Deep Polynomial Chaos Expansion

Johannes Exenberger
CAIML, TU Wien
TU Graz

Sascha Ranftl*
Brown University
Courant Institute NYU

Robert Peharz*
TU Graz
Graz Center for Machine Learning

Abstract

Polynomial chaos expansion (PCE) is a classical and widely used surrogate modeling technique in physical simulation and uncertainty quantification. By taking a linear combination of a set of basis polynomials—orthonormal with respect to the distribution of uncertain input parameters—PCE enables tractable inference of key statistical quantities such as (conditional) means, variances, covariances, and Sobol sensitivity indices, which are essential for understanding the modeled system and identifying influential parameters and their interactions. The applicability of PCE to high-dimensional problems is limited by poor scalability, as the number of basis functions grows exponentially with the number of parameters. In this paper, we address this challenge by combining PCE with ideas from tractable probabilistic circuits, resulting in *deep polynomial chaos expansion* (DeepPCE)—a deep generalization of PCE that scales effectively to high-dimensional input spaces. DeepPCE achieves predictive performance comparable to that of multilayer perceptrons (MLPs), while retaining PCE's ability to compute *exact* statistical inferences via simple forward passes. In contrast, such computations in MLPs require costly and often inaccurate approximations, such as Monte Carlo integration.

We provide the code for DeepPCE at `https://github.com/jxnb/deep-polynomial-chaos-expansion`.

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

1 INTRODUCTION

Numerical simulations of partial differential equations (PDEs) are key tools in thermodynamics, solid-state physics, fluid dynamics, and many other areas of the physical sciences. The accuracy of PDE solvers typically depends on several uncertain input parameters, such as boundary conditions, initial states, and physical properties. Bayesian methods are often used to estimate these parameters, but they become infeasible with increasing input dimension or computationally demanding simulations. A common workaround is to train a *surrogate model* on a limited number of simulations, enabling efficient exploration of the parameter space.

Polynomial chaos expansion (PCE) is a widely used surrogate model, originally introduced by Wiener (1938) for Gaussian inputs and later extended to a broader range of distributions (Xiu and Karniadakis, 2002). The key idea is to approximate the *response function*—i.e., the simulation output—as a linear combination of polynomial basis functions that are orthonormal with respect to the input distribution. This orthonormality offers both theoretical and computational advantages. In theory, PCE yields an optimal polynomial approximation in least-squares sense and, in the context of stochastic differential equations, allows coefficient computation similar to Galerkin projection (Galerkin, 1915; Ghanem and Spanos, 1991). In practice, PCE enhances data efficiency and simplifies the computation of (conditional) expectations, variances, covariances, and Sobol sensitivity indices (Sobol', 1990). These quantities have closed-form expressions when derived from PCE surrogates (Crestaux et al., 2009; Sudret, 2008).

However, PCE scales poorly to high-dimensional parameter spaces, as the number of polynomial terms grows combinatorially with the number of inputs. Truncation schemes (Mühlpfordt et al., 2018), sparse methods (Lüthen et al., 2021), and adaptive or greedy approaches (Blatman and Sudret, 2011; Lüthen et al., 2022) improve scalability to a certain extent, but ultimately

* Shared last authorship.

break down in high-dimensional settings. Due to these limitations, neural networks are increasingly used as surrogate models (Karniadakis et al., 2021; Raissi et al., 2019), as they often achieve better data fit for complex, high-dimensional physical systems. A major limitation of neural surrogates is that expectations and Sobol indices are intractable and must be estimated via expensive and often imprecise Monte Carlo simulations.

In this paper, we propose a scalable and principled extension of PCE to high-dimensional inputs by combining it with a deep circuit architecture. This idea is inspired by the framework of *probabilistic circuits* (Choi et al., 2020), which generalize shallow mixture models such as Gaussian mixture models (GMMs) into deep and structured representations. Like PCE, GMMs and other mixture models suffer from the curse of dimensionality. Deep probabilistic circuits overcome this problem by *compactly representing exponentially many mixture components through circuit depth,* enabling effective modeling of high-dimensional distributions (Peharz et al., 2020a,b). We apply the same principle to introduce *deep polynomial chaos expansion* (DeepPCE), which can represent exponentially many orthogonal polynomial terms compactly.

In experiments, we show that DeepPCE scales PCE to problems with thousands of input parameters, far exceeding the capabilities of existing approaches. In terms of data fit, DeepPCE performs comparably to other domain-agnostic neural network classes like multilayer perceptrons (MLPs). Importantly, we derive exact and efficient methods for computing (conditional) means, variances, and covariances in DeepPCE, from which exact Sobol indices follow. On benchmarks with known ground truth Sobol indices, DeepPCE matches these faithfully, whereas neural surrogates approximate them coarsely, even when using orders of magnitude more compute for Monte Carlo estimation. In summary, our main contributions are:

- We propose a deep generalization of classical PCE, the *deep polynomial chaos expansion* (DeepPCE), based on a deep circuit structure over orthogonal polynomial basis functions. Unlike classical PCE, DeepPCE compactly represents a combinatorial number of polynomial terms and scales to high-dimensional parameter spaces.
- We derive exact formulas for statistical moments (means, variances, covariances) and Sobol sensitivity indices within the DeepPCE framework.
- In experiments, we show that DeepPCE (i) matches the predictive performance of MLPs on high-dimensional regression tasks, (ii) enables scalable sensitivity analysis on common synthetic uncertainty quantification benchmarks and (iii) serves as a scalable surrogate for high-dimensional PDE benchmarks, namely Darcy flow and steady-state diffusion.

2 BACKGROUND

Random variables are written as uppercase letters (e.g. X, Y, Z) and corresponding values with lowercase letters (e.g. x, y, z). The set of integers $\{0, \ldots, K\}$ is denoted by $[K]$.

2.1 Polynomial Chaos Expansion (PCE)

Numerical methods for modeling physical systems are often governed by a large number of uncertain input parameters, such as boundary and initial conditions or material properties. We denote these parameters by $\mathbf{X} = \{X_1, X_2, \ldots, X_D\}$, forming a D-dimensional random vector. In PCE, it is common to assume a factorized input distribution $p(\mathbf{x}) = \prod_d p(x_d)$, although this assumption can be relaxed (Navarro et al., 2014). The *response function* $f^*(\mathbf{X})$, i.e., the output of the numerical simulation, is approximated using a polynomial expansion.

For a single input variable $X = X_1$, the PCE of order K is given as the linear combination

$$f_{\text{PCE}}(x) = \sum_{i=0}^{K} w_i \, \psi_i(x), \tag{1}$$

where $\psi_i(x)$ are polynomial basis functions of degree i and w_i are real-valued coefficients. Crucially, the polynomials are chosen to be orthonormal with respect to the *inner product induced by the distribution of X:*

$$\mathbb{E}_X[\psi_i(X)\, \psi_j(X)] = \int p(x)\, \psi_i(x)\, \psi_j(x)\, \mathrm{d}x = \delta_{i,j}, \tag{2}$$

where $\delta_{i,j}$ is the Kronecker delta. The original PCE formulation by Wiener (1938) assumed X to be Gaussian, in which case the polynomials $\{\psi_i\}_{i=0}^{K}$ are Hermite polynomials. Xiu and Karniadakis (2002) extended this framework using the Askey scheme of hypergeometric polynomials, deriving orthonormal polynomials for a wide range of input distributions, such as Legendre polynomials for uniform distributions, Jacobi polynomials for Beta distributions, Krawtchouk polynomials for binomial distributions, etc.

For multiple inputs $\mathbf{X} = \{X_1, \ldots, X_D\}$, multivariate basis functions are constructed as tensor products of univariate, orthonormal polynomials

$$\mathbf{\Phi}_{\boldsymbol{\alpha}}(\mathbf{x}) = \prod_{d=1}^{D} \psi_{\alpha_d}(x_d), \tag{3}$$

yielding the multivariate expansion

$$f_{\mathrm{PCE}}(\mathbf{x}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} w_{\boldsymbol{\alpha}} \boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{x}). \tag{4}$$

Here, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_D) \in \mathcal{A}$ is a multi-index, where each $\alpha_d \in [K]$ denotes the polynomial degree associated with X_d, and $\mathcal{A}$ is the set of multi-indices for all terms in the expansion. Because the input distribution $p(\mathbf{x})$ is assumed to be factorized, and the univariate polynomials ψ_{α_d} in Eq. (3) are orthonormal for each X_d, the tensor product polynomials are also orthonormal:

$$\mathbb{E}_{\mathbf{X}}[\boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X})\, \boldsymbol{\Phi}_{\boldsymbol{\alpha}'}(\mathbf{X})] = \\ = \int p(\mathbf{x})\, \boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{x})\, \boldsymbol{\Phi}_{\boldsymbol{\alpha}'}(\mathbf{x})\, \mathrm{d}\mathbf{x} = \delta_{\boldsymbol{\alpha},\boldsymbol{\alpha}'}. \tag{5}$$

The number of possible basis functions is bounded by $|\mathcal{A}| \leq \frac{(K+D)!}{K!D!}$, which grows combinatorially with the number of inputs D and the polynomial order K. As this number quickly becomes infeasible with increasing dimensionality, truncation schemes are often applied (Sudret, 2015). A common approach is to include only multi-indices satisfying $\|\boldsymbol{\alpha}\|_p \leq t$, where $\|\cdot\|_p$ denotes the p-(quasi)norm for $0 < p \leq 1$, and t is a threshold, usually set to the maximum polynomial order K. Truncation schemes or sparse adaptive methods (Lüthen et al., 2021, 2022) improve the scalability of PCE only to a certain extent and require assumptions on the rank and sparsity of the basis sets. Note that here, for simplicity, we describe the case of a scalar response function. The case for multi-valued response functions is straightforward, by devising a PCE for each output.

2.2 Sensitivity Analysis

PCE has closed-form solutions for the expectation and variance of f. Since the zeroth-order polynomial term, corresponding to $\boldsymbol{\alpha}_0 = (0, \ldots, 0)$, is a constant, it follows from Eq. (5) that $\mathbb{E}[\boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X})] = 0, \forall \boldsymbol{\alpha} \neq \boldsymbol{\alpha}_0$, hence the expectation is

$$\mathbb{E}[f(\mathbf{X})] = w_{\boldsymbol{\alpha}_0}. \tag{6}$$

Moreover, again due to Eq. (5), all multiplicative cross-terms cancel in expectation and we get

$$\mathrm{Var}(f(\mathbf{X})) = \mathbb{E}[f^2_{\mathrm{PCE}}(\mathbf{X})] - w^2_{\boldsymbol{\alpha}_0} = \sum_{\boldsymbol{\alpha} \in \mathcal{A} \setminus \{\boldsymbol{\alpha}_0\}} w^2_{\boldsymbol{\alpha}}. \tag{7}$$

PCE also allows to compute "advanced" statistical quantities, such as variances of conditional expectations. Let $\mathcal{I} \subseteq \{1, \ldots, D\}$ be an index set for some parameters in $\mathbf{X}$ and $\neg\mathcal{I} = \{1, \ldots, D\} \setminus \mathcal{I}$. It can be shown that

$$\mathrm{Var}_{\mathbf{X}_{\mathcal{I}}}(\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[f(\mathbf{X}) \mid \mathbf{X}_{\mathcal{I}}]) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_{\mathcal{I}}} w^2_{\boldsymbol{\alpha}}, \tag{8}$$

where $\mathcal{A}_{\mathcal{I}} = \{\boldsymbol{\alpha} \colon \forall j \in \neg\mathcal{I} \colon \alpha_j = 0\}$. These variances of conditional expectations allow to compute prominent sensitivity measures such as *Sobol indices* (Sobol', 1990). These indices are normalized variances of the so-called *Sobol decomposition*, which describes f as a superposition of all possible 2^D interaction terms (Saltelli et al., 2008), including a constant, D individual terms, $\binom{D}{2}$ pairwise terms, etc.

The *first order Sobol indices* S_i describe the direct contribution of each parameter X_i to the variance of the response function, excluding contributions of X_i via all interactive terms with other parameters:

$$S_i = \frac{\mathrm{Var}_{X_i}(\mathbb{E}_{\mathbf{X}_{\neg i}}[f(\mathbf{X}) \mid X_i])}{\mathrm{Var}(f(\mathbf{X}))}. \tag{9}$$

The *total effect Sobol indices* S_{T_i} include also the variance contributions of X_i caused by all interaction terms:

$$S_{T_i} = \frac{\mathbb{E}_{\mathbf{X}_{\neg i}}[\mathrm{Var}_{X_i}(f(\mathbf{X}) \mid \mathbf{X}_{\neg i})]}{\mathrm{Var}(f(\mathbf{X}))} \\ = 1 - \frac{\mathrm{Var}_{\mathbf{X}_{\neg i}}(\mathbb{E}_{X_i}[f(\mathbf{X}) \mid \mathbf{X}_{\neg i}])}{\mathrm{Var}(f(\mathbf{X}))}. \tag{10}$$

In PCE, the indices are given via Eqs. (7) and (8). Other Sobol indices describing, for example, only pairwise interactions can also be computed in PCE.

2.3 (Probabilistic) Circuits

Probabilistic circuits (Choi et al., 2020) are deep architectures composed of sum and product operations, whose structural constraints enable various tractable computations. Although primarily used for probabilistic modeling, they can be applied to regression tasks, in which case they are typically referred to as (structured) circuits.

Definition 1 ((Probabilistic) Circuit)**.** A circuit $\mathcal{C} \equiv (\mathcal{G}, \mathbf{W})$ is a computational graph $\mathcal{G}$ with parameters $\mathbf{W}$ defining a function $\mathcal{C}(\mathbf{X})$ with input variables $\mathbf{X} = \{X_1, \ldots, X_D\}$. Here, $\mathcal{G}$ is an acyclic directed graph with three types of nodes: *input nodes*, *sum nodes* and *product nodes*. In the following, let $\mathrm{in}(c)$ be the set of input nodes of some node $c \in \mathcal{G}$. An *input* node c (i.e. $\mathrm{ch}(c) = \emptyset$) represents a parameterized function $g_c(\mathbf{X}_c)$ over a subset of inputs $\mathbf{X}_c \subseteq \mathbf{X}$, called its *scope*. Internal nodes (i.e. $\mathrm{in}(c) \neq \emptyset$) are either *product* nodes or *sum* nodes, representing operations $\prod_{c' \in \mathrm{in}(c)} f_{c'}(\mathbf{X}_{c'})$ and $\sum_{c' \in \mathrm{in}(c)} w_{c,c'} f_{c'}(\mathbf{X}_{c'})$, respectively, where $w_{c,c'}$ are the sum node parameters. The scope of a sum or product node is recursively given as $\mathbf{X}_c = \bigcup_{c' \in \mathrm{in}(c)} \mathbf{X}_{c'}$. A circuit is a *probabilistic circuit* if $\mathcal{C}(\mathbf{X})$ computes a probability distribution $p(\mathbf{X})$ (Choi et al., 2020; Vergari et al., 2021).

A circuit is a special case of neural network, with non-linear functions in the input layer and linear units

(sum nodes) and multiplicative units (product nodes) in subsequent layers. The set of parameters $\mathbf{W}$ is the union of all parameters of input functions and all sum weights. Gradients of these are easily obtained by backpropagation. Circuits enable tractable inference of different quantities through specific structural properties (Vergari et al., 2021) (Fig. 1). A central structural constraint for our purpuse is *(structured) decomposability*:

Definition 2 ((Structured) Decomposability)**.** A circuit is decomposable when the children of every product node have pairwise disjoint scope. Formally, $\mathbf{X}_{c'} \cap \mathbf{X}_{c''} = \emptyset \ \forall c', c'' \in \text{in}(c) \ \forall c \in \Pi$, where Π is the set of product nodes. A circuit is *structured decomposable* if it is decomposable and if any pair of product nodes with the same scope decomposes the scope in the same way.

Decomposability allows to rewrite high-dimensional integrals w.r.t. inputs $\mathbf{X}$ as nested low-dimensional integrals by observing that (i) integration interchanges with summation and (ii) integration interchanges with decomposable multiplication (Peharz et al., 2015). By induction, an integral over the circuit reduces to the integrals of the input nodes—each over the respective scope—followed by a single forward-pass. Consequently, as long as integration over the input functions is tractable (e.g. closed-form), the integral over the function represented by the whole circuit is tractable as well.

Often circuits are also assumed to be smooth (Choi et al., 2020), which means that for each sum node, all input nodes have the same scope. While not required for tractable integrations, smoothness yields well-defined *probabilistic* circuits, i.e. circuits that represent probability distributions, where tractable integrations allows efficient marginalization and conditioning. In this paper, we use circuits to establish a deep generalization of PCE that still allows to compute expectations, variances and Sobol indices exactly and efficiently.

3 DEEP POLYNOMIAL CHAOS EXPANSION

The core idea of DeepPCE is to use orthonormal PCE basis functions as input nodes in circuits, achieving a structured representation of a PCE that preserves the ability of PCEs to efficiently compute statistical moments and sensitivity indices at the input nodes while also allowing tractable inference based on circuit properties (Fig. 2). For tractable inference to work in circuit structures, it is required that the input nodes encode functions which have closed-form solutions; these functions do not necessarily have to be probability distributions. The integrals for the expectation $\mathbb{E}[f(\mathbf{X})]$

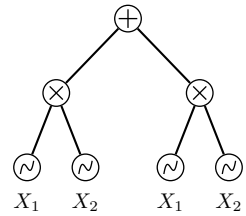

Figure 1: A simple *smooth* and *structured decomposable* circuit with input variables $\{X_1, X_2\}$. The input nodes encode tractable functions with inputs X_1 or X_2, followed by products over nodes with *distinct* scopes $\{X_1\}$ or $\{X_2\}$. The root node computes a sum over product nodes with the same scope $\{X_1, X_2\}$.

and the variance $\text{Var}(f(\mathbf{X}))$ in PCEs can be computed efficiently from the expansion coefficients, as shown in Eqs. (6) and (7).

From a computational perspective, DeepPCE is just as scalable as other circuits; constructing a PCE via hierarchical sums and products of PCEs with smaller scopes enables the modeling of high-dimensional functions beyond the reach of traditional PCE approaches, without any assumption on rank or sparsity of the polynomial basis. This approach follows from two key observations:

Observation 1 (A PCE is a circuit)**.** Circuits can be interpreted as representations of polynomials (Choi et al., 2020; Vergari et al., 2021). Therefore, recalling Eq. (4), a PCE can be represented as a circuit, computing tensor products over polynomials $\mathbf{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}) = \prod_{i \in \mathcal{I}} \psi_{\alpha_i}(X_i)$, where $\mathcal{I} \subseteq \{1, \dots, D\}$ are the indices of the variables in $\mathbf{X}$, followed by weighted sums over these products.

Observation 2 (A DeepPCE recovers a PCE)**.** The distributive law of multiplication over addition allows to obtain a shallow representation $\mathcal{S}$ for any circuit $\mathcal{C}$ (Choi et al., 2020). The shallow representation is

$$\mathcal{S}(\mathbf{X}) = \sum_{i=1}^{P} w_i \prod_{j=1}^{M_i} g_{c_j}^{(i)}(\mathbf{X}_{c_j}^{(i)}) \tag{11}$$

where P denotes the number of product nodes in the circuit and M_i is the number of input nodes reachable from the ith product node, each encoding the function $g_{c_j}^{(i)}$ over scope $\mathbf{X}_{c_j}$. Recalling Eq. (3) and Eq. (4), it becomes visible that a deep circuit with inputs encoding PCEs $(g_{c_1}(\mathbf{X}_{c_1}), \dots g_{c_N}(\mathbf{X}_{c_N}))$ over scopes $\mathbf{X}_{c_n} \subseteq \mathbf{X}$ recovers a PCE over the whole scope $f_{\text{PCE}}(\mathbf{X})$.

3.1 Architecture

We implement the circuit as a layerwise tensorized computational graph (Loconte et al., 2025), following the structure of modern circuit implementations (Peharz et al., 2020a; Vergari et al., 2019). We denote the multivariate circuit output as $\mathbf{Y} = \{Y_1, \ldots, Y_O\}$, representing the function $\mathbf{Y} = f(\mathbf{X})$, where the input nodes are defined as PCEs as described in Sec. 2.1.

PCE Layer Recall that the first layer of a circuit consists of multiple input nodes, each associated with a scope $\mathbf{X}_c \subseteq \mathbf{X}$. In contrast to probabilistic circuits, where input nodes encode probability distributions, each DeepPCE input node c represents a PCE over scope $\mathbf{X}_c$, denoted as $g_c(\mathbf{X}_c)$ in the form of Eq. (4). We use overparameterized circuits (Loconte et al., 2025), with multiple input nodes encoding a PCE $g_{c,n}(\mathbf{X}_c)$ over the same scope, each with a distinct set of weights. In the special case where $\mathbf{X}_c = \mathbf{X}$, the entire input space is used as a single scope, recovering a shallow PCE. We adopt a random partitioning strategy for assigning scopes to input nodes, randomly permuting the input vector $\mathbf{X}$ and splitting it into scopes of equal or similar length, which are then assigned to the input nodes. This approach is inspired by Peharz et al. (2020b); rather than mixing multiple randomly partitioned circuits at the root node, as proposed by the authors, we restrict the model to a single random partition of the input vector $\mathbf{X}$ to preserve circuit decomposability for tractable inference (Vergari et al., 2021). Methods for learning the optimal structure of a circuit are actively developed (Adel et al., 2015; Gens and Domingos, 2013; Yang et al., 2023), which could be integrated in the DeepPCE framework.

Sum-Product Layer Tractable circuits are composed of sequential sum and product nodes. Each product node computes the element-wise or outer product from its input vectors $\mathbf{u}, \mathbf{v}$ with length M and scopes $\mathbf{X}_{c'}$ and $\mathbf{X}_{c''}$ respectively. In tensorized circuit representations, multiple such nodes are computed in parallel, either in Kronecker or Hadamard product layers (Loconte et al., 2025). For each product node p_n, there exist $L \geq 1$ individual sum nodes, each computing a sum over the same scope induced by the input node p_n but with a distinct set of weights. The output of a sum node is $s_{n,l} = \sum_m^M w_m^{l,n} o_m$, where $w_m^{l,n}$ are the elements of the sum nodes' weight vector $\mathbf{w}_{l,n} \in \mathbf{W}_{L \times N \times M}$ and $\mathbf{W}$ is the complete weight tensor of the sum layer. The number of sum nodes L for each product node is a hyperparameter controlling the overparameterization of the circuit. At the output layer, $L = 1$ and each sum s_n simply corresponds to one output dimension $Y_n \in \mathbf{Y}$. The depth of the DeepPCE is determined by the input dimensionality D and the size of the input scopes $\mathbf{X}_c$. In tensorized circuits, product and sum operations are usually abstracted in a single module for computational efficiency as shown in Peharz et al. (2020a), yielding a sum-product layer.

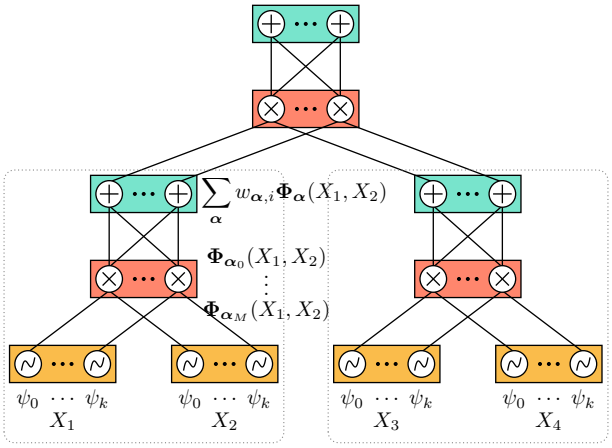

Figure 2: A DeepPCE over inputs $\{X_1, X_2, X_3, X_4\}$ with two input nodes, encoding PCEs over scopes $\{X_1, X_2\}$ and $\{X_3, X_4\}$. Inputs are expanded as orthogonal polynomials $\{\psi_i\}_{i=0}^K$. A layer of outer-product nodes $\otimes$ forms the multivariate basis functions (tensor products) $\{\boldsymbol{\Phi}_{\boldsymbol{\alpha}_0}, \ldots, \boldsymbol{\Phi}_{\boldsymbol{\alpha}_M}\}$. A sum layer $\oplus$ with trainable network weights composes multiple PCEs over the same input scope but with distinct weights, overparameterizing the PCE layer for better expressivity. The input layer is followed by one or several blocks of Kronecker $\otimes$ or Hadamard products $\odot$ and weighted sums $\oplus$ over these products. Repeated sum-product layers can then be stacked to deeper structures. The depth of the DeepPCE is determined by the input dimensionality and the size of the input scopes.

3.2 Training

Unlike classical PCE, DeepPCE does not admit a closed-form solution for the maximum likelihood estimate of the weights. Training therefore relies on standard gradient-based optimization. Because PCEs are computed at the DeepPCE input layer, the values at the input nodes can span several orders of magnitude. This is especially problematic for unbound input distributions such as the normal distribution, potentially destabilizing training. This issue is further amplified by the multiplicative interactions in product nodes, which, in high-dimensional settings, often lead to vanishing or exploding outputs. As a result, DeepPCE may converge to poor local minima or fail to converge entirely.

This challenge is not unique to DeepPCE; training circuit-based models in high-dimensional regimes is known to be difficult with existing optimizers (Liu

et al., 2022). To mitigate the impact of large values introduced by higher-order polynomial terms, we draw inspiration from truncation strategies in the PCE literature (Blatman and Sudret, 2011). Specifically, we initialize the weights of higher-order polynomial terms in the PCE input layer with lower variances than those of lower-order terms:

$$w_{\boldsymbol{\alpha}} = x \cdot \lambda^r \quad \text{with } r = \sum_{\alpha \in \boldsymbol{\alpha}} \alpha, \quad x \sim \mathcal{N}(0, \sigma^2) \tag{12}$$

$\lambda \leq 1$ is a decay hyperparameter and r is the rank of the polynomial term, given by the sum of the expansion terms' multi-index $\boldsymbol{\alpha}$. We find that this variance scaling improves both training stability and convergence, but does not mitigate the problem entirely. Robust parameter initialization strategies for the DeepPCE are thus an important part of further research.

To further improve training stability, we apply batch normalization after each sum layer. Importantly, at test time, batch normalization simply amounts to a linear layer that can be absorbed into the constant (zeroth-degree) term of the polynomial expansion, preserving both orthogonality and tractability of the inference pass (see App. D). For the experiments presented in the paper, we perform randomized training of multiple models, dropping runs with bad parameter sets.

3.3 Inference

By building on both PCE and circuit properties, DeepPCE retains the ability to compute *exact* statistical moments and sensitivity indices. Due to the properties of PCEs, the operations reduce to simple masking operations of individual weights at the input nodes (the PCEs) of the DeepPCE, followed by a standard forward pass through the network. This way, we are able to compute expectations $\mathbb{E}[f(\mathbf{X})]$, covariances $\text{cov}(f(\mathbf{X}), f(\mathbf{X}))$, conditional expectations $\mathbb{E}[f(\mathbf{X}) \mid \mathbf{x}]$, conditional covariances $\text{cov}(f(\mathbf{X}), f(\mathbf{X}) \mid \mathbf{x})$, expectations of conditional covariances $\mathbb{E}[\text{cov}(f(\mathbf{X}), f(\mathbf{X}) \mid \mathbf{X}_c \subset \mathbf{X})]$ and covariances of conditional expectations $\text{cov}(\mathbb{E}[f(\mathbf{X}) \mid \mathbf{X}_c \subset \mathbf{X}], \mathbb{E}[f(\mathbf{X}) \mid \mathbf{X}_c \subset \mathbf{X}])$. These quantities directly feed into the computation of Sobol indices for sensitivity analysis (see Sec. 2.2). As a sanity check, we compare all derived analytical moments with large-sample Monte Carlo estimates for evaluation. Results and detailed derivations of the inference pass are shown in App. B.

Expectations, covariances As an example, we show how to compute the total model expectation $\mathbb{E}[f(\mathbf{X})]$ and covariances $\text{cov}(f(\mathbf{X}))$, making use of both PCE and circuit properties. Under the assumption that $p(\mathbf{X})$ factorizes, the expectation $\mathbb{E}[f(\mathbf{X})]$ is represented by the same circuit as $f(\mathbf{X})$, following from the fact that (i) sums commute with expectations, $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$, and (ii) factorized expectations distribute over decomposable products, $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ if $X \perp\!\!\!\perp Y$. The single-dimensional expectations can hence be "pulled down" to the input nodes (Choi et al., 2020; Peharz et al., 2015). At the input nodes, the expectation $\mathbb{E}[g(\mathbf{X}_c)]$ of a PCE node with scope $\mathbf{X}_c$ can be estimated directly from the weights as shown in Eq. (6) by simply setting all weights $w_{\boldsymbol{\alpha}_i} = 0 \; \forall i > 0$. We get $\mathbb{E}[f(\mathbf{X})]$ by using $\mathbb{E}[g(\mathbf{X}_c)]$ and performing a standard forward pass through the circuit. Similarly, we get $\text{cov}(f(\mathbf{X}), f(\mathbf{X}))$ by computing $\text{cov}(g(\mathbf{X}_c), g(\mathbf{X}_c))$ at the input PCE—multiplicative cross-terms cancel due to orthogonality—according to Eq. (7), followed by a forward pass. Applying the same rationale, we are able to compute other statistical quantities, such as conditional expectations, conditional covariances, expectations of conditional covariances and covariances of conditional expectations, as shown in App. B.

4 RELATED WORK

PCE Numerous methods have been proposed to mitigate the curse of dimensionality in PCEs, aiming to extend their applicability to higher input dimensions and higher polynomial orders, including various (hyperbolic) truncation schemes (Mühlpfordt et al., 2018), compressed sensing (Mathelin and Gallivan, 2012), variational inference (Bhattacharyya, 2020), and sparse (Lüthen et al., 2021), adaptive (Blatman and Sudret, 2011; Hampton and Doostan, 2018; Jakeman et al., 2015) or greedy (Doostan and Owhadi, 2011) approaches, and combinations thereof (Alemazkoor and Meidani, 2017; Lüthen et al., 2022). Various approaches have explored integrating PCEs with machine learning (Torre et al., 2019a), and their performance has been systematically compared in the literature (Kumar, 2025; Shahane et al., 2019; Shahzadi and Soulaïmani, 2021). Common strategies include (i) learning the PCE regression coefficients with (physics-informed) neural networks (Lütjens et al., 2021; Yao et al., 2023; Zheng et al., 2022), (ii) learning the PCE basis functions with neural networks (Bahmani et al., 2025; Zhang et al., 2019), (iii) approximating the entire PCE as a neural network (Cooper, 2021; Schwab and Zech, 2019) or (iv) constructing the PCE on a learned, low-dimensional (latent) representations of input or output spaces (Kontolati et al., 2022; Shustin et al., 2025). All these approaches aim at scalability at the expense of tractable inference. Oladyshkin et al. (2023) proposed orthonormal polynomial bases as neural network acti-

vation functions, constructing a PCE in each neuron. The basis functions are chosen to be orthonormal w.r.t. the previous layer's outputs using the arbitrary PCE approach (Oladyshkin and Nowak, 2012), requiring re-orthonormalization after each weight update. Unlike our DeepPCE framework, this method does not leverage structured, orthogonality-preserving hierarchical tensor products and does not recover a PCE. Aside neural network-based approaches, PCEs have also been combined with bagging (El Moçayd and Seaid, 2021) and Gaussian process regression (Ranftl and von der Linden, 2021; Schobi et al., 2015). Lastly, PCEs can be extended to arbitrary probability measures (Oladyshkin and Nowak, 2012; Wan and Karniadakis, 2006), dependent parameters (Jakeman et al., 2019; Navarro et al., 2014; Torre et al., 2019b), discrete variables (Zheng et al., 2015) and Bayesian PCEs (Arnst et al., 2010; Karagiannis and Lin, 2014; Ranftl and von der Linden, 2021; Tan, 2015).

Probabilistic Circuits Orthogonality has been extensively studied in neural networks, particularly in weight matrices (Li et al., 2021), latent spaces (Ha et al., 2021), and other settings (Achour et al., 2022; Mashhadi et al., 2021; Yang and Tseng, 1996). In contrast, orthonormal constructions within probabilistic circuits remain largely underexplored. Loconte and Vergari (2025) introduced orthonormal circuits that constrain input layers to orthonormal bases and additionally orthonormalize the circuit weights, which yields normalized squared circuits and faster marginalization. Their notion of orthonormality is aiming at enabling faster probabilistic inferences, while DeepPCE leverages orthonormality and circuit properties to enable exact computations of statistical moments. Earlier, Adel et al. (2015) employed SVD-based orthogonality for training circuits, but without constructing orthonormal structures.

Sensitivity Analysis and Uncertainty Quantification Feature importance and sensitivity analysis play a critical role in uncertainty quantification, allowing to identify the most influential input parameters affecting a model's output. Proposed methods comprise model-agnostic, sampling-based methods, such as Shapley values (Ewald et al., 2024; Lundberg and Lee, 2017), and derivative-based (Lamboni, 2020; Lamboni and Kucherenko, 2021; Sobol' and Kucherenko, 2009) or variance-based methods (Homma and Saltelli, 1996; Lo Piano et al., 2021) such as Sobol sensitivity indices (Sobol', 1990). Notably, the orthogonality of PCEs has been exploited to derive analytic expressions for Sobol sensitivity indices (Crestaux et al., 2009; Sudret, 2008). Most generic machine learning models lack a built-in mechanism for variance decomposition, necessitating the use of expensive and potentially inaccurate Monte Carlo-based methods. Regarding uncertainty quantification Abdar et al. (2021), ensemble and Bayesian methods remain popular in the physical sciences (Psaros et al., 2023). Unlike MLPs, probabilistic circuits like DeepPCE allow exact and scalable propagation of input uncertainties.

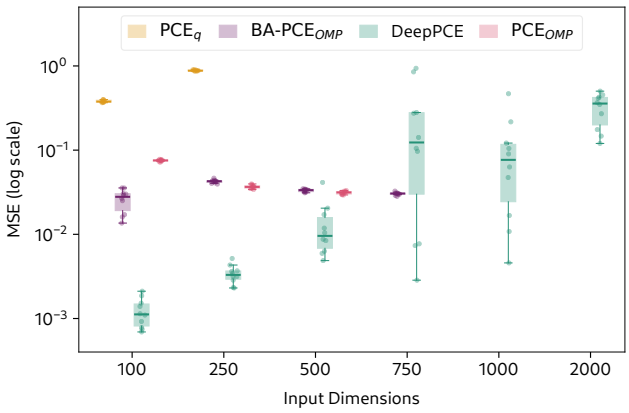

Figure 3: Test performances using the Bratley function with $D = [100, 250, 500, 750, 1000, 2000]$. In contrast to classical PCE approaches, the DeepPCE manages to scale beyond 750 input dimensions and also outperforms the baseline PCEs in terms of predictive performance in lower dimensional settings.

5 EXPERIMENTS

We evaluate the scalability and sensitivity analysis performance of the DeepPCE on several common benchmarks from the uncertainty quantification literature and show its performance as surrogate model on two high-dimensional PDE benchmarks compared to a UNet and a classic MLP. The hyperparameters of the DeepPCE and other neural networks are tuned for each experiment. Each experiment is performed 10 times for each model. Additional experiments, comprehensive results of all runs and hyperparameter settings are described in App. E.

PCE Benchmarks We compare our DeepPCE with three classical PCEs using different approaches to generate the basis sets: (i) a PCE with a hyperbolic truncation scheme ($||\boldsymbol{\alpha}||_{0.6} \leq K$) with $K = 5$ (Blatman and Sudret, 2011) (PCE_q), (ii) a sparse PCE based on orthogonal matching pursuit (OMP) (PCE_OMP) (Baptista et al., 2019), initialized with the same basis as PCE_q, and (iii) a sparse PCE using OMP and the basis adaptive algorithm by Jakeman et al. (2015) (BA-PCE_OMP) to incrementally build a suitable basis. Various sparse and adaptive methods exist; we chose the sparse and adaptive variants based on their popu-

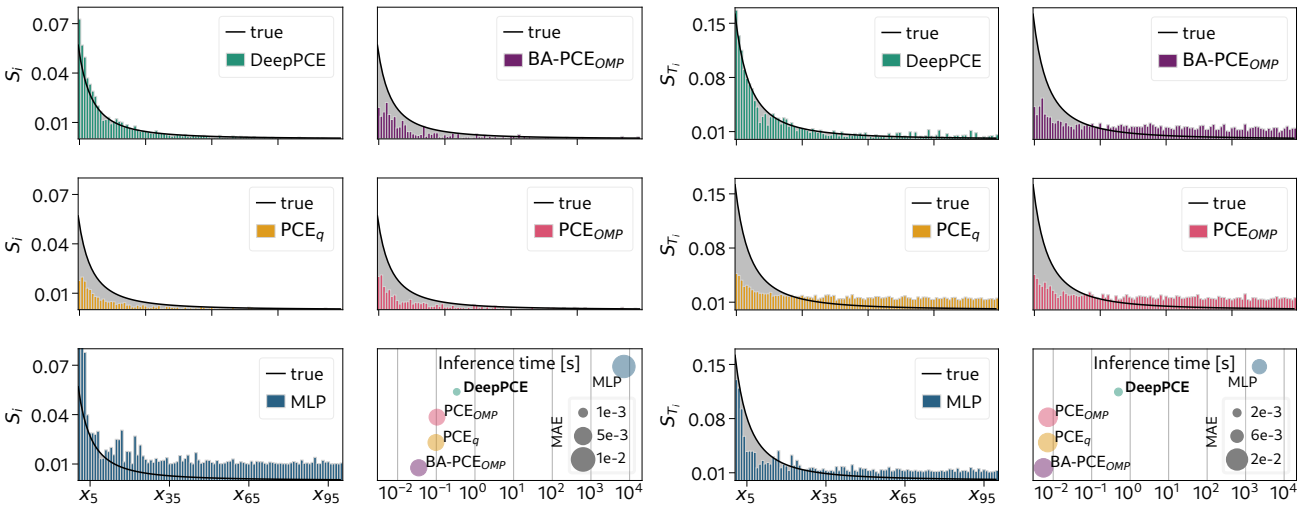

Figure 4: First order Sobol indices (left) and total effect Sobol indices (right) for Sobol G* function with $D = 100$ (panels 1 - 5) and the measured wall clock time for an inference pass as well as mean average errors for the Sobol indices (panel 6). The DeepPCE closely approximates to true variance contributions of each variable, outperforming all classical PCE variants. The MLP's Sobol indices are obtained by Monte Carlo, using a total of 10^8 samples to approximate the Sobol indices for each input X_i. While it shows the worst performance of all models tested and fails to accurately represent the true variance contributions, the computation time to approximate the Sobol indices is longer by a factor of 10^3 to 10^4 compared to the DeepPCE (wall clock time).

larity and performance results from the comprehensive studies of Lüthen et al. (2021, 2022), although they did not show a clear winner. Based on the collection of test functions provided by Wicaksono and Hecht (2023), we chose two common benchmark datasets for uncertainty quantification and sensitivity analysis that can be scaled to arbitrary dimensions: the *Bratley Sum* function (Bratley et al., 1992; Wicaksono and Hecht, 2023) and the *Sobol G** function (Saltelli et al., 2010; Wicaksono and Hecht, 2023), which have known analytical expressions for Sobol sensitivity indices. For each experiment, we use 10000 training samples and an additional 2000 samples for final evaluation. The training procedures for the DeepPCE and BA-PCE$_{\text{OMP}}$ demand a separate validation set, for which we used 2000 samples from the training set, keeping the total number of samples consistent for each model. We provide detailed information on the function definitions and training results as well as additional experiments in App. E.

Fig. 3 shows the performance of all benchmark PCEs and the DeepPCE on the Bratley Sum function for input dimensions $D = [100, 250, 500, 750, 1000, 2000]$. While the standard PCE$_q$ already breaks down with $D = 250$, the sparse methods scale considerably better, with the BA-PCE$_{\text{OMP}}$ managing to scale up to 750 input dimensions. DeepPCE scales beyond this threshold and outperforms the sparse methods in terms of data fit in lower dimensional settings.

First order and total effect Sobol indices (Eqs. (9) and (10)) for the Sobol G* function for $D = 100$ are shown in Fig. 4. The DeepPCE achieves the best approximation of both first order and total effect Sobol indices, outperforming all shallow PCE variants which have a tendency to underestimate the variances. For the MLP, we obtain Sobol indices by Monte Carlo sampling with 10^8 samples. As shown in Fig. 4, the MLP does not manage to provide accurate estimates especially for first order Sobol indices. Additionally, the Monte Carlo estimation has by far the longest runtime. The wall clock time for computing MLP Sobol indices is longer by a factor of 10^3 to 10^4 compared to the analytical computation with the DeepPCE (hours vs. less than a second).

PDE datasets We use two PDE datasets to evaluate the capability of the DeepPCE as a surrogate model for high dimensional physics problems: (i) Darcy flow, a two-dimensional PDE and a common benchmark for data-driven physics models where the input is a permeability field following a log-normal distribution and the output is a velocity field, and (ii) a two-dimensional steady-state diffusion with a diffusion coefficient as input modeled by the logarithm of a Gaussian random field and a pressure field as output. In both datasets, the inputs are 64×64 dimensional fields, resulting in a data dimensionality of 4096—PCEs have so far not been applicable to such high dimensional problems. We use the Darcy flow dataset provided by Zhu and Zabaras (2018), the dataset for the steady-state diffu-

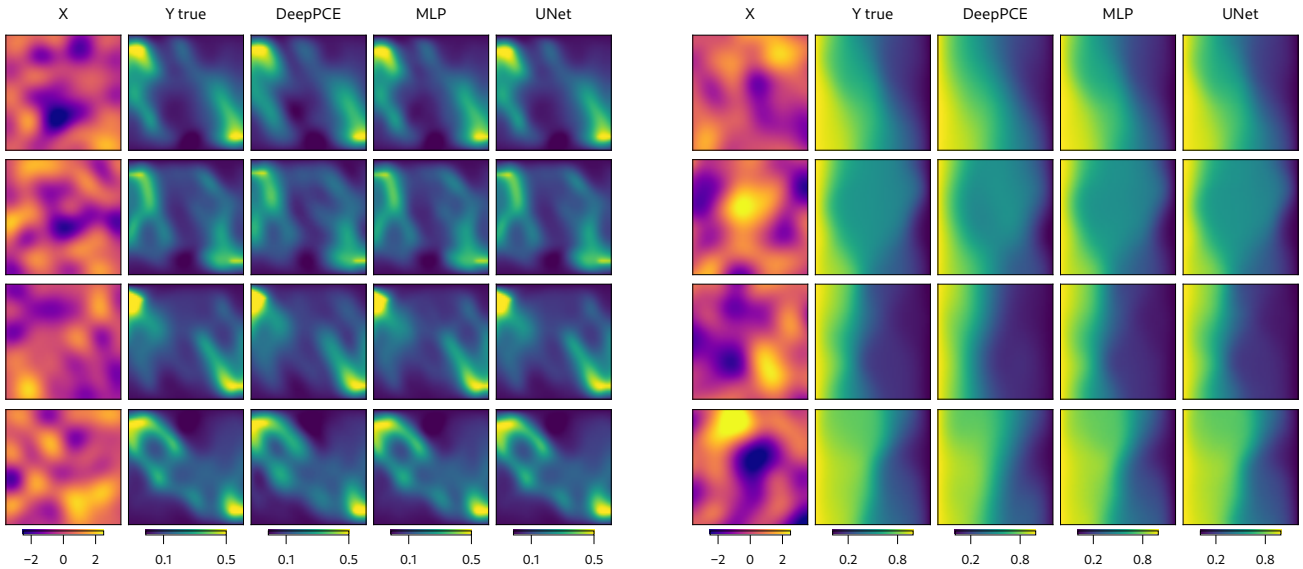

Figure 5: Random test samples for the Darcy flow dataset (left) and the steady-state diffusion dataset (right) from DeepPCE, MLP and UNet. The first column of panels show random input samples, the second column the corresponding true targets. The results show that the DeepPCE manages to closely approximate the PDE results, offering predictive performance that is competitive against the UNet and the MLP, while also offering tractable and exact inference.

sion PDE is based on Tripathy and Bilionis (2018). For both datasets, we use 8000 training samples and 2000 validation samples. The test dataset for the Darcy flow PDE consists of 980 samples, while 2000 test samples are used for the steady state diffusion PDE. Note that the input data in both PDEs is correlated; Sobol indices can thus not be inferred directly, yet the functions can still be approximated by the DeepPCE, allowing to evaluate its predictive performance in high dimensional settings. Random test samples for both datasets are shown in Fig. 5. The UNet shows the best performance, due to its inductive bias specifically suited for image-type data, with relative MSEs in the range of $7 \cdot 10^{-5}$ for both PDE datasets. The average relative MSEs across all then runs for the MLP are 0.003 for the Darcy flow experiment and for 0.0002 for the steady state diffusion. The DeepPCE shows average relative MSEs across all runs of 0.006 for the Darcy flow experiment and 0.006 for the steady state diffusion experiment. While performing comparable to the MLP, the DeepPCE is the only model of the ones included in the comparsion that offers tractable and exact inference.

6 CONCLUSION

In this paper, we introduced Deep Polynomial Chaos Expansion (DeepPCE), a generalization of classical PCE that combines orthonormal polynomial bases within a deep, tractable circuit architecture. DeepPCE overcomes the curse of dimensionality of PCE, without any assumptions on sparsity or on the order of interaction terms, while retaining PCE's ability to exactly compute statistical moments and Sobol sensitivity indices In experiments, DeepPCE manages to accurately compute sensitivity indices and achieves predictive performance comparable to MLPs on high-dimensional PDE problems.

One limitation of DeepPCE is its sensitivity to the weight initialization, which can lead to convergence to bad local minima. While our approach mitigates the problem to a certain extend, adressing parameter initialization and training techniques for circuits with polynomial basis functions is a promising direction for further research. The reliance on factorized input distributions for sensitivity analysis suggests future extensions on correlated inputs. Extending the DeepPCE structure from simple random scopes by including, for example, structure learning methods is another relevant topic for further exploration. Lastly, our results suggest further exploration of orthogonality and (probabilistic) circuits, which remains largely underexplored.

Acknowledgements

This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/COE12 and the Austrian Research Promotion Agency (FFG) (910263) under the AI for Green program. S.R. was financially supported by the Austrian Science Fund (FWF), Grant DOI: 10.55776/J4774.

References

Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., Makarenkov, V., and Nahavandi, S. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297.

Achour, E. M., Malgouyres, F., and Mamalet, F. (2022). Existence, Stability and Scalability of Orthogonal Convolutional Neural Networks. *Journal of Machine Learning Research*, 23(347):1–56.

Adel, T., Balduzzi, D., and Ghodsi, A. (2015). Learning the Structure of Sum-Product Networks via an SVD-based Algorithm. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*.

Alemazkoor, N. and Meidani, H. (2017). Divide and conquer: An incremental sparsity promoting compressive sampling approach for polynomial chaos expansions. *Computer Methods in Applied Mechanics and Engineering*, 318:937–956.

Arnst, M., Ghanem, R., and Soize, C. (2010). Identification of Bayesian posteriors for coefficients of chaos expansions. *Journal of Computational Physics*, 229(9):3134–3154.

Bahmani, B., Kevrekidis, I. G., and Shields, M. D. (2025). Neural chaos: A spectral stochastic neural operator. *Journal of Computational Physics*, 539:114233.

Baptista, R., Stolbunov, V., and Nair, P. B. (2019). Some greedy algorithms for sparse polynomial chaos expansions. *Journal of Computational Physics*, 387:303–325.

Bhattacharyya, B. (2020). Global sensitivity analysis: A Bayesian learning based polynomial chaos approach. *Journal of Computational Physics*, 415:109539.

Blatman, G. and Sudret, B. (2011). Adaptive sparse polynomial chaos expansion based on least angle regression. *Journal of Computational Physics*, 230(6):2345–2367.

Bratley, P., Fox, B. L., and Niederreiter, H. (1992). Implementation and tests of low-discrepancy sequences. *ACM Trans. Model. Comput. Simul.*, 2(3):195–213.

Choi, Y., Vergari, A., and Van den Broeck, G. (2020). Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models. Technical report, University of California, Los Angeles.

Cooper, R. G. (2021). *Augmented Neural Network Surrogate Models for Polynomial Chaos Expansions and Reduced Order Modeling*. PhD thesis, Virginia Tech.

Crestaux, T., Le Maı^tre, O., and Martinez, J.-M. (2009). Polynomial chaos expansion for sensitivity analysis. *Reliability Engineering & System Safety*, 94(7):1161–1172.

Doostan, A. and Owhadi, H. (2011). A non-adapted sparse approximation of PDEs with stochastic inputs. *Journal of Computational Physics*, 230(8):3015–3034.

El Moçayd, N. and Seaid, M. (2021). Data-driven polynomial chaos expansions for characterization of complex fluid rheology: Case study of phosphate slurry. *Reliability Engineering & System Safety*, 216:107923.

Ewald, F. K., Bothmann, L., Wright, M. N., Bischl, B., Casalicchio, G., and König, G. (2024). A Guide to Feature Importance Methods for Scientific Inference. In Longo, L., Lapuschkin, S., and Seifert, C., editors, *Explainable Artificial Intelligence*, pages 440–464, Cham. Springer Nature Switzerland.

Galerkin, BG. (1915). On electrical circuits for the approximate solution of the Laplace equation. *Vestnik Inzh*, 19:897–908.

Gens, R. and Domingos, P. (2013). Learning the Structure of Sum-Product Networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 873–880. PMLR.

Ghanem, R. G. and Spanos, P. D. (1991). *Stochastic Finite Elements: A Spectral Approach*. Springer, New York, NY.

Ha, W., Singh, C., Lanusse, F., Upadhyayula, S., and Yu, B. (2021). Adaptive wavelet distillation from neural networks through interpretations. In *Advances in Neural Information Processing Systems*, volume 34, pages 20669–20682. Curran Associates, Inc.

Hampton, J. and Doostan, A. (2018). Basis adaptive sample efficient polynomial chaos (BASE-PC). *Journal of Computational Physics*, 371:20–49.

Homma, T. and Saltelli, A. (1996). Importance measures in global sensitivity analysis of nonlinear models. *Reliability Engineering & System Safety*, 52(1):1–17.

Jakeman, J. D., Eldred, M. S., and Sargsyan, K. (2015). Enhancing $\ell 1$-minimization estimates of polynomial chaos expansions using basis selection. *Journal of Computational Physics*, 289:18–34.

Jakeman, J. D., Franzelin, F., Narayan, A., Eldred, M., and Plfüger, D. (2019). Polynomial chaos expansions for dependent random variables. *Computer Methods in Applied Mechanics and Engineering*, 351:643–666.

Karagiannis, G. and Lin, G. (2014). Selection of polynomial chaos bases via Bayesian model uncertainty methods with applications to sparse approximation of PDEs with stochastic inputs. *Journal of Computational Physics*, 259:114–134.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440.

Kontolati, K., Loukrezis, D., dos Santos, K. R. M., Giovanis, D. G., and Shields, M. D. (2022). Manifold learning-based polynomial chaos expansions for high-dimensional surrogate models. *International Journal for Uncertainty Quantification*, 12(4).

Kucherenko, S., Rodriguez-Fernandez, M., Pantelides, C., and Shah, N. (2009). Monte Carlo evaluation of derivative-based global sensitivity measures. *Reliability Engineering & System Safety*, 94(7):1135–1148.

Kumar, R. (2025). Comparative analysis of Polynomial Chaos Expansion and Neural Networks for inferring elastic modulus from beam deflection with Matérn covariance. *Journal of Computational Science*, 91:102691.

Lamboni, M. (2020). Derivative-based generalized sensitivity indices and Sobol' indices. *Mathematics and Computers in Simulation*, 170:236–256.

Lamboni, M. and Kucherenko, S. (2021). Multivariate sensitivity analysis and derivative-based global sensitivity measures with dependent variables. *Reliability Engineering & System Safety*, 212:107519.

Li, S., Jia, K., Wen, Y., Liu, T., and Tao, D. (2021). Orthogonal Deep Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(4):1352–1368.

Liu, A., Zhang, H., and den Broeck, G. V. (2022). Scaling Up Probabilistic Circuits by Latent Variable Distillation. In *The Eleventh International Conference on Learning Representations (ICLR)*.

Lo Piano, S., Ferretti, F., Puy, A., Albrecht, D., and Saltelli, A. (2021). Variance-based sensitivity analysis: The quest for better estimators and designs between explorativity and economy. *Reliability Engineering & System Safety*, 206:107300.

Loconte, L., Mari, A., Gala, G., Peharz, R., de Campos, C., Quaeghebeur, E., Vessio, G., and Vergari, A. (2025). What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)? *Transactions on Machine Learning Research*.

Loconte, L. and Vergari, A. (2025). On Faster Marginalization with Squared Circuits via Orthonormalization. In *AAAI 25 Workshop on Connecting Lowrank Representations in AI*.

Lundberg, S. M. and Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Lüthen, N., Marelli, S., and Sudret, B. (2021). Sparse Polynomial Chaos Expansions: Literature Survey and Benchmark. *SIAM/ASA Journal on Uncertainty Quantification*, 9(2):593–649.

Lüthen, N., Marelli, S., and Sudret, B. (2022). Automatic selection of basis-adaptive sparse polynomial chaos expansions for engineering applications. *International Journal for Uncertainty Quantification*, 12(3).

Lütjens, B., Crawford, C. H., Veillette, M., and Newman, D. (2021). PCE-PINNs: Physics-Informed Neural Networks for Uncertainty Propagation in Ocean Modeling. In *ICLR 2021 Workshop on AI for Modeling Oceans and Climate Change*.

Mashhadi, P. S., Nowaczyk, S., and Pashami, S. (2021). Parallel orthogonal deep neural network. *Neural Networks*, 140:167–183.

Mathelin, L. and Gallivan, K. A. (2012). A Compressed Sensing Approach for Partial Differential Equations with Random Input Data. *Communications in Computational Physics*, 12(4):919–954.

Mühlpfordt, T., Findeisen, R., Hagenmeyer, V., and Faulwasser, T. (2018). Comments on Truncation Errors for Polynomial Chaos Expansions. *IEEE Control Systems Letters*, 2(1):169–174.

Navarro, M., Witteveen, J., and Blom, J. (2014). Polynomial Chaos Expansion for general multivariate distributions with correlated variables.

Oladyshkin, S. and Nowak, W. (2012). Data-driven uncertainty quantification using the arbitrary polynomial chaos expansion. *Reliability Engineering & System Safety*, 106:179–190.

Oladyshkin, S., Praditia, T., Kroeker, I., Mohammadi, F., Nowak, W., and Otte, S. (2023). The deep arbitrary polynomial chaos neural network or how Deep Artificial Neural Networks could benefit from data-driven homogeneous chaos theory. *Neural Networks*, 166:85–104.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Broeck, G. V. D., Kersting, K., and Ghahramani, Z. (2020a). Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7563–7574. PMLR.

Peharz, R., Tschiatschek, S., Pernkopf, F., and Domingos, P. (2015). On Theoretical Properties of Sum-Product Networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 744–752. PMLR.

Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., and Ghahramani, Z.

(2020b). Random Sum-Product Networks: A Simple and Effective Approach to Probabilistic Deep Learning. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, pages 334–344. PMLR.

Psaros, A. F., Meng, X., Zou, Z., Guo, L., and Karniadakis, G. E. (2023). Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *Journal of Computational Physics*, 477:111902.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

Ranftl, S. and von der Linden, W. (2021). Bayesian Surrogate Analysis and Uncertainty Propagation. *Physical Sciences Forum*, 3(1):6.

Saltelli, A., Annoni, P., Azzini, I., Campolongo, F., Ratto, M., and Tarantola, S. (2010). Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. *Computer Physics Communications*, 181(2):259–270.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. (2008). *Global Sensitivity Analysis: The Primer*. John Wiley & Sons.

Saltelli, A. and Sobol', I. M. (1995). About the use of rank transformation in sensitivity analysis of model output. *Reliability Engineering & System Safety*, 50(3):225–239.

Schobi, R., Sudret, B., and Wiart, J. (2015). Polynomial-chaos-based kriging. *International Journal for Uncertainty Quantification*, 5(2).

Schwab, C. and Zech, J. (2019). Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in UQ. *Analysis and Applications*, 17(01):19–55.

Shahane, S., Aluru, N. R., and Vanka, S. P. (2019). Uncertainty quantification in three dimensional natural convection using polynomial chaos expansion and deep neural networks. *International Journal of Heat and Mass Transfer*, 139:613–631.

Shahzadi, G. and Soulaïmani, A. (2021). Deep Neural Network and Polynomial Chaos Expansion-Based Surrogate Models for Sensitivity and Uncertainty Propagation: An Application to a Rockfill Dam. *Water*, 13(13).

Shustin, P. F., Ubaru, S., Zimoń, M. J., Lu, S., Kalantzis, V., Horesh, L., and Avron, H. (2025). PCENet: High Dimensional Surrogate Modeling for Learning Uncertainty.

Sobol', I. M. (1990). On sensitivity estimation for nonlinear mathematical models. *Matematicheskoe modelirovanie*, 2(1):112–118.

Sobol', I. M. and Kucherenko, S. (2009). Derivative based global sensitivity measures and their link with global sensitivity indices. *Mathematics and Computers in Simulation*, 79(10):3009–3017.

Sudret, B. (2008). Global sensitivity analysis using polynomial chaos expansions. *Reliability Engineering & System Safety*, 93(7):964–979.

Sudret, B. (2015). Polynomial chaos expansions and stochastic finite element methods. *Risk and Reliability in Geotechnical Engineering*, pages 265–300.

Tan, M. H. Y. (2015). Sequential Bayesian Polynomial Chaos Model Selection for Estimation of Sensitivity Indices. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):146–168.

Torre, E., Marelli, S., Embrechts, P., and Sudret, B. (2019a). Data-driven polynomial chaos expansion for machine learning regression. *Journal of Computational Physics*, 388:601–623.

Torre, E., Marelli, S., Embrechts, P., and Sudret, B. (2019b). A general framework for data-driven uncertainty quantification under complex input dependencies using vine copulas. *Probabilistic Engineering Mechanics*, 55:1–16.

Tripathy, R. K. and Bilionis, I. (2018). Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, 375:565–588.

Vergari, A., Choi, Y., Liu, A., Teso, S., and Van den Broeck, G. (2021). A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference. In *Advances in Neural Information Processing Systems*, volume 34, pages 13189–13201. Curran Associates, Inc.

Vergari, A., Di Mauro, N., and Esposito, F. (2019). Visualizing and understanding Sum-Product Networks. *Machine Learning*, 108(4):551–573.

Wan, X. and Karniadakis, G. E. (2006). Beyond Wiener–Askey Expansions: Handling Arbitrary PDFs. *Journal of Scientific Computing*, 27(1):455–464.

Wicaksono, D. and Hecht, M. (2023). UQTestFuns: A Python3 library of uncertainty quantification (UQ) test functions. *Journal of Open Source Software*, 8(90):5671.

Wiener, N. (1938). The Homogeneous Chaos. *American Journal of Mathematics*, 60(4):897–936.

Xiu, D. and Karniadakis, G. E. (2002). The Wiener-Askey Polynomial Chaos for Stochastic Differential

Equations. *SIAM Journal on Scientific Computing*, 24(2):619–644.

Yang, S.-S. and Tseng, C.-S. (1996). An orthogonal neural network for function approximation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(5):779–785.

Yang, Y., Gala, G., and Peharz, R. (2023). Bayesian Structure Scores for Probabilistic Circuits. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, pages 563–575. PMLR.

Yao, W., Zheng, X., Zhang, J., Wang, N., and Tang, G. (2023). Deep adaptive arbitrary polynomial chaos expansion: A mini-data-driven semi-supervised method for uncertainty quantification. *Reliability Engineering & System Safety*, 229:108813.

Zhang, D., Lu, L., Guo, L., and Karniadakis, G. E. (2019). Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850.

Zheng, M., Wan, X., and Karniadakis, G. E. (2015). Adaptive multi-element polynomial chaos with discrete measure: Algorithms and application to SPDEs. *Applied Numerical Mathematics*, 90:91–110.

Zheng, X., Yao, W., Zhang, Y., and Zhang, X. (2022). Consistency regularization-based deep polynomial chaos neural network method for reliability analysis. *Reliability Engineering & System Safety*, 227:108732.

Zhu, Y. and Zabaras, N. (2018). Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes] Detailed explanations of all models are in App. E.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Not Applicable]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes] We will publish the code in github after acceptance.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes] The theoretical derivation of the inference pass is shown in App. B
   (c) Clear explanations of any assumptions. [Yes] We state all relevant assumptions in the paper, such as in Section 2.

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes] The experiment scripts. including random seeds, are published with the source code on Github.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes] See Sec. 5 and App. E.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] See App. E

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary Material

A CIRCUITS

The DeepPCE shares its structure with (probabilistic) circuits — the constraints imposed on the structure of circuits allow the tractable computation of different inference queries (Vergari et al., 2021). For detailed explanations and proofs of the theoretical foundations and necessary and sufficient conditions for tractability, we refer to Choi et al. (2020); Vergari et al. (2021); Loconte et al. (2025). Here, we will provide a higher-level summary of the important concepts. Similar to other frameworks (Peharz et al., 2020a; Loconte et al., 2025), the DeepPCE is implemented as layerwise tensorized computational graph, parallelizing the operations inside the circuit. It still consists of the same basic computational *nodes*, which we explain briefly. An inner node c (a node that is not an input node) in the circuit receives inputs from one or more other circuit nodes, its *inputs*, which we denote as $\text{in}(c)$. Each node encodes a function, denoted as h_c over a subset of input variables $\mathbf{X}_c \subseteq \mathbf{X}$, also called the *scope* of a node. A circuit has only three distinct computational nodes:

Input node A *input node* c encodes an integrable parameterized function over input variables $\mathbf{X}_c \subseteq \mathbf{X}$. While a input node in probabilistic circuits encodes a probability distribution, in the case of DeepPCE it encodes a PCE, which we denote as g. Recalling Sec. 2.1, a PCE over scope $\mathbf{X}_c$ has the form

$$g(\mathbf{X}_c) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_c} w_{\boldsymbol{\alpha}} \boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}_c). \tag{13}$$

Product node A *product node* c represents a product $\prod_{c \in \text{in}(n)} h_c(\mathbf{X}_c)$. The scope of a product node is the union of the scope of its input nodes, $\bigcup_{c' \in \text{in}(c)} \mathbf{X}_{c'}$. For tractable inference, it is required that a circuit is *structured decomposable* (Definition 2). A circuit is decomposable if all inputs of product units have *disjoint scopes*, so that $\mathbf{X}_{c'} \cap \mathbf{X}_{c''} = \emptyset \; \forall c', c'' \in \text{in}(c)$. For structural decomposability, it is further required that all pairs of product nodes n, m with same scope decompose the scope in the same way, so that $\mathbf{X}_{\text{in}_i(n)} = \mathbf{X}_{\text{in}_i(m)} \; \forall i \in \{1, \ldots, C\}$, where C is the number of inputs in $\text{in}(n)$ and $\text{in}(m)$.

Sum node A *sum node* computes the sum $s(\mathbf{X}_c) = \sum_{c' \in \text{in}(c)} w_{c,c'} h_{c'}(\mathbf{X}_{c'})$, where $\mathbf{w}_n = [w_{c,1}, \ldots, w_{c,C}]$ are the trainable parameters of the node and C are the number of inputs of node c. All sum nodes in a circuit have to satisfy the structural condition *smoothness* for tractable inference. A circuit is smooth if, for all sum nodes, the inputs of a sum node have *same scope*, so $\mathbf{X}_c = \mathbf{X}_{c'} \; \forall c, c' \in \text{in}(c)$.

If a circuit is *structured decomposable*, statistical moments of $f(\mathbf{X})$ can be computed by decomposing the complex integral into smaller, easier to solve integrals based on the observation that integration interchanges with summation and integration interchanges with decomposable multiplication (Choi et al., 2020). Essentially, the integral gets "pushed down" to the input nodes of the circuit, which can be evaluated easily. It is then sufficient to perform a forward pass through the circuit using the evaluated integrals at the circuit leaves to compute the desired statistical moment of the whole circuit $f(\mathbf{X})$.

In tensorized circuit representations, multiple nodes are parallelized and grouped to circuit *layers*. Usually, a layer includes multiple nodes encoding the *same function* over the *same scope*, but each with a distinct set of parameters (Peharz et al., 2020a), increasing the number of parameters and expressivity of the circuit. The layerwise representation yields two types of product layers: (i) product layers that compute the *outer product* $\mathbf{A}_{N \times N} = \mathbf{u} \otimes \mathbf{v}$, where $\mathbf{u}$ is the vector of all N outputs from children nodes with scope $\mathbf{X}_c$ and $\mathbf{v}$ is the vector of all N outputs from children nodes with scope $\mathbf{X}_{c'}$, also referred to in the circuit literature as a *Kronecker product* (Loconte et al., 2025), (ii) product layers computing the *Hadamard product* $\mathbf{a} = \mathbf{u} \odot \mathbf{v}$. Aside from the PCE input layer computing the tensor product $\boldsymbol{\Phi}$, which requires a Kronecker product layer to compute all expansion terms of the full tensor product basis, both Hadamard or Kronecker product layers can be used.

B TRACTABLE INFERENCE

In this section, we derive the tractable computations of statistical moments in DeepPCE via simple forward passes, leveraging PCE and circuit properties.

B.1 Expectations

First, we want to compute the expectation of the whole circuit $f(\mathbf{X}) = \mathbf{Y}$:

$$\mathbb{E}[\mathbf{Y}] = \int p(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{14}$$

We first compute the expectation at the input nodes of the circuit and show how we can pass them through product and sum nodes to get the result at the output.

PCE input layer

We recall that, following from the orthogonality property of PCE (Eq. (5)), the expectation of a single input node PCE g_n over scope $\mathbf{X}_c$ is

$$\mathbb{E}[g_n(\mathbf{X}_c)] = w_{n,\boldsymbol{\alpha}_0} \tag{15}$$

To compute the expectation for an input node, we thus simply have to set all $w_{n,\boldsymbol{\alpha}_i} = 0 \ \forall \ i > 0$.

Product layer

The DeepPCE is a *decomposable* circuit — inputs to product nodes always have disjoint scopes $\mathbf{X}_c \cap \mathbf{X}_{c'} = \emptyset$. The product $p_{n,m}$ of two inputs g_n, g_m is

$$p_{n,m} = g_n(\mathbf{X}_c) g_m(\mathbf{X}_{c'}) \tag{16}$$

Because the circuit is decomposable and the input distribution $p(\mathbf{X})$ factorizes, we get that $g_n(\mathbf{X}_c) \perp\!\!\!\perp g_m(\mathbf{X}_{c'})$. We can thus decompose the expectation of the product:

$$\begin{aligned} \mathbb{E}[p_{n,m}] &= \mathbb{E}[g_n(\mathbf{X}_c) g_m(\mathbf{X}_{c'})] \\ &= \mathbb{E}[g_n(\mathbf{X}_c)]\mathbb{E}[g_m(\mathbf{X}_{c'})] \qquad g_n(\mathbf{X}_c) \perp\!\!\!\perp g_m(\mathbf{X}_{c'}) \end{aligned} \tag{17}$$

We only have to pass the expectations computed in App. B.1 to the product node to compute its expectation.

Sum layer

In case of an outer product layer (sec. App. A), a sum node with index k computes the sum

$$s_k = \sum_{n,m} w_{k,n,m} p_{n,m} \tag{18}$$

The expectation is then

$$\mathbb{E}[s_k] = \sum_{n,m} w_{k,n,m} \mathbb{E}[p_{n,m}] \tag{19}$$

We know the expectation of the product $\mathbb{E}[p_{n,m}]$ from App. B.1. As deeper circuits are only sequences of sum and product nodes, it sufficies to compute the expectations at the input nodes and perform a forward pass through the circuit to compute $\mathbb{E}[\mathbf{Y}]$.

B.2 Covariances

Next we want to compute the covariance of all DeepPCE outputs $\text{cov}(\mathbf{Y}, \mathbf{Y})$, which is a $O \times O$ positive definite matrix. We show how covariances are computed at the input layer can be propagated through the network.

PCE input layer

First, we compute the covariances at the input nodes, which are:

$$\text{cov}(g_n(\mathbf{X}_c), g_m(\mathbf{X}_{c'})) = \mathbb{E}[g_n(\mathbf{X}_c)\ g_m(\mathbf{X}_{c'})] - \mathbb{E}[g_n(\mathbf{X}_{c'})]\mathbb{E}[g_m(\mathbf{X}_{c'})] \tag{20}$$

Recall that we assume a factorized distribution of the inputs $\mathbf{X}$. The covariance for all pairs of input node PCEs $g_n(\mathbf{X}_c), g_m(\mathbf{X}_{c'})$ with disjoint scopes $\mathbf{X}_c \neq \mathbf{X}_{c'}$ is thus zero, so we only have to compute the covariances from input nodes that share the same scope. We know how to compute $\mathbb{E}[g_n(\mathbf{X}_c)]$ and $\mathbb{E}[g_m(\mathbf{X}_c)]$ from Eq. (15). For a PCE input node, we can compute the second moment, i.e. the expectation of a product of two PCEs g_n, g_m, over the same scope as

$$\begin{aligned}\mathbb{E}[g_n(\mathbf{X}_c)\ g_m(\mathbf{X}_c)] &= \mathbb{E}\left[\sum_{\boldsymbol{\alpha}\in\mathcal{A}_c}\sum_{\boldsymbol{\beta}\in\mathcal{A}_c} w_{n,\boldsymbol{\alpha}} w_{m,\boldsymbol{\beta}} \boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}_c)\boldsymbol{\Phi}_{\boldsymbol{\beta}}(\mathbf{X}_c)\right] \\ &= \sum_{\boldsymbol{\alpha},\boldsymbol{\beta}\in\mathcal{A}_c} w_{n,\boldsymbol{\alpha}} w_{m,\boldsymbol{\beta}} \underbrace{\mathbb{E}[\boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}_{c,\mathcal{I}})\boldsymbol{\Phi}_{\boldsymbol{\beta}}(\mathbf{X}_{c,\mathcal{I}})]}_{=\delta_{\boldsymbol{\alpha},\boldsymbol{\beta}}}\end{aligned} \tag{21}$$

where $\boldsymbol{\alpha}$ is the multi-index of the PCE g_n and $\boldsymbol{\beta}$ is the multi-index of the PCE g_m. We know again from the orthogonality property that $\mathbb{E}[\boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X})\boldsymbol{\Phi}_{\boldsymbol{\beta}}(\mathbf{X})] = \delta_{\boldsymbol{\alpha},\boldsymbol{\beta}}$, where $\delta_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ is the Kronecker delta. The second moment is thus

$$\mathbb{E}[g_n(\mathbf{X}_c)\ g_m(\mathbf{X}_c)] = \sum_{\boldsymbol{\alpha}\in\mathcal{A}_c} w_{n,\boldsymbol{\alpha}} w_{m,\boldsymbol{\alpha}} \tag{22}$$

We get the covariance using the result for the expectation from Eq. (15):

$$\begin{aligned}\text{cov}(g_n(\mathbf{X}_c), g_m(\mathbf{X}_c)) &= \mathbb{E}[g_n(\mathbf{X}_c)g_m(\mathbf{X}_c)] - \mathbb{E}[g_n(\mathbf{X}_c)]\mathbb{E}[g_m(\mathbf{X}_c)] \\ &= \sum_{\boldsymbol{\alpha}\in\mathcal{A}_c} w_{n,\boldsymbol{\alpha}} w_{m,\boldsymbol{\alpha}} - w_{n,\boldsymbol{\alpha}_0} w_{m,\boldsymbol{\alpha}_0} \\ &= \sum_{\boldsymbol{\alpha}\in\mathcal{A}_c\setminus\{\boldsymbol{\alpha}_0\}}\ \sum_{\boldsymbol{\beta}\in\mathcal{A}_c\setminus\{\boldsymbol{\beta}_0\}} w_{n,\boldsymbol{\alpha}} w_{m,\boldsymbol{\beta}}\end{aligned} \tag{23}$$

Product layer

To compute second moments at the product layer, we again apply $g(\mathbf{X}_c) \perp\!\!\!\perp g(\mathbf{X}_{c'})$ due to decomposability:

$$\begin{aligned}\mathbb{E}[p_{n,m}p_{k,l}] &= \mathbb{E}[g_n(\mathbf{X}_c)g_m(\mathbf{X}_{c'})g_k(\mathbf{X}_c)g_l(\mathbf{X}_{c'})] \\ &= \mathbb{E}[g_n(\mathbf{X}_c)g_k(\mathbf{X}_c)]\mathbb{E}[g_m(\mathbf{X}_{c'})g_l(\mathbf{X}_{c'})]\end{aligned} \tag{24}$$

We know how to compute the expectation of products over the same scope from Eq. (22). We can now compute the covariance

$$\text{cov}(p_{n,m}, p_{k,l}) = \mathbb{E}[p_{n,m}p_{k,l}] - \mathbb{E}[p_{n,m}]\mathbb{E}[p_{k,l}] \tag{25}$$

by applying Eq. (24) and Eq. (17). We are thus able to compute second moments at a product node by propagating second moments from the input nodes. Covariances can be computed similarly, with the additional need of computing expectations.

Sum layer

Second moments of two sums s_i, s_j over same scope are

$$\begin{aligned}\mathbb{E}[s_i s_j] &= \mathbb{E}\left[\sum_{n,m} w_{i,n,m} p_{n,m} \sum_{k,l} w_{j,k,l} p_{k,l}\right] \\ &= \sum_{n,m,k,l} w_{i,n,m} w_{j,k,l} \mathbb{E}\left[p_{n,m} p_{k,l}\right]\end{aligned} \tag{26}$$

We know how to compute the expectation of the product of two product nodes from Eq. (24). We finally are able to compute the covariance

$$\mathrm{cov}(s_i, s_j) = \mathbb{E}[s_i s_j] - \mathbb{E}[s_i]\mathbb{E}[s_j] \tag{27}$$

using the result for the second moment of sums in Eq. (26) and the expectation of sums in Eq. (19). We show that statistical queries for the second moment $\mathbb{E}[\mathbf{Y}\mathbf{Y}]$ and the covariance $\mathrm{cov}(\mathbf{Y}, \mathbf{Y})$ can be answered by computing those quantities at the input node PCEs. Leveraging the orthogonality property, they reduce to simple operations only involving the learned weights of input nodes.

B.3 Conditional expectations

Now, we consider the case of the conditional expectation $\mathbb{E}[\mathbf{Y}|\mathbf{x}_\mathcal{I}]$, where $\mathcal{I} \subseteq \{1, \ldots, D\}$ is an index set of some parameters in $\mathbf{X}$. The set of marginal variables is denoted as $\mathbf{X}_{\neg\mathcal{I}}$ with $\neg\mathcal{I} = \{1, \ldots, D\} \setminus \mathcal{I}$, so that $\mathbf{X}_\mathcal{I} \cup \mathbf{X}_{\neg\mathcal{I}} = \mathbf{X}$. The conditional expectation is

$$\begin{aligned}\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{x}_\mathcal{I}] &= \int p(\mathbf{x}_{\neg\mathcal{I}}|\mathbf{x}_\mathcal{I}) f(\overbrace{\mathbf{x}_{\neg\mathcal{I}}, \mathbf{x}_\mathcal{I}}^{\mathbf{x}}) \mathrm{d}\mathbf{x}_{\neg\mathcal{I}} \\ &= \int p(\mathbf{x}_{\neg\mathcal{I}}) f(\mathbf{x}_{\neg\mathcal{I}}, \mathbf{x}_\mathcal{I}) \mathrm{d}\mathbf{x}_{\neg\mathcal{I}}\end{aligned} \tag{28}$$

The second step again follows from the assumption that $p(\mathbf{X})$ factorizes, so that $p(\mathbf{x}_{\neg\mathcal{I}}|\mathbf{x}_\mathcal{I}) = p(\mathbf{x}_{\neg\mathcal{I}})$. We can write the conditional expectation of a PCE at the input layer as

$$\mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_n(\mathbf{X}_c)|\mathbf{x}_{c,\mathcal{I}}] = \mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}\left[\sum_{\boldsymbol{\alpha}\in\mathcal{A}_c} w_{n,\boldsymbol{\alpha}} \boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}_{c,\neg\mathcal{I}}) \hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{x}_{c,\mathcal{I}})\right] \tag{29}$$

where $\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{x}_{c,\neg\mathcal{I}})$ is the tensor product of the conditional variables:

$$\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{x}_{c,\mathcal{I}}) = \prod_{x_e \in \mathbf{x}_{c,\mathcal{I}}} \psi_{\alpha_e}(x_e) \tag{30}$$

Based on Eq. (29), we get

$$\begin{aligned}\mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_n(\mathbf{X}_c)|\mathbf{x}_{c,\mathcal{I}}] &= \sum_{\boldsymbol{\alpha}\in\mathcal{A}_c} w_{n,\boldsymbol{\alpha}}\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{x}_{c,\mathcal{I}}) \underbrace{\mathbb{E}[\boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}_{c,\neg\mathcal{I}})]}_{\substack{=1 \text{ if } \alpha_i=0\ \forall\ \alpha_i\in\boldsymbol{\alpha},\\ \text{else } 0}} \\ &= \sum_{\boldsymbol{\alpha}\in\mathcal{A}_{c,\mathcal{I}}} w_{n,\boldsymbol{\alpha}}\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{x}_{c,\mathcal{I}}) \quad \text{with } \mathcal{A}_{c,\mathcal{I}} = \{\boldsymbol{\alpha}:\ \forall j\in\neg\mathcal{I}:\alpha_j=0\} \end{aligned} \tag{31}$$

It follows from the orthogonality property of the PCE that $\mathbb{E}[\boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}_{c,\neg\mathcal{I}})] = 1$ only if $\{\boldsymbol{\alpha}:\ \forall j\in\neg\mathcal{I}:\alpha_j=0\}$. To compute the conditional expectation, we set all input layer PCE weights to zero not meeting this requirement. Recycling our computations for expectations in product and sum layers (App. B.1), we only have to perform a forward pass to compute $\mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{x}_{\mathcal{I}}]$.

B.4 Conditional covariances

We compute the conditial covariance $\text{cov}_{\mathbf{X}_{\neg\mathcal{I}}}(\mathbf{Y},\mathbf{Y}|\mathbf{x}_{\mathcal{I}})$ using the same method as described in App. B.2. The PCE conditional second moment is

$$\begin{aligned}&\mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_n(\mathbf{X}_{c,\neg\mathcal{I}}|\mathbf{x}_{\mathcal{I}})\ g_m(\mathbf{X}_{c,\neg\mathcal{I}}|\mathbf{x}_{\mathcal{I}})] = \\ &\qquad = \mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}\left[\sum_{\boldsymbol{\alpha},\boldsymbol{\beta}\in\mathcal{A}_c} w_{n,\boldsymbol{\alpha}}w_{m,\boldsymbol{\beta}}\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{x}_{\mathcal{I}})\hat{\boldsymbol{\Phi}}_{\boldsymbol{\beta}}(\mathbf{x}_{\mathcal{I}})\boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}_{c,\neg\mathcal{I}})\boldsymbol{\Phi}_{\boldsymbol{\beta}}(\mathbf{X}_{c,\neg\mathcal{I}})\right] \\ &\qquad = \sum_{\boldsymbol{\alpha},\boldsymbol{\beta}\in\mathcal{A}_c} w_{n,\boldsymbol{\alpha}}w_{m,\boldsymbol{\beta}}\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{x}_{\mathcal{I}})\hat{\boldsymbol{\Phi}}_{\boldsymbol{\beta}}(\mathbf{x}_{\mathcal{I}})\underbrace{\mathbb{E}\left[\boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}_{c,\neg\mathcal{I}})\boldsymbol{\Phi}_{\boldsymbol{\beta}}(\mathbf{X}_{c,\neg\mathcal{I}})\right]}_{=\delta_{\boldsymbol{\alpha},\boldsymbol{\beta}}} \\ &\qquad = \sum_{\boldsymbol{\alpha}\in\mathcal{A}_c\setminus\{\boldsymbol{\alpha}_0\}}\ \sum_{\boldsymbol{\beta}\in\mathcal{A}_c\setminus\{\boldsymbol{\beta}_0\}} w_{n,\boldsymbol{\alpha}}w_{m,\boldsymbol{\beta}}\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{x}_{\mathcal{I}})\hat{\boldsymbol{\Phi}}_{\boldsymbol{\beta}}(\mathbf{x}_{\mathcal{I}}) \end{aligned} \tag{32}$$

We can now compute the covariance using Eq. (20). Based on App. B.2, we perform a forward pass to compute $\text{cov}_{\mathbf{X}_{\neg\mathcal{I}}}(\mathbf{Y},\mathbf{Y}|\mathbf{x}_{\mathcal{I}})$.

B.5 Expectations of covariances and covariances of expectations

Additional to computing statistical moments, we are interested in performing sensitivity analysis by identifying the input variables $X_d \in \mathbf{X}$ that contribute most to the variance of $f(\mathbf{X}) = \mathbf{Y}$ represented by the DeepPCE.

Recalling the definition of Sobol indices from Eqs. (9) and (10), we are ultimately interested in computing the covariance of expectations conditioned on the random variables $\mathbf{X}_{\mathcal{I}} \subset \mathbf{X}$, $\text{cov}_{\mathbf{X}_{\mathcal{I}}}(\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}])$ as well as the expectation of conditional covariances: $\mathbb{E}_{\mathbf{X}_{\mathcal{I}}}[\text{cov}_{\mathbf{X}_{\neg\mathcal{I}}}(\mathbf{Y}|\mathbf{X}_{\mathcal{I}})]$, quantifying the expectation or covariance based *only* on the randomness of $\mathbf{X}_{\mathcal{I}}$. We again apply the standard formula for the covariance

$$\begin{aligned}\text{cov}_{\mathbf{X}_{\mathcal{I}}}(\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}],\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}]) &= \mathbb{E}_{\mathbf{X}_{\mathcal{I}}}[\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}]\ \mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}]] - \mathbb{E}_{\mathbf{X}_{\mathcal{I}}}[\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}]]^2 \\ &= \mathbb{E}_{\mathbf{X}_{\mathcal{I}}}[\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}]\ \mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}]] - \mathbb{E}[\mathbf{Y}]^2 \end{aligned} \tag{33}$$

For the simplification in the second step, we use the law of total expectation, which states that $\mathbb{E}[\mathbb{E}[Y|X]] = \mathbb{E}[Y]$. We know how to compute $\mathbb{E}[\mathbf{Y}]$ based on the derivation in App. B.1. We further apply our results from Eq. (31) to compute the second moment of the distribution of the conditional expectation at the PCE nodes:

$$\begin{aligned}\mathbb{E}_{\mathbf{X}_{c,\mathcal{I}}}\Big[\mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_n(\mathbf{X}_{c,\neg\mathcal{I}})|\mathbf{X}_{c,\mathcal{I}}]\ \mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_m(\mathbf{X}_{c,\neg\mathcal{I}})|\mathbf{X}_{c,\mathcal{I}}]\Big] &= \\ &= \mathbb{E}\left[\sum_{\boldsymbol{\alpha}\in\mathcal{A}_{c,\mathcal{I}}} w_{n,\boldsymbol{\alpha}}\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{X}_{\mathcal{I}})\sum_{\boldsymbol{\beta}\in\mathcal{A}_{c,\mathcal{I}}} w_{m,\boldsymbol{\beta}}\hat{\boldsymbol{\Phi}}_{\boldsymbol{\beta}}(\mathbf{X}_{\mathcal{I}})\right] \\ &= \sum_{\boldsymbol{\alpha},\boldsymbol{\beta}\in\mathcal{A}_{c,\mathcal{I}}} w_{n,\boldsymbol{\alpha}}w_{m,\boldsymbol{\beta}}\underbrace{\mathbb{E}\left[\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{X}_{\mathcal{I}})\hat{\boldsymbol{\Phi}}_{\boldsymbol{\beta}}(\mathbf{X}_{\mathcal{I}})\right]}_{=\delta_{\boldsymbol{\alpha},\boldsymbol{\beta}}} \end{aligned} \tag{34}$$

where $\mathcal{A}_{c,\mathcal{I}} = \{\boldsymbol{\alpha} :\ \forall j \in \neg\mathcal{I} : \alpha_j = 0\}$ as defined in Eq. (31). Based the orthogonality property of the PCE, the equation reduces to

$$\begin{aligned}\mathbb{E}_{\mathbf{X}_{c,\mathcal{I}}}\Big[\mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_n(\mathbf{X}_{c,\neg\mathcal{I}}|\mathbf{X}_{c,\mathcal{I}})]\ \mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_m(\mathbf{X}_{c,\neg\mathcal{I}}|\mathbf{X}_{c,\mathcal{I}})]\Big] &= \\ &= \sum_{\boldsymbol{\alpha}\in\mathcal{A}_{c,\mathcal{I}}} w_{n,\boldsymbol{\alpha}}w_{m,\boldsymbol{\alpha}}\underbrace{\mathbb{E}\left[\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{X}_{\mathcal{I}})\hat{\boldsymbol{\Phi}}_{\boldsymbol{\alpha}}(\mathbf{X}_{\mathcal{I}})\right]}_{=1} \\ &= \sum_{\boldsymbol{\alpha}\in\mathcal{A}_{c,\mathcal{I}}} w_{n,\boldsymbol{\alpha}}w_{m,\boldsymbol{\alpha}} \end{aligned} \tag{35}$$

We are now able to compute the covariance of conditional expectations at the PCE nodes according to Eq. (33) by inserting Eq. (35):

$$\begin{aligned}\mathrm{cov}_{\mathbf{X}_{c,\mathcal{I}}}(\mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_n(\mathbf{X}_c)|\mathbf{X}_{c,\mathcal{I}}], \mathbb{E}_{\mathbf{X}_{c,\neg\mathcal{I}}}[g_n(\mathbf{X}_c)|\mathbf{X}_{c,\mathcal{I}}]) &= \sum_{\boldsymbol{\alpha}\in\mathcal{A}_{c,\mathcal{I}}} w_{n,\boldsymbol{\alpha}}w_{m,\boldsymbol{\alpha}} - w_{n,\boldsymbol{\alpha}_0}w_{m,\boldsymbol{\alpha}_0} \\ &= \sum_{\boldsymbol{\alpha}\in\mathcal{A}_{c,\mathcal{I}}\setminus\{\boldsymbol{\alpha}_0\}} w_{n,\boldsymbol{\alpha}}w_{m,\boldsymbol{\alpha}} \end{aligned} \tag{36}$$

Finally, we are also able to compute the expectation of conditional covariances using the law of total covariance, stating that $\mathrm{cov}(X,Y) = \mathbb{E}[\mathrm{cov}(X,Y|Z)] + \mathrm{cov}(\mathbb{E}[X|Z], \mathbb{E}[Y|Z])$:

$$\mathbb{E}_{\mathbf{X}_{\mathcal{I}}}[\mathrm{cov}_{\mathbf{X}_{\neg\mathcal{I}}}(\mathbf{Y},\mathbf{Y}|\mathbf{X}_{\mathcal{I}})] = \mathrm{cov}(\mathbf{Y},\mathbf{Y}) - \mathrm{cov}_{\mathbf{X}_{\mathcal{I}}}(\mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}], \mathbb{E}_{\mathbf{X}_{\neg\mathcal{I}}}[\mathbf{Y}|\mathbf{X}_{\mathcal{I}}]), \tag{37}$$

by computing the covariance according to Eq. (20) and the covariance of conditional expectations according to Eq. (36).

C VALIDATION OF MODEL INFERENCE

We test all inference queries by comparing the analytical results with Monte Carlo approximations of those quantities from the same DeepPCE. We compute Monte Carlo approximations with different sample sizes $S = [10^5, 10^6, 10^7, 10^8]$, performing 30 individual Monte Carlo runs per sample size setting. We then test the convergence with increasing sample size and compare it to the analytical solution derived above by performing one sample t-tests for all outputs, showing that the difference between the analytical solutions and the Monte Carlo approximations is not statistical significant for any outputs (Tab. 1). Fig. 6 shows results for a DeepPCE with 8 inputs $\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and 8 outputs, from which we randomly chose a single output for visualization. We iteratively validate the analytical solutions and use them in further Monte Carlo tests as described below.

Expectation and covariance

We first test the result of the analytical computation obtained with the model, $\mathbb{E}[\mathbf{Y}] = \mathbb{E}[\mathcal{M}(\mathbf{X})]$, by comparing it with the Monte Carlo estimate $\mathbb{E}[\mathbf{Y}]_{MC}$:

$$\mathbb{E}[\mathbf{Y}]_{MC} = \frac{1}{n}\sum_{i}^{N}\mathcal{M}(\mathbf{X}) \tag{38}$$

After confirming that $\mathbb{E}[\mathbf{Y}] = \mathbb{E}[\mathbf{Y}]_{MC}$, we apply the analytical expectation in the Monte Carlo approximation of the covariance:

$$\text{cov}(\mathbf{Y},\mathbf{Y})_{MC} = \frac{1}{n-1}\sum_{i}^{N}(\mathcal{M}(\mathbf{X}) - \underbrace{\mathbb{E}[\mathbf{Y}]}_{\text{tested}})^2 \tag{39}$$

Conditional expectation and conditional covariance

For the validation of conditional moments, we condition on variables $\mathbf{X}_{\mathcal{I}} = \{X_1, X_2, X_3, X_5\}$. For tests with fixed conditional values, we chose conditioned values as $\mathbf{X}_{\mathcal{I}} \sim \mathcal{U}(-2, 2)$. We test the conditional expectation similar to Eq. (38):

$$\mathbb{E}[\mathbf{Y}|\mathbf{x}_{\mathcal{I}}]_{MC} = \frac{1}{n}\sum_{i}^{N}\mathcal{M}(\mathbf{X}_{\neg\mathcal{I}}, \mathbf{x}_{\mathcal{I}}) \tag{40}$$

After confirming the results, we use it for the Monte Carlo approximation of the conditional covariance

$$\text{cov}(\mathbf{Y},\mathbf{Y}|\mathbf{x}_{\mathcal{I}})_{MC} = \frac{1}{n-1}\sum_{i}^{N}(\mathcal{M}(\mathbf{X}_{\neg\mathcal{I}}, \mathbf{x}_{\mathcal{I}}) - \underbrace{\mathbb{E}[\mathbf{Y}|\mathbf{x}_{\mathcal{I}}]}_{\text{tested}})^2 \tag{41}$$

The Monte Carlo approximation for the expected conditional covariance is then simply

$$\mathbb{E}[\text{cov}(\mathbf{Y},\mathbf{Y}|\mathbf{X}_{\mathcal{I}})]_{MC} = \frac{1}{n}\sum_{i}^{N}\underbrace{\text{cov}(\mathbf{Y},\mathbf{Y}|\mathbf{x}_i)}_{\text{tested}} \tag{42}$$

Because we get the covariance of conditional expectations just from known quantities (Eq. (37)), we do not have to perform a separate evaluation with Monte Carlo.

Table 1: P-values computed using one sample t-tests, comparing the analytical solution of the statistical inferences to Monte Carlo approximations. The results are not statistical significant for any outputs, indicating that the analytical solution is not different from the Monte Carlo solution.

Outputs	p-values				
	$\mathbb{E}[Y]$	$\text{Var}(Y)$	$\mathbb{E}[Y\|x]$	$\text{Var}(Y\|x)$	$\mathbb{E}[\text{Var}(Y\|X)]$
Y_1	0.875	0.950	0.581	0.172	0.493
Y_2	0.300	0.119	0.575	0.583	0.986
Y_3	0.108	0.158	0.643	0.660	0.873
Y_4	0.132	0.223	0.706	0.495	0.970
Y_5	0.915	0.816	0.521	0.204	0.522
Y_6	0.093	0.058	0.679	0.776	0.916
Y_7	0.382	0.148	0.567	0.559	0.951
Y_8	0.250	0.104	0.591	0.610	0.964

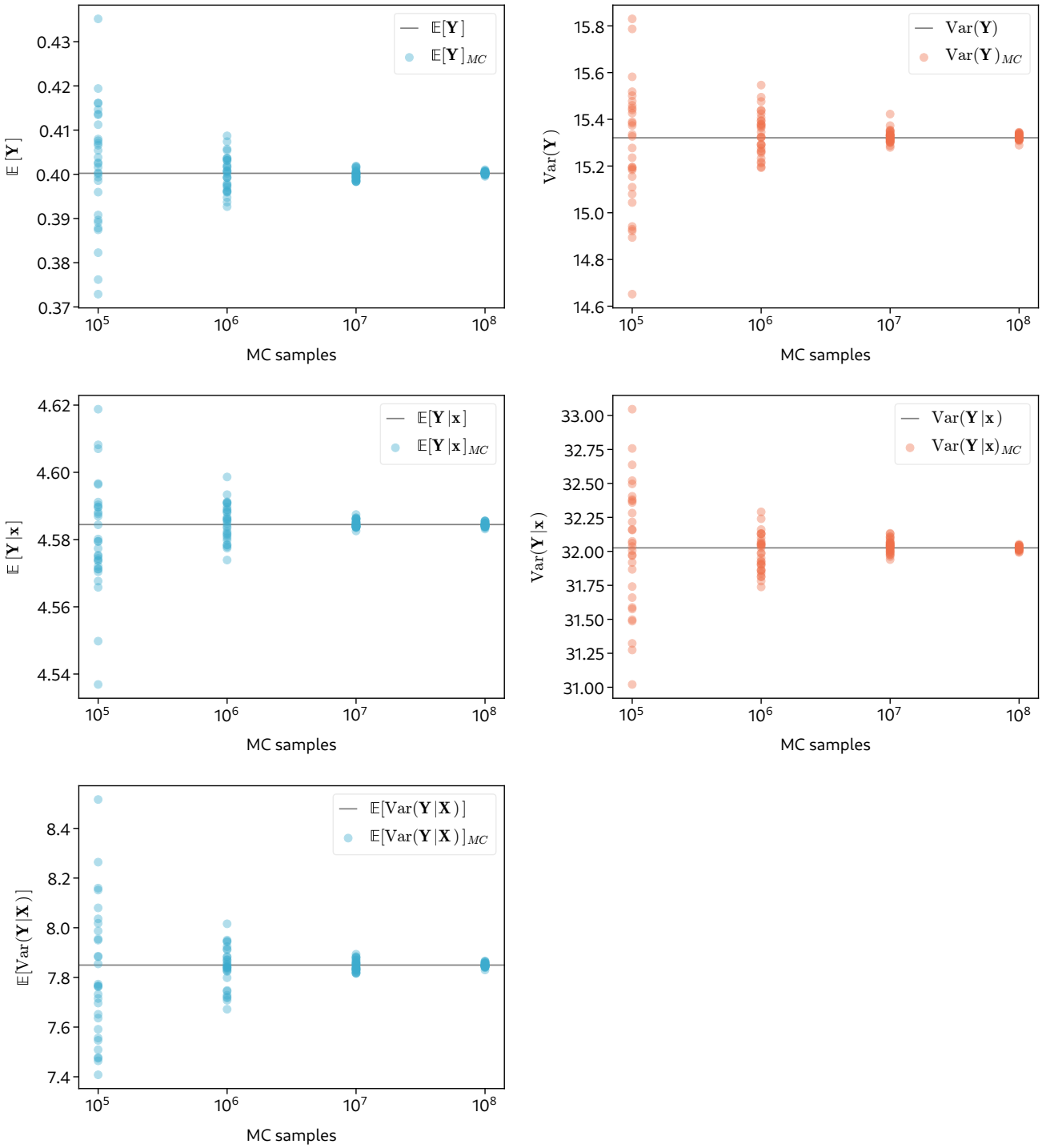

Figure 6: Inference pass evaluation comparing the analytical solution (grey line) to Monte Carlo approximations (blue/orange circles). The first row show unconditional expectation $\mathbb{E}[Y]$ (left) and unconditional variance $\mathrm{Var}(Y)$, the second row conditional expectation $\mathbb{E}[Y|\mathbf{x}]$ and conditional variance $\mathrm{Var}(Y|\mathbf{x})$, the third row shows the expectation of the conditional variance $\mathbb{E}[\mathrm{Var}(Y|\mathbf{X}_{\mathcal{I}})]$

D BATCH NORMALIZATION AND ORTHOGONALITY

We show that orthogonality is preserved in DeepPCE also when batch normalization is applied at the sum layers. Batch normalization, denoted as the function z, applies an affine transformation at inference time:

$$z(\mathbf{Y}) = \gamma \frac{\mathbf{Y} - \mathbb{E}[\mathbf{Y}]}{\sqrt{\mathrm{Var}(\mathbf{Y}) + \varepsilon}} + \beta \tag{43}$$

where $\mathbf{Y}$ is previous hidden layer's output, γ and β are learnable parameters of the batch norm layer and $\mathbb{E}[\mathbf{Y}]$ and $\mathrm{Var}(\mathbf{Y})$ are the estimates for mean and variance respectively and ε is a small constant added for numerical stability. At inference time, the batch statistics are independent from the inputs. The equation can be reformulated as

$$z(\mathbf{Y}) = \hat{\gamma}\mathbf{Y} + \hat{\beta} \tag{44}$$

$$\text{with } \hat{\gamma} = \frac{\gamma}{\sqrt{\mathrm{Var}[\mathbf{Y}] + \varepsilon}}, \quad \hat{\beta} = \beta - \frac{\gamma \mathbb{E}[\mathbf{Y}]}{\sqrt{\mathrm{Var}[\mathbf{Y}] + \varepsilon}} \tag{45}$$

Applied to a PCE $f(\mathbf{X}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} w_{\boldsymbol{\alpha}} \boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X})$ we get:

$$z(f(\mathbf{X})) = \hat{\gamma}\left(\sum_{\boldsymbol{\alpha} \in \mathcal{A}} w_{\boldsymbol{\alpha}} \boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}) \right) + \hat{\beta} \tag{46}$$

The batch norm coefficients γ and β can be subsumed in the weights and the zeroth-order term of the expansion, i.e. the constant $w_{\boldsymbol{\alpha}_0}$, respectively, thus not affecting orthogonality:

$$z(f(\mathbf{X})) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} \hat{\gamma}\ w_{\boldsymbol{\alpha}} \boldsymbol{\Phi}_{\boldsymbol{\alpha}}(\mathbf{X}) + b_{\boldsymbol{\alpha}} \quad \text{with } b_{\boldsymbol{\alpha}} = \begin{cases} \hat{\beta} & \text{if } \alpha_i = 0\ \forall \alpha_i \in \boldsymbol{\alpha} \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

E EXPERIMENTS

E.1 Benchmark functions

We adopt benchmark functions that are widely used in the uncertainty quantification and sensitivity analysis literature. The Sobol G* and Bratley Sum functions are our own re-implementations of examples from the comprehensive collection of test functions in the `uqtestfuns` library (Wicaksono and Hecht, 2023)—we refer the reader there for more detailed information.

E.1.1 Sobol G*

The Sobol G* function (Saltelli and Sobol', 1995) is defined as

$$f(\mathbf{X}, \mathbf{a}, \boldsymbol{\delta}, \boldsymbol{\alpha}) = \prod_{i=1}^{D} g_k(x_i, a_i, \delta_i, \alpha_i) \tag{48}$$

$$\text{where} \quad g_k(x_i, a_i, \delta_i, \alpha_i) = \frac{(1+\alpha_i)|2(x_i + \delta_i - \lfloor x_i + \delta_i \rfloor) - 1|^{\alpha_i} + a_i}{1 + a_i},$$

with $\mathbf{X} \sim \mathcal{U}[0,1]^d$. Each parameter $a_i \in \mathbf{a} \geq 0$ controls the "importance" of the input variable $X_i \in \mathbf{X}$. Smaller values of a_i translate to a higher contribution of X_i to the global variance $\mathrm{Var}(\mathbf{X})$. We generate $\mathbf{a}$ with $a_i = \frac{i-1}{D-1} \cdot 10$. $\boldsymbol{\delta} \sim \mathcal{U}[0,1]^d$ is a shift parameter that influences the shape of the function but not the sensitivity indices and is thus sampled randomly. $\boldsymbol{\alpha} = \mathbf{\frac{1}{2}}$ controls the curvature of the function. The floor operator introduces additional nonlinearity and makes this function an interesting test case. Analytical solutions to first order and total effect Sobol indices are known.

E.1.2 Bratley Sum

This function was introduced in Bratley et al. (1992) and later used in Kucherenko et al. (2009) and Saltelli et al. (2010):

$$f(\mathbf{X}) = \sum_{i=1}^{D} (-1)^i \prod_{j=1}^{i} x_i \tag{49}$$

The joint input distribution is a uniform distribution on the unit hypercube, $\mathbf{X} \sim \mathcal{U}[0,1]^d$. This function features only low-order interactions and favors approximations using sparse, strongly truncated bases. Analytical solution for the total effect Sobol indices are known from Saltelli et al. (2010).

E.1.3 PCE benchmark

This function from Lüthen et al. (2021) is usually used with $D = 100$:

$$f(\mathbf{X}) = 3 - \frac{5}{d}\sum_{i=1}^{d} iX_i + \frac{1}{d}\sum_{i=1}^{d} iX_i^3 + \frac{1}{3d}\sum_{i=1}^{d} i\ln(X_i^2 + X_i^4) + X_1X_2^2 + X_2X_4 - X_3X_5 + X_{51} + X_{50}X_{54}^2 \tag{50}$$

where inputs $\mathbf{X} = \{X_1, \dots X_D\}$ are sampled from uniform distributions $X_i \sim \mathcal{U}(1,2)\ \forall i \neq 20$ and $X_{20} \sim \mathcal{U}(1,3)$. This test function is interesting due to different intervals of the input distributions and the presence of interaction variables. It is usually used to benchmark sparse PCEs, as only low-order interactions are present.

E.2 Experiment details

Each experiment was performed in isolation on a SLURM cluster, using a single NVIDIA A40 GPU (48 GB Memory) with 2 CPU cores and 64 GB memory. We peformed each experiment 10 times for each model to provide estimates of error variances. The DeepPCE, MLP and UNet were trained for a maximum of 500 epochs, using early stopping to prevent overfitting. As described in Sec. 3.2, the DeepPCE is susceptible to bad parameter initializations. To mitigate this, we initialized 10 DeepPCEs and continued training with the best one after 50 epochs. This resulted in no runs converging to bad local minima. Due to the consistent performance of MLPs and UNets, this training procedure was not necessary for those two model classes. For all experiments, we report relative MSEs as

$$R = \frac{\frac{1}{N}\sum_i^N (y - \hat{y})^2}{\frac{1}{N}\sum_i^N y^2} \tag{51}$$

where y are the true values and $\hat{y}$ are the model predictions.

E.3 Hyperparameters

For each experiment, the DeepPCE, MLP and UNet were tuned using the same computational budget. We tuned the learning rate (lr) for all models, using values $[0.001, 0.005, 0.01, 0.05, 0.1]$ and a fixed batch size of 128. Additional tuned parameters for the DeepPCE include the number of sum nodes per layer (num sums), the maximum order of the polynomial expansion (max order), the standard deviation of the Gaussian distribution used for weight initialization (std) as well as the variance decay parameter (λ) described in Eq. (12). We used a fixed input node scope size of 1 and Hadamard product layers. Additional MLP tuning parameters include the number of hidden layers (num hidden) and the number of units per hidden layer (num units), specific UNet tuning parameters are the number of channels at the bottleneck of the encoder (num channels). The hyperparameters for all experiments are shown in Tab. 2, Tab. 3 shows details on model sizes and training time.

For the sparse PCE variants (PCE$_{OMP}$, BA-PCE$_{OMP}$), the stopping criterion for the orthogonal matching pursuit algorithm is the maximum number of active dimensions A, with $A \approx \frac{1}{10}D$, where D is the number of input dimensions.

Table 2: Experiment hyperparameter settings.

	DeepPCE		MLP		UNet	
Darcyflow	lr	0.005	lr	0.001	lr	0.005
	num sums	100	num hidden	2	num channels	1024
	max order	3	num units	2048		
	std	0.9				
	λ	0.3				
Steady state diffusion	lr	0.001	lr	0.001	lr	0.001
	num sums	50	num hidden	2	num channels	1024
	max order	3	num units	2048		
	std	0.15				
	λ	0.15				
Sobol G*	lr	0.01	lr	0.001		
	num sums	20	num hidden	3		
	max order	6	num units	512		
	std	0.7				
	λ	0.15				
XD Benchmark	lr	0.005	lr	0.001		
	num sums	20	num hidden	2		
	max order	6	num units	256		
	std	0.8				
	λ	0.2				
Bratley Sum $<$ 500	lr	0.001				
	num sums	20				
	max order	6				
	std	0.5				
	λ	0.15				
Bratley Sum $>=$ 500	lr	0.001				
	num sums	75				
	max order	6				
	std	0.9				
	λ	0.12				

Table 3: Model parameters.

experiment	model	num params	model size [MB]	max memory [MB]	train time [sec/iter]
Darcy flow	DeepPCE	43049558	164.456	4699.239	19.942
	MLP	20996098	80.094	1469.283	0.470
	UNet	31054163	118.462	4710.679	7.913
Steady state diffusion	DeepPCE	11320558	43.419	2509.667	12.361
	MLP	20996098	80.094	1472.880	0.437
	UNet	7707471	29.402	4368.906	6.688

E.4 Main paper experiments: additional results

Bratley Sum Relative MSEs for each of the 10 training runs for the Bratley Sum function with dimensions $D = [100, 250, 500, 750, 1000, 2000]$ for DeepPCE, PCE$_q$, PCE$_{OMP}$ and BA-PCE$_{OMP}$ are shown in Tab. 4.

PDEs Relative MSEs for each of the 10 training runs for the Dracy flow and steady state diffusion experiments are shown in Tab. 5.

Table 4: Relative MSEs for all 10 runs of the Bratley Sum experiment with dimensions $D = [100, 250, 500, 750, 1000, 2000]$. Experiments with traditional PCEs failed in higher dimensions due to memory constraints.

$D = 100$			
DeepPCE	PCE$_q$	PCE$_{OMP}$	BA-PCE$_{OMP}$
0.00037	0.12071	0.02520	0.00819
0.00023	0.12683	0.02415	0.01159
0.00069	0.11904	0.02398	0.00523
0.00035	0.12808	0.02447	0.00993
0.00025	0.12881	0.02521	0.00975
0.00049	0.12326	0.02552	0.00984
0.00059	0.11832	0.02441	0.00562
0.00046	0.12064	0.02458	0.00878
0.00024	0.12938	0.02364	0.00460
0.00031	0.12412	0.02495	0.01170

$D = 250$			
DeepPCE	PCE$_q$	PCE$_{OMP}$	BA-PCE$_{OMP}$
0.00121	0.29169	0.01165	0.01320
0.00076	0.29343	0.01169	0.01357
0.00093	0.29658	0.01139	0.01418
0.00144	0.29268	0.01267	0.01543
0.00077	0.29097	0.01232	0.01457
0.00105	0.29084	0.01305	0.01405
0.00122	0.29580	0.01211	0.01424
0.00169	0.28663	0.01290	0.01412
0.00115	0.29051	0.01195	0.01352
0.00104	0.29655	0.01226	0.01458

$D = 500$			$D = 750$		$D = 1000$	$D = 2000$
DeepPCE	PCE$_{OMP}$	BA-PCE$_{OMP}$	DeepPCE	BA-PCE$_{OMP}$	DeepPCE	DeepPCE
0.00210	0.01052	0.01053	0.34076	0.01121	0.02101	0.04947
0.00159	0.01083	0.01134	0.05260	0.01110	0.00372	0.05870
0.00284	0.01056	0.01050	0.00266	0.01202	0.01621	0.08810
0.00673	0.01014	0.01100	0.03848	0.01056	0.15814	0.15262
0.00197	0.01088	0.01143	0.00279	0.01110	0.00157	0.14441
0.01389	0.01067	0.01146	0.03562	0.01125	0.04136	0.16543
0.00291	0.00986	0.01084	0.10126	0.01136	0.03061	0.12488
0.00391	0.01044	0.01120	0.30875	0.01149	0.00569	0.04012
0.00575	0.01000	0.01107	0.10065	0.01097	0.03548	0.13877
0.00353	0.01097	0.01170	0.00103	0.01038	0.07428	0.11783

Table 5: Relative MSEs of DeepPCE, MLP and UNet on the Darcy flow dataset for all 10 test runs.

Darcy flow			Steady state diffusion		
DeepPCE	MLP	UNet	DeepPCE	MLP	UNet
0.002919	0.001632	0.000062	0.000754	0.000250	0.000074
0.003497	0.001668	0.000064	0.000813	0.000257	0.000076
0.003954	0.001698	0.000066	0.001164	0.000257	0.000078
0.005309	0.001885	0.000068	0.001186	0.000261	0.000079
0.005517	0.002159	0.000069	0.001376	0.000262	0.000080
0.006018	0.002181	0.000069	0.002412	0.000264	0.000080
0.006077	0.002353	0.000070	0.003373	0.000265	0.000081
0.006196	0.003312	0.000070	0.003886	0.000266	0.000083
0.007252	0.003597	0.000070	0.008011	0.000267	0.000087
0.018164	0.011898	0.000073	0.039037	0.000277	0.000090

E.5 Additional experiments

E.5.1 PCE Benchmark

We additionally computed total effect Sobol indices for the PCE benchmark function (Eq. (50)), comparing the DeepPCE with the other PCE variants (PCE_q, PCE_{OMP}, BA-PCE_{OMP}) and Monte Carlo-approximated Sobol indices of an MLP, using 10^8 samples. The results are shown in Fig. 7. While all models slightly overestimate the variance contributions of each variable, the DeepPCE and the sparse PCE variants (PCE_{OMP}, BA-PCE_{OMP}) manage to capture the general pattern of the total effect Sobol indices. The PCE using a simple hyperbolic truncation (PCE_q) shows the worst performance of all PCEs. The accuracy of the MLP Sobol indices is comparable to that of the other models, but the computation time is prohibitive.

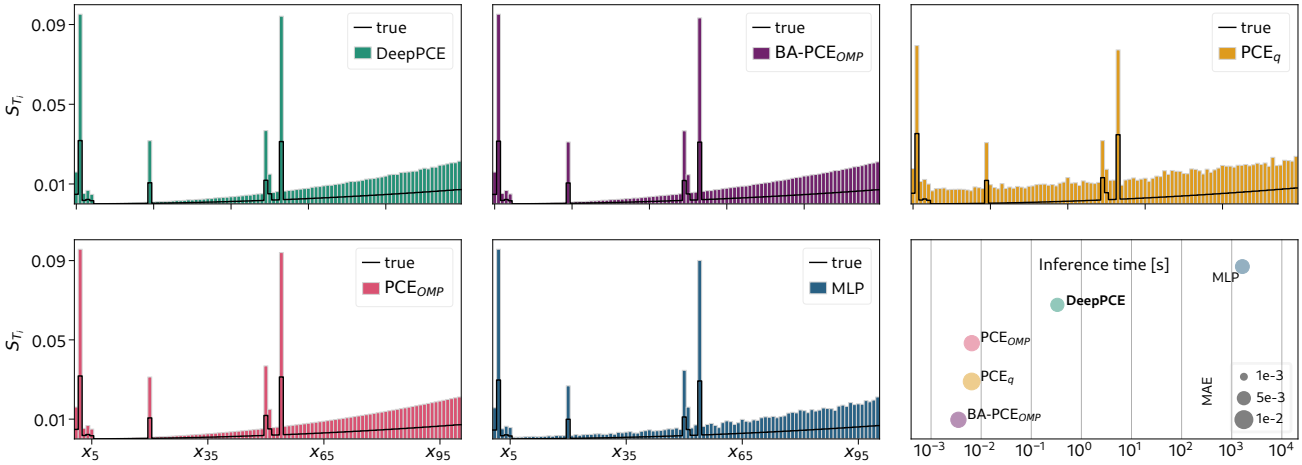

Figure 7: Total effect Sobol indices for the PCE benchmark dataset. The DeepPCE and the sparse PCE variants (PCE_{OMP}, BA-PCE_{OMP}) are equally able to reproduce the true Sobol indices, while PCE_q shows slightly worse performance. Using 10^8 Monte Carlo samples, the Sobol indices obtained from the MLP are comparable to the results of the other models, but exhibiting high computational cost.

F ABLATION STUDIES

We performed ablation studies to assess convergence performance with different scope sizes on the Bratley dataset with $D = 100$ and the steady state diffusion PDE, using scope sizes $S = [1, 2, 3]$ while keeping the other hyperparameters fixed according to Tab. 2. Convergence rates and predictive performance deteriorates with larger scope sizes, especially for the larger steady state diffusion dataset. A likely reason is the presence of

higher rank interaction terms in the expansion, amplifying the problem of vanishing/exploding values in the circuit. Additionally, the initialization is based on the variance scaling scheme shown in Sec. 3.2, using a decay hyperparameter λ tuned specifically for a scope size of 1. Changing the scope sizes at the input layer thus requires retuning of λ.

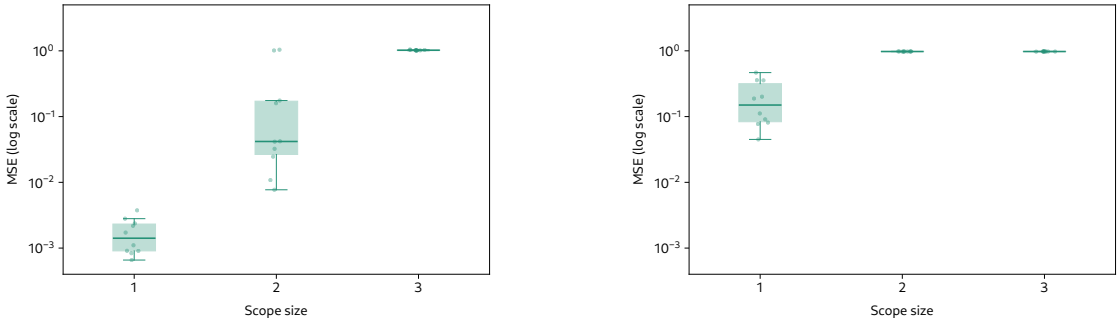

Figure 8: Test performance from 10 runs for a DeepPCE with different scope size parameter for the Bratyel function with $D = 100$ (left) and the steady-state diffusion dataset (right). Results show that the scope size has a strong effect on performance and convergence rates of the DeepPCE.