features.* As no feature distribution enters, the attributions are *model-based* rather than dependence-aware: a feature is credited for its direct functional contribution, not for being correlated with one the model uses, unlike under observational SHAP [Gevaert and Saeys, 2024, Sec. 5.3]. Neither choice is universally correct (Appendix B.3); empirically, near-duplicate features receive near-equal credit under ours (Appendix G.2).

Given this value function, we next derive an efficient algorithm to compute exact Shapley values.

3.2 Exact Shapley Values Computation

Building on the value function above, we now show that the Shapley value of any feature admits a recursive form. This recursion underlies the computational gains of our approach and provides the pathway to exact computation.

Theorem 3. *Let $\mathcal{Z} := \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d\}$ with $\boldsymbol{z}_i := k_i(\mathbf{X}_i, x_i) \in \mathbb{R}^n$, $\mathcal{Z}_{-j} = \mathcal{Z} \setminus \{\boldsymbol{z}_j\}$, and $e_q(\mathcal{Z})$ the elementary symmetric polynomials (ESPs) of order q over $\mathcal{Z}$, defined as*

$$e_q(\mathcal{Z}_{-j}) = \frac{1}{q} \sum_{r=1}^{q} (-1)^{r-1} e_{q-r}(\mathcal{Z}_{-j}) \odot p_r(\mathcal{Z}_{-j}),$$

where $p_r(\mathcal{Z}) = \sum_{\boldsymbol{z}_i \in \mathcal{Z}} \boldsymbol{z}_i^r$ is the element-wise degree-r power sum. For product-kernel learning methods with coefficients $\boldsymbol{\alpha}$, the Shapley value $\phi_j^{\boldsymbol{x}}$ for feature j of instance $\boldsymbol{x}$ under the functional decomposition value function can then be expressed as

$$\phi_j^{\boldsymbol{x}} = \boldsymbol{\alpha}^\top \big((\boldsymbol{z}_j - \mathbf{1}) \bigodot \sum_{q=0}^{d-1} \mu(q)\, e_q(\mathcal{Z}_{-j}) \big). \tag{3}$$

The recursion follows from Newton's identities for ESPs [Egge, 2019]; we defer the details to Appendix C. Intuitively, under the functional decomposition value function, the multiplicative structure of the product kernel allows us to factorize the marginal contribution $v_{\boldsymbol{x}}(\mathcal{S} \cup \{j\}) - v_{\boldsymbol{x}}(\mathcal{S})$ as $\boldsymbol{\alpha}^\top \big((k_j(\mathbf{X}_j, x_j) - \mathbf{1}) \odot k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}}) \big)$. This factorization lets us pull the sum over $\mathcal{S} \subseteq \mathcal{D} \setminus \{j\}$ inside the inner product with $\boldsymbol{\alpha}$ and express it directly in terms of weighted ESPs $e_q(\mathcal{Z}_{-j})$, where $\boldsymbol{z}_i = k_i(\mathbf{X}_i, x_i)$.

From one to all Shapley values. The recursion in Theorem 3 shows that a *single* Shapley value $\phi_j^{\boldsymbol{x}}$ can be computed in $O(d^2)$ time, since each e_q follows from the power sums $p_r(\mathcal{Z}_{-j})$ via Newton's identities. Applying this procedure independently to every $j \in \mathcal{D}$, however, ignores substantial overlap between the sets $\mathcal{Z}_{-j}$: the ESPs $e_q(\mathcal{Z}_{-j})$ across different j share many intermediate quantities, which can be reused.

To this end, we develop a numerically stable method that exploits structural properties of ESPs to compute the sum $\sum_q \mu(q) e_q(\mathcal{Z}_{-j})$ from equation (3). The key tool is the generating-polynomial representation of ESPs, which organizes interaction terms by degree and makes intermediate quantities reusable across features. For any ordered index set $\mathcal{A} \subseteq \mathcal{D}$, define $p_{\mathcal{A}}(s) := \bigodot_{i \in \mathcal{A}} \big(\mathbf{1} + \boldsymbol{z}_i\, s\big)$, where s is a univariate variable. Expanding the product gives $p_{\mathcal{A}}(s) = \sum_{q=0}^{|\mathcal{A}|} e_q(\mathcal{A})\, s^q$, so the coefficient of s^q is the degree-q ESP over the vectors indexed by $\mathcal{A}$. For a fixed index $j \in \mathcal{D}$, we construct the ESPs over $\mathcal{Z}_{-j}$ in (3) by splitting the remaining vectors into a *prefix* (indices before j) and a *suffix* (indices after j):

$$p_{j-1}^{\mathrm{pref}}(s) := \bigodot_{i=1}^{j-1} \big(\mathbf{1} + \boldsymbol{z}_i\, s\big) = \sum_{t=0}^{j-1} \mathbf{\Gamma}_{j-1,t}\, s^t, \qquad p_{j+1}^{\mathrm{suf}}(s) := \bigodot_{i=j+1}^{d} \big(\mathbf{1} + \boldsymbol{z}_i\, s\big) = \sum_{u=0}^{d-j} \mathbf{R}_{j+1,u}\, s^u,$$

where $\mathbf{\Gamma}_{j-1,t}$ and $\mathbf{R}_{j+1,u}$ are the element-wise ESPs of degrees t and u over the vectors preceding and following index j, respectively. The leave-one-out polynomial then factorizes as $p_{-j}(s) = p_{j-1}^{\mathrm{pref}}(s) \odot p_{j+1}^{\mathrm{suf}}(s)$, involving only vectors in $\mathcal{Z}_{-j}$ by construction.

Shapley value computation thus reduces to evaluating the inner sum of weighted degree-q ESPs. The following proposition shows that prefix–suffix polynomial constructions enable the reuse of intermediate ESP quantities across features, yielding efficient computation of all Shapley values.

Proposition 4. *Let $\mathbf{\Gamma}_{j-1,t}$ and $\mathbf{R}_{j+1,u}$ be as above, and $\mathbf{G}_{j+1,t} := \sum_{u=0}^{d-j} \mu(t+u)\, \mathbf{R}_{j+1,u}$. Then $\sum_{q=0}^{d-1} \mu(q)\, e_q(\mathcal{Z}_{-j}) = \sum_{t=0}^{j-1} \mathbf{\Gamma}_{j-1,t} \odot \mathbf{G}_{j+1,t}$.*

Time and memory complexity. Proposition 4 reduces the computation of all Shapley values to constructing the prefix coefficients $\mathbf{\Gamma}_{j,t}$ and the aggregated suffix quantities $\mathbf{G}_{j,u}$ for $j \in \mathcal{D}$, $t < j$, $u \leq d - j$: given these, each Shapley value can be obtained in linear time. We now show that both quantities admit simple recursive constructions and can themselves be computed in $O(d^2)$ time.

Proposition 5. *The prefix coefficients $\mathbf{\Gamma} = \{\mathbf{\Gamma}_{j,t}\}$ and aggregated suffix quantities $\mathbf{G} = \{\mathbf{G}_{j,t}\}$ can be computed and stored in $O(d^2 n)$ time and $O(d^2 n)$ memory.*

Proposition 5 shows that the prefix and suffix ESP coefficients can be computed in $O(d^2 n)$ time, after which each Shapley value $\phi_j^{\boldsymbol{x}}$ is obtained in $O(dn)$ time. **Computing all d Shapley values therefore costs $O(d^2 n)$ in total, or $O(dn)$ per feature on average.**

Memory and blockwise computation. The $O(d^2 n)$ memory in Proposition 5 is not intrinsic: the recursions are independent across kernel centres, since the contribution of centre i involves only the i-th entries of $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d$. Processing the active centres in blocks of size B—computing the prefix and aggregated suffix tables for the block, contracting them with the corresponding α_i, accumulating the d-dimensional partial Shapley vector, and discarding the tables—is exact, keeps the $O(d^2 n)$ time, and reduces the working memory to $O(Bd^2)$. Moreover, only centres with $\alpha_i \neq 0$ contribute, so for support vector machines n is the number of support vectors. On seven real datasets, blockwise PKeX-Shapley with $B = 256$ returns identical values while cutting peak memory by up to $8.9\times$ at a modest runtime cost (Appendix F).

Comparison to TreeSHAP. A value function with similar multiplicative structure arises in TreeSHAP, where interpolation-based algorithms recover leave-one-out ESPs via polynomial division [Yu et al., 2022, Jiang et al., 2025]. These methods work well when the polynomial degree is bounded by the tree depth $D \ll d$, giving Linear TreeSHAP an $\mathcal{O}(DL)$ per-tree complexity. For product-kernel methods, however, the polynomial degree is $d - 1$, and interpolation becomes numerically unstable already at moderate dimensions (e.g., $d \approx 50$); see Appendix E. PKeX-Shapley avoids this by relying only on element-wise additions and multiplications, remaining stable at large d with $\mathcal{O}(d^2 n)$ total cost. Notably, applying Linear TreeSHAP to kernel methods would view the model as an ensemble of n trees—one per training point, each with depth and leaf count d—also yielding $\mathcal{O}(d^2 n)$ overall, but at the cost of the numerical instability above.

4 Explaining kernel-based statistical discrepancies

We extend the removal-operator framework to two widely used kernel-based statistical discrepancies: the *Maximum Mean Discrepancy* (MMD) and the *Hilbert–Schmidt Independence Criterion* (HSIC). The same principle applies in both cases: removing a feature replaces its kernel factor with the multiplicative identity, yielding a parameter-free value function in closed form. We state the resulting Shapley formulations in terms of leave-one-out ESPs via Newton's identities for readability; in practice we compute them with the prefix–suffix construction of Propositions 4–5, which transfers verbatim with vectors replaced by matrices of pairwise kernel evaluations (see the end of this section).

4.1 Distributing the discrepancy: Explaining MMD

The MMD quantifies the difference between two probability distributions in terms of their kernel mean embeddings: $\mathrm{MMD}^2(\mathbb{P}, \mathbb{Q}) := \|\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\|^2_{\mathcal{H}_k}$ where $\mathcal{H}_k$ is the RKHS associated with the kernel k and $\mu_{\mathbb{P}} := \int k(\boldsymbol{x}, \cdot)\, d\mathbb{P}(\boldsymbol{x}) \in \mathcal{H}_k$ is the kernel mean embedding of $\mathbb{P}$ [Muandet et al., 2017, Sec. 3.5]. The embedding $\mu_{\mathbb{Q}}$ is defined analogously. Based on the samples $\{\boldsymbol{x}^{(i)}\}_{i=1}^n \sim \mathbb{P}$ and $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim \mathbb{Q}$, the empirical estimate of MMD, expressed entirely in terms of k, is given by

$$\widehat{\mathrm{MMD}}^2(\mathbb{P}, \mathbb{Q}) = \frac{1}{n(n-1)} \sum_{i \neq j} k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) + \frac{1}{m(m-1)} \sum_{i \neq j} k(\boldsymbol{z}^{(i)}, \boldsymbol{z}^{(j)}) - \frac{2}{nm} \sum_{i,j} k(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(j)}).$$

In this section, we propose an attribution method that allocates the overall discrepancy measured by MMD among the variables involved. Such attributions are useful in many settings, including explaining MMD-based test statistics in hypothesis testing [Gretton et al., 2012] and identifying the variables driving covariate shift [Zhang et al., 2020]. For a product kernel k, removing a feature subset $\mathcal{T}$ replaces each factor k_q, $q \in \mathcal{T}$, with 1, leaving only the sub-kernel $k_{\mathcal{D}\setminus\mathcal{T}}$. Applied to MMD, this removal induces a natural, parameter-free value function, and the corresponding functional decomposition is uniquely determined.

Theorem 6. *For product kernel MMD, the removal operator $k_q \mapsto 1$ for $q \in \mathcal{T}$ induces the following:*

(i) The value function representing the contribution of the variables in $\mathcal{S}$ is $v_{MMD}(\mathcal{S}) = \frac{1}{n(n-1)} \sum_{i \neq j} k_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}^{(i)}, \boldsymbol{x}_{\mathcal{S}}^{(j)}) + \frac{1}{m(m-1)} \sum_{i \neq j} k_{\mathcal{S}}(\boldsymbol{z}_{\mathcal{S}}^{(i)}, \boldsymbol{z}_{\mathcal{S}}^{(j)}) - \frac{2}{nm} \sum_{i,j} k_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}^{(i)}, \boldsymbol{z}_{\mathcal{S}}^{(j)})$.

(ii) The MMD function admits the unique functional decomposition into subset contributions induced by this removal operator:

$$\widehat{MMD}^2(\mathbb{P},\mathbb{Q}) = \sum_{\mathcal{S}\subseteq\mathcal{D}} \Bigg[\frac{1}{n(n-1)} \sum_{i\neq j} \prod_{q\in\mathcal{S}} \left(k_q(x_q^{(i)}, x_q^{(j)}) - 1\right) + \frac{1}{m(m-1)} \sum_{i\neq j} \prod_{q\in\mathcal{S}} \left(k_q(z_q^{(i)}, z_q^{(j)}) - 1\right) - \frac{2}{nm} \sum_{i,j} \prod_{q\in\mathcal{S}} \left(k_q(x_q^{(i)}, z_q^{(j)}) - 1\right) \Bigg].$$

Applying the same construction as in Theorem 3 to v_{MMD} yields an analogous recursive formulation. For the first term in v_{MMD}, the multiplicative structure of the product kernel gives $v_{\mathrm{MMD}}(\mathcal{S}\cup\{q\}) - v_{\mathrm{MMD}}(\mathcal{S}) = k_{\mathcal{S}\cup\{q\}}(\boldsymbol{x}^{(i)}_{\mathcal{S}\cup\{q\}}, \boldsymbol{x}^{(j)}_{\mathcal{S}\cup\{q\}}) - k_{\mathcal{S}}(\boldsymbol{x}^{(i)}_{\mathcal{S}}, \boldsymbol{x}^{(j)}_{\mathcal{S}}) = \left(k_q(x_q^{(i)}, x_q^{(j)}) - 1\right) k_{\mathcal{S}}(\boldsymbol{x}^{(i)}_{\mathcal{S}}, \boldsymbol{x}^{(j)}_{\mathcal{S}})$. Pulling $k_q(x_q^{(i)}, x_q^{(j)}) - 1$ outside the summation in the Shapley value lets us express the total sum over $\mathcal{S}\subseteq\mathcal{D}\setminus\{q\}$ as a weighted ESP. The following proposition formalizes this.

Proposition 7. *Let $\mathcal{Z}^{(\boldsymbol{x},\boldsymbol{x}')} = \{k_1(x_1, x_1'), \ldots, k_d(x_d, x_d')\}$, and $e_r(\mathcal{Z}^{(\boldsymbol{x},\boldsymbol{x}')}_{-q})$ determined as*

$$e_r(\mathcal{Z}^{(\boldsymbol{x},\boldsymbol{x}')}_{-q}) = \frac{1}{r}\sum_{s=1}^{r}(-1)^{s-1} e_{r-s}(\mathcal{Z}^{(\boldsymbol{x},\boldsymbol{x}')}_{-q})\, p_s(\mathcal{Z}^{(\boldsymbol{x},\boldsymbol{x}')}_{-q}),$$

where $p_s(\mathcal{Z}) = \sum_{z\in\mathcal{Z}} z^s$ represents the degree-s power sum. Further, let $\gamma_q(\boldsymbol{x},\boldsymbol{x}')$ be defined as $\gamma_q(\boldsymbol{x},\boldsymbol{x}') := (k_q(x_q, x_q') - 1)\sum_{r=0}^{d-1}\mu(r) e_r(\mathcal{Z}^{(\boldsymbol{x},\boldsymbol{x}')}_{-q})$. Then, for product kernels, the Shapley value for the MMD can be recursively computed as

$$\phi_q^{MMD} = \tfrac{1}{n(n-1)}\sum_{i\neq j}\gamma_q(\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}) + \tfrac{1}{m(m-1)}\sum_{i\neq j}\gamma_q(\boldsymbol{z}^{(i)},\boldsymbol{z}^{(j)}) - \tfrac{2}{nm}\sum_{i,j}\gamma_q(\boldsymbol{x}^{(i)},\boldsymbol{z}^{(j)}).$$

The Shapley values ϕ_q^{MMD} allow us to allocate the overall distributional discrepancy between $\mathbb{P}$ and $\mathbb{Q}$ across the variables, identifying the most influential ones in distinguishing the two distributions.

4.2 Distributing the dependence: Explaining HSIC

The HSIC is a kernel-based dependence measure between two random variables. Let X and Y be random variables equipped with reproducing kernels $k(\cdot,\cdot)$ and $l(\cdot,\cdot)$, respectively, where k is a product kernel; l can be any valid kernel, since it enters only through the fixed matrix $\mathbf{L}$ below. Then $\mathrm{HSIC}(X,Y) := \|\mathcal{C}_{XY}\|^2_{\mathrm{HS}}$, where $\mathcal{C}_{XY}$ is the cross-covariance operator and $\|\cdot\|_{\mathrm{HS}}$ is the Hilbert–Schmidt (HS) norm; see Muandet et al. [2017, Sec. 3.6] for technical details. Given a sample $\{(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})\}_{i=1}^n \sim \mathbb{P}(X,Y)$, HSIC admits the empirical estimator $\widehat{\mathrm{HSIC}}(X,Y) = \frac{1}{(n-1)^2}\,\mathrm{tr}(\mathbf{KHLH})$, where $\mathbf{K},\mathbf{L}\in\mathbb{R}^{n\times n}$ are the kernel matrices with $\mathbf{K}_{ij} = k(\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)})$ and $\mathbf{L}_{ij} = l(\boldsymbol{y}^{(i)},\boldsymbol{y}^{(j)})$, and $\mathbf{H} = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ is the centreing matrix. Computing $\mathrm{HSIC}(X,Y)$ gives the overall dependence between X and Y. We now consider the setting where X is a feature vector $\boldsymbol{x}$ and Y is a scalar outcome y, and ask how the overall dependence can be distributed across the features of $\boldsymbol{x}$. This is particularly useful for feature selection and global sensitivity analysis, where the target is typically univariate.

For a product kernel k on X, the kernel matrix factorizes element-wise as $\mathbf{K} = \bigodot_{j\in\mathcal{D}}\mathbf{K}_j$, where $(\mathbf{K}_j)_{ab} = k_j(x_j^{(a)}, x_j^{(b)})$. Removing feature j corresponds to replacing $\mathbf{K}_j$ with $\mathbf{1}\mathbf{1}^\top$, the all-ones matrix, which is the element-wise multiplicative identity ($\mathbf{A}\odot\mathbf{1}\mathbf{1}^\top = \mathbf{A}$) and the kernel matrix for the constant kernel $k_j\equiv 1$. Applied to HSIC, this yields a natural, parameter-free value function, and the corresponding functional decomposition is uniquely determined by this removal operator.

Theorem 8. *For the HSIC with the product kernel on X, the removal operator $\mathbf{K}_j \mapsto \mathbf{1}\mathbf{1}^\top$ for $j\notin\mathcal{S}$ induces the following:*

(i) The value function that quantifies the contribution of the variable set $\mathcal{S}$ to the overall dependence is $v_{HSIC}(\mathcal{S}) = (n-1)^{-2}\mathrm{tr}\big(\mathbf{HLHK}_{\mathcal{S}}\big)$.

(ii) The HSIC function admits the unique functional decomposition into subset contributions induced by this removal operator: $\widehat{HSIC}(X,Y) = \frac{1}{(n-1)^2}\sum_{\mathcal{S}\subseteq\mathcal{D}}\mathrm{tr}\left(\mathbf{HLH}\bigodot_{j\in\mathcal{S}}(\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top)\right)$.

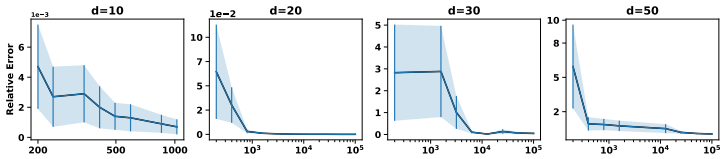

Figure 2: Relative estimation error of regression-based Shapley values versus the exact recursive values, shown across coalition sample sizes for feature dimensions $d = \{10, 20, 30, 50\}$.

Applying the same trick as in Proposition 7 to the value function v_{HSIC} yields a similar recursive formulation of Shapley values for HSIC.

Proposition 9. *For the product kernel, the Shapley value for HSIC can be recursively computed as:*

$$\phi_j^{HSIC} = \frac{1}{(n-1)^2}\operatorname{tr}\left(\mathbf{HLH}\left((\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top)\bigodot \sum_{q=0}^{d-1}\mu(q)E_q(\mathcal{K}_{-j})\right)\right).$$

where $\mathcal{K} = \{\mathbf{K}_1, \ldots, \mathbf{K}_d\}$ with $\mathbf{K}_i$ being the kernel matrix for feature i only, and $E_q(\mathcal{K}_{-j}) = \frac{1}{q}\sum_{r=1}^{q}(-1)^{r-1}E_{q-r}(\mathcal{K}_{-j})\bigodot P_r(\mathcal{K}_{-j})$, is the ESPs with $P_r(\mathcal{K}) = \sum_{\mathbf{K}_i\in\mathcal{K}}\mathbf{K}_i^{\odot r}$ being the element-wise degree-r power sum, where $\mathbf{K}_i^{\odot r}$ denotes the element-wise (Hadamard) r-th power of $\mathbf{K}_i$.

The Shapley values ϕ_j^{HSIC} allow us to allocate the overall dependence between X and Y across individual features fairly, identifying the most influential ones for prediction. Our results can be generalized to scenarios when both X and Y are multivariate. The attribution of overall dependence to the involved multivariate variables has other applications in statistical inference as well, e.g., kernel-based (conditional) independence testing [Albert et al., 2022, Gretton et al., 2007].

Time and memory complexity. Each kernel factor now contributes a matrix of pairwise evaluations rather than a length-n vector, and the recursions act independently on every sample pair. With n samples per distribution (resp. n paired observations), all d attributions cost $O(d^2n^2)$ time; a fully materialized prefix–suffix table needs $O(d^2n^2)$ memory, which the blockwise scheme of Section 3, applied to blocks of B pairs weighted by the MMD or HSIC coefficients, reduces to $O(Bd^2)$ without changing the time complexity or the exactness (Appendix F).

5 Experiments

We demonstrate the effectiveness of PKeX-Shapley for product-kernel methods through a series of experiments conducted on a 24-core machine with 16GB RAM. We further provide the experimental setups, training procedure, and extra experiments in Appendix G, including a feature selection case study using HSIC (Appendix G.4).

5.1 Effectiveness of Recursion in Local Explanations

Recursion vs. regression formulation. We first compare PKeX-Shapley with sampling-based approximations such as Kernel SHAP on four synthetic datasets, each with 1000 samples and $d \in \{10, 20, 30, 50\}$ features drawn independently from a standard normal distribution. Labels follow a linear model with additive Gaussian noise ($\sigma = 0.1$). For each dataset, we train a support vector regression model with an RBF kernel and use it to compare explanation methods.

For a fair comparison, both methods use the same value function. We compute the exact Shapley values $\phi_j^{\boldsymbol{x}}$ with PKeX-Shapley, then approximate them using a regression-based estimator analogous to Kernel SHAP. The estimator follows the paired coalition sampling strategy of Covert and Lee [2021], draws between 200 and 10^5 coalitions, and obtains $\hat{\phi}_j^{\boldsymbol{x}}$ by solving the resulting weighted linear regression. We report the average relative deviation $\sum_{j=1}^{d}|\phi_j^{\boldsymbol{x}} - \hat{\phi}_j^{\boldsymbol{x}}|/|\phi_j^{\boldsymbol{x}}|$ over 100 randomly selected instances.

Figure 2 plots the relative error against sample size for each dataset, averaged over the 100 instances. At $d = 10$, 200 samples (roughly 20% of all possible coalitions) already drive the error to about 0.005. At $d = 50$, by contrast, 200 samples leave the error above 9.0; 10^4 samples still leave it

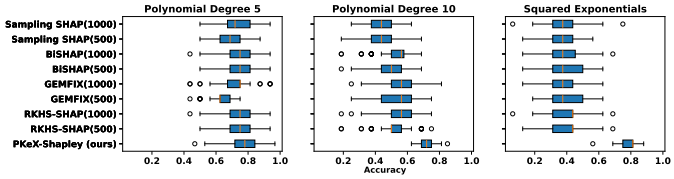

Figure 3: Recovery rate of true active features by each method on the three synthetic tasks.

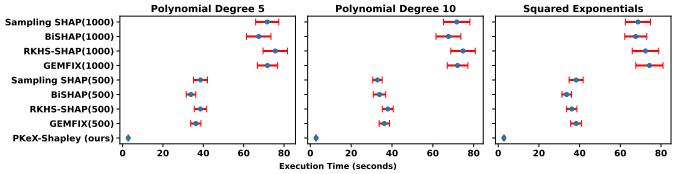

Figure 4: Average Per-instance explanation time for each method at 500 and 1000 coalition samples.

above 1.0; and only at 10^5 does it approach 0.05. Sampling-based methods thus become unreliable in high-dimensional settings, underscoring the value of fast exact computation.

Effectiveness of PKeX-Shapley. We evaluate the effectiveness of PKeX-Shapley through three *synthetic* regression datasets with $n = 1000$ samples in $\mathbb{R}^{50}$ where only the first 17 features (denoted by $\mathcal{S}$) influence the response, and the remaining 33 features are redundant. The target functions over $\mathcal{S}$ are: a degree-5 polynomial, a degree-10 polynomial, and a squared-exponential response of the form $y = \exp\big(-\sum_{i\in\mathcal{S}} x_i^2\big)$. For each dataset, we train a support vector regressor with an RBF kernel and compute feature attributions using PKeX-Shapley alongside several baselines: RKHS-SHAP [Chau et al., 2022], GEMFIX [Mohammadi et al., 2025], BiSHAP [Masoomi et al., 2022], and Sampling SHAP [Štrumbelj and Kononenko, 2014], with 500 and 1000 coalition samples. All baseline methods use a fixed background set of 100 points for value-function estimation. The baselines thus estimate their standard interventional value functions, whereas PKeX-Shapley uses the neutral-factor one, so the methods do not target identical Shapley values; this is an end-to-end feature-recovery comparison in which all methods explain the same model and are scored against the same external ground truth (the controlled same-value-function comparison is Figure 2).

Attribution accuracy is evaluated over 100 independent test instances by selecting the top-17 features ranked by each method and computing the fraction of truly active features recovered. Figure 3 plots the average accuracy across the three tasks. PKeX-Shapley achieves competitive or superior performance in all settings. In contrast, RKHS-SHAP, GEMFIX, and Sampling SHAP exhibit decreasing accuracy as the complexity of the target function increases, especially for the degree-10 polynomial and the exponential tasks.

Figure 4 reports per-instance execution times. Baselines use their authors' public implementations (Sampling SHAP via the official `shap` package), unmodified and with their native vectorization, on the same model, instances, background set, and hardware; only the attribution stage is timed. PKeX-Shapley is consistently much faster than all competing methods, even when the baselines use only 500 coalition samples: computing the full set of Shapley values takes, on average, slightly more than two seconds per instance, versus on the order of 30 seconds for the sampling-based methods, and the gap widens further as the number of coalition samples increases.

Real-world datasets. Real data provide no ground-truth Shapley values, so we complement the synthetic benchmarks with three checks on seven UCI regression datasets ($17 \le d \le 81$) with RBF support vector regressors; protocols and full tables are in Appendix G.2. First, despite optimizing different value functions, PKeX-Shapley and RKHS-SHAP select substantially overlapping important features (mean overlap 0.67 for the top-20% and 0.80 for the top-30% features), so the neutral-factor value function does not lead to qualitatively different conclusions. Second, five planted Gaussian null features receive 0.18–0.49 of the mean absolute attribution of the real features (lower than RKHS-SHAP on four datasets, higher on three), and when the two most important features are duplicated up to small noise, the original receives 47–63% of the pair's attribution (ideal: 50%), closer to an even split than RKHS-SHAP on every dataset. Third, exact blockwise PKeX-Shapley ($B = 256$) is 46×–303× faster than RKHS-SHAP with 500 coalition samples and uses 2.8×–10.5× less peak memory (Table 7).

5.2 Explaining Distribution Discrepancy using MMD with PKeX-Shapley

To illustrate how PKeX-Shapley explains distributional discrepancies measured by MMD, we attribute the MMD between two distributions to individual input variables in two synthetic experiments following the standard two-sample testing setup [Schrab, 2025]; one is presented below and the other in Appendix G.3. All MMD experiments use the RBF kernel with the bandwidth selected via the median heuristic. We generate datasets X and Z, each comprising 20 variables. The first ten variables $X_1, \ldots, X_{10}$ and $Z_1, \ldots, Z_{10}$ are sampled from the same multivariate normal distribution, ensuring identical distributions. The remaining 10 variables $X_{11}, \ldots, X_{20}$ differ, with variables in X sampled from a multivariate normal distribution and those in Z from a Student's t-distribution of the same mean. This introduces differences in higher-order moments and covariance structure, resulting in a measurable discrepancy.

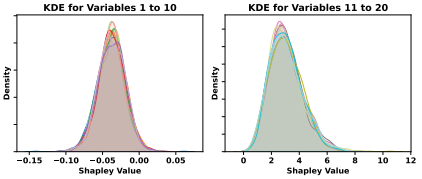

Figure 5: KDE of Shapley values. Variables 1–10 have negative Shapley values (lowering MMD); 11–20 are positive (raising discrepancy).

The left subplot in Figure 5 displays the KDE plots for variables 1–10, while the right subplot corresponds to variables 11–20. For the first 10 variables, the consistently negative Shapley values indicate a reduction in the overall MMD, effectively "pulling" the discrepancy closer to a lower value, as expected for variables that are identically distributed across both datasets. In contrast, variables 11–20 exhibit positive Shapley values, reflecting their contribution to the increase in MMD and, thus, their role in capturing the divergence between the distributions. Moreover, the KDE plots of the Shapley values within each group (variables 1–10 and 11–20, respectively) are nearly identical, which is consistent with the symmetric construction of the synthetic data.

6 Conclusion, Limitation, and Discussion

This work introduces PKeX-Shapley, an algorithm that computes exact Shapley values for all d features of product-kernel methods in $O(d^2 n)$ time—amortised linear time per feature. The method rests on a distribution-free removal operator intrinsic to the product-kernel structure, replacing each absent feature's kernel factor with the multiplicative identity, which yields a parameter-free value function requiring no sampling and uniquely determines the corresponding functional decomposition of the model. The framework extends to kernel-based statistical discrepancies—MMD and HSIC—enabling exact and efficient interpretability of distributional differences and dependence.

The limitations of our approach concern its scope. First, the algorithm applies only to product kernels. This trade-off is largely unavoidable: tractable computation requires structural constraints (Appendix A lists common kernels that are and are not of product form). Second, the attributions are exact for the neutral-factor value function and relative to the model's factorization (Section 3); interventional and observational SHAP, retraining-based games, and performance-attribution games such as Shapley value regression [Lipovetsky and Conklin, 2001] lack the multiplicative structure our recursions exploit and remain outside their scope; follow-up work shows that the empirical interventional value function can nevertheless be written as a weighted sum of product games and handled, up to a prescribed tolerance, by quadrature [Mohammadi et al., 2026c]. Third, PKeX-Shapley is post-hoc: kernel hyperparameters such as bandwidths or ARD length-scales are fixed at training time and the explainer adds none; changing them changes the model and hence its attributions, a sensitivity that belongs to model training rather than to the explanation algorithm. We see several directions for future work: relaxing the product-kernel assumption without sacrificing tractability; extending the approach to higher-order attribution methods such as Shapley interaction indices [Sundararajan et al., 2020, Muschalik et al., 2024]; and investigating whether our computational techniques can accelerate other explanation methods, including partial dependence plots and integrated gradients.

Acknowledgments

This work was funded by the German Federal Ministry of Education and Research (BMBF). The first author was partially supported by a DAAD Postdoc-NeT-AI short-term research visit scholarship. We

thank the anonymous reviewers and the area chair for their constructive comments, which substantially improved the paper.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

Mélisande Albert, Béatrice Laurent, Amandine Marrel, and Anouar Meynaoui. Adaptive test of independence based on HSIC measures. *The Annals of Statistics*, 50(2):858–879, 2022.

F Berzal, Juan C Cubero, Fernando Cuenca, and Juan Miguel Medina. Relational decomposition through partial functional dependencies. *Data & Knowledge Engineering*, 43(2):207–234, 2002.

Siu Lun Chau, Robert Hu, Javier Gonzalez, and Dino Sejdinovic. RKHS-SHAP: Shapley values for kernel methods. *Advances in neural information processing systems*, 35:13050–13063, 2022.

Siu Lun Chau, Krikamol Muandet, and Dino Sejdinovic. Explaining the uncertain: Stochastic Shapley values for Gaussian process models. *Advances in Neural Information Processing Systems*, 36: 50769–50795, 2023.

Siu Lun Chau, Antonin Schrab, Arthur Gretton, Dino Sejdinovic, and Krikamol Muandet. Credal two-sample tests of epistemic uncertainty. In *International Conference on Artificial Intelligence and Statistics*, pages 127–135. PMLR, 2025.

Hugh Chen, Joseph D Janizek, Scott Lundberg, and Su-In Lee. True to the model or true to the data? *arXiv preprint arXiv:2006.16234*, 2020.

Kacper Chwialkowski, Heiko Strathmann, and Arthur Gretton. A kernel test of goodness of fit. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2606–2615, 2016.

Héctor Climente-González, Chloé-Agathe Azencott, Samuel Kaski, and Makoto Yamada. Block HSIC Lasso: model-free biomarker detection for ultra-high dimensional data. *Bioinformatics*, 35 (14):i427–i435, 2019.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

Ian Covert and Su-In Lee. Improving KernelSHAP: Practical Shapley value estimation using linear regression. In *International Conference on Artificial Intelligence and Statistics*, pages 3457–3465. PMLR, 2021.

Ian Covert, Scott M Lundberg, and Su-In Lee. Understanding global feature contributions with additive importance measures. *Advances in Neural Information Processing Systems*, 33:17212–17223, 2020.

David K Duvenaud, Hannes Nickisch, and Carl Rasmussen. Additive Gaussian processes. *Advances in neural information processing systems*, 24, 2011.

Eric S Egge. *An introduction to symmetric functions and their combinatorics*, volume 91. American Mathematical Soc., 2019.

Fabian Fumagalli, Maximilian Muschalik, Eyke Hüllermeier, Barbara Hammer, and Julia Herbinger. Unifying feature-based explanations with functional ANOVA and cooperative game theory. In *International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*. PMLR, 2025.

Jacob Gardner, Geoff Pleiss, Ruihan Wu, Kilian Weinberger, and Andrew Wilson. Product kernel interpolation for scalable Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 1407–1416. PMLR, 2018.

Arne Gevaert and Yvan Saeys. Unifying attribution-based explanations using functional decomposition. *arXiv preprint arXiv:2412.13623*, 2024.

Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19, 2006.

Arthur Gretton, Kenji Fukumizu, Choon Teo, Le Song, Bernhard Schölkopf, and Alex Smola. A kernel statistical test of independence. *Advances in neural information processing systems*, 20, 2007.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

Trevor J Hastie. Generalized additive models. *Statistical models in S*, pages 249–307, 2017.

Dominik Janzing, Lenon Minorics, and Patrick Blöbaum. Feature relevance quantification in explainable ai: A causal problem. In *International Conference on artificial intelligence and statistics*, pages 2907–2916. PMLR, 2020.

Zhongli Jiang, Min Zhang, and Dabao Zhang. Fast calculation of feature contributions in boosting trees. In *Proceedings of the Forty-First Conference on Uncertainty in Artificial Intelligence*, pages 1859–1875, 2025.

I Elizabeth Kumar, Suresh Venkatasubramanian, Carlos Scheidegger, and Sorelle Friedler. Problems with Shapley-value-based explanations as feature importance measures. In *International conference on machine learning*, pages 5491–5500. PMLR, 2020.

Stan Lipovetsky and Michael Conklin. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17(4):319–330, 2001.

Feng Liu, Wenkai Xu, Jie Lu, Guangquan Zhang, Arthur Gretton, and Danica J Sutherland. Learning deep kernels for non-parametric two-sample tests. In *International conference on machine learning*, pages 6316–6326. PMLR, 2020.

Qiang Liu, Jason Lee, and Michael Jordan. A kernelized stein discrepancy for goodness-of-fit tests. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 276–284, 2016.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 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.

Aria Masoomi, Davin Hill, Zhonghui Xu, Craig P Hersh, Edwin K Silverman, Peter J Castaldi, Stratis Ioannidis, and Jennifer Dy. Explanations of black-box models based on directional feature interactions. In *International Conference on Learning Representations*, 2022.

Jovana Mitrovic, Dino Sejdinovic, and Yee Whye Teh. Causal inference via kernel deviance measures. *Advances in neural information processing systems*, 31, 2018.

Majid Mohammadi, Ilaria Tiddi, and Annette Ten Teije. Unlocking the game: Estimating games in möbius representation for explanation and high-order interaction detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 19512–19519, 2025.

Majid Mohammadi, Siu Lun Chau, and Krikamol Muandet. PKeX-Shapley: Python implementation and experiments. https://github.com/Majeed7/PKeX-Shapley, 2026a.

Majid Mohammadi, Krikamol Muandet, Ilaria Tiddi, Annette Ten Teije, and Siu Lun Chau. Exact Shapley attributions in quadratic-time for FANOVA Gaussian processes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026b.

Majid Mohammadi, Grigory Reznikov, Pavel Sinitcyn, Krikamol Muandet, and Siu Lun Chau. QuadraSHAP: ϵ-exact Shapley values for product games in logarithmic parallel time. *arXiv preprint arXiv:2605.05870*, 2026c.

Christoph Molnar. *Interpreting machine learning models with SHAP: a guide with Python examples and theory on Shapley values*. Christoph Molnar, 2025.

Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, Bernhard Schölkopf, et al. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends® in Machine Learning*, 10(1-2):1–141, 2017.

Maximilian Muschalik, Hubert Baniecki, Fabian Fumagalli, Patrick Kolpaczki, Barbara Hammer, and Eyke Hüllermeier. shapiq: Shapley interactions for machine learning. *Advances in Neural Information Processing Systems*, 37:130324–130357, 2024.

Masha Naslidnyk, Siu Lun Chau, Francois-Xavier Briol, and Krikamol Muandet. Kernel quantile embeddings and associated probability metrics. In *International Conference on Machine Learning*, pages 45770–45800. PMLR, 2025.

Art B Owen. Variance components and generalized Sobol' indices. *SIAM/ASA Journal on Uncertainty Quantification*, 1(1):19–41, 2013.

Art B Owen. Sobol' indices and Shapley value. *SIAM/ASA Journal on Uncertainty Quantification*, 2 (1):245–251, 2014.

Antonin Schrab. A unified view of optimal kernel hypothesis testing. *arXiv preprint arXiv:2503.07084*, 2025.

Lloyd S. Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker, editors, *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematics Studies*, pages 307–317. Princeton University Press, Princeton, NJ, 1953.

Lars Ståhle and Svante Wold. Analysis of variance (ANOVA). *Chemometrics and intelligent laboratory systems*, 6(4):259–272, 1989.

Mark Stitson, Alex Gammerman, Vladimir Vapnik, Volodya Vovk, Chris Watkins, and Jason Weston. Support vector regression with ANOVA decomposition kernels. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 285–291. MIT Press, 1999.

Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41:647–665, 2014.

Mukund Sundararajan and Amir Najmi. The many Shapley values for model explanation. In *International conference on machine learning*, pages 9269–9278. PMLR, 2020.

Mukund Sundararajan, Kedar Dhamdhere, and Ashish Agarwal. The Shapley Taylor interaction index. In *International conference on machine learning*, pages 9259–9268. PMLR, 2020.

Zoltán Szabó and Bharath K Sriperumbudur. Characteristic and universal tensor product kernels. *Journal of Machine Learning Research*, 18(233):1–29, 2018.

Peng Yu, Albert Bifet, Jesse Read, and Chao Xu. Linear TreeShap. *Advances in Neural Information Processing Systems*, 35:25818–25828, 2022.

Wenju Zhang, Xiang Zhang, Long Lan, and Zhigang Luo. Maximum mean and covariance discrepancy for unsupervised domain adaptation. *Neural Processing Letters*, 51:347–366, 2020.

Yuqi Zhang, Krikamol Muandet, Dino Sejdinovic, Edwin Fong, and Siu Lun Chau. Instrumental and proximal causal inference with Gaussian processes. *arXiv preprint arXiv:2603.02159*, 2026.

Appendix

The appendix provides additional information and proofs related to the material presented in the main paper. It includes detailed explanations, proofs, algorithms, and experiments relevant to explaining product-kernel models. The structure of the appendix is as follows:

- **Product kernels in kernel methods**: §A discusses the construction and examples of product kernels, including their definitions and properties, and lists common kernels that are not of product form.
- **The neutral-factor removal operator**: §B justifies the choice $k_j \mapsto 1$ via the Functional Null axiom, states the factorization convention, discusses the interpretation under correlated features, and lists value functions outside the scope of PKeX-Shapley.
- **Newton's identities** : §C presents the main result of Newton's identities for elementary symmetric polynomials.
- **Proofs**: §D provides detailed proofs of key theorems and lemmas used in the main paper.
- **Interpolation-Based ESP Recovery Methods**: Section E investigates whether interpolation-based approaches, similar to those used in TreeSHAP, can be applied to product-kernel methods and examines the numerical stability of the resulting algorithms.
- **Blockwise computation and memory**: §F describes the exact blockwise implementation of PKeX-Shapley, its memory complexity for predictive models, MMD, and HSIC, and measures peak memory on real datasets.
- **Experiments**: §G reports experimental setup and extra experiments on explaining product-kernel models, including real-world predictive models (§G.2).
- **FAQ**: §H responds to main question about PKeX-Shapley.

A Product Kernels in Kernel Methods

Product kernels provide a powerful mechanism for constructing high-dimensional similarity measures by combining kernels defined on individual dimensions or feature subsets. This section discusses key examples of product kernels.

Radial Basis Function (RBF) Kernels as Product Kernels The RBF kernel is a canonical example of product kernels. The RBF kernel is defined based on a distance metric between two instances, with the two well-known metrics being Euclidean (L2 norm) and Manhattan distances (L1 norm). We refer to the former as RBF, and the latter as Laplacian RBF to distinguish these two kernel functions. In addition, when we have only one kernel bandwidth parameter σ, the RBF kernel is referred to as *isotropic*. The RBF kernel with both distance metrics inherently decomposes into products of univariate kernels across dimensions:

- **RBF Kernel**:

$$K_{\text{RBF}}(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2}\right) = \prod_{i=1}^{d} \exp\left(-\frac{(x_i - z_i)^2}{2\sigma^2}\right).$$

- **Laplacian RBF Kernel**:

$$K_{\text{Laplacian RBF}}(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|_1}{\sigma}\right) = \prod_{i=1}^{d} \exp\left(-\frac{|x_i - z_i|}{\sigma}\right).$$

When alternative distance metrics are incorporated into the RBF kernel, such as the Mahalanobis distance, which involves a covariance matrix, the resulting kernel might lose its product decomposition.

Automatic Relevance Determination (ARD) in Gaussian Processes ARD extends RBF kernels by assigning independent length-scale parameters σ_i to each dimension:

$$K_{\text{ARD}}(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\sum_{i=1}^{d} \frac{(x_i - z_i)^2}{2\sigma_i^2}\right) = \prod_{i=1}^{d} \exp\left(-\frac{(x_i - z_i)^2}{2\sigma_i^2}\right).$$

The ARD is extensively used in Gaussian processes for feature selection via learned σ_i, and to enhance interpretability and adaptability. ARD is also referred to as *anisotropic*.

Cauchy Kernel The Cauchy kernel, inspired by the Cauchy distribution, is another example of a product kernel:

$$K_{\text{Cauchy}}(\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{d} \frac{1}{1 + \frac{(x_i - z_i)^2}{\sigma^2}}.$$

Product of Base Kernels A popular way for constructing product kernels is by first defining a base kernel for each individual feature and then computing the overall kernel function by multiplying the base kernels over individual features. The product of PSD kernels remains PSD by the Schur product theorem:

$$K(\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{d} K_i(x_i, z_i).$$

This type of kernel introduces flexibility in combining base kernels while maintaining validity as a (product) kernel function. The base kernels need not be univariate or real-valued: a factor may act on a group of features or on a structured input (e.g., a categorical variable with a delta kernel, or a sequence with a string kernel), in which case PKeX-Shapley attributes at the level of the factors.

ANOVA Kernel The (all-order) ANOVA kernel [Stitson et al., 1999, Duvenaud et al., 2011]

$$K_{\text{ANOVA}}(\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{d} \big(1 + k_i(x_i, z_i)\big)$$

is a product kernel with factors $1 + k_i$; expanding the product shows that it contains all interactions of all orders among the base kernels. Its variant without the constant term, $\prod_i (1 + k_i) - 1$, differs only by a constant that cancels in every marginal contribution and therefore yields the same Shapley values.

Supported and Unsupported Kernels To summarize, PKeX-Shapley applies to any kernel that decomposes into multiplicative factors, including: the Gaussian (RBF) kernel with isotropic or feature-specific (ARD) lengthscales, the Laplacian RBF kernel with the L_1 distance, the Cauchy kernel, the all-order ANOVA kernel, and coordinate-wise products of arbitrary base kernels. Kernels that couple the features through a non-separable operation are not directly supported. Common examples are the polynomial kernel $(\boldsymbol{x}^\top \boldsymbol{z} + c)^p$, the inverse multiquadric kernel $(\|\boldsymbol{x} - \boldsymbol{z}\|^2 + c^2)^{-1/2}$, Gaussian kernels with a full (Mahalanobis) covariance, and the standard Matérn family with the Euclidean distance (including the exponential kernel $\exp(-\|\boldsymbol{x} - \boldsymbol{z}\|_2/\sigma)$, which, unlike its L_1 counterpart above, does not factorize). Such kernels can be explained by PKeX-Shapley only after being replaced by a product-form surrogate, and we do not study the resulting approximation here.

Deep Kernels PKeX-Shapley may also be applied to deep kernels of the form $k(\boldsymbol{x}, \boldsymbol{z}) = \kappa(\varphi(\boldsymbol{x}), \varphi(\boldsymbol{z}))$ with a product kernel κ on a learned representation φ, by treating the latent coordinates of $\varphi(\boldsymbol{x})$ as the explanatory variables. The resulting attributions describe the learned representation rather than the original input features. Deep kernels that mix a product kernel on the latent space with non-separable terms, such as the full construction of Liu et al. [2020], are not directly supported even at the latent-coordinate level.

B The Neutral-Factor Removal Operator: Justification, Convention, and Scope

This appendix expands on the remarks of Section 3 concerning the neutral-factor removal operator $k_j \mapsto 1$ that underlies PKeX-Shapley.

B.1 The Functional Null Axiom and the Choice of the Neutral Factor

Beyond algebraic convenience, the neutral factor of one is singled out by the *Functional Null* axiom of Gevaert and Saeys [2024, Section 5.3.1, Definition 10], which requires an attribution method to assign zero importance to any feature on which the model does not functionally depend. In the canonical product representation of a product-kernel model, a structurally inactive feature j is one whose factor is identically equal to one,

$$k_j(\cdot,\cdot) \equiv 1 \quad \Longrightarrow \quad \phi_j^{\boldsymbol{x}} = 0.$$

Removing such a factor is a no-op because one is the multiplicative identity, so the coalition value is unchanged whether or not the inactive feature is present, $v_{\boldsymbol{x}}(\mathcal{S} \cup \{j\}) - v_{\boldsymbol{x}}(\mathcal{S}) = 0$ for every $\mathcal{S}$, and the Shapley value of j is exactly zero.

Replacing an absent factor by a constant $c \neq 1$ defines another formally admissible removal operator, but it violates the Functional Null axiom: the coalition values then depend on whether the inactive feature is counted as present (factor 1) or absent (factor c), and the feature receives a nonzero marginal contribution even though the model ignores it. Table 1 illustrates this with a controlled null-feature experiment, in which we compute the attribution of a feature with factor identically one while absent factors are replaced by the constant c. Only the multiplicative identity $c = 1$ guarantees zero attribution for the null feature.

Table 1: Attribution assigned to a null feature (a feature the model does not depend on) when absent kernel factors are replaced by the constant c instead of the multiplicative identity.

Replacement constant c	Attribution of the null feature
1.0	0.000000
0.9	-0.228486
0.5	-0.459949
1.3	$+1.479523$

We note that the neutral factor is also the choice under which the value function of Proposition 2 is available in closed form without any expectation over a background distribution; other functional-decomposition centerings (e.g., centering each factor at its marginal mean, which recovers the interventional value function) require such expectations and forfeit the parameter-free, sampling-free computation that PKeX-Shapley exploits.

B.2 Factorization and Scaling Convention

A product kernel admits multiple factorizations: for any $\beta > 0$, $k_0 k_1 = (\beta k_0)(\beta^{-1} k_1)$. Since PKeX-Shapley explains the factors, two such factorizations of the same kernel can yield different attributions. We therefore make the convention explicit. We assume that the multiplicative decomposition is specified by the trained model—the individual factors define how each feature enters the product kernel, as in coordinate-wise product kernels and the RBF, ARD, Laplacian, Cauchy, and ANOVA kernels of Appendix A—and PKeX-Shapley explains this specified factorization. The rescaling above is not an alternative reading of a single specification but a reassignment of a constant between factors; because such a constant does not depend on the feature, it represents no contribution of that feature. We recommend the convention that any constant scale be absorbed into the model coefficients $\boldsymbol{\alpha}$ rather than into a kernel factor. This convention fixes the factorization uniquely and leaves the predictions unchanged, and because it amounts only to an element-wise rescaling of the vectors $\boldsymbol{z}_j$, it modifies neither the ESP recursions of Section 3 nor their time and memory complexity.

B.3 Interpretation under Correlated Features

Under correlated features, PKeX-Shapley should be interpreted as a *model-based* rather than a dependence-aware attribution. Neutral-factor removal measures the direct functional contribution of each kernel factor while leaving the remaining factors unchanged; it does not use the joint feature distribution. Consequently, if two correlated features both enter the model, they receive credit according to their respective functional contributions, but a feature is not credited merely because it is correlated with another feature used by the model.

This distinction reflects the "indirect influence" debate discussed by Gevaert and Saeys [2024, Section 5.3]. In their controlled Gaussian example (Section 5.3.1 therein), observational (conditional) SHAP assigns nonzero importance to a feature that the model does not use, solely because it is correlated with an influential feature. As emphasized in that framework, neither removal operator is universally correct: they answer different explanatory questions. PKeX-Shapley is appropriate for explaining the model's internal functional dependence on its inputs, whereas conditional SHAP is preferable when the user wishes to include dependence-mediated influence. The near-duplicate experiment of Appendix G.2 (Table 6) shows that, when a feature is duplicated up to small noise, the neutral-factor value function splits the credit between the two copies close to evenly, as expected for features that encode nearly identical information.

B.4 Value Functions Outside the Scope of PKeX-Shapley

The recursions of Section 3 rely on the multiplicative structure of the neutral-factor value function $v_{\boldsymbol{x}}(\mathcal{S}) = \boldsymbol{\alpha}^\top k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}})$, which lets the marginal contribution of a feature factorize into a feature-specific term and an elementary symmetric polynomial over the remaining factors. They do not apply to value functions that lack this structure, including: the *interventional* value function $\mathbb{E}_{X_{\mathcal{D}\setminus\mathcal{S}}}[f(\boldsymbol{x}_{\mathcal{S}}, X_{\mathcal{D}\setminus\mathcal{S}})]$ and the *observational* value function $\mathbb{E}[f(X) \mid X_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}]$ of Section 2, as estimated for kernel methods by RKHS-SHAP [Chau et al., 2022] through kernel mean embeddings; *retraining-based* games, in which $v(\mathcal{S})$ is the prediction of a model retrained on the features in $\mathcal{S}$; and *performance-attribution* games such as Shapley value regression [Lipovetsky and Conklin, 2001] and SAGE [Covert et al., 2020], which distribute model fit or predictive performance across features rather than decompose a fixed product-kernel expansion. Explanations for these targets must still be obtained with sampling- or regression-based estimators.

C Newton's Identities

To explain Newton's identities, we begin with a specific set of variables $\mathcal{Z}_4 = \{z_1, z_2, z_3, z_4\}$ before generalizing it to sets of arbitrary size $\mathcal{Z}_d = \{z_1, z_2, \ldots, z_d\}$. The *elementary symmetric polynomials* (ESPs) of degree q is defined as

$$e_q(\mathcal{Z}_4) = \sum_{1 \le i_1 < i_2 < \cdots < i_q \le 4} z_{i_1} z_{i_2} \cdots z_{i_q},$$

with the conventions $e_0(\mathcal{Z}_4) = 1$ and $e_q(\mathcal{Z}_4) = 0$ for $q > 4$. For example:

$$\begin{aligned} e_1(\mathcal{Z}_4) &= z_1 + z_2 + z_3 + z_4, \\ e_2(\mathcal{Z}_4) &= z_1 z_2 + z_1 z_3 + z_1 z_4 + z_2 z_3 + z_2 z_4 + z_3 z_4, \\ e_4(\mathcal{Z}_4) &= z_1 z_2 z_3 z_4. \end{aligned}$$

The *power sum* of degree r is given by

$$p_r(\mathcal{Z}_4) = z_1^r + z_2^r + z_3^r + z_4^r.$$

In particular, $p_1(\mathcal{Z}_4) = e_1(\mathcal{Z}_4)$ and, for example, $p_2(\mathcal{Z}_4) = z_1^2 + z_2^2 + z_3^2 + z_4^2$. Then, Newton's identities relate the ESPs to the power sum recursively. For $q \ge 1$,

$$e_q(\mathcal{Z}_4) = \frac{1}{q} \sum_{r=1}^{q} (-1)^{r-1} e_{q-r}(\mathcal{Z}_4)\, p_r(\mathcal{Z}_4).$$

For the set $\mathcal{Z}_4 = \{z_1, z_2, z_3, z_4\}$, the identities yield:

$$\begin{aligned}
e_1(\mathcal{Z}_4) &= \frac{1}{1}[e_0(\mathcal{Z}_4)\, p_1(\mathcal{Z}_4)] = p_1(\mathcal{Z}_4) = z_1 + z_2 + z_3 + z_4, \\
e_2(\mathcal{Z}_4) &= \frac{1}{2}\Big[e_1(\mathcal{Z}_4)\, p_1(\mathcal{Z}_4) - e_0(\mathcal{Z}_4)\, p_2(\mathcal{Z}_4)\Big] \\
&= \frac{(z_1 + z_2 + z_3 + z_4)^2 - (z_1^2 + z_2^2 + z_3^2 + z_4^2)}{2}, \\
e_3(\mathcal{Z}_4) &= \frac{1}{3}\Big[e_2(\mathcal{Z}_4)\, p_1(\mathcal{Z}_4) - e_1(\mathcal{Z}_4)\, p_2(\mathcal{Z}_4) + e_0(\mathcal{Z}_4)\, p_3(\mathcal{Z}_4)\Big], \\
e_4(\mathcal{Z}_4) &= \frac{1}{4}\Big[e_3(\mathcal{Z}_4)\, p_1(\mathcal{Z}_4) - e_2(\mathcal{Z}_4)\, p_2(\mathcal{Z}_4) \\
&\qquad + e_1(\mathcal{Z}_4)\, p_3(\mathcal{Z}_4) - e_0(\mathcal{Z}_4)\, p_4(\mathcal{Z}_4)\Big].
\end{aligned}$$

The identities presented above can be extended to sets of arbitrary size $\mathcal{Z}_d = \{z_1, z_2, \ldots, z_d\}$ as

$$e_q(\mathcal{Z}_d) = \frac{1}{q}\sum_{r=1}^{q}(-1)^{r-1} e_{q-r}(\mathcal{Z}_d)\, p_r(\mathcal{Z}_d), \quad \text{for } q \geq 1,$$

with $e_0(\mathcal{Z}_d) = 1$ and $e_q(\mathcal{Z}_d) = 0$ if $q > d$ or $q < 0$.

D Proofs

This section provides the detailed proofs of the theoretical results presented in the main paper. First of all, we present a theorem that plays a key role in the other proofs.

Proposition 10. *Let the kernel function for feature j be denoted by $k_j(x_j, x'_j)$, and $\prod_{j\in\emptyset}(k_j(x_j, x'_j) - 1) = 1$ by convention. Then, the following equation holds for product kernel functions:*

$$\prod_{j\in\mathcal{D}} k_j(x_j, x'_j) = \sum_{\mathcal{S}\subseteq\mathcal{D}} \prod_{j\in\mathcal{S}} (k_j(x_j, x'_j) - 1).$$

Proof. We prove the statement by induction on $|\mathcal{D}|$.

Base case ($|\mathcal{D}| = 1$)**:** Let $\mathcal{D} = \{j\}$. Then $k_j(x_j, x'_j) = 1 + (k_j(x_j, x'_j) - 1)$, as desired.

Inductive step: Assume the equation holds for any set of size m. Consider a set $\mathcal{D}$ with size $m + 1$, and let $a \in \mathcal{D}$. We write:

$$\prod_{j\in\mathcal{D}} k_j(x_j, x'_j) = k_a(x_a, x'_a) \prod_{j\in\mathcal{D}\setminus\{a\}} k_j(x_j, x'_j).$$

Using the induction hypothesis for $\mathcal{D} \setminus \{a\}$, we have

$$\prod_{j\in\mathcal{D}} k_j(x_j, x'_j) = k_a(x_a, x'_a) \sum_{\mathcal{T}\subseteq\mathcal{D}\setminus\{a\}} \prod_{j\in\mathcal{T}} (k_j(x_j, x'_j) - 1).$$

Since $k_a(x_a, x'_a) = (k_a(x_a, x'_a) - 1) + 1$, expanding gives:

$$\prod_{j\in\mathcal{D}} k_j(x_j, x'_j) = [(k_a(x_a, x'_a) - 1) + 1] \sum_{\mathcal{T}\subseteq\mathcal{D}\setminus\{a\}} \prod_{j\in\mathcal{T}} (k_j(x_j, x'_j) - 1).$$

Expanding this product, we obtain:

$$\prod_{j\in\mathcal{D}} k_j(x_j, x'_j) = \sum_{\mathcal{T}\subseteq\mathcal{D}\setminus\{a\}} (k_a(x_a, x'_a) - 1) \prod_{j\in\mathcal{T}} (k_j(x_j, x'_j) - 1) \; + \sum_{\mathcal{T}\subseteq\mathcal{D}\setminus\{a\}} \prod_{j\in\mathcal{T}} (k_j(x_j, x'_j) - 1).$$

The first sum on the right-hand side of the equation covers all subsets that contain a, and the second sum covers all subsets that do not contain a. Thus, together they sum over all subsets $\mathcal{S} \subseteq \mathcal{D}$:

$$\prod_{j\in\mathcal{D}} k_j(x_j, x'_j) = \sum_{\mathcal{S}\subseteq\mathcal{D}} \prod_{j\in\mathcal{S}} (k_j(x_j, x'_j) - 1).$$

Thus, by induction, the theorem is proved. □

We now provide the proofs for all the theorems and lemmas in the manuscript.

Proof of Proposition 1

Using the product kernel, the decision function can be rewritten as:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \prod_{j \in \mathcal{D}} k_j(x_j, x_j^{(i)}).$$

Expanding the product using Proposition 10:

$$\prod_{j \in \mathcal{D}} k_j(x_j, x_j^{(i)}) = \sum_{\mathcal{S} \subseteq \mathcal{D}} \prod_{j \in \mathcal{S}} \big(k_j(x_j, x_j^{(i)}) - 1\big).$$

Substituting this expansion into the decision function:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \sum_{\mathcal{S} \subseteq \mathcal{D}} \prod_{j \in \mathcal{S}} \big(k_j(x_j, x_j^{(i)}) - 1\big).$$

Rearranging the summations:

$$f(\boldsymbol{x}) = \sum_{\mathcal{S} \subseteq \mathcal{D}} \sum_{i=1}^{n} \alpha_i \prod_{j \in \mathcal{S}} \big(k_j(x_j, x_j^{(i)}) - 1\big).$$

This establishes the functional decomposition:

$$f(\boldsymbol{x}) = \sum_{\mathcal{S} \subseteq \mathcal{D}} f_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}),$$

where each term is defined as:

$$f_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \sum_{i=1}^{n} \alpha_i \prod_{j \in \mathcal{S}} \big(k_j(x_j, x_j^{(i)}) - 1\big).$$

This establishes the existence of the decomposition. To see uniqueness relative to the chosen removal family, let $R_{\mathcal{S}} f := P_{\mathcal{D} \setminus \mathcal{S}} f$. By the canonical additive decomposition (CAD) construction of Gevaert and Saeys [2024], the induced components satisfy

$$R_{\mathcal{S}} f(\boldsymbol{x}) = \sum_{\mathcal{T} \subseteq \mathcal{S}} f_{\mathcal{T}}(\boldsymbol{x}_{\mathcal{T}}), \qquad \mathcal{S} \subseteq \mathcal{D}.$$

By Möbius inversion on the subset lattice,

$$f_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \sum_{\mathcal{T} \subseteq \mathcal{S}} (-1)^{|\mathcal{S}|-|\mathcal{T}|} R_{\mathcal{T}} f(\boldsymbol{x}).$$

For the product-kernel removal, $R_{\mathcal{T}} f(\boldsymbol{x}) = \boldsymbol{\alpha}^\top k_{\mathcal{T}}(\mathbf{X}_{\mathcal{T}}, \boldsymbol{x}_{\mathcal{T}})$, so

$$f_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \boldsymbol{\alpha}^\top \sum_{\mathcal{T} \subseteq \mathcal{S}} (-1)^{|\mathcal{S}|-|\mathcal{T}|} \bigodot_{j \in \mathcal{T}} \boldsymbol{z}_j = \sum_{i=1}^{n} \alpha_i \prod_{j \in \mathcal{S}} \big(k_j(x_j, x_j^{(i)}) - 1\big),$$

where the last equality is the standard inclusion-exclusion identity $\sum_{\mathcal{T} \subseteq \mathcal{S}} (-1)^{|\mathcal{S}|-|\mathcal{T}|} \prod_{j \in \mathcal{T}} z_j = \prod_{j \in \mathcal{S}} (z_j - 1)$. Thus these are the unique CAD components induced by the chosen removal family. □

Proof of Proposition 2

By Proposition 1, the product-kernel predictor admits the functional decomposition

$$f(\boldsymbol{x}) = \sum_{\mathcal{T} \subseteq \mathcal{D}} f_{\mathcal{T}}(\boldsymbol{x}_{\mathcal{T}}), \qquad f_{\mathcal{T}}(\boldsymbol{x}_{\mathcal{T}}) = \sum_{i=1}^{n} \alpha_i \prod_{j \in \mathcal{T}} \big(k_j(x_j, x_j^{(i)}) - 1\big).$$

Following the decomposition-based removal view of Gevaert and Saeys [2024], removing the variables in $\mathcal{D} \setminus \mathcal{S}$ retains only those decomposition components whose support is contained in $\mathcal{S}$. Hence,

$$P_{\mathcal{D} \setminus \mathcal{S}}(f)(\boldsymbol{x}) := \sum_{\mathcal{T} \subseteq \mathcal{S}} f_{\mathcal{T}}(\boldsymbol{x}_{\mathcal{T}}).$$

Substituting the product-kernel components gives

$$P_{\mathcal{D}\setminus\mathcal{S}}(f)(\boldsymbol{x}) = \sum_{\mathcal{T}\subseteq\mathcal{S}} \sum_{i=1}^{n} \alpha_i \prod_{j\in\mathcal{T}} \big(k_j(x_j, x_j^{(i)}) - 1\big)$$
$$= \sum_{i=1}^{n} \alpha_i \sum_{\mathcal{T}\subseteq\mathcal{S}} \prod_{j\in\mathcal{T}} \big(k_j(x_j, x_j^{(i)}) - 1\big).$$

For each fixed i, the inner sum expands as

$$\sum_{\mathcal{T}\subseteq\mathcal{S}} \prod_{j\in\mathcal{T}} \big(k_j(x_j, x_j^{(i)}) - 1\big) = \prod_{j\in\mathcal{S}} \Big(1 + \big(k_j(x_j, x_j^{(i)}) - 1\big)\Big) = \prod_{j\in\mathcal{S}} k_j(x_j, x_j^{(i)}).$$

Therefore,

$$P_{\mathcal{D}\setminus\mathcal{S}}(f)(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \prod_{j\in\mathcal{S}} k_j(x_j, x_j^{(i)}) = \boldsymbol{\alpha}^\top k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}}).$$

Thus, for product-kernel models, decomposition-based removal is algebraically equivalent to replacing all absent kernel factors by the multiplicative identity 1. This is natural since $k_j = 1$ is the neutral element for the multiplicative structure.

Following the pointwise cooperative-game construction induced by the removal operator [Gevaert and Saeys, 2024, Definition 40], we define

$$v_{\boldsymbol{x}}(\mathcal{S}) = P_{\mathcal{D}\setminus\mathcal{S}}(f)(\boldsymbol{x}) - P_{\mathcal{D}}(f)(\boldsymbol{x}).$$

Since

$$P_{\mathcal{D}}(f)(\boldsymbol{x}) = f_\emptyset = \sum_{i=1}^{n} \alpha_i,$$

we obtain

$$v_{\boldsymbol{x}}(\mathcal{S}) = \boldsymbol{\alpha}^\top k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}}) - f_\emptyset.$$

In particular,

$$v_{\boldsymbol{x}}(\emptyset) = f_\emptyset - f_\emptyset = 0, \qquad v_{\boldsymbol{x}}(\mathcal{D}) = f(\boldsymbol{x}) - f_\emptyset.$$

This proves the claimed value function.

Proof of Theorem 3

We can write the Shapley value as follows:

$$\phi_j^{\boldsymbol{x}} := \sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\}} \mu(|\mathcal{S}|)\big(v_{\boldsymbol{x}}(\mathcal{S}\cup\{j\}) - v_{\boldsymbol{x}}(\mathcal{S})\big)$$
$$= \sum_{q=0}^{d-1} \mu(q) \sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\} \\ |\mathcal{S}|=q}} \big(v_{\boldsymbol{x}}(\mathcal{S}\cup\{j\}) - v_{\boldsymbol{x}}(\mathcal{S})\big).$$

We substitute the functional decomposition value function $v_{\boldsymbol{x}}(\mathcal{S}) = \boldsymbol{\alpha}^\top k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}})$. The marginal contribution of feature j to coalition $\mathcal{S}$ is:

$$v_{\boldsymbol{x}}(\mathcal{S}\cup\{j\}) - v_{\boldsymbol{x}}(\mathcal{S}) = \boldsymbol{\alpha}^\top \big(k_{\mathcal{S}\cup\{j\}}(\mathbf{X}_{\mathcal{S}\cup\{j\}}, \boldsymbol{x}_{\mathcal{S}\cup\{j\}}) - k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}})\big)$$
$$= \boldsymbol{\alpha}^\top \big((k_j(\mathbf{X}_j, x_j) - \mathbf{1}) \odot k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}})\big),$$

where we used the product kernel factorization $k_{\mathcal{S}\cup\{j\}} = k_j \cdot k_{\mathcal{S}}$ and factored out $k_{\mathcal{S}}$.

Let $\boldsymbol{z}_i := k_i(\mathbf{X}_i, x_i)$ and $\mathcal{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d\}$. Substituting into the Shapley sum:

$$\phi_j^{\boldsymbol{x}} = \boldsymbol{\alpha}^\top \left(\big(k_j(\mathbf{X}_j, x_j) - \mathbf{1}\big) \odot \sum_{q=0}^{d-1} \mu(q) \sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\} \\ |\mathcal{S}|=q}} k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}}) \right), \tag{4}$$

where all products and subtractions are element-wise. By definition of the elementary symmetric polynomials (ESPs) over the vectors $\{\boldsymbol{z}_i\}$:

$$\sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\}\\|\mathcal{S}|=q}} k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}},\boldsymbol{x}_{\mathcal{S}}) = \sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\}\\|\mathcal{S}|=q}} \bigodot_{i\in\mathcal{S}} \boldsymbol{z}_i = e_q(\mathcal{Z}_{-j}).$$

Substituting back into (4):

$$\phi_j^{\boldsymbol{x}} = \boldsymbol{\alpha}^\top \left((\boldsymbol{z}_j - \mathbf{1}) \odot \sum_{q=0}^{d-1} \mu(q)\, e_q(\mathcal{Z}_{-j}) \right).$$

The ESP vectors $e_q(\mathcal{Z}_{-j})$ can be recursively computed via Newton's identities, with base case $e_0(\mathcal{Z}_{-j}) = \mathbf{1}$ (the all-ones vector in $\mathbb{R}^n$) and for $q \geq 1$:

$$e_q(\mathcal{Z}_{-j}) = \frac{1}{q} \sum_{r=1}^{q} (-1)^{r-1} e_{q-r}(\mathcal{Z}_{-j}) \odot p_r(\mathcal{Z}_{-j}),$$

where $p_r(\mathcal{Z}) = \sum_{\boldsymbol{z}_i\in\mathcal{Z}} \boldsymbol{z}_i^r$ and the power is applied element-wise. This completes the proof.

Proof of Proposition 4

To justify the decomposition of $e_q(\mathcal{Z}_{-j})$ into prefix and suffix contributions, we start from the polynomial representation introduced in the main body. Consider the leave-one-out polynomial

$$p_{-j}(s) := \bigodot_{i\in\mathcal{D}\setminus\{j\}} \big(\mathbf{1} + \boldsymbol{z}_i\, s\big),$$

whose coefficient of s^q is, by definition, the degree-q ESP $e_q(\mathcal{Z}_{-j})$. Since $\mathcal{D}\setminus\{j\} = \{1,\dots,j-1\}\cup\{j+1,\dots,d\}$ is a disjoint union, the polynomial $p_{-j}(s)$ factorizes as the product of a prefix and a suffix polynomial,

$$p_{-j}(s) = \left(\bigodot_{i=1}^{j-1} \big(\mathbf{1} + \boldsymbol{z}_i\, s\big) \right) \bigodot \left(\bigodot_{i=j+1}^{d} \big(\mathbf{1} + \boldsymbol{z}_i\, s\big) \right).$$

Using the definitions of the prefix and suffix polynomials, this can be written as

$$p_{-j}(s) = \left(\sum_{t=0}^{j-1} \boldsymbol{\Gamma}_{j-1,t}\, s^t \right) \odot \left(\sum_{u=0}^{d-j} \mathbf{R}_{j+1,u}\, s^u \right).$$

Expanding the product yields

$$p_{-j}(s) = \sum_{t=0}^{j-1} \sum_{u=0}^{d-j} \big(\boldsymbol{\Gamma}_{j-1,t} \odot \mathbf{R}_{j+1,u}\big)\, s^{t+u}.$$

Collecting terms with the same total degree $q = t + u$, the coefficient of s^q is therefore given by

$$e_q(\mathcal{Z}_{-j}) = \sum_{\substack{t,u\geq 0\\ t+u=q}} \boldsymbol{\Gamma}_{j-1,t} \odot \mathbf{R}_{j+1,u},$$

where only indices $t \in \{0,\dots,j-1\}$ and $u \in \{0,\dots,d-j\}$ contribute nonzero terms. This establishes the claimed prefix–suffix decomposition of the ESPs over $\mathcal{Z}_{-j}$.

Substituting this expression into $\sum_{q=0}^{d-1} \mu(q)\, e_q(\mathcal{Z}_{-j})$ yields

$$\begin{aligned}
\sum_{q=0}^{d-1} \mu(q)\, e_q(\mathcal{Z}_{-j}) &= \sum_{q=0}^{d-1} \mu(q) \sum_{\substack{t,u\geq 0\\ t+u=q}} \boldsymbol{\Gamma}_{j-1,t} \odot \mathbf{R}_{j+1,u} \\
&= \sum_{t=0}^{j-1} \sum_{u=0}^{d-j} \mu(t+u) \big(\boldsymbol{\Gamma}_{j-1,t} \odot \mathbf{R}_{j+1,u}\big).
\end{aligned}$$

For fixed t, the factor $\mathbf{\Gamma}_{j-1,t}$ does not depend on u and can be factored out of the inner sum, giving

$$\sum_{q=0}^{d-1} \mu(q)\, e_q(\mathcal{Z}_{-j}) = \sum_{t=0}^{j-1} \mathbf{\Gamma}_{j-1,t} \odot \left(\sum_{u=0}^{d-j} \mu(t+u)\, \mathbf{R}_{j+1,u} \right).$$

Recognizing the inner term as $\mathbf{G}_{j+1,t}$ completes the proof.

Proof of Proposition 5

The prefix coefficients $\mathbf{\Gamma}_{j,t}$ arise as the polynomial coefficients of the prefix polynomial

$$p_j^{\text{pref}}(s) = \bigodot_{i=1}^{j} (\mathbf{1} + \boldsymbol{z}_i s) = \sum_{t=0}^{j} \mathbf{\Gamma}_{j,t} s^t.$$

Multiplying $p_{j-1}^{\text{pref}}(s)$ by $(\mathbf{1}+\boldsymbol{z}_j s)$ and matching coefficients of equal powers of s yields the recurrence

$$\mathbf{\Gamma}_{j,t} = \mathbf{\Gamma}_{j-1,t} + \boldsymbol{z}_j \odot \mathbf{\Gamma}_{j-1,t-1}, \qquad j = 1, \ldots, d,$$

with the convention $\mathbf{\Gamma}_{j,-1} = \mathbf{0}$. This allows all prefix coefficients to be computed in a single forward pass. The aggregated suffix quantities are defined from the suffix ESPs by

$$\mathbf{G}_{j,t} := \sum_{u=0}^{d-j+1} \mu(t+u)\, \mathbf{R}_{j,u}.$$

The suffix ESP $\mathbf{R}$ has a similar recurrence to $\mathbf{\Gamma}$. Substituting the suffix recurrence $\mathbf{R}_{j,u} = \mathbf{R}_{j+1,u} + \boldsymbol{z}_j \odot \mathbf{R}_{j+1,u-1}$ into this definition gives

$$\begin{aligned} \mathbf{G}_{j,t} &= \sum_{u} \mu(t+u) \Big(\mathbf{R}_{j+1,u} + \boldsymbol{z}_j \odot \mathbf{R}_{j+1,u-1} \Big) \\ &= \underbrace{\sum_{u} \mu(t+u) \mathbf{R}_{j+1,u}}_{=\mathbf{G}_{j+1,t}} + \boldsymbol{z}_j \odot \underbrace{\sum_{u \geq 1} \mu(t+u) \mathbf{R}_{j+1,u-1}}_{=\mathbf{G}_{j+1,t+1}}, \end{aligned}$$

where the second identity follows from the index shift $u \mapsto u-1$. Consequently, $\mathbf{G}$ satisfies the backward recursion

$$\mathbf{G}_{j,t} = \mathbf{G}_{j+1,t} + \boldsymbol{z}_j \odot \mathbf{G}_{j+1,t+1}, \qquad j = d, \ldots, 1,$$

with boundary condition $\mathbf{G}_{d+1,t} = \mu(t)\mathbf{1}$ induced by the empty-suffix polynomial.

We set $\mu(q) = 0$ for $q \notin \{0, \ldots, d-1\}$ so the boundary terms in the backward recurrence vanish automatically. Using these recursive formulations, computing and storing the triangular tables $\mathbf{\Gamma}$ and $\mathbf{G}$ requires $O(d^2 n)$ time and $O(d^2 n)$ memory, since each table entry $\mathbf{\Gamma}_{j,t}$ or $\mathbf{G}_{j,t}$ is a vector in $\mathbb{R}^n$. Given $\mathbf{\Gamma}$ and $\mathbf{G}$, each Shapley value $\phi_j^{\boldsymbol{x}}$ can be computed on the fly in $O(dn)$ time and $O(n)$ additional memory. Therefore, the overall time complexity for computing all d Shapley values is $O(d^2 n)$.

Proof of Theorem 6

(ii) We begin by expressing the full kernel function as a product of feature-wise kernels:

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \prod_{q \in \mathcal{D}} k_q(x_q^{(i)}, x_q^{(j)}).$$

Using Proposition 10, we expand the product:

$$\prod_{q \in \mathcal{D}} k_q(x_q^{(i)}, x_q^{(j)}) = \sum_{\mathcal{S} \subseteq \mathcal{D}} \prod_{q \in \mathcal{S}} \big(k_q(x_q^{(i)}, x_q^{(j)}) - 1 \big).$$

Substituting this into the MMD formulation for $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$, $k(\boldsymbol{z}^{(i)}, \boldsymbol{z}^{(j)})$, and $k(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(j)})$ yields

$$\begin{aligned} \widehat{\text{MMD}}^2(\mathbb{P}, \mathbb{Q}) = &\frac{1}{n(n-1)} \sum_{i \neq j} \sum_{\mathcal{S} \subseteq \mathcal{D}} \prod_{q \in \mathcal{S}} \big(k_q(x_q^{(i)}, x_q^{(j)}) - 1 \big) \\ &+ \frac{1}{m(m-1)} \sum_{i \neq j} \sum_{\mathcal{S} \subseteq \mathcal{D}} \prod_{q \in \mathcal{S}} \big(k_q(z_q^{(i)}, z_q^{(j)}) - 1 \big) - \frac{2}{nm} \sum_{i,j} \sum_{\mathcal{S} \subseteq \mathcal{D}} \prod_{q \in \mathcal{S}} \big(k_q(x_q^{(i)}, z_q^{(j)}) - 1 \big). \end{aligned}$$

By the linearity of summation, we have

$$\widehat{\text{MMD}}^2(\mathbb{P},\mathbb{Q}) = \sum_{\mathcal{S}\subseteq\mathcal{D}} \Bigg[\frac{1}{n(n-1)} \sum_{i\neq j} \prod_{q\in\mathcal{S}} \big(k_q(x_q^{(i)}, x_q^{(j)}) - 1\big) + \frac{1}{m(m-1)} \sum_{i\neq j} \prod_{q\in\mathcal{S}} \big(k_q(z_q^{(i)}, z_q^{(j)}) - 1\big) - \frac{2}{nm} \sum_{i,j} \prod_{q\in\mathcal{S}} \big(k_q(x_q^{(i)}, z_q^{(j)}) - 1\big) \Bigg].$$

This establishes the functional decomposition of MMD:

$$\widehat{\text{MMD}}^2(\mathbb{P},\mathbb{Q}) = \sum_{\mathcal{S}\subseteq\mathcal{D}} f_{\mathcal{S}}^{\text{MMD}}(\mathbb{P},\mathbb{Q}),$$

where $f_{\mathcal{S}}^{\text{MMD}}$ is defined as

$$f_{\mathcal{S}}^{\text{MMD}}(\mathbb{P},\mathbb{Q}) = \frac{1}{n(n-1)} \sum_{i\neq j} \prod_{q\in\mathcal{S}} \big(k_q(x_q^{(i)}, x_q^{(j)}) - 1\big) + \frac{1}{m(m-1)} \sum_{i\neq j} \prod_{q\in\mathcal{S}} \big(k_q(z_q^{(i)}, z_q^{(j)}) - 1\big) - \frac{2}{nm} \sum_{i,j} \prod_{q\in\mathcal{S}} \big(k_q(x_q^{(i)}, z_q^{(j)}) - 1\big).$$

(i) Following the same reasoning as in Proposition 2, the functional decomposition of MMD in part (ii) induces a removal operator that sets $k_j = 1$ for features in the absent set. The corresponding value function is defined as $v_{\text{MMD}}(\mathcal{S}) := P_{\mathcal{D}\setminus\mathcal{S}}(\widehat{\text{MMD}}^2) - P_{\mathcal{D}}(\widehat{\text{MMD}}^2)$. Since $P_{\mathcal{D}}(\widehat{\text{MMD}}^2) = f_{\emptyset}^{\text{MMD}} = 0$ (setting all kernels to 1 yields equal cross-terms that cancel), this reduces to

$$v_{\text{MMD}}(\mathcal{S}) = \sum_{\mathcal{T}\subseteq\mathcal{S}} f_{\mathcal{T}}^{\text{MMD}}(\mathbb{P},\mathbb{Q}).$$

Applying this to MMD, we define:

$$\begin{aligned} v_{\text{MMD}}(\mathcal{S}) &= \sum_{\mathcal{T}\subseteq\mathcal{S}} \frac{1}{n(n-1)} \sum_{i\neq j} \prod_{q\in\mathcal{T}} \big(k_q(x_q^{(i)}, x_q^{(j)}) - 1\big) \\ &\quad + \frac{1}{m(m-1)} \sum_{i\neq j} \prod_{q\in\mathcal{T}} \big(k_q(z_q^{(i)}, z_q^{(j)}) - 1\big) - \frac{2}{nm} \sum_{i,j} \prod_{q\in\mathcal{T}} \big(k_q(x_q^{(i)}, z_q^{(j)}) - 1\big) \\ &= \frac{1}{n(n-1)} \sum_{i\neq j} \sum_{\mathcal{T}\subseteq\mathcal{S}} \prod_{q\in\mathcal{T}} \big(k_q(x_q^{(i)}, x_q^{(j)}) - 1\big) \\ &\quad + \frac{1}{m(m-1)} \sum_{i\neq j} \sum_{\mathcal{T}\subseteq\mathcal{S}} \prod_{q\in\mathcal{T}} \big(k_q(z_q^{(i)}, z_q^{(j)}) - 1\big) - \frac{2}{nm} \sum_{i,j} \sum_{\mathcal{T}\subseteq\mathcal{S}} \prod_{q\in\mathcal{T}} \big(k_q(x_q^{(i)}, z_q^{(j)}) - 1\big). \end{aligned}$$

By applying Proposition 10 to the inner sum, we can simplify the above equation as:

$$v_{\text{MMD}}(\mathcal{S}) = \frac{1}{n(n-1)} \sum_{i\neq j} k_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}^{(i)}, \boldsymbol{x}_{\mathcal{S}}^{(j)}) + \frac{1}{m(m-1)} \sum_{i\neq j} k_{\mathcal{S}}(\boldsymbol{z}_{\mathcal{S}}^{(i)}, \boldsymbol{z}_{\mathcal{S}}^{(j)}) - \frac{2}{nm} \sum_{i,j} k_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}^{(i)}, \boldsymbol{z}_{\mathcal{S}}^{(j)}),$$

which completes the proof. □

Proof of Proposition 7

By substituting $v_{\text{MMD}}(\mathcal{S})$ into the Shapley value formula, we obtain:

$$\begin{aligned}\phi_q^{\text{MMD}} &= \sum_{r=0}^{d-1} \mu(r) \sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{q\} \\ |\mathcal{S}|=r}} \Big(v_{\text{MMD}}(\mathcal{S}\cup\{q\}) - v_{\text{MMD}}(\mathcal{S}) \Big) \\ &= \sum_{r=0}^{d-1} \mu(r) \sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{q\} \\ |\mathcal{S}|=r}} \Bigg(\frac{1}{n(n-1)} \sum_{i\neq j} \big(k_q(x_q^{(i)}, x_q^{(j)}) - 1\big) k_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}^{(i)}, \boldsymbol{x}_{\mathcal{S}}^{(j)}) \\ &\quad + \frac{1}{m(m-1)} \sum_{i\neq j} \big(k_q(z_q^{(i)}, z_q^{(j)}) - 1\big) k_{\mathcal{S}}(\boldsymbol{z}_{\mathcal{S}}^{(i)}, \boldsymbol{z}_{\mathcal{S}}^{(j)}) \\ &\quad - \frac{2}{nm} \sum_{i,j} \big(k_q(x_q^{(i)}, z_q^{(j)}) - 1\big) k_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}^{(i)}, \boldsymbol{z}_{\mathcal{S}}^{(j)}) \Bigg).\end{aligned}$$

Let $\mathcal{Z}^{(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})} = \{k_1(x_1^{(i)}, x_1^{(j)}), \ldots, k_d(x_d^{(i)}, x_d^{(j)})\}$ and define the elementary symmetric polynomial as

$$e_r(\mathcal{Z}_{-q}^{(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})}) = \sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{q\} \\ |\mathcal{S}|=r}} k_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}^{(i)}, \boldsymbol{x}_{\mathcal{S}}^{(j)}).$$

Then, it follows from Newton's identities recurrence, with base case $e_0(\mathcal{Z}_{-q}^{(\boldsymbol{x}, \boldsymbol{x}')}) = 1$ and for $r \geq 1$:

$$e_r(\mathcal{Z}_{-q}^{(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})}) = \frac{1}{r} \sum_{s=1}^{r} (-1)^{s-1} e_{r-s}(\mathcal{Z}_{-q}^{(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})}) p_s(\mathcal{Z}_{-q}^{(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})}),$$

where $p_s(\mathcal{Z}) = \sum_{z\in\mathcal{Z}} z^s$. Finally, we obtain

$$\begin{aligned}\phi_q^{\text{MMD}} &= \frac{1}{n(n-1)} \sum_{i\neq j} \Bigg(\big(k_q(x_q^{(i)}, x_q^{(j)}) - 1\big) \sum_{r=0}^{d-1} \mu(r) e_r(\mathcal{Z}_{-q}^{(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})}) \Bigg) \\ &\quad + \frac{1}{m(m-1)} \sum_{i\neq j} \Bigg(\big(k_q(z_q^{(i)}, z_q^{(j)}) - 1\big) \sum_{r=0}^{d-1} \mu(r) e_r(\mathcal{Z}_{-q}^{(\boldsymbol{z}^{(i)}, \boldsymbol{z}^{(j)})}) \Bigg) \\ &\quad - \frac{2}{nm} \sum_{i,j} \Bigg(\big(k_q(x_q^{(i)}, z_q^{(j)}) - 1\big) \sum_{r=0}^{d-1} \mu(r) e_r(\mathcal{Z}_{-q}^{(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(j)})}) \Bigg),\end{aligned}$$

and that completes the proof. □

Proof of Theorem 8

(ii) We begin by expressing the full kernel matrix as a product of feature-wise kernels:

$$\mathbf{K} = \bigodot_{j\in\mathcal{D}} \mathbf{K}_j.$$

Since $\mathbf{K}$ is the element-wise product of the $\mathbf{K}_j$, we can apply Proposition 10 for each element in the kernel matrix and write:

$$\bigodot_{j\in\mathcal{D}} \mathbf{K}_j = \sum_{\mathcal{S}\subseteq\mathcal{D}} \bigodot_{j\in\mathcal{S}} (\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top).$$

Since HSIC is dependent on $\text{tr}(\mathbf{KHLH})$, substituting the decomposition of $\mathbf{K}$ gives:

$$\widehat{\text{HSIC}}(X, Y) = \frac{1}{(n-1)^2} \text{tr} \left(\mathbf{HLH} \sum_{\mathcal{S}\subseteq\mathcal{D}} \bigodot_{j\in\mathcal{S}} (\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top) \right).$$

Using the linearity of the trace operator, we obtain:

$$\widehat{\text{HSIC}}(X,Y) = \frac{1}{(n-1)^2} \sum_{\mathcal{S}\subseteq\mathcal{D}} \text{tr}\left(\mathbf{HLH}\left[\bigodot_{j\in\mathcal{S}}(\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top)\right]\right).$$

This completes the proof of the functional decomposition.

(i) Following the same reasoning as in Proposition 2, the functional decomposition of HSIC in part (ii) induces a removal operator that sets $k_j = 1$ for features in the absent set. The corresponding value function is $v_{\text{HSIC}}(\mathcal{S}) := P_{\mathcal{D}\setminus\mathcal{S}}(\widehat{\text{HSIC}}) - P_{\mathcal{D}}(\widehat{\text{HSIC}})$. Since $P_{\mathcal{D}}(\widehat{\text{HSIC}}) = f_{\emptyset}^{\text{HSIC}} = \frac{1}{(n-1)^2}\text{tr}(\mathbf{HLH11}^\top) = 0$ (as $\mathbf{H1} = \mathbf{0}$), this reduces to

$$v_{\text{HSIC}}(\mathcal{S}) = \sum_{\mathcal{T}\subseteq\mathcal{S}} \frac{1}{(n-1)^2}\text{tr}\left(\mathbf{HLH}\left[\bigodot_{j\in\mathcal{T}}(\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top)\right]\right).$$

By applying Proposition 10, we simplify:

$$\sum_{\mathcal{T}\subseteq\mathcal{S}} \bigodot_{j\in\mathcal{T}}(\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top) = \bigodot_{j\in\mathcal{S}} \mathbf{K}_j.$$

Thus, the value function for feature subset $\mathcal{S}$ is:

$$v_{\text{HSIC}}(\mathcal{S}) = \frac{1}{(n-1)^2}\text{tr}\left(\mathbf{HLH}\bigodot_{j\in\mathcal{S}}\mathbf{K}_j\right) = \frac{1}{(n-1)^2}\text{tr}\left(\mathbf{HLHK}_{\mathcal{S}}\right),$$

which completes the proof. □

Proof of Proposition 9

By substituting $v_{\text{HSIC}}(\mathcal{S}) = \frac{1}{(n-1)^2}\text{tr}(\mathbf{K}_{\mathcal{S}}\mathbf{HLH})$ into the Shapley value formula, we obtain:

$$\begin{aligned}
\phi_j^{\text{HSIC}} &= \frac{1}{(n-1)^2}\text{tr}\left(\mathbf{HLH}\sum_{q=0}^{d-1}\mu(q)\sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\}\\|\mathcal{S}|=q}}\left(\mathbf{K}_{\mathcal{S}\cup\{j\}} - \mathbf{K}_{\mathcal{S}}\right)\right)\\
&= \frac{1}{(n-1)^2}\text{tr}\left(\mathbf{HLH}\sum_{q=0}^{d-1}\mu(q)\sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\}\\|\mathcal{S}|=q}}\left(\mathbf{K}_j\odot\mathbf{K}_{\mathcal{S}} - \mathbf{K}_{\mathcal{S}}\right)\right)\\
&= \frac{1}{(n-1)^2}\text{tr}\left(\mathbf{HLH}\left((\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top)\odot\sum_{q=0}^{d-1}\mu(q)\sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\}\\|\mathcal{S}|=q}}\mathbf{K}_{\mathcal{S}}\right)\right).
\end{aligned}$$

Letting $\mathcal{K} = \{\mathbf{K}_1, ..., \mathbf{K}_d\}$, we express:

$$\sum_{\substack{\mathcal{S}\subseteq\mathcal{D}\setminus\{j\}\\|\mathcal{S}|=q}}\mathbf{K}_{\mathcal{S}} = E_q(\mathcal{K}_{-j}),$$

which is computed recursively via Newton's identities, with base case $E_0(\mathcal{K}_{-j}) = \mathbf{1}\mathbf{1}^\top$ and for $q \geq 1$:

$$E_q(\mathcal{K}_{-j}) = \frac{1}{q}\sum_{r=1}^{q}(-1)^{r-1}E_{q-r}(\mathcal{K}_{-j})\odot P_r(\mathcal{K}_{-j}),$$

where $P_r(\mathcal{K}) = \sum_{\mathbf{K}_i \in \mathcal{K}} \mathbf{K}_i^{\odot r}$ is the element-wise power sum (with $\mathbf{K}_i^{\odot r}$ denoting the element-wise r-th power of $\mathbf{K}_i$). Substituting this back, we obtain:

$$\phi_j^{\text{HSIC}} = \frac{1}{(n-1)^2} \operatorname{tr}\left(\mathbf{HLH}\left((\mathbf{K}_j - \mathbf{1}\mathbf{1}^\top) \bigodot \sum_{q=0}^{d-1} \mu(q) E_q(\mathcal{K}_{-j}) \right) \right),$$

which completes the proof. □

E Interpolation-Based ESP Recovery Methods

This appendix describes the polynomial interpolation approaches used to recover the elementary symmetric polynomials (ESPs) appearing in the recursive Shapley formulation of Theorem 3. Both the Chebyshev and Fourier constructions encode the ESPs

$$\{e_q(\mathcal{Z}_{-j})\}_{q=0}^{d-1}, \qquad j \in \{1, \ldots, d\},$$

as the coefficients of a univariate polynomial and attempt to reconstruct these coefficients from evaluations of the polynomial at a fixed set of interpolation nodes. Although highly effective for tree-based models—where the relevant polynomial degree is the depth of the tree—these methods become numerically unstable for product kernels, where the polynomial degree equals $d-1$, the total number of features. We describe each method in detail and analyze its numerical behavior.

E.1 Chebyshev Interpolation

Polynomial Representation. Theorem 3 shows that the Shapley value for feature j involves the weighted sum

$$\sum_{q=0}^{d-1} \mu(q)\, e_q(\mathcal{Z}_{-j}),$$

where $e_q(\mathcal{Z}_{-j})$ is the degree-q ESP over the leave-one-out set $\mathcal{Z}_{-j} = \mathcal{Z} \setminus \{\boldsymbol{z}_j\}$. Define the polynomial

$$H_j(y) \;:=\; \sum_{q=0}^{D} e_q(\mathcal{Z}_{-j})\, y^q, \qquad D := d-1.$$

Recovering the ESPs $\{e_q(\mathcal{Z}_{-j})\}$ is therefore equivalent to recovering the coefficient vector of H_j.

Interpolation at Chebyshev Nodes. Let

$$y_\ell = \cos\left(\frac{\pi \ell}{D}\right), \qquad \ell = 0, \ldots, D,$$

be the $(D+1)$ Chebyshev points of the second kind on $[-1, 1]$. Evaluating the polynomial at these nodes yields

$$g_{j,\ell} = H_j(y_\ell) = \sum_{q=0}^{D} e_q(\mathcal{Z}_{-j})\, y_\ell^q.$$

Define the vector $e^{(-j)} = (e_0(\mathcal{Z}_{-j}), \ldots, e_D(\mathcal{Z}_{-j}))^\top$ and the Chebyshev–Vandermonde matrix $V \in \mathbb{R}^{(D+1)\times(D+1)}$ with entries

$$V_{\ell q} = y_\ell^q.$$

Then

$$g_j = V\, e^{(-j)}.$$

Inverting this system recovers $e^{(-j)} = V^{-1} g_j$. In practice, instead of repeatedly solving this system, it is convenient to precompute a weight vector $N \in \mathbb{R}^{D+1}$ satisfying

$$\mu^\top e^{(-j)} = N^\top g_j,$$

so that the Shapley contribution can be evaluated without explicitly materializing $e^{(-j)}$.

Numerical Instability. Although optimal in an approximation-theoretic sense, Chebyshev nodes do not prevent numerical ill-conditioning when the polynomial degree D is moderate or large. Even with Chebyshev spacing, the condition number of V grows exponentially with D: for $D \approx 40$, one typically obtains $\mathrm{cond}(V) \gtrsim 10^{12}$ in double precision, and for $D \approx 60$ the matrix becomes nearly singular. In decision trees, the polynomial degree equals the depth, often < 15. For product kernels, the degree is $D = d - 1$, which can be up to hundreds, making instability intrinsic. These issues make the inversion $V^{-1}g_j$ highly unreliable beyond $d \approx 40$, even if the Chebyshev grid is used.

E.2 Fourier / Roots-of-Unity Interpolation

The Fourier approach mirrors the Chebyshev strategy but replaces Chebyshev nodes with equally spaced complex roots of unity. The algebraic structure becomes particularly simple due to the unitary nature of the discrete Fourier transform (DFT).

Roots-of-Unity Evaluation. Let $N := D + 1$ and

$$\omega_\ell = \exp\left(\tfrac{2\pi i \ell}{N}\right), \qquad \ell = 0, \dots, N-1.$$

Evaluating H_j at these points gives

$$g_{j,\ell} = H_j(\omega_\ell) = \sum_{q=0}^{D} e_q(\mathcal{Z}_{-j})\, \omega_\ell^q.$$

Define the DFT matrix $F \in \mathbb{C}^{N \times N}$ by $F_{\ell q} = \omega_\ell^q$. Then

$$g_j = F\, e^{(-j)}, \qquad e^{(-j)} = F^{-1} g_j = \frac{1}{N} F^* g_j.$$

Relation to the Chebyshev Construction. The Fourier method uses the same polynomial representation as the Chebyshev method, but replaces the real Vandermonde matrix V with the complex DFT matrix F. The main steps differ only in the choice of nodes and the linear system that is solved:

$$\text{Chebyshev:} \quad g_j = V e^{(-j)}, \qquad \text{Fourier:} \quad g_j = F e^{(-j)}.$$

Both methods require evaluating H_j at $D+1$ distinct interpolation points and both require recovering the coefficients of a degree-D polynomial from these evaluations.

Numerical Instability. Although the DFT matrix F is perfectly conditioned in exact arithmetic, numerical instability arises for product kernels due to various reasons. First and foremost, the values $H_j(\omega_\ell)$ involve complex products $\prod_{i \neq j}(1 + \omega_\ell \boldsymbol{z}_i)$ whose magnitudes grow roughly exponentially in d. Thus g_j spans an enormous dynamic range. In addition, the inverse DFT involves alternating complex phases. When some entries of g_j are extremely large and others small, cancellation magnifies floating-point error. Moreover, as with the Chebyshev method, the instability is structural: the degree $D = d - 1$ is simply too large for stable recovery of polynomial coefficients by interpolation.

Even though the Fourier transform itself is numerically stable, the interpolation problem it is solving is not. In our experiments, instability appears as early as $d \approx 50$ for product kernels.

E.3 Numerical Stability of ESP-Based Shapley Computation

We empirically evaluate the numerical stability and computational efficiency of different elementary symmetric polynomial (ESP) aggregation strategies used to compute exact Shapley values in product-kernel models.

Experimental setup. Synthetic regression datasets are generated using `sklearn.make_regression` with $n = 500$ samples and varying feature dimension $d \in \{10, 15, 20, 25, 30, 40, 50, 70, 80, 100, 200, 500, 1000\}$. For each d, the data are split into 90% training and 10% test sets. A Gaussian Process Regressor with an RBF kernel is trained on the training set, using a small observation noise term ($\alpha = 10^{-2}$) and no hyperparameter restarts unless otherwise stated.

Table 2: Numerical accuracy and runtime of ESP-based Shapley computation. We report the average absolute Shapley-sum error $\mathbb{E}[|\sum_j \phi_j - (f(x) - v(\emptyset))|]$ and average wall-clock time per explanation call (seconds), averaged over 5 trials.

d	PKeX-Shapley		FFT		Chebyshev	
	Error	Time (s)	Error	Time (s)	Error	Time (s)
10	2.26×10^{-12}	1.22×10^{-3}	4.83×10^{-12}	1.55×10^{-3}	4.35×10^{-10}	1.38×10^{-3}
15	2.34×10^{-12}	2.10×10^{-3}	2.07×10^{-10}	2.97×10^{-3}	4.86×10^{-7}	2.28×10^{-3}
20	1.14×10^{-12}	2.88×10^{-3}	2.07×10^{-9}	4.13×10^{-3}	2.61×10^{-4}	2.71×10^{-3}
25	7.16×10^{-13}	3.75×10^{-3}	5.05×10^{-8}	5.04×10^{-3}	2.29×10^{-1}	3.67×10^{-3}
30	1.01×10^{-12}	4.78×10^{-3}	1.03×10^{-6}	6.94×10^{-3}	1.70×10^{2}	4.52×10^{-3}
40	1.93×10^{-13}	7.36×10^{-3}	7.43×10^{-4}	9.93×10^{-3}	4.59×10^{7}	7.02×10^{-3}
50	3.64×10^{-13}	1.05×10^{-2}	9.77×10^{-3}	1.68×10^{-2}	2.77×10^{10}	9.89×10^{-3}
70	4.55×10^{-13}	1.81×10^{-2}	8.15×10^{0}	2.97×10^{-2}	2.99×10^{16}	1.79×10^{-2}
80	7.28×10^{-13}	2.23×10^{-2}	1.59×10^{4}	3.66×10^{-2}	3.08×10^{18}	2.20×10^{-2}
100	1.07×10^{-12}	3.38×10^{-2}	5.28×10^{7}	5.55×10^{-2}	1.53×10^{22}	3.20×10^{-2}
200	5.46×10^{-13}	1.18×10^{-1}	8.60×10^{27}	2.19×10^{-1}	3.93×10^{45}	1.15×10^{-1}
500	2.50×10^{-13}	6.98×10^{-1}	3.57×10^{85}	1.71×10^{0}	1.10×10^{102}	9.61×10^{-1}
1000	1.73×10^{-12}	2.65×10^{0}	2.90×10^{183}	1.36×10^{1}	3.72×10^{199}	3.75×10^{0}

Local explanations are computed using a product-kernel Shapley explainer, instantiated with three different ESP aggregation strategies: (i) *quadratic*, a dynamic-programming method based on prefix–suffix recursion over elementary symmetric polynomials; (ii) *FFT*, which aggregates ESPs via roots-of-unity evaluation and fast Fourier transforms; and (iii) *Chebyshev*, which interpolates the ESP polynomial using Chebyshev nodes and solves the resulting Vandermonde system.

Evaluation protocol. For each trained model and each ESP method, we randomly select five test instances and compute per-feature Shapley values. Correctness is assessed using the Shapley efficiency identity

$$\sum_{j=1}^{d} \phi_j \;=\; f(x) - v(\emptyset), \tag{5}$$

where $f(x)$ is the GP prediction and $v(\emptyset)$ denotes the null-game value used by the explainer. We report the *average absolute Shapley-sum error*, $\mathbb{E}\big[\,|\sum_j \phi_j - (f(x) - v(\emptyset))|\,\big]$, as well as the average wall-clock time per explanation call. All results are averaged over the five trials.

Results. Table 2 reports numerical error and runtime across feature dimensions. The quadratic dynamic-programming method exhibits consistently low numerical error, on the order of 10^{-13}–10^{-12}, across all tested dimensions, including $d = 1000$. Runtime increases smoothly with d, remaining below one second up to $d = 500$ and around 2.6 seconds at $d = 1000$.

In contrast, both FFT- and Chebyshev-based approaches are accurate for small to moderate dimensions ($d \lesssim 20$–25), achieving near machine-precision error, but exhibit rapid numerical degradation as d increases. Beyond this regime, errors grow exponentially and quickly become catastrophic, despite these methods maintaining low wall-clock times.

Discussion. These results highlight a clear trade-off between computational speed and numerical robustness. While FFT- and Chebyshev-based ESP aggregation can be attractive for low-dimensional problems, their current formulations amplify rounding and conditioning errors as the polynomial degree grows, leading to numerical instability at moderate to large d. The quadratic dynamic-programming approach, although asymptotically slower, is the only method that remains numerically reliable across all tested dimensions, making it the appropriate default for exact Shapley computation in high-dimensional product-kernel models.

F Blockwise Computation and Memory Complexity

Predictive models. Proposition 5 bounds the memory of the prefix and aggregated suffix tables by $O(d^2 n)$, since each of the $O(d^2)$ table entries is a vector in $\mathbb{R}^n$ holding one value per kernel centre. This bound describes a fully materialized implementation; it is not intrinsic to the algorithm. The ESP recursions are computed independently across kernel centres: the contribution of centre i to $\phi_j^{\boldsymbol{x}}$ depends only on its own d kernel factors $\{(\boldsymbol{z}_1)_i, \ldots, (\boldsymbol{z}_d)_i\}$ (see Equation (3)) and does not interact with the ESP quantities of any other centre. The active kernel centres (those with $\alpha_i \neq 0$) can therefore be partitioned into blocks of size B. For each block we (i) compute the prefix coefficients $\mathbf{\Gamma}$ and the aggregated suffix quantities $\mathbf{G}$ restricted to the centres in the block, (ii) contract them with the corresponding coefficients α_i and factors $(\boldsymbol{z}_j)_i - 1$ as in Proposition 4 to obtain a d-dimensional partial Shapley vector, (iii) accumulate it into the running total, and (iv) discard the block's tables before processing the next block. Every active centre is processed exactly once, so the computation remains exact and retains the $O(d^2 n)$ time complexity, while the auxiliary working memory drops from $O(d^2 n)$ to $O(Bd^2)$. In addition, kernel predictions often depend on only a subset of the training data: for support vector machines only the support vectors have non-zero coefficients, so n can be replaced by the number of support vectors, which is frequently much smaller than the training set.

MMD and HSIC. For the statistical discrepancies of Section 4 the same observation holds with kernel centres replaced by sample pairs. For MMD with n samples per distribution, the within- and cross-distribution terms involve $O(n^2)$ sample pairs; for HSIC each feature contributes an $n \times n$ kernel matrix. The ESP recursions act independently on each pair (matrix entry), so all d attributions cost $O(d^2 n^2)$ time, and a fully materialized implementation requires $O(d^2 n^2)$ memory for the prefix–suffix tables. Processing B pairs at a time, weighting their prefix–suffix quantities by the corresponding MMD coefficients ($1/n(n-1)$, $1/m(m-1)$, $-2/nm$) or HSIC coefficients (the entries of $1/(n-1)^2$ $\mathbf{HLH}$), and accumulating the result into the d-dimensional Shapley vector keeps the computation exact and the time $O(d^2 n^2)$ while reducing the memory to $O(Bd^2)$.

Empirical peak memory. To evaluate the practical impact of this optimization, we implemented the blockwise algorithm and benchmarked it on seven real-world regression datasets from the UCI repository. For each dataset we trained an RBF support vector regressor (SVR) on at most $4{,}000$ training instances and compared the original fully materialized implementation against the exact blockwise implementation with block size $B = 256$. Table 3 reports the mean per-instance explanation time and the peak memory usage over 30 explanation calls, measuring only the explanation routine and excluding model training and other wrapper overhead; memory denotes the peak Python-level allocation traced during one explanation call. The blockwise implementation produces identical Shapley values while reducing peak memory consumption by up to $8.9\times$. The largest savings occur for models with many support vectors and high-dimensional feature spaces, where the original implementation stores the largest number of intermediate ESP quantities. The reduction in memory comes at the cost of a modest increase in runtime, as the kernel factors are processed sequentially across blocks. When the number of support vectors is already small (e.g., KEGG Undirected, with 247 support vectors $< B$), the memory savings are naturally less pronounced and the runtime overhead is correspondingly minimal.

Table 3: Fully materialized versus blockwise ($B = 256$) PKeX-Shapley on real datasets: number of training instances N, features d, support vectors (SV), mean per-instance explanation time, and peak memory of the explanation call. Both variants return identical Shapley values.

				PKeX-Shapley		PKeX-Shapley (B=256)		
Dataset	N	d	SV	Time (ms)	Mem. (MB)	Time (ms)	Mem. (MB)	Mem. reduction
UCI Superconductivity	21,263	81	2,440	74.21	207.31	164.88	23.19	8.9×
SkillCraft	3,337	19	1,761	4.83	10.29	8.69	1.73	5.9×
Pol	14,999	26	1,413	7.74	14.35	12.59	2.85	5.0×
Bike Sharing	17,378	17	1,063	3.27	5.14	8.22	1.35	3.8×
KEGG Undirected	15,901	27	247	3.90	2.69	4.02	2.69	1.0×
Parkinson	5,874	20	1,512	4.67	9.66	8.37	1.84	5.3×
SML	4,136	26	512	4.89	5.21	5.58	2.66	2.0×

G Experiments

G.1 Experimental Setup

SVM Optimization Using Optuna [Akiba et al., 2019] When using SVM in our experiments, we optimized the Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel using Optuna [Akiba et al., 2019], a robust hyperparameter optimization framework. The target type (either 'regression' or 'classification') was determined to guide the selection of the appropriate SVM model (`SVR` for regression and `SVC` for classification). The hyperparameters `C` and `gamma`, critical for the RBF kernel's performance, were optimized within an extensive range using a log-uniform distribution. Specifically, we defined the hyperparameters: `C` between 10^{-5} and 10^{5} and `gamma` between 10^{-5} and 10^{3}, and utilized 5-fold cross-validation to ensure reliable evaluation. The optimization process aimed to minimize the mean squared error for regression tasks and maximize accuracy for classification tasks. After conducting the specified number of trials (n=100), the best hyperparameters were used to train a final SVM model on the entire dataset, yielding both the optimal model configuration and the best cross-validation score achieved during the optimization process.

Training Gaussian Process (GP) Using K-Fold Cross-Validation When using GP, we trained a model using k-fold cross-validation to ensure robust evaluation and generalization performance. We defined the GP kernel as `C(1.0, (1e-4, 1e1)) * RBF(1.0, (1e-4, 10))`, suitable for both regression and classification tasks. For classification problems, `GaussianProcessClassifier` was utilized, while `GaussianProcessRegressor` was used for regression tasks. We employed `K-Fold` with $K = 5$ for cross-validation to evaluate the model's performance across different folds. All the hyperparameters, including the kernel width of the RBF kernel, are determined in the training process using optimization. During the cross-validation process, the model was trained on each fold, and predictions were made on the validation fold. Performance metrics were chosen based on the problem type: accuracy for classification models and mean absolute percentage error (MAPE) for regression models. The scores from each fold were aggregated to compute the average and standard deviation of the scores.

G.2 Real-World Predictive Models

Real-world datasets do not provide ground-truth Shapley values, as this would require complete knowledge of the joint data-generating distribution. Consequently, unlike in the synthetic experiments of Section 5.1, attribution accuracy cannot be quantified on real data, and the synthetic experiments remain the appropriate benchmark for quantitative evaluation. We therefore use real data for three complementary checks: an agreement analysis with RKHS-SHAP, two sanity checks based on planted features, and a runtime and peak-memory comparison. All experiments use seven regression datasets from the UCI repository (UCI Superconductivity, SkillCraft, Pol, Bike Sharing, KEGG Undirected, Parkinson, and SML; see Table 3 for their sizes), on each of which we train an RBF support vector regressor. RKHS-SHAP [Chau et al., 2022] is run with the implementation of its authors and a background set of 100 observations.

Agreement with interventional RKHS-SHAP. We first compare the feature rankings produced by PKeX-Shapley and by RKHS-SHAP, the kernel-specific explanation method based on the interventional value function. The purpose of this experiment is not to regard the interventional value function as the ground truth, nor to argue that higher agreement implies a better explanation: the two methods are derived from fundamentally different value functions and capture different notions of feature contribution, so some disagreement is both expected and theoretically justified (Appendix B). Rather, the experiment is a practical robustness check: since RKHS-SHAP has a model-specific interventional value function for kernel methods, we wish to verify that replacing it with the neutral-factor value function does not fundamentally alter the resulting feature-importance ranking. For each dataset we explain 25 held-out observations with both methods; RKHS-SHAP uses $1{,}000$ coalition samples. Features are ranked by their Shapley values, each method's top-$k\%$ features are selected, and we report the proportion of features appearing in both sets, repeated over 10 random runs that vary both the explained observations and the RKHS-SHAP background set. Table 4 shows that, although the two methods optimize different value functions, they identify substantially overlapping sets of important features across all seven datasets: on average, the overlap is 0.67 for the top 20% and 0.80 for the top 30% of the features. These results should not be interpreted as validating one value

function against the other; they indicate that adopting the neutral-factor value function does not lead to dramatically different feature-importance rankings from those obtained with the conventional interventional value function, while benefiting from the computational advantages of PKeX-Shapley.

Table 4: Agreement between PKeX-Shapley and RKHS-SHAP feature rankings: proportion of features appearing in both methods' top-$k\%$ sets (mean $\pm$ standard deviation over 10 runs).

Dataset	d	Top-20%	Top-30%
UCI Superconductivity	81	0.68 ± 0.03	0.79 ± 0.03
SkillCraft	19	0.71 ± 0.08	0.80 ± 0.07
Pol	26	0.62 ± 0.08	0.78 ± 0.07
Bike Sharing	17	0.65 ± 0.15	0.82 ± 0.09
KEGG Undirected	27	0.64 ± 0.13	0.83 ± 0.06
Parkinson	20	0.69 ± 0.10	0.77 ± 0.07
SML	26	0.71 ± 0.08	0.83 ± 0.04
Overall (mean)	–	0.67 ± 0.09	0.80 ± 0.06

Planted null features. Following a common sanity-check protocol for feature attribution, we augment each dataset with five null features generated independently from a standard Gaussian distribution, and therefore statistically unrelated to the target. A well-behaved explanation method should assign negligible importance to these features, since they carry no predictive information. For each dataset we train an RBF-SVR on the augmented data and compute PKeX-Shapley and RKHS-SHAP values for 30 held-out test instances. We then report the ratio between the mean absolute attribution assigned to the planted null features and that assigned to the original features,

$$\frac{\mathbb{E}\big[\,|\phi_{\text{null}}|\,\big]}{\mathbb{E}\big[\,|\phi_{\text{real}}|\,\big]},$$

where smaller values indicate stronger suppression of irrelevant features: a ratio close to zero means that the null features receive negligible attribution, whereas a ratio close to one (or larger) means that they are treated like genuine predictive features. Table 5 reports the results. PKeX-Shapley consistently assigns substantially lower importance to the planted null features than to the original ones (ratios between 0.18 and 0.49). Compared with RKHS-SHAP, it attains a lower ratio on four of the seven datasets (UCI Superconductivity, SkillCraft, Parkinson, and SML) and a higher ratio on the remaining three (Pol, Bike Sharing, and KEGG Undirected). These results suggest that the neutral-factor value function does not spuriously attribute importance to irrelevant variables.

Table 5: Null-feature attribution: ratio of the mean absolute attribution of five planted Gaussian null features to that of the original features (lower is better).

Dataset	PKeX-Shapley	RKHS-SHAP
UCI Superconductivity	0.280	0.595
SkillCraft	0.310	0.355
Pol	0.340	0.322
Bike Sharing	0.490	0.313
KEGG Undirected	0.337	0.228
Parkinson	0.312	0.395
SML	0.183	0.211

Near-duplicate features. To evaluate whether importance is shared fairly between highly redundant variables (cf. Appendix B.3), we extend the planted-feature protocol by copying the two most important original features of each dataset and adding a small amount of Gaussian perturbation to the copies. For each original feature and its planted near-duplicate, we compute the share of the pair's combined absolute attribution assigned to the original feature,

$$\frac{|\phi_{\text{original}}|}{|\phi_{\text{original}}| + |\phi_{\text{duplicate}}|},$$

where a value of approximately 0.5 indicates that the two features receive equal attribution, as expected when they encode nearly identical information. Table 6 summarizes the results. Across

all datasets, PKeX-Shapley produces original-feature shares between 0.473 and 0.627, consistently close to the ideal value of 0.5, and on every dataset its share is closer to an even split than that of RKHS-SHAP. The neutral-factor value function therefore distributes attribution more symmetrically among highly redundant features in this setting.

Table 6: Share of the combined absolute attribution of an original feature and its near-duplicate that is assigned to the original feature (ideal: 0.5).

Dataset	PKeX-Shapley	RKHS-SHAP
UCI Superconductivity	0.517	0.469
SkillCraft	0.515	0.539
Pol	0.606	0.705
Bike Sharing	0.507	0.699
KEGG Undirected	0.473	0.817
Parkinson	0.607	0.655
SML	0.627	0.718

Runtime and peak memory. Finally, we compare the exact blockwise PKeX-Shapley ($B = 256$; Appendix F) with RKHS-SHAP on the same fitted models and held-out observations, with RKHS-SHAP using 100 background observations and a budget of 500 coalition samples. To isolate the explanation cost, we measure only the explanation call, excluding model training, explainer setup, and other wrapper operations. Runtime is averaged over 30 test observations, and memory denotes the peak Python-level allocation traced during one explanation call. As Table 7 shows, despite computing exact Shapley values rather than a sampled approximation, PKeX-Shapley is $46\times$ to $303\times$ faster and uses $2.8\times$ to $10.5\times$ less peak memory than RKHS-SHAP across the seven datasets.

Table 7: Explanation time and peak memory of exact blockwise PKeX-Shapley ($B = 256$) and RKHS-SHAP (500 coalition samples, 100 background observations) on real datasets.

	PKeX-Shapley (B=256)		RKHS-SHAP		
Dataset	Time (ms)	Mem. (MB)	Time (ms)	Mem. (MB)	Speedup
UCI Superconductivity	164.88	23.19	7587.80	65.71	46×
SkillCraft	8.69	1.73	2630.03	15.75	303×
Pol	12.59	2.85	2401.81	21.34	191×
Bike Sharing	8.22	1.35	1595.14	14.14	194×
KEGG Undirected	4.02	2.69	468.90	22.13	117×
Parkinson	8.37	1.84	2319.59	16.55	277×
SML	5.58	2.66	903.81	21.32	162×

G.3 MMD Experiments

In addition to the synthetic experiments for MMD, we first provide another experiment for the cases when there is no distribution discrepancy. To that end, X and Z are sampled from the same multivariate normal distribution across all 20 variables. The MMD is near zero, indicating the distributions are equivalent. Shapley values are computed and replicated 1000 times, with histograms plotted for each variable in Figure 6. The near-identical distributions of Shapley values across all variables reflect the uniform contribution of these variables to the MMD close to zero, consistent with the absence of any significant difference between the distributions.

We extend our analysis to the UCI Diabetes dataset, consisting of 442 samples and 10 baseline variables, including age, sex, body mass index (BMI), average blood pressure, and six blood serum measurements (shown by s1 to s6 features). The dataset is split into male and female subsets using the second variable (sex), which is excluded from the analysis, leaving nine variables for comparison.

Using MMD, we calculate the dissimilarity between male and female groups and then compute Shapley values to attribute variable contributions to the MMD. Figure 7 displays the Shapley values for the nine variables in the Diabetes dataset. The results show that $s3$ and $s4$ contribute most significantly to the MMD, followed by bp, $s6$, age, $s5$, and $s2$. In contrast, bmi and $s1$ contribute nearly zero to the MMD, indicating their alignment across the two groups.

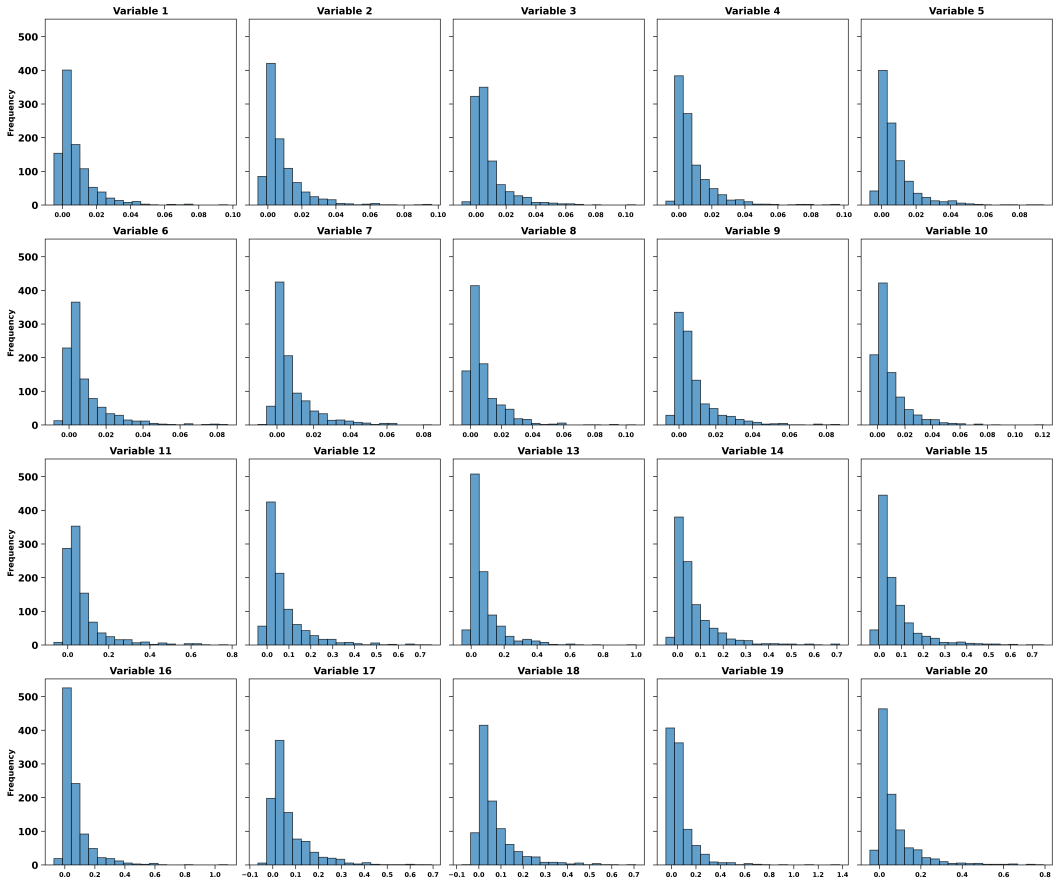

Figure 6: Shapley values for the synthetic data sets with equal distributions. All variables contribute equally to the near-zero MMD.

To validate these results, we analyze the marginal distributions of variables for males and females, as shown in Figure 8. The analysis confirms that variables with distinct marginal distributions between males and females (e.g., $s3$ and $s4$) have high positive Shapley values, reflecting their role in increasing the MMD. Conversely, variables with similar distributions (e.g., $s1$) exhibit zero or infinitesimal Shapley values, implying they do not contribute to the overall MMD and the discrepancy between the two distributions.

G.4 HSIC Feature Selection Case Study

We demonstrate how attributing the HSIC between input features and the target variable to individual features can support feature selection by quantifying their respective contributions to the overall statistical dependence. We compare our approach with five feature importance methods: HSICLasso [Climente-González et al., 2019], Mutual Information (MI), Lasso, K–Best, and Tree Ensemble (with feature permutations). Experiments were conducted on seven datasets, where we trained Gaussian Process (GP) models with an RBF kernel, using only the top 20% of features ranked by each selection method.

Table 8 reports the five-fold cross-validated mean and standard deviation for the GP models trained on the selected features. We use mean absolute percentage error (MAPE) for regression tasks and accuracy for classification tasks. For kernel computation in HSIC, we use an RBF kernel for features and regression targets, and a categorical kernel for classification targets, with bandwidth selected via the median heuristic. PKeX-Shapley consistently delivers strong results across all datasets. On regression problems, it yields better MAPE on *breast cancer*, *skillcraft*, and *parkinson*, while other methods often incur higher error or greater variance. For classification, PKeX-Shapley attains accuracies above 80% on all three datasets and is competitive with the baselines: it is the best method

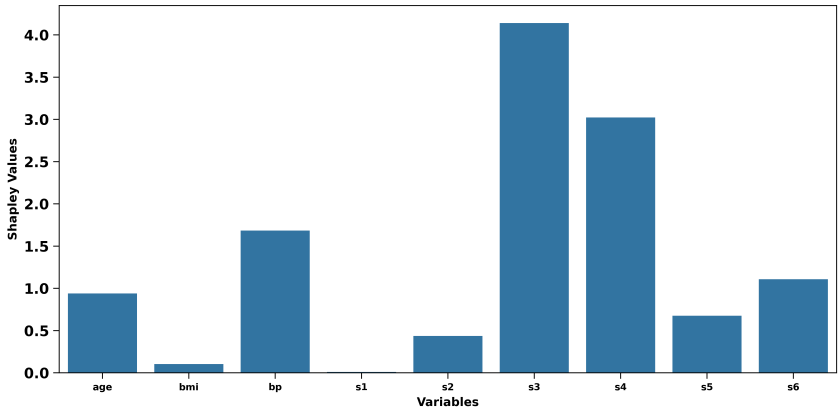

Figure 7: Shapley values explaining MMD between male and female subsets in the UCI Diabetes data set.

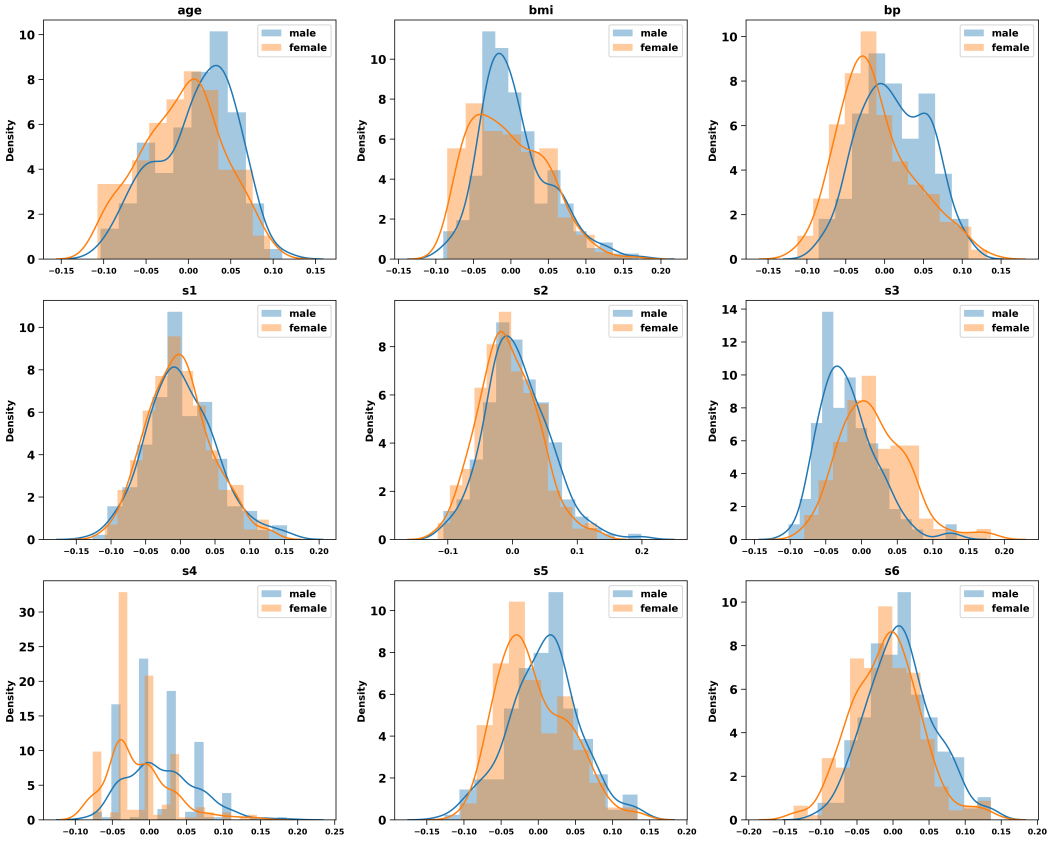

Figure 8: Marginal distributions of variables for male and female subsets in the UCI Diabetes dataset.

on *Wisconsin* (tied with K–Best), whereas on *sonar* and *ionosphere* Mutual Information, Lasso, and Tree Ensemble achieve higher accuracy. Notably, it outperforms HSICLasso—a method specifically tailored for feature selection—on 5 out of 7 datasets, tying on one and only falling short on *ionosphere*. Several baselines report a MAPE of exactly 1.000 on the regression datasets. This is the error of a predictor that outputs zero on a zero-mean (standardized) target: the GP trained on the features selected by these baselines reverted to predicting the target mean, i.e., the selected features carried no usable signal for the GP.

Table 8: Performance (mean±standard deviation) when training on the top 20% of features. Datasets *breast cancer*, *skillcraft*, *sml*, and *parkinson* are regression (MAPE, lower is better); *sonar*, *Wisconsin*, *ionosphere* are classification (accuracy, higher is better).

Method	**sonar**	**Wisconsin**	**ionosphere**	**breast cancer**	**skillcraft**	**sml**	**parkinson**
	accuracy (↑)			mean absolute percentage error (↓)			
PKeX-Shapley	$0.808_{\pm 0.030}$	$\mathbf{0.909_{\pm 0.015}}$	$0.878_{\pm 0.036}$	$\mathbf{0.850_{\pm 0.010}}$	$\mathbf{1.000_{\pm 0.020}}$	$0.999_{\pm 0.001}$	$\mathbf{0.125_{\pm 0.022}}$
HSICLasso	$0.808_{\pm 0.044}$	$0.884_{\pm 0.010}$	$0.912_{\pm 0.035}$	$1.000_{\pm 0.080}$	$2.175_{\pm 0.429}$	$1.354_{\pm 0.726}$	$1.170_{\pm 0.269}$
MI	$\mathbf{0.875_{\pm 0.053}}$	$0.900_{\pm 0.015}$	$\mathbf{0.937_{\pm 0.023}}$	$1.000_{\pm 0.080}$	$1.134_{\pm 0.099}$	$\mathbf{0.196_{\pm 0.071}}$	$0.214_{\pm 0.004}$
Lasso	$0.842_{\pm 0.038}$	$0.900_{\pm 0.024}$	$0.932_{\pm 0.021}$	$1.000_{\pm 0.080}$	$1.821_{\pm 0.680}$	$1.000_{\pm 0.000}$	$1.000_{\pm 0.000}$
K–Best	$0.779_{\pm 0.040}$	$\mathbf{0.909_{\pm 0.015}}$	$0.869_{\pm 0.018}$	$1.000_{\pm 0.080}$	$1.134_{\pm 0.099}$	$0.257_{\pm 0.086}$	$1.018_{\pm 0.076}$
Tree Ens.	$0.837_{\pm 0.059}$	$0.887_{\pm 0.033}$	$0.926_{\pm 0.031}$	$1.000_{\pm 0.080}$	$2.175_{\pm 0.429}$	$1.000_{\pm 0.000}$	$0.214_{\pm 0.004}$

H FAQ

How Does PKeX-Shapley Differ from Mohammadi et al. [2026b]?

In Mohammadi et al. [2026b], the authors propose an exact Shapley computation method for stochastic attribution in FANOVA GP models. Similar to PKeX-Shapley, their approach leverages Newton's identities and symmetric polynomial representations to obtain closed-form attributions. However, the two methods differ in both scope and capability.

PKeX-Shapley is designed specifically for kernel methods with product kernels. It employs the functional decomposition value function to compute exact Shapley values. While this formulation enables exact computation of the mean Shapley values, it does not extend naturally to higher-order moments. In particular, when applied to Gaussian processes, computing the variance or covariance structure of Shapley values is nontrivial, since orthogonality does not generally hold for product kernels. As a result, our method cannot be directly extended to explain GP models in a stochastic way. In addition, PKeX-Shapley computes all d Shapley values in $O(d^2 n)$, while the one proposed by Mohammadi et al. [2026b] has the same complexity for computing a single Shapley value.

In addition, Mohammadi et al. [2026b] focus on FANOVA GPs, whose kernels admit a functional ANOVA decomposition. Compared to product kernels, this class of kernels is more restrictive and often harder to train in practice. Nonetheless, the additional structure provides a key advantage: it enables exact polynomial-time computation not only of the mean but also of the variance and covariance of stochastic Shapley values—something that product kernels cannot achieve. This distinction highlights the central difference between their method and ours.

How Does PKeX-Shapley Differ from RKHS-SHAP [Chau et al., 2022]?

RKHS-SHAP is the first kernel method-specific SHAP-based algorithm. While the authors still employ the conditional expectation value function $\tilde{\nu}_{\boldsymbol{x}}(S) = \mathbb{E}[f(X) \mid X_S = \boldsymbol{x}_S]$, they use the fact that for function f in the RKHS $\mathcal{H}_k$, it is possible to estimate $\tilde{\nu}_{\boldsymbol{x}}(S)$ non-parametrically utilizing a tool known as conditional kernel mean embedding. Specifically, this leads to the following expression:

$$\tilde{\nu}_{\boldsymbol{x}}(\mathcal{S}) = \boldsymbol{\alpha}^\top \mathbf{K}(\mathbf{K}_{\mathcal{S}} + n\lambda I)^{-1} k_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}}, \boldsymbol{x}_{\mathcal{S}})$$

which of course, is different from our functional decomposition value function.

Another crucial difference between PKeX-Shapley and Chau et al. [2022] is that the former uses two recurrences to compute all Shapley values in quadratic time, while the latter uses a subset of coalition samples and computes the Shapley value using a regressor, similar to Kernel SHAP.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state that PKeX-Shapley computes exact Shapley values for product-kernel methods in $O(d^2 n)$ time (amortized linear per feature), using a parameter-free functional-decomposition value function. All claims are supported by theoretical results and empirical experiments.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion explicitly discusses the limitations: the method applies only to product kernels (with supported and unsupported kernels listed in Appendix A), the attributions are exact only for the neutral-factor value function and are relative to the model's factorization (Section 3 and Appendix B), and other value functions are outside the scope of the recursions. Memory requirements and the blockwise implementation are discussed in Section 3 and Appendix F. The introduction also notes that this narrows scope compared to generic kernel methods. Future directions (higher-order attributions, extensions beyond product kernels) are also discussed.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or when images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All propositions and theorems are proved in full in Appendix D. The main text provides proof sketches and intuition (e.g., the Newton's-identity recursion and the prefix–suffix construction). Assumptions (product kernel structure, coefficients $\boldsymbol{\alpha}$) are stated explicitly in each result.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Experimental details (datasets, kernel choices, hyperparameter optimisation via Optuna, cross-validation procedure) are provided in Appendix G, with the blockwise implementation in Appendix F. The code is publicly available at the link given in the introduction (`https://github.com/Majeed7/PKeX-Shapley`).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: A Python implementation is publicly available (link in the introduction). The repository includes code to reproduce all experiments reported in the paper.

   Guidelines:

   - The answer NA means that the paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix G describes the experimental setup in full, including the SVM hyperparameter optimisation via Optuna, the GP training procedure with k-fold cross-validation, dataset sources, kernel choices, and bandwidth selection via the median heuristic; the protocols of the real-world experiments are given in Appendix G.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All experiments that compare methods report mean $\pm$ standard deviation (error bars in figures, subscript notation in tables). The MMD experiments replicate over 1 000 runs and show KDE plots of the Shapley-value distributions.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experiments section states that all experiments were run on a 24-core machine with 16 GB RAM. PKeX-Shapley is CPU-only; no GPU resources were used.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The paper conforms to the NeurIPS Code of Ethics.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This is a foundational methods paper on exact Shapley-value computation for kernel models. It introduces no new data collection, generative models, or systems with direct societal deployment. No negative societal impacts are foreseen.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This is not applicable to this study.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: To our knowledge, we have acknowledged all the relevant works and code bases we used in our study.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: A new Python package implementing PKeX-Shapley is released alongside the paper (link in the introduction). The repository includes a tutorial notebook and instructions for reproducing all experiments.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Not applicable to our study.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: We only used LLMs for polishing the text.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2026/LLM) for what should or should not be described.