Highlights

OrPoNet: An Interpretable and Scalable Neuralization of Polynomial Expansions

Siyuan Xing, Daniel M. Tartakovsky

- Universal approximation is proved and scalable structural extensions are introduced.
- OrPoNet achieves interpretability by embedding orthogonal polynomial expansions into neural networks.
- OrPoNet is more accurate than MLPs, with fewer parameters.
- OrPoNet enables robust uncertainty quantification in problems with up to 2000 dimensions.
- Spectral-like error decay is observed for a smooth 2D Brusselator problem.

Accepted manuscript

Published article: https://doi.org/10.1016/j.jcp.2026.115599

© 2026. This manuscript is available under CC BY-NC-ND 4.0

OrPoNet: An Interpretable and Scalable Neuralization of Polynomial Expansions

Siyuan Xing[a,*], Daniel M. Tartakovsky[b,*]

[a]Mechanical Engineering Department, California Polytechnic State University, San Luis Obispo, CA, 93407, USA
[b]Department of Energy Science and Engineering, Stanford University, Stanford, CA, 94305, USA

ARTICLE INFO

Keywords:
interpretable neural networks
polynomial chaos expansion
uncertainty quantification
physics-informed machine learning
spectral methods

ABSTRACT

Polynomial expansions provide compact, interpretable, and rapidly convergent representations for smooth functions commonly arising in scientific computing. However, their use in high-dimensional problems is limited by the combinatorial growth of basis terms. To address this challenge, we introduce OrPoNet, an interpretable neural architecture that represents orthogonal polynomial expansions through a structured multilayer network. Instead of composing affine maps with nonlinear activation functions, OrPoNet sequentially evaluates univariate orthogonal polynomial bases of input coordinates and combines them with previously constructed basis products through trainable weights and Hadamard products. This forward propagation rule preserves an explicit polynomial input-output structure and reduces the parameter growth from that of a full multivariate expansion. We demonstrate that OrPoNet admits a constrained functional tensor-train representation and prove that OrPoNet is a universal approximator on compact domains, i.e., any finite orthogonal-polynomial expansion can be represented by the network. To improve trainability in high dimensions, we introduce basis-aware initialization together with residual connections and additive basis branches. The resulting polynomial structure enables coefficient reconstruction and direct computation of statistical moments and global-sensitivity indices from network weights. In smooth-function experiments, OrPoNet has lower median test errors than a multilayer perceptron (MLP) after architecture selection, while comparisons with Kolmogorov-Arnold networks depend on the function, dimension, and matched parameter count. On the high-dimensional Bratley benchmark, OrPoNet achieves lower mean errors than DeepPCE, while using substantially fewer trainable parameters. When used in physics-informed neural networks, OrPoNet attains comparable or lower losses than MLPs, while using fewer parameters. With suitable sampling and optimization strategies, OrPoNet also exhibits spectral-like error decay in a two-dimensional Brusselator example.

1. Introduction

Function approximation lies at the core of scientific computing, where one seeks compact and accurate representations of maps from inputs (e.g., spatial coordinates, physical parameters, random inputs) to quantities of interest. It underpins various research fields, including surrogate modeling [1], uncertainty quantification [2], and the recent approximation of PDE-governed dynamics through physics-informed neural networks (PINNs) [3, 4] and neural operators [5, 6].

Existing approaches can be either implicit or explicit. Neural networks (NNs) are a popular implicit representation for high-dimensional function approximation due to their universality [7, 8] and scalability [9]. PINNs, for instance, encode physical laws into the loss function during NN training, providing a mesh-free and differentiable approach that reduces reliance on mesh-based discretizations in high-dimensional problems. Building on this, neural operators further extend these capabilities by learning mappings between function spaces, providing efficient surrogates for PDE systems under varying initial/boundary conditions or coefficients. While they have been applied to many problems, conventional multilayer-perceptron-based surrogates typically lack the explicit basis structure, orthogonality, and interpretability that make classical approximation methods attractive. For example, in uncertainty quantification, quantities such as statistical moments, probability distributions, and Sobol sensitivity indices usually require computationally expensive Monte Carlo sampling or post-processing when the model is represented by a black-box NN [10].

Explicit approximation methods approach the same task by expanding the target function in a set of predefined basis functions, including polynomials, Fourier series, wavelets, and radial basis functions. Among these methods,

*Corresponding authors
sixing@calpoly.edu (S. Xing); tartakovsky@stanford.edu (D.M. Tartakovsky)
ORCID(s): 0000-0003-4502-3006 (S. Xing); 0000-0001-9019-8935 (D.M. Tartakovsky)

orthogonal polynomial expansions play a central role in spectral methods [11] for solving partial differential equations, surrogate modeling for low-cost computer simulations [1], and polynomial chaos expansion (PCE) [12] for uncertainty quantification. When a target function is sufficiently regular, as is often the case for many scientific and engineering problems, such expansions can achieve rapid, often exponential, convergence. This means a few terms can approximate the function with sufficient accuracy, producing a more compact and interpretable representation than black-box models. In addition, this explicit representation can be directly used to compute statistical quantities and sensitivity measures.

A main limitation of orthogonal polynomial expansions is their poor scalability in high dimensions. For a total-degree expansion of order p in d dimensions, the number of basis terms grows combinatorially with d and p (see preliminaries). This curse of dimensionality makes direct polynomial expansions infeasible for high-dimensional approximation tasks. To mitigate this growth, various sparse, adaptive, and low-rank strategies have been developed. Truncated PCE [12, 13] selects a subset of basis functions based on total degree or hyperbolic cross criteria to reduce the number of terms. Sparse PCE [13, 14, 15] identifies a parsimonious set of basis functions that contribute most to the approximation, through compressed sensing [16], greedy algorithms such as least angle regression [17] and orthogonal matching pursuit [18], and more recent basis-adaptive strategies [19]. Low-rank tensor methods [20, 21, 22] seek to mitigate the curse of dimensionality by approximating the coefficient tensor using factorized formats, such as tensor-train representations [23, 24, 25, 26]. Recent developments have further extended PCE beyond its traditional regression and projection formulations. Data-driven and sensitivity-enhanced PCE methods incorporate simulation data, gradient information, or adaptive learning strategies to improve sample efficiency in uncertainty quantification [27, 28]. Physics-informed and physics-constrained PCE methods embed governing equations, boundary conditions, or other physical constraints into the construction of the polynomial surrogate, thereby reducing the amount of training data required while preserving the analytical structure of PCE [10, 29]. PCE-inspired neural operator models combine spectral stochastic representations with trainable neural architectures [30]. Despite these advances, existing polynomial-expansion methods often depend on sparsity assumptions, low-rank structure, adaptive basis selection, or prescribed truncation rules.

We address the curse of dimensionality of polynomial expansions by developing an interpretable and scalable multilayer network representation of orthogonal polynomial expansions. Such a representation bridges the interpretability and convergence properties of orthogonal polynomial expansions with the scalability of deep NNs. We refer to this process as the neuralization of orthogonal polynomial expansions: classical basis expansions are reorganized into trainable multilayer architectures while retaining their explicit polynomial structure. The motivation for this work can be understood through an analogy with NN architectures (Fig. 1). The deep compositional structure of multilayer networks can approximate certain classes of complex functions with far fewer parameters than a single-layer network [31], resulting in better computational efficiency and scalability on modern hardware. In contrast, a classical orthogonal polynomial expansion can be viewed as a single-layer network (bottom left) whose hidden units are predefined polynomial basis functions and whose output layer linearly combines these basis functions. The evolution from single-layer to multilayer perceptrons raises the following question: Can orthogonal polynomial expansions be organized into a multilayer architecture to address the curse of dimensionality?

To answer this question, we introduce OrPoNet, a NN whose layers are built from univariate orthogonal polynomial bases. While conventional NNs apply nonlinear activations to affine transformations of previous-layer outputs, OrPoNet evaluates univariate polynomial bases of the input coordinates at each layer and combines them through trainable weights and Hadamard products. Through forward propagation, the network constructs interpretable multivariate polynomial basis products in a layered form. This construction represents classical orthogonal polynomial expansions using a network architecture, allowing us to leverage the interpretability and convergence properties of polynomial expansions while benefiting from modern machine learning techniques. As demonstrated in section 3.2, the vanilla OrPoNet architecture can also be viewed as a constrained functional tensor train [25, 26]. The network structure also connects to recent efforts to develop interpretable neural architectures for scientific computing. For example, Kolmogorov-Arnold Networks (KANs) [32] are based on Kolmogorov-Arnold approximation as a potential alternative to a multilayer perceptron (MLP). KAN places learnable univariate functions, often represented by splines, on edges and sums their outputs at nodes, showing that functions with local-basis representations can be embedded directly into network layers.

OrPoNet approximates functions using global polynomial bases. We prove that any finite orthogonal polynomial expansion can be transformed into an OrPoNet. We propose a specific initialization scheme and incorporate residual connections to scale OrPoNet for high-dimensional problems that are computationally challenging for traditional methods. Across smooth-function benchmarks, OrPoNet achieves lower median errors than MLPs and remains competitive with KANs on lower-dimensional problems. Moreover, OrPoNet achieves lower mean errors than DeepPCE [33] at most tested dimensions, while using substantially fewer trainable parameters in the high-dimensional Bratley benchmark,

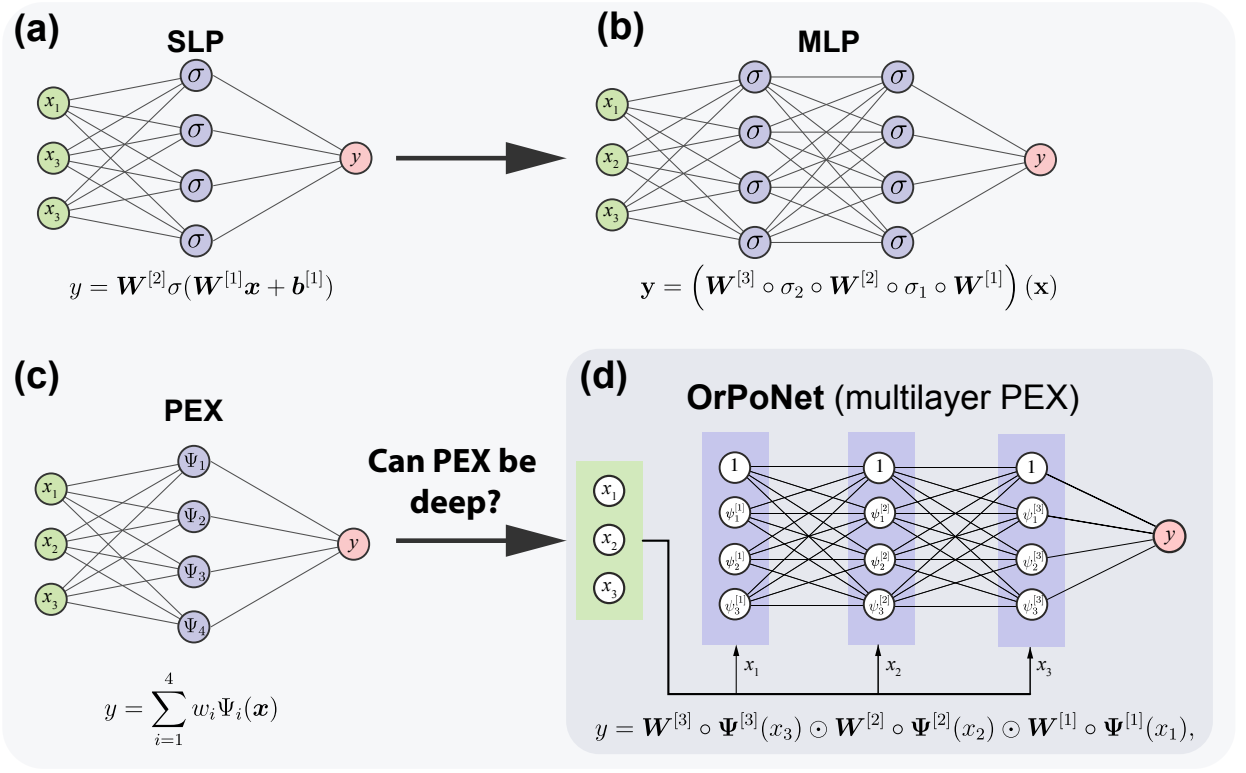


Figure 1: Evolution of neural architectures from standard multilayer perceptron (MLP) to OrPoNet. (a) Single-Layer Perceptron (SLP) and (b) MLP rely on nested nonlinear activations. (c) Polynomial Expansion (PEX) represents a function as a sum of polynomial basis functions; we focus on orthogonal polynomial bases. (d) OrPoNet extends the PEX framework into a high-dimensional, multilayer architecture, where univariate orthogonal-polynomial bases are introduced sequentially and multiplied with previously constructed basis products through trainable layer weights. This structure preserves the interpretability of polynomial expansions while leveraging the hierarchical depth of NNs to capture complex, multi-dimensional interactions.

tested up to 2000 dimensions. We also demonstrate its application to physics-informed learning of PDE solutions, with substantially fewer parameters than MLPs. Finally, with suitable sampling and optimization, OrPoNet exhibits spectral-like error decay in a two-dimensional Brusselator example.

The rest of this paper is organized as follows. Section 2 contains an brief overview of orthogonal polynomial expansions and the curse of dimensionality. In section 3, we present the architecture of OrPoNet, prove its universal approximation property, and discuss its interpretability and connection to polynomial chaos expansion. We also introduce techniques to scale OrPoNet to high-dimensional problems. In section 4, we demonstrate the performance of OrPoNet on various function approximation tasks, uncertainty quantification, and physics-informed learning problems. We conclude, in section 5 with a discussion of the limitations of OrPoNet and future research directions.

2. Preliminaries

We focus on polynomial expansions constructed from orthogonal polynomial basis functions. Unless otherwise specified, polynomial expansion refers to this orthogonal-polynomial setting hereafter. Given a target function $f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$, we approximate it by a finite orthogonal-polynomial expansion,

$$\bar{f}(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha}\in\mathcal{A}} c_{\boldsymbol{\alpha}}\, \Psi_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha}\in\mathcal{A}} c_{\boldsymbol{\alpha}} \prod_{i=1}^{d} \psi^{[i]}_{\alpha_i}(x_i), \tag{1}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d) \in \mathbb{N}^d$ is a multi-index, $\mathcal{A} \subset \mathbb{N}^d$ defines the set of included polynomial terms (e.g., $|\boldsymbol{\alpha}| \leq p$), $\psi_{\alpha_i}^{[i]}$ is the α_i-th degree univariate orthogonal polynomial in the i-th coordinate, and $c_{\boldsymbol{\alpha}}$ are expansion coefficients.

Each univariate polynomial basis $\psi_n^{(i)}(\xi_i)$ is orthogonal under a chosen nonnegative weight function $w_i(\xi_i)$ on domain Ω_i:

$$\int_{\Omega_i} \psi_m^{(i)}(\xi_i)\psi_n^{(i)}(\xi_i)w_i(\xi_i)\,\mathrm{d}\xi_i = \delta_{mn}. \tag{2}$$

With tensor-product bases and product weight $w(\boldsymbol{\xi}) = \prod_{i=1}^{d} w_i(\xi_i)$, the multivariate basis $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi})$ is globally orthogonal,

$$\int_{\Omega} \psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi})\psi_{\boldsymbol{\beta}}(\boldsymbol{\xi})w(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi} = \prod_{i=1}^{d} \delta_{\alpha_i\beta_i} = \delta_{\boldsymbol{\alpha},\boldsymbol{\beta}}. \tag{3}$$

The number of terms N in this expansion is given by the binomial coefficient,

$$N = \frac{(n+p)!}{n!p!}. \tag{4}$$

It grows combinatorially with the input dimension n and polynomial degree p, leading to the curse of dimensionality. This combinatorial growth makes direct finite polynomial expansions infeasible for high-dimensional approximations, motivating the need for scalable representations such as OrPoNet.

3. Methodology

3.1. Network structure

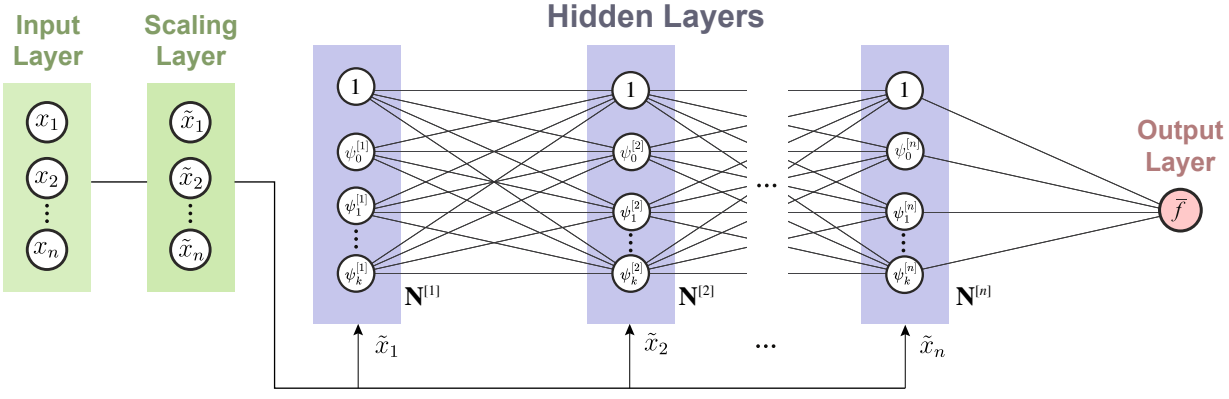


Figure 2: The network structure of OrPoNet, shown for a single weight copy. The input is first scaled to the valid domain through a scaling layer. The transformed coordinates are then sequentially fed into subsequent layers, generating univariate basis functions that replace neurons' activations in traditional NNs. During forward propagation, the vector of these basis functions of each layer is multiplied by a weight matrix, followed by a Hadamard product with the next layer's basis functions (see Eq. 8). The output of the last layer results in a linear combination of all basis functions produced from forward propagation. When multiple weight copies are used, the same propagation is applied in parallel to each copy channel, and the copy channels are combined only at the output layer.

Figure 1 provides a conceptual view of OrPoNet as a multilayer extension of polynomial expansions. A specific network structure used in its current implementation is shown in Fig. 2. Compared with the conceptual schematic, the implemented architecture includes an input scaling layer, which maps each input coordinate to the domain associated with the chosen orthogonal polynomial family (e.g., $[-1, 1]$ for Chebyshev and Legendre polynomials). After scaling,

each coordinate $\tilde{x}_i$ ($i = 1, 2, \dots, n$) is assigned to one layer in sequence. In the i-th layer, $\tilde{x}_i$ is evaluated by a set of activation functions, which are the corresponding univariate orthogonal polynomial basis functions,

$$\boldsymbol{\Psi}^{[i]}(\tilde{x}_i) = \left[1, \psi_0^{[i]}(\tilde{x}_i), \psi_1^{[i]}(\tilde{x}_i), \dots, \psi_k^{[i]}(\tilde{x}_i)\right]^\top, \tag{5}$$

where $\psi_j^{[i]}$ is the j-th degree univariate orthogonal polynomial in the i-th coordinate, and k is the maximum degree of the univariate polynomials in each layer. Here, order k indicates that all polynomial bases from degree 0 to k are included in the layer. Although the explicit constant 1 and ψ_0 do not enlarge the polynomial span, the duplicated constant channels create additional paths through which neurons across layers can interact. Different from traditional NNs such as multilayer perceptrons, the activation functions do not compose the output of the previous layer; instead, they incorporate the scaled coordinates to form univariate basis functions, which will then be used to construct multivariate basis functions through forward propagation.

To understand the forward propagation of the OrPoNet, we first formulate the process from the $(i-1)$-th to i-th layer for a single copy, which yields

$$\boldsymbol{y}^{[i]} = \boldsymbol{\Psi}^{[i]}(\tilde{x}_i) \odot \boldsymbol{W}^{[i-1]} \boldsymbol{y}^{[i-1]}, \tag{6}$$

where the output of the $(i-1)$-th layer is first multiplied by a weight matrix, generating linear combinations of basis products already constructed from previous coordinates. The Hadamard product with $\boldsymbol{\Psi}^{[i]}(\tilde{x}_i)$ combines these terms with univariate polynomial bases of the current coordinate, thereby forming multivariate basis products involving coordinates up to x_i. Theorem 1 shows that multiple copies of these basis neurons may be needed to represent a general polynomial expansion. Rather than directly duplicating these neurons within each layer, we define a copy dimension in the weight matrices, expanding the weights as third-order tensors. Let C denote the number of copies and r_i be the number of neurons in layer i. The hidden states and layer weights are, respectively,

$$\boldsymbol{Y}^{[i]} = [\boldsymbol{y}_1^{[i]}, \dots, \boldsymbol{y}_C^{[i]}] \in \mathbb{R}^{r_i \times C} \quad \text{and} \quad \mathcal{W}^{[i-1]} \in \mathbb{R}^{r_i \times r_{i-1} \times C},$$

wherein cth slice is the weight matrix $\boldsymbol{W}_c^{[i-1]}$ for copy c. Each copy is propagated independently,

$$\boldsymbol{y}_c^{[i]} = \boldsymbol{\Psi}^{[i]}(\tilde{x}_i) \odot \boldsymbol{W}_c^{[i-1]} \boldsymbol{y}_c^{[i-1]}, \qquad c = 1, \dots, C, \tag{7}$$

and the copy channels are combined only at the output layer. This tensorized representation increases expressiveness while making the number of hidden-layer weights grow linearly with C, rather than quadratically as in a fully connected duplicated-neuron implementation. We denote network structures as $[p_1, \dots, p_L] \times C$, where p_i is the maximum polynomial order of layer i and C is the number of copies. Continuing this process from the first layer to the last gives

$$\bar{f}(\boldsymbol{x}) = \boldsymbol{N}(\boldsymbol{x}) = \boldsymbol{W}^{[n]} \boldsymbol{\Psi}^{[n]}(\tilde{x}_n) \odot \boldsymbol{W}^{[n-1]} \dots \boldsymbol{\Psi}^{[2]}(\tilde{x}_2) \odot \boldsymbol{W}^{[1]} \boldsymbol{\Psi}^{[1]}(\tilde{x}_1) \tag{8}$$

for a single copy, where the expression is evaluated from right to left. When $C > 1$, the same propagation is applied independently to each copy channel, and the final output layer linearly combines all copy channels. Eventually, the output is a linear combination of the basis functions generated during forward propagation, with coefficients determined by a compact set of trainable weights. Related efforts have explored neural architectures for structured symbolic or polynomial representations, including P-nets [34], Equation Learner [35], and SREINet [36]. A more closely related line in polynomial-chaos modeling is the deep arbitrary polynomial chaos NN, which embeds data-driven arbitrary polynomial-chaos representations into deep NNs [28]. These methods are related in their use of structured polynomial representations, but OrPoNet differs in its goal of constructing a multilayer orthogonal-polynomial expansion with explicit coefficient reconstruction and weight-space moment and Sobol-index calculations. Prior to discussing the advantage of this network structure, we show that OrPoNet is a universal approximator.

Theorem 1. *OrPoNet is a universal approximator for continuous functions on compact domains.*

Proof. We prove the universal approximation property of OrPoNet in two steps. First, by the Stone-Weierstrass approximation theorem [37, 38], any continuous function $f : \mathbb{K} \to \mathbb{R}$ on a compact domain can be uniformly approximated by polynomials. Since orthogonal polynomials span the same polynomial space, for any $\epsilon > 0$, there exists a finite orthogonal-polynomial expansion

$$\|f(\boldsymbol{x}) - \sum_{\boldsymbol{\alpha} \in \mathcal{A}} c_{\boldsymbol{\alpha}} \psi_{\boldsymbol{\alpha}}(\boldsymbol{x})\|_\infty < \epsilon, \tag{9}$$

where $|\boldsymbol{\alpha}| \leq p$ for some finite p.

Second, we show that the finite expansion

$$P_p(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha}\in\mathcal{A}} c_{\boldsymbol{\alpha}}\psi_{\boldsymbol{\alpha}}(\boldsymbol{x}), \quad \psi_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \prod_{j=1}^{d} \psi_{\alpha_j}^{[j]}(x_j),$$

with $|\boldsymbol{\alpha}| \leq p$ can be represented exactly by an OrPoNet through explicit construction. We start with one copy ($C = 1$) and increase C only when an additional isolated path is needed. The proof proceeds by induction on the graded-lexicographic ordering of multi-indices. We enumerate $\mathcal{A}$ by this order: $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$ when $|\boldsymbol{\alpha}| < |\boldsymbol{\beta}|$; if $|\boldsymbol{\alpha}| = |\boldsymbol{\beta}|$, compare the first index q with $\alpha_q \neq \beta_q$ and set $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$ if $\alpha_q < \beta_q$. This gives

$$\mathcal{A} = \{\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_M\},$$

and each $\boldsymbol{\alpha} \in \mathcal{A}$ has a unique label $i \in \{1, \dots, M\}$. Define partial sums in the form

$$P_m(\boldsymbol{x}) = \sum_{i=1}^{m} c_{\boldsymbol{\alpha}_i}\psi_{\boldsymbol{\alpha}_i}(\boldsymbol{x}), \quad m = 1, \dots, M.$$

We construct an OrPoNet with $L = d$ layers and use induction on m as follows.

Base case. For $m = 1$, there is one basis term $c_{\boldsymbol{\alpha}_1}\psi_{\boldsymbol{\alpha}_1}(\boldsymbol{x})$. With $C = 1$, a forward path selects the required univariate bases $\psi_{\alpha_{1j}}^{[j]}(x_j)$ across layers. Assign weights on this path so their product equals $c_{\boldsymbol{\alpha}_1}$, and set all other weights to zero. Then the network output is exactly P_1.

Inductive Hypothesis. Assume that for some $k < M$, there exists an OrPoNet with a finite copy number C_k that represents P_k exactly.

Inductive Step. Consider $P_{k+1}(\boldsymbol{x}) = P_k(\boldsymbol{x}) + c_{\boldsymbol{\alpha}_{k+1}}\psi_{\boldsymbol{\alpha}_{k+1}}(\boldsymbol{x})$. If the new basis product can be realized by adding a path within the existing copies without changing the coefficients of the first k terms or creating unintended terms, we assign the path weights so that the new path-product equals $c_{\boldsymbol{\alpha}_{k+1}}$. Otherwise, we increase the copy number from C_k to $C_k + 1$ and use the new copy as an isolated branch. In this branch, we select the univariate bases $\psi_{\alpha_{k+1,j}}^{[j]}(x_j)$ across layers and assign the weights so that the branch realizes $c_{\boldsymbol{\alpha}_{k+1}}\psi_{\boldsymbol{\alpha}_{k+1}}(\boldsymbol{x})$. All weights that would couple this new copy to previously constructed paths are set to zero. Therefore, the new term is added without modifying P_k or introducing spurious terms.

Thus, in both cases, there exists an OrPoNet with a finite copy number $C_{k+1} \leq C_k + 1$ that represents P_{k+1} exactly. Combining Steps 1 and 2, for any $\epsilon > 0$, there exists an OrPoNet output $\bar{f}$ with a finite copy number C such that $\|f - \bar{f}\|_\infty < \epsilon$ on $\mathbb{K}$. Therefore, OrPoNet is a universal approximator on compact domains. □

Remark 1. *Although OrPoNet can universally approximate any continuous function on a compact domain, the network may require repeated basis functions, or multiple copies of the same univariate polynomial basis within each layer. As described above, these copies are implemented as independent channels rather than fully connected duplicated neurons. An immediate consequence of this representation is that the number of trainable parameters (or coefficients) can be significantly reduced when C is small. A full polynomial expansion has $\mathcal{O}((n+p)!/p!/n!)$ terms for total degree p in n dimensions. In contrast, the number of parameters in OrPoNet is*

$$N_W = C(n-1)(2+p)^2 + C(p+2) = \mathcal{O}(Cnp^2), \tag{10}$$

i.e., it scales linearly in the dimension and number of copies and quadratically in the polynomial degree.[1] *Although this structure can considerably reduce the parameter count when C is small, it still faces challenges when scaled up to high dimensions due to vanishing of gradients. We address this scaling issue in section 3.4.*

Remark 2 (Interpretability)**.** *A distinct feature of OrPoNet is its inherent interpretability, as it allows for reconstruction of the polynomial expansion* (1)*, producing an analytical formula for input-output mapping in three steps: i) enumerate all paths of forward propagation, ii) multiply the weights and basis functions along each path, and iii) aggregate path contributions that correspond to the same multivariate polynomial basis.*[2]

[1] A detailed analysis of parameter counts, including the residual and additive extensions, is provided in Appendix A.

[2] An example in Appendix D.7 illustrates the ability of OrPoNet to reconstruct the polynomial expansion of a given function.

Although Theorem 1 establishes representability, it does not guarantee that the required copy number remains small as the dimension or polynomial degree increases. The practical parameter savings therefore depend on whether the target polynomial can be accurately represented with a modest number of copies. In addition, OrPoNet provides a structured parameterization of multivariate polynomials through a factorization of their coefficient tensor. We show in section 3.2 that the vanilla architecture admits a constrained functional tensor-train (FT) representation.

3.2. OrPoNet as a constrained functional tensor train

To connect the OrPoNet to the functional tensor-train (FT) framework [24, 25, 26], consider a function $f(x_1, \dots, x_n)$ represented by an FT decomposition,

$$f(x_1, \dots, x_n) = \boldsymbol{G}^{[1]}(x_1)\boldsymbol{G}^{[2]}(x_2)\cdots\boldsymbol{G}^{[n]}(x_n), \tag{11}$$

where each functional core is a matrix-valued univariate function $\boldsymbol{G}^{[i]} : \mathcal{X}_i \to \mathbb{R}^{\gamma_{i-1}\times\gamma_i}$, with boundary dimensions $\gamma_0 = \gamma_n = 1$. We use γ_i for the FT ranks to distinguish them from the network layer widths r_i. The entries of the tensor cores can be represented, e.g., by linear combinations of univariate basis functions [26],

$$\boldsymbol{G}^{[i]}(x_i) = \sum_{k=0}^{p_i} \boldsymbol{A}_k^{[i]}\psi_k^{[i]}(x_i), \tag{12}$$

where p_i is the maximum polynomial degree in coordinate i and $\boldsymbol{A}_k^{[i]}$ are independently parameterized coefficient matrices. In the following, we prove that the vanilla OrPoNet architecture admits a constrained FT representation.

Proposition 1 (FT representation of OrPoNet)**.** *A vanilla OrPoNet with one copy ($C = 1$) and $n \geq 2$ coordinates can be written in the FT form of Eq.* (11)*.*

Proof. Let $\boldsymbol{\Psi}^{[i]} = [1, \psi_0^{[i]}, \dots, \psi_{p_i}^{[i]}]^\top$, with $\psi_0^{[i]}(x_i) \equiv 1$ and $r_i = p_i + 2$, be the basis vector. We define $\boldsymbol{D}^{[i]} = \mathrm{diag}(\boldsymbol{\Psi}^{[i]})$. For notational convenience, we omit the tilde in the following and write x_i for the scaled coordinate. The Hadamard product in Eq. (6) is rewritten as

$$\begin{aligned}\boldsymbol{y}^{[i]} &= \boldsymbol{D}^{[i]}(x_i)\boldsymbol{W}^{[i-1]}\boldsymbol{y}^{[i-1]} = \sum_{k=1}^{p_i+1}\left(\delta_k^1\boldsymbol{e}_0\boldsymbol{e}_0^\top + \boldsymbol{e}_k\boldsymbol{e}_k^\top\right)\boldsymbol{W}^{[i-1]}\psi_{k-1}^{[i]}(x_i)\boldsymbol{y}^{[i-1]} \\ &= \boldsymbol{G}_{\mathrm{net}}^{[i]}(x_i)\boldsymbol{y}^{[i-1]},\end{aligned} \tag{13}$$

where δ_k^1 is the Kronecker delta, and $\boldsymbol{G}_{\mathrm{net}}^{[i]}(x_i) = \boldsymbol{D}^{[i]}(x_i)\boldsymbol{W}^{[i-1]}$ for $i = 2, \dots, n-1$. Here, $\boldsymbol{e}_0, \dots, \boldsymbol{e}_{p_i+1} \in \mathbb{R}^{r_i}$ form the standard coordinate basis in the same order as the entries of $\boldsymbol{\Psi}^{[i]}$. Their outer products select the corresponding rows of $\boldsymbol{W}^{[i-1]}$. To include the first layer and the output layer, we define the boundary cores

$$\boldsymbol{G}_{\mathrm{net}}^{[1]}(x_1) = \boldsymbol{\Psi}^{[1]}(x_1), \qquad \boldsymbol{G}_{\mathrm{net}}^{[n]}(x_n) = \boldsymbol{W}^{[n]}\boldsymbol{D}^{[n]}(x_n)\boldsymbol{W}^{[n-1]}, \tag{14}$$

so that the single-copy network output in Eq. (8) reads

$$\bar{f}(\boldsymbol{x}) = \boldsymbol{G}_{\mathrm{net}}^{[n]}(x_n)\boldsymbol{G}_{\mathrm{net}}^{[n-1]}(x_{n-1})\cdots\boldsymbol{G}_{\mathrm{net}}^{[1]}(x_1). \tag{15}$$

Since the output is a scalar, transposing this product gives the FT form defined in Eq. (11). □

Remark 3. *Although OrPoNet is expressible in the FT form, it is a constrained FT representation. In a general polynomial FT core, each matrix entry can contain an independently chosen linear combination of the univariate basis functions. In contrast, each row of an internal OrPoNet matrix $\boldsymbol{D}^{[i]}\boldsymbol{W}^{[i-1]}$ is proportional to one prescribed component of $\boldsymbol{\Psi}^{[i]}$. Its weights change the connection amplitudes, but cannot independently change the polynomial dependence of each entry. In a general polynomial FT representation, the FT ranks can be specified independently of the polynomial degree. In OrPoNet, the FT ranks are not specified directly. Instead, they are constrained by the layer widths and the number of copies. These constraints reduce the number of free parameters and couple the polynomial coefficients through shared weights.*

The constraints in Remark 3 raise a question about the ability of OrPoNet to approximate general functions despite the restricted form of its cores. As established in Theorem 1, any finite polynomial expansion can be represented by OrPoNet with a sufficient polynomial degree and number of copies. Multiple copies provide additional flexibility by summing independently parameterized chains, while preserving the constrained structure within each copy.

The sum of these chains can itself be written as an FT with FT ranks Cr_i (see Appendix B for detail), so the minimal FT ranks satisfy

$$\gamma_i^{\min} \leq Cr_i, \quad \text{for} \quad i = 1, \dots, n-1.$$

This upper bound neither determines the actual FT ranks nor guarantees that a small number of copies is sufficient. The practical benefit of the parameter scaling $\mathcal{O}(Cnp^2)$ therefore depends on attaining the desired accuracy with moderate polynomial degree and a small number of copies. In addition, our numerical experiments reported in section 4.1 suggest that, similar to FT, OrPoNet is sensitive to input coordinate ordering at fixed model capacity, even with residual connections. This sensitivity can be mitigated by additional constant basis channels. A systematic numerical comparison with general FT methods is beyond the scope of this work.

3.3. OrPoNet as a deep variant of polynomial chaos expansion

The architectural design of OrPoNet can also be interpreted as a deep analog of PCE. To see this connection, consider the classical PCE framework where a target function $f(\boldsymbol{\xi})$ is represented as a series of orthogonal polynomials [12]. Assuming the input random variables ξ_i are independent with known probability distributions $\rho_i(\xi_i)$, the multivariate output $f(\boldsymbol{\xi})$ can be approximated as a finite series,

$$f(\boldsymbol{\xi}) \approx \sum_{\boldsymbol{\alpha} \in \mathcal{A}} c_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} c_{\boldsymbol{\alpha}} \prod_{i=1}^{d} \psi_{\alpha_i}^{[i]}(\xi_i) \tag{16}$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_d) \in \mathbb{N}^d$ is a multi-index, and $\mathcal{A} \subset \mathbb{N}^d$ defines the set of polynomial terms with a total degree $|\boldsymbol{\alpha}| \leq p$. The univariate polynomial bases $\psi_n^{(i)}(\xi_i)$ satisfy an orthonormality condition with respect to density $\rho_i(\xi_i)$,

$$\int \psi_m^{(i)}(\xi_i)\psi_n^{(i)}(\xi_i)\rho_i(\xi_i)\,\mathrm{d}\xi_i = \delta_{mn}. \tag{17}$$

These orthonormal polynomial bases can be chosen based on the distribution of the input random variables according to the Wiener-Askey scheme [12]. In this context, OrPoNet's weights serve as trainable parameters to determine the coefficients $c_{\boldsymbol{\alpha}}$, and its neurons form basis products or the multivariate basis functions. This provides a deep parameterization of PCE that can reduce the number of trainable parameters when the desired accuracy is attained with moderate polynomial degree and a small number of copies.

The PCE representation by OrPoNet is particularly useful for uncertainty quantification because the variance-based sensitivity measures can be evaluated analytically. Let $Y = f(\boldsymbol{\xi})$ and let $\boldsymbol{\xi}_{\sim i}$ denote all input variables except ξ_i. The first-order and total Sobol indexes are defined, respectively, as

$$S_i = \frac{\mathrm{Var}_{\xi_i}(\mathbb{E}_{\boldsymbol{\xi}_{\sim i}}[Y \mid \xi_i])}{\mathrm{Var}(Y)} \quad \text{and} \quad S_i^{\mathrm{T}} = 1 - \frac{\mathrm{Var}_{\boldsymbol{\xi}_{\sim i}}(\mathbb{E}_{\xi_i}[Y \mid \boldsymbol{\xi}_{\sim i}])}{\mathrm{Var}(Y)}. \tag{18}$$

For a PCE, these conditional variances reduce to sums of squared expansion coefficients. OrPoNet obtains the same quantities either by explicitly reconstructing the implied PCE coefficients or, in high dimensions, by propagating first and second moments directly through the trained network weights (Appendix C).

3.4. Scaling to high-dimensional problems

The vanilla OrPoNet has two limitations when scaled to high-dimensional problems. The first is vanishing gradients. Because the network depth equals the input dimension, a 20-dimensional function would require a 20-layer OrPoNet. In practice, this serial structure is likely to cause vanishing gradients during backpropagation, making network training challenging. The second limitation is that the compact structure of OrPoNet comes at the cost of strong coupling between neurons. For complex functions, one may require more copies to compensate for this coupling, which yields wide layers and reduces the structural advantage of OrPoNet. We solve the scaling issue by taking inspiration from two well-known techniques: weight initialization and residual connection.

3.4.1. Basis-aware weight initialization

For shallow networks, it makes sense to generate initial weights using a random distribution with a small variance. However, random initialization will lead to the vanishing of gradients for deep networks [39, 40], including OrPoNet. Moreover, as the forward propagation of OrPoNet includes additional calculation of basis functions, it should also be counted in weight initialization. A basis-aware initialization method for OrPoNet ameliorates these challenges.

Proposition 2 (Basis-aware weight initialization)**.** *Assume that adjacent layers of an OrPoNet have equal widths, $r_{l+1} = r_l$. Let $y_j^{[l]}$ denote the j-th output of the l-th layer, assumed to be independent and identically distributed (i.i.d.) at initialization. To maintain the same distribution at the $(l+1)$-th layer, the weights should be initialized as*

$$w_{ij}^{[l]} \sim \mathcal{N}(0, \sigma_i^2) \quad with \quad \sigma_i^{[l]} = \frac{1}{\sqrt{r_l \mathbb{E}[(\psi_i^{[l+1]})^2]}}, \tag{19}$$

where $\psi_i^{[l+1]}(\cdot)$ represents the basis function of the i-th unit of the $(l+1)$-th layer, and r_l is the number of neurons in the l-th layer.

Proof. By Eq. (6), the i-th output of the $(l+1)$-th layer reads

$$y_i^{[l+1]} = \psi_i^{[l+1]}(\cdot)\left(\sum_{j=1}^{r_l} w_{ij}^{[l]} y_j^{[l]}\right) = \sum_{j=1}^{r_l} \psi_i^{[l+1]}(\cdot) w_{ij}^{[l]} y_j^{[l]}. \tag{20}$$

Following [39, 40], we assume that the weights are independent of the layer inputs and of the input-dependent basis values, with $\mathbb{E}[w_{ij}^{[l]}] = 0$. Therefore, the mean of output $y_i^{[l+1]}$ becomes

$$\mathbb{E}[y_i^{[l+1]}] = \sum_{j=1}^{r_l} \mathbb{E}[\psi_i^{[l+1]}]\mathbb{E}[w_{ij}^{[l]}]\mathbb{E}[y_j^{[l]}] = 0. \tag{21}$$

Since $y_i^{[l+1]}$ is zero mean at initialization, its variance equals its second moment. Accordingly, the variance of $y_i^{[l+1]}$ is

$$\begin{aligned} \mathrm{Var}[y_i^{[l+1]}] &= \mathbb{E}[(y_i^{[l+1]})^2] = \sum_{j=1}^{r_l} \mathbb{E}[(\psi_i^{[l+1]})^2]\,\mathbb{E}[(w_{ij}^{[l]})^2]\,\mathbb{E}[(y_j^{[l]})^2] \\ &= (\sigma_i^{[l]})^2 r_l \mathbb{E}[(y_j^{[l]})^2]\,\mathbb{E}[(\psi_i^{[l+1]})^2]. \end{aligned} \tag{22}$$

The mean of $\psi_i^{[l+1]}(\cdot)$ may not be zero, indicating $\mathbb{E}[\psi_i^{[l+1]}(\cdot)^2] \neq \mathrm{Var}[\psi_i^{[l+1]}(\cdot)]$. To avoid exponential amplification or attenuation of input signals, the condition

$$\mathrm{Var}[y_i^{[l+1]}] = \mathrm{Var}[y_i^{[l]}] \tag{23}$$

should be satisfied. Substituting Eq. (22) into Eq. (23), we obtain

$$\sigma_i^{[l]} = \frac{1}{\sqrt{r_l \mathbb{E}[(\psi_i^{[l+1]})^2]}}. \tag{24}$$

To evaluate the gradient of backward propagation, we consider a loss value L whose gradient with respect to $y_i^{[l]}$ is

$$\frac{\partial L}{\partial y_i^{[l]}} = \sum_{k=1}^{r_{l+1}} \frac{\partial L}{\partial y_k^{[l+1]}} \psi_k^{[l+1]}(\cdot) w_{ki}^{[l]}. \tag{25}$$

Following [39, 40], we treat $\partial L/\partial y_k^{[l+1]}$ as i.i.d. with a common variance denoted by v_{l+1}. Since $w_{ki}^{[l]}$ has zero mean, $\mathbb{E}[\partial L/\partial y_i^{[l]}] = 0$. Using the same second-moment approximation,

$$\mathrm{Var}\left[\frac{\partial L}{\partial y_i^{[l]}}\right] \approx \sum_{k=1}^{r_{l+1}} \mathrm{Var}\left[\frac{\partial L}{\partial y_k^{[l+1]}}\right] \mathbb{E}[(\psi_k^{[l+1]})^2](\sigma_k^{[l]})^2. \tag{26}$$

Substitution of Eq. (24) gives $\mathbb{E}[(\psi_k^{[l+1]})^2](\sigma_k^{[l]})^2 = 1/r_l$ for every k and, hence,

$$\mathrm{Var}\left[\frac{\partial L}{\partial y_i^{[l]}}\right] \approx v_{l+1}\sum_{k=1}^{r_{l+1}}\frac{1}{r_l} = v_{l+1}. \tag{27}$$

Thus, the backward gradient variance is approximately preserved under the stated equal-width assumption.

□

Remark 4. *Theoretically, the expected power of the basis function $\psi_i^{[l+1]}$, $\mathbb{E}[(\psi_i^{[l+1]}(\cdot))^2]$, can be estimated from the training dataset. However, in mini-batch gradient descent, this empirical estimate may fluctuate, leading to unintended amplification or attenuation of variance across layers. To mitigate this instability, we multiply the weight initialization in Eq. (19) by a scaling factor.*[3]

3.4.2. Residual connection with optional gating and additive basis injection

In addition to weight initialization, another effective technique for training deep networks is the use of residual connections [41]. Inspired by this technique, we augment OrPoNet with residual connections and additive basis branches to improve both scalability and the expressiveness of OrPoNet. Unless stated otherwise, residual OrPoNet refers to OrPoNet with both residual connections and additive basis branches in the following.

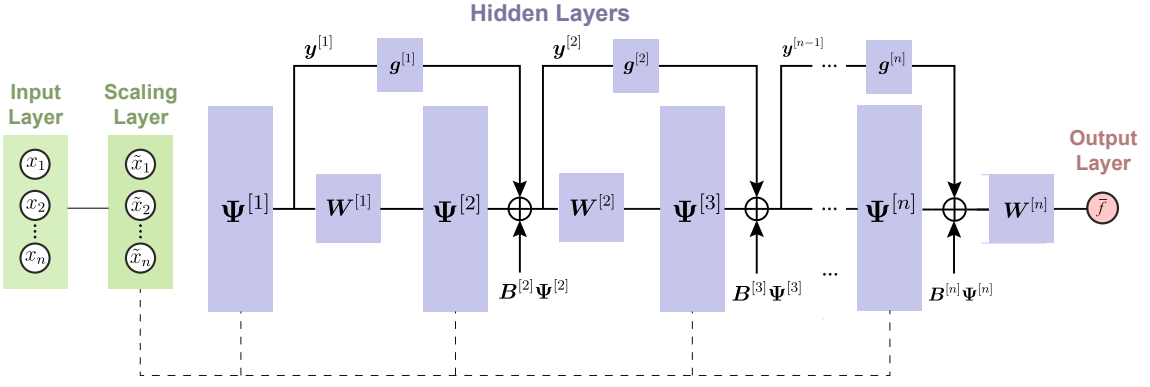


Figure 3: OrPoNet with residual connections (with optional gates), and additive basis branches. The residual connection helps mitigate vanishing gradients, while the gate provides a learnable modulation of the skip channels. The additive basis branch injects terms related to the current coordinate that are independent of basis products produced by previous layers, improving the expressiveness of the network.

The augmented OrPoNet (Fig. 3) retains the polynomial layers and weight matrices, but replaces layer updates with

$$\boldsymbol{y}^{[i]} = F_{i-1}(\boldsymbol{y}^{[i-1]}, \boldsymbol{\Psi}^{[i]}) + \boldsymbol{g}^{[i]} \circ S_{i-1}(\boldsymbol{y}^{[i-1]}) + B_{i-1}(\boldsymbol{\Psi}^{[i]}). \tag{28}$$

Here, $F_{i-1}(\boldsymbol{y}^{[i-1]}, \boldsymbol{\Psi}^{[i]})$ is the original multiplicative polynomial update in Eq. (6) and $S_{i-1}(\boldsymbol{y}^{[i-1]})$ is the residual branch. If two adjacent layers have the same width, S_{i-1} is set to an identity operator; otherwise, it is a learnable linear projection. A soft gate $\boldsymbol{g}^{[i]}$ is applied elementwise to the residual branch and is parameterized by learnable logits $\boldsymbol{A}^{[i]}$:

$$\boldsymbol{g}^{[i]} = \sigma(\boldsymbol{A}^{[i]}/\tau), \tag{29}$$

where $\sigma(\cdot)$ is the sigmoid function and τ is a temperature parameter controlling the sharpness of the gate. A smaller τ makes the gate more binary, while a larger τ produces smoother gate values. The gate $\boldsymbol{g}^{[i]}$ directly scales the residual branch, allowing the network to continuously adjust the amount of information carried through each skip channel [42, 43]. In addition to the residual branch, we inject one additive input $B_{i-1}(\boldsymbol{\Psi}^{[i]})$, which is a learnable linear combination of the

[3] Results of our ablation study (Table D.7) demonstrate that the basis-aware weights substantially improve the scale of initial outputs and gradients vis-à-vis random initialization.

local basis vector $\mathbf{\Psi}^{[i]}$. This branch creates paths for terms related to the current coordinate that are independent of basis products composed in previous layers. Together, the weight initialization scheme, residual connection, and additive basis branch enrich the expressiveness of OrPoNet while avoiding the vanishing gradient problem.[4] The residual gate is treated as an optional modulation mechanism; in the nonsparse benchmark used for the ablation study, it provides only marginal additional improvement over the standard residual connection.

4. Results

We use a series of numerical experiments to demonstrate salient features of OrPoNet and compare its performance to that of other deep-learning models.

4.1. Function approximation

To gain an initial understanding of the approximation ability of OrPoNet, we begin with three two-dimensional analytic functions from Fig. 4. For each function, we compare the loss curve and parameter count of OrPoNet with those of MLP and KAN. To ensure a fair comparison, we generate and use the same dataset of 1024 points sampled from a Sobol sequence within the rectangle $[-1, 1]^2$, accompanied with a dense evaluation set with 2048 points. The training process comprises an initial 400-epoch phase using the Adam optimizer with a learning rate of 0.001 and a batch size of 128, followed by fine-tuning with 40-step L-BFGS with a default learning rate. We employ a two-layer vanilla OrPoNet with Chebyshev polynomials and degrees ranging from 2 to 12 in increments of 2. The MLP configurations use the tanh activation function with depths of $\{2, 3, 4\}$ and widths of $\{8, 16, 20\}$. Regarding the KAN models, we maintain a depth of 1 (one hidden layer) and a spline order of 3, while varying the widths among $\{2, 5, 8\}$ and grid sizes across $\{3, 5, 7, 10\}$. For each configuration, we perform three runs with distinct random seeds. In these examples, the KAN grid resolution is held constant within each run. The subsequent benchmark comparison incorporates progressive grid refinement for KAN.

Figure 4a shows the loss curves of the best-performing configurations of OrPoNet, MLP, and KAN, selected for each function using the minimum mean squared error (MMSE) within the search space. The selected KAN has one hidden layer with width 8 and grid size 10 for all three functions. While KAN exhibits faster initial convergence when using the Adam optimizer, OrPoNet achieves better accuracy upon refinement with L-BFGS, especially for the polynomial function (which is expected as Chebyshev polynomials form a basis for the polynomial space). Figure 4b reveals that OrPoNet outperforms both MLP and KAN in terms of parameter efficiency: it achieves higher or at least comparable accuracy than either MLP or KAN, while utilizing fewer parameters. This advantage may stem from the exponential convergence rate of a polynomial expansion when approximating analytic functions.

Next, we evaluate the performance of the residual OrPoNet on eight benchmark smooth (C^∞) functions listed in Table 1. These functions encompass a set of diverse geometric features such as a localized peak, valley, plate, S-shaped, and multi-frequency components (functions f_5 through f_8) that are challenging for MLPs [44]. These functions are drawn from the standard optimization benchmark set [45], with minor modifications to the coefficients of the Rosenbrock and Zakharov functions to address the ill-conditioned growth and extreme scale disparity inherent in their high-dimensional formulations. The data domains of each function are selected to balance numerical stability with feature representation. More specifically, a tighter interval $[-1, 1]^d$ is used for polynomial and oscillatory functions (f_2, f_3, f_7, f_8) to prevent high-dimensional numerical explosion, while a wider interval $[-2, 2]^d$ is adopted for exponential and plateau-like functions (f_1, f_4, f_5, f_6) to ensure their characteristics are well captured.

We focus on the approximation accuracy and parameter count of OrPoNet in high-dimensional scenarios, comparing them to those of MLP and KAN in 10, 20, 50, and 100 dimensions. As the function values vary across multiple scales, we evaluate the approximation error by the relative l_2 error,

$$e_{\text{rel}} = \frac{\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2}{\|\boldsymbol{y}\|_2}. \tag{30}$$

For each function and dimension, we adopt seven candidate configurations per model under the same per-candidate training budget and select the final configuration using validation errors.[5]

[4] The ablation study of these added components (Table D.8) demonstrates that this architectural design significantly improves the performance of OrPoNet on high-dimensional problems without using multiple copies (Appendix D.3).

[5] Details of the search scheme, data generation, and training settings are provided in Appendix D.5.

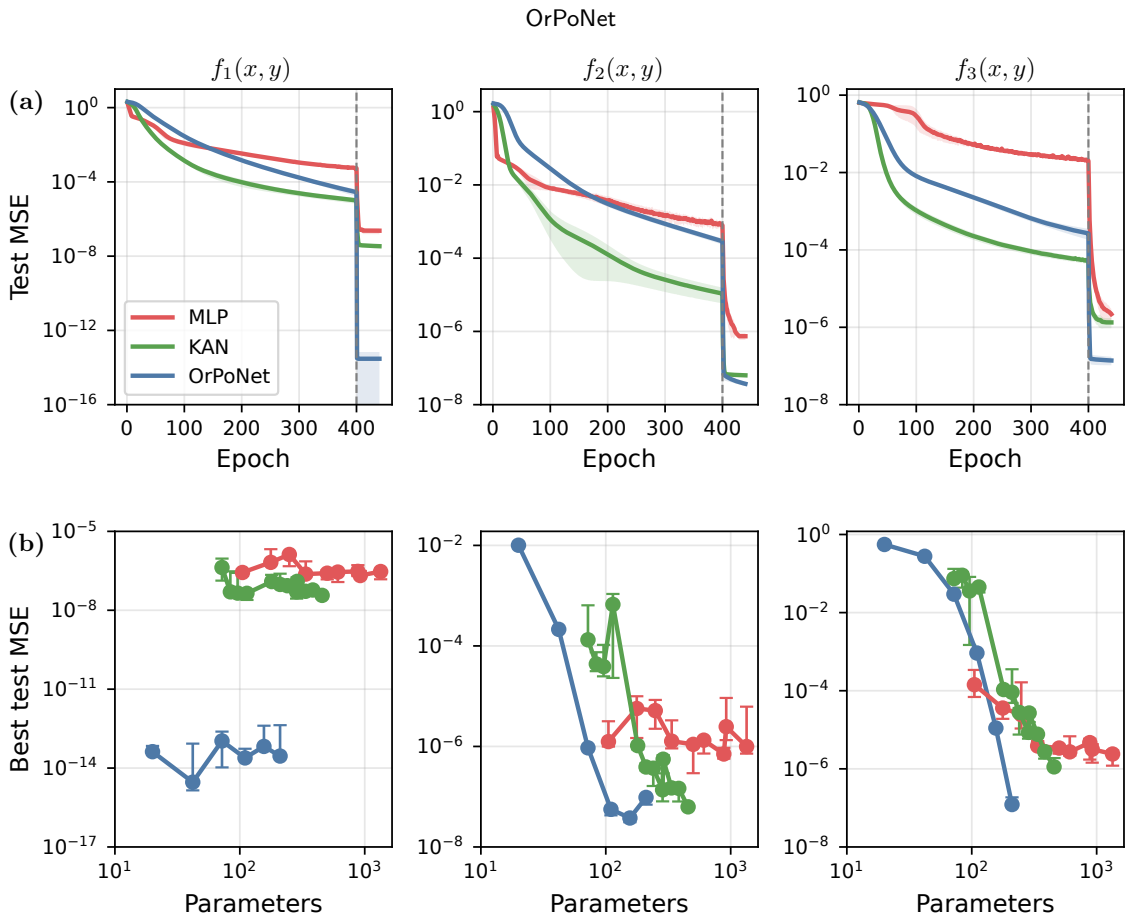


Figure 4: Comparison of OrPoNet, MLP, and KAN on the two-dimensional functions $f_1(x,y) = 1+x+y+xy+x^2-y^2+0.5x^2y$, $f_2(x,y) = \exp(0.8x-0.4y)+0.3\sin(\pi x)\cos(\pi y)+0.2xy$, and $f_3(x,y) = \sin(2\pi x+\pi y)+0.5\cos(3\pi x/2-\pi y)+0.15\exp(xy)$. The results are generated using three runs with different random seeds. OrPoNet demonstrates superior accuracy and parameter efficiency on polynomial functions. For non-polynomial smooth functions, f_2 and f_3, OrPoNet exhibits a much steeper scaling curve, achieving Best Test MSE using about 2 times fewer parameters than KAN and 5 times fewer than MLP. Panel (a) shows the mean evaluation MSE with one population standard deviation for the configuration selected in each model. Panel (b) shows the median of the minimum evaluation MSE, with error bars spanning the minimum-to-maximum range across seeds.

Figure 5 summarizes test errors after the architecture selection, for each function and dimension. Each point is the average over three training seeds for the selected configuration, and the boxplots describe variation across the eight functions. The median error of OrPoNet is lower than that of MLP, but higher than KAN's, at all four dimensions. OrPoNet achieves lower mean test errors than KAN on Griewank at 10D, 20D, and 50D and on modified Rosenbrock at 20D, 50D, and 100D, while KAN achieves lower mean errors on modified Zakharov and Log-Sum-Exp throughout.

To compare the accuracy at similar model sizes, we conduct a parameter-matched experiment at $d = 10$ and $d = 100$. MLP and KAN configurations are selected using validation errors, with their parameter counts constrained to within $\pm 5\%$ of five targets defined by OrPoNet. Details of the parameter-count targets, architecture search, and training settings are provided in Appendix D.5. Figure 6 shows that OrPoNet has lower mean test errors than MLP in 25 of the 40 function/target combinations at 10D and 33 of the 40 combinations at 100D. Its comparison with KAN depends on the function, dimension, and parameter count. At 10D, OrPoNet has lower mean errors than KAN in 17 of the 40 combinations. At 100D, KAN has lower mean errors in 37 of the 40 combinations. In general, OrPoNet achieves lower mean test errors than MLP in most parameter-matched configurations at both dimensions. KAN achieves lower mean errors in most of the higher-dimensional configurations tested, while OrPoNet is more competitive with KAN in the lower-dimensional cases. Increasing the polynomial order can substantially reduce approximation errors, but the improvement is not monotonic and higher orders may introduce optimization difficulties. Low-order OrPoNet

Table 1
Benchmark functions used for function approximation.

ID	Name	Feature	Expression
f_1	Gaussian	Localized peak	$10\exp(-2\sum_{j=1}^{d} x_j^2/d)$
f_2	Modified Rosenbrock	Valley	$\sum_{j=1}^{d-1}[2(x_{j+1}-x_j^2)^2+(1-x_j)^2]$
f_3	Modified Zakharov	Plate	$\sum_{j=1}^{d} x_j^2/d+(\sum_{j=1}^{d} jx_j/d^2)^2+(\sum_{j=1}^{d} jx_j/d^2)^4$
f_4	Log-Sum-Exp	Curved plate	$\ln[\sum_{j=1}^{d}\exp(x_j)]$
f_5	Sigmoid	S-shape	$1/[1+\exp(-\sum_{j=1}^{d} x_j/d)]$
f_6	Griewank	Rugged parabola	$\sum_{j=1}^{d} x_j^2/4000-\prod_{j=1}^{d}\cos(x_j/\sqrt{j})+1$
f_7	Rastrigin	Rugged parabola	$10d+\sum_{j=1}^{d}[x_j^2-10\cos(2\pi x_j)]$
f_8	Levy	Multi-frequency	$\sin^2(\pi w_1)+\sum_{j=1}^{d-1}(w_j-1)^2[1+10\sin^2(\pi w_j+1)]+(w_d-1)^2[1+\sin^2(2\pi w_d)]$ with $w_j=1+(x_j-1)/4$

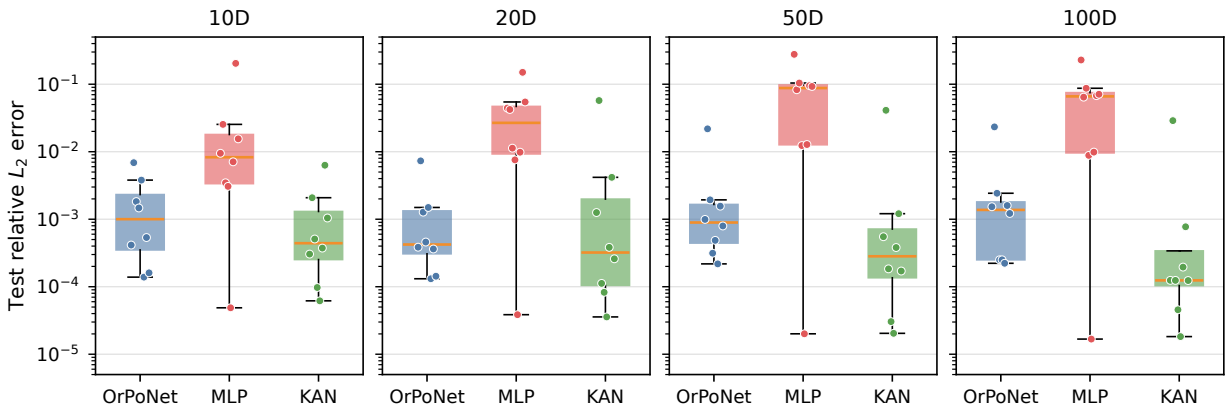


Figure 5: Statistical distribution of the test relative-l_2 error e_{rel} for residual OrPoNet, MLP, and KAN across $d = 10, 20, 50,$ and 100 dimensions. The boxplots summarize the distribution across the eight functions, with each scattered point representing the mean test error over three training seeds. The hyperparameter-selection scheme is given in Appendix D.5. The residual OrPoNet achieves lower median errors than MLP across all tested dimensions; KAN has lower median errors than OrPoNet across these dimensions.

models can be effective at small parameter budgets, as illustrated by the modified Rosenbrock results. The stronger competitiveness of OrPoNet in lower-dimensional settings is consistent with the trends in Figs. 4 and 5.

Additional studies on sampling strategies and polynomial-order effects are provided in Appendices D.1 and D.2. The performance of OrPoNet on nonsmooth functions is demonstrated in Appendix D.4. We restrict this general function-approximation benchmark to 100 dimensions and evaluate higher dimensional scalability in the UQ setting, where the orthogonal-polynomial structure directly enables moment and Sobol-index computations.

Appendix D.9, similar to FT, OrPoNet is sensitive to input coordinate ordering at fixed model capacity, even with residual connections.

Besides the efficiency in approximating smooth functions, another distinct feature of OrPoNet is its interpretability. Through forward propagation, OrPoNet can produce an analytical expression of input-output mapping in the form of truncated series expansion. We demonstrate this on function $f_2(x_1, x_2, x_3) = \exp(0.5x_1x_2) + 0.7\sin(1.2x_3)$.

4.2. Application to uncertainty quantification

We now apply OrPoNet to uncertainty quantification (UQ), a particularly important regime for robust design optimization and reliability analysis. To motivate this application, we first recall its connection to polynomial chaos expansion (PCE), which represents stochastic model outputs using orthogonal polynomial bases and is widely used for uncertainty propagation. We use this connection to evaluate whether OrPoNet can retain the statistical structure of PCE

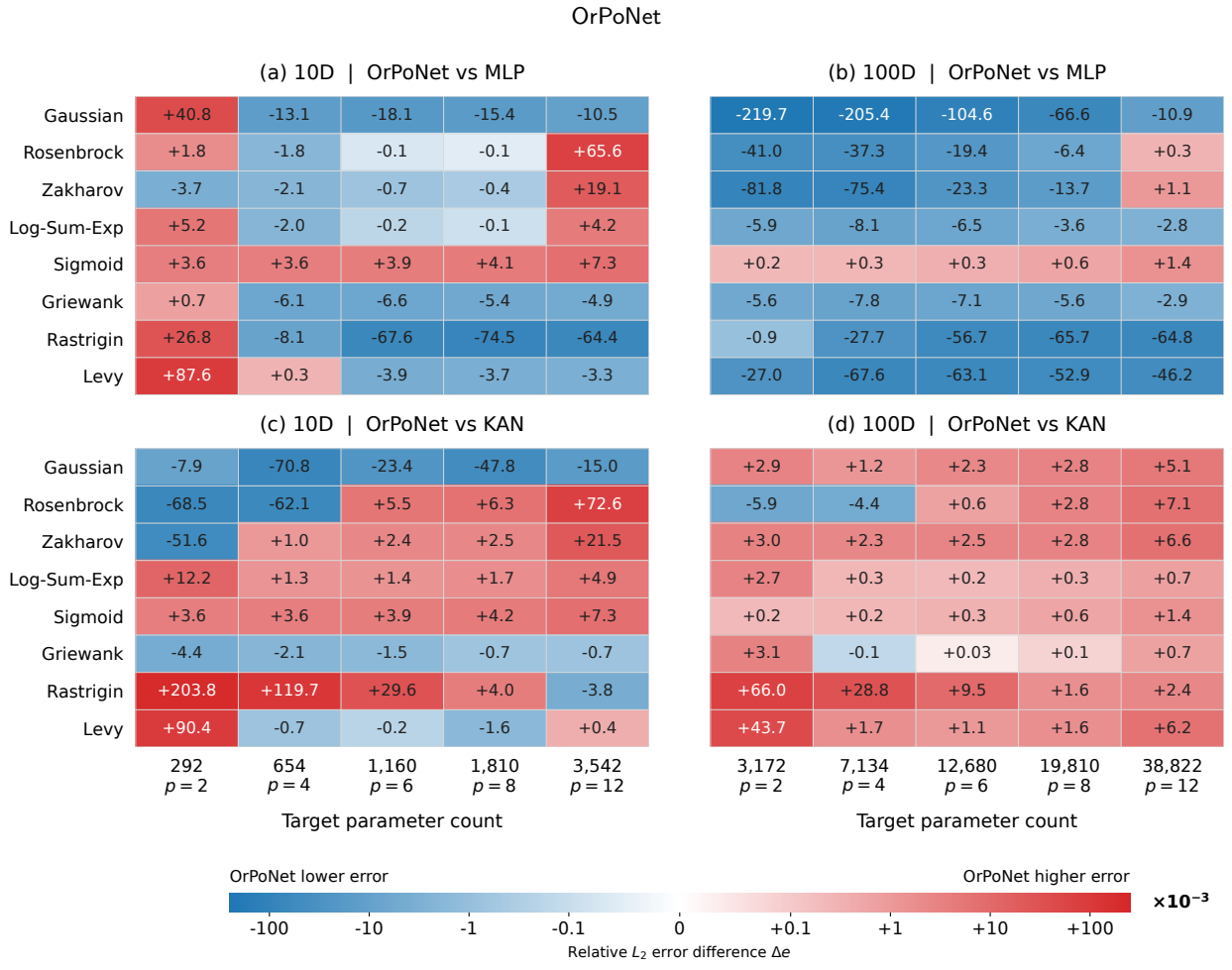


Figure 6: Differences in test relative-L_2 errors at approximately matched trainable parameter counts. The left and right columns show 10D and 100D results, respectively; the top and bottom rows compare residual OrPoNet with MLP and KAN. Each cell shows $\Delta e = e_{\text{OrPoNet}} - e_{\text{baseline}}$. OrPoNet has lower mean errors than MLP in most tested configurations, while its advantages over KAN are more frequent at lower dimensions.

while scaling more effectively to high-dimensional settings. We first consider benchmarking the Bratley Sum function in the form

$$f(\mathbf{x}) = \sum_{j=1}^{d} (-1)^j \prod_{k=1}^{j} x_k, \qquad \mathbf{x} = (x_1, \ldots, x_d) \in [0,1]^d. \tag{31}$$

This benchmark function is used to compare DeepPCE [33] and the variants of truncated PCEs. The DeepPCE combines PCE bases with a structured circuit architecture inspired by probabilistic circuits [46] such that it can compactly represent high-dimensional polynomial terms while preserving tractable computation of moments and Sobol indices. This model has shown scalability beyond the variants of PCE methods (e.g., truncated PCEs and adaptive-basis truncated PCEs). Herein, we use residual OrPoNet to learn the PCE of the Bratley Sum function and compare it with the DeepPCE model and truncated PCE (PCE_q). The sparse PCE variants reported in [33] are not included in Fig. 7, because their effective model size depends on the adaptive basis-selection procedure and is not directly comparable to the trainable-parameter counts of DeepPCE and OrPoNet.[6]

[6]We use the values of DeepPCE and PCE_q from [33]; details on the experimental setting and results of 10 OrPoNet runs are reported in Appendix D.6.

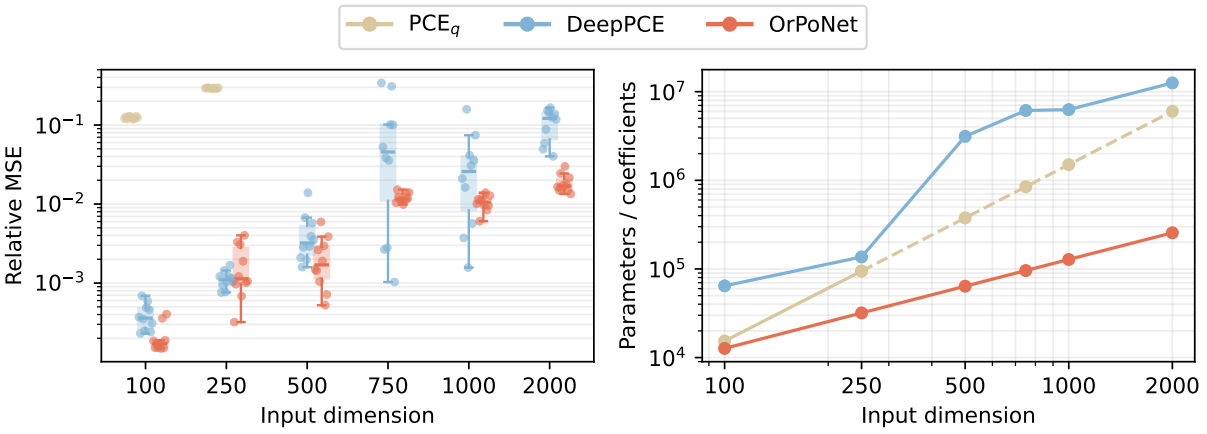


Figure 7: Bratley benchmark comparison across input dimensions. Left panel: test relative MSE distributions over 10 runs for truncated PCE (PCE$_q$), DeepPCE, and residual OrPoNet. Right panel: corresponding model sizes, measured as trainable parameters for DeepPCE and OrPoNet and candidate coefficients for PCE$_q$. The dashed PCE$_q$ curve beyond 250 dimensions indicates coefficient counts extrapolated from the fixed hyperbolic truncation rule rather than available PCE$_q$ relative MSE results. The results of PCE$_q$ and DeepPCE are taken from [33].

Figure 7 reveals that the traditional truncated PCE baseline (PCE$_q$) becomes difficult to scale beyond 250 dimensions because the number of polynomial basis functions grows rapidly with dimension. DeepPCE remains trainable up to 2000 dimensions, but its test relative MSE increases and shows larger variance across training runs in higher dimensions. DeepPCE is sensitive to weight initialization and has vanishing or exploding outputs in high-dimensional settings [33]. For the Bratley benchmark, the protocol in [33] first trains ten randomly initialized models for 50 epochs each and then continues training the candidate with the lowest validation loss. In contrast, each OrPoNet run reported here starts from a single initialization. In general, OrPoNet achieves lower mean test relative MSE and smaller sample standard deviations than DeepPCE except at 250 dimensions, while using approximately 4–64 times fewer trainable parameters. The parameter count of OrPoNet scales linearly in dimension for a fixed polynomial degree and copy number.

We further evaluate the performance of OrPoNet in high-dimensional sensitivity analysis through the computation of the Sobol indices, defined in Sec. 3.3. For this purpose, we consider the Sobol G^* function:

$$f(\mathbf{X}) = \prod_{i=1}^{d} \frac{(1+\alpha)|2\left(x_i + \delta_i - \lfloor x_i + \delta_i \rfloor\right) - 1|^{\alpha} + a_i}{1 + a_i}, \qquad \mathbf{x} \sim \mathcal{U}[0,1]^d, \tag{32}$$

which admits closed-form Sobol indices for comparison. Herein, $\alpha = 0.5$, $\delta_i \sim \mathcal{U}[0,1]^d$ is a fixed random parameter, and a_i is linearly spaced from 0 to 10, i.e., $a_i = 10(i-1)/(d-1)$. We set $d = 100$.

The Sobol indices of OrPoNet and DeepPCE are extracted from their polynomial representations, whereas those of MLP are estimated by Monte Carlo sampling with 10^7 samples. We sample 8,000 training points, 2,000 validation points, and 2,000 test points from $[0,1]^d$ with $d = 100$. We use sixth-order Legendre polynomials for both OrPoNet and DeepPCE and train both models with Adam for up to 500 epochs.

In the first six panels of Fig. 8, we compare the predicted first-order Sobol indices (S_i) and total-effect indices (S_{T_i}) for $d = 100$ against the ground truth (solid black line). OrPoNet (first row) demonstrates a significantly closer fit to the true sensitivity distribution compared to DeepPCE (second row) and MLP (third row). The bottom scatter plots show the estimation error of Sobol indices and the recorded computation time. OrPoNet achieves the lowest mean absolute errors of the first-order and total-effect Sobol indices.

4.3. Application to PINNs

Another substantial application of OrPoNet is in the domain of solving PDEs with PINNs [3, 4]. The method encodes residuals of PDEs as well as the corresponding boundary and initial conditions into the loss function and learns

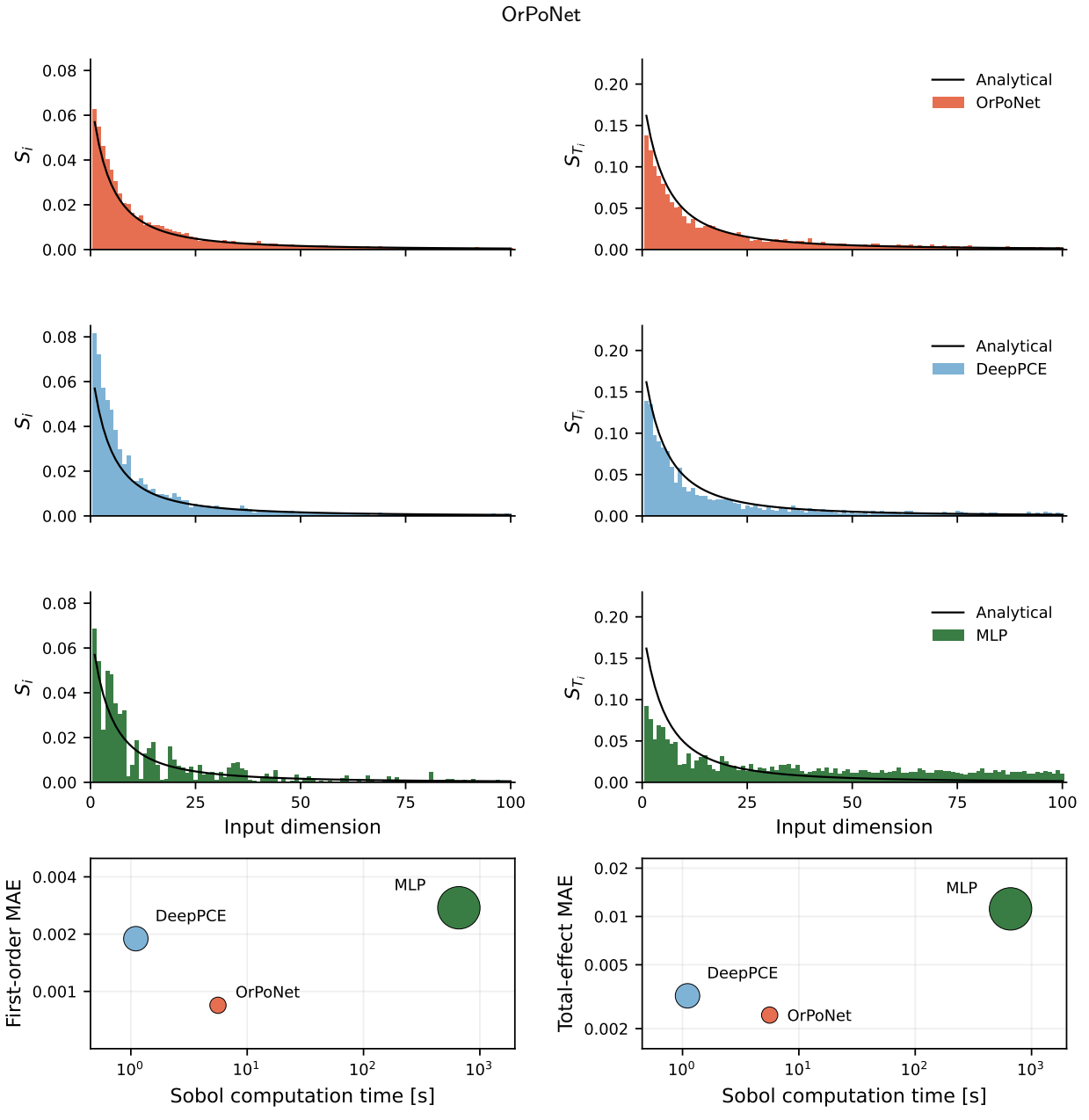


Figure 8: Performance benchmarking of OrPoNet against DeepPCE and MLP in high-dimensional sensitivity analysis. The top six panels compare the predicted first-order Sobol indices (S_i) and total-effect indices (S_{T_i}) across 100 input dimensions against the ground truth (solid black line). OrPoNet (orange) demonstrates a significantly closer fit to the true sensitivity distribution compared to DeepPCE (blue) and MLP (green), which exhibit higher variance and bias. The bottom scatter plots show estimation error (Mean Absolute Error of indices) and recorded computation time. The circle areas are proportional to the square root of the number of trainable parameters. OrPoNet achieves the lowest mean absolute errors.

the PDE solutions by minimizing the loss at a collection of sampled points. This allows one to bypass mesh generation and numerical discretization, which is particularly advantageous for high-dimensional PDEs.

We showcase the capability of OrPoNet to solve PDEs within the PINN framework and use MLP as a representative baseline; other possibilities include specialized or hybrid physics-informed models, such as hybrid polynomial-series/neural-network schemes [47]. In this context, OrPoNet provides a bridge between spectral methods and machine learning, maintaining the explicit polynomial-series structure of spectral approximations while using trainable network

Table 2
Steady Brusselator final logged losses and model sizes. Architecture lists neurons per layer; ×1 denotes one OrPoNet copy.

Model	Architecture	Params (per network)	$\|\mathcal{F}(u,v)\|^2$	$\|\mathcal{G}(u,v)\|^2$
OrPoNet	$[16, 16] \times 1$	272	1.43×10^{-5}	3.81×10^{-5}
MLP	[32, 32, 32]	2,241	2.59×10^{-3}	1.40×10^{-3}

parameters to scale the representation beyond traditional matrix-based formulations. We therefore focus on PDEs with smooth solutions, for which polynomial or spectral expansions are known to provide highly efficient approximations.

4.3.1. Steady-state solution of Brusselator system

W apply OrPoNet to solve the steady-state response of a Brusselator system under external excitation,

$$\begin{aligned} \frac{\partial u}{\partial t} &= \mathcal{F}(u,v) \equiv D_u \Delta u + a - (b+1)u + u^2 v - f(x,y), \\ \frac{\partial v}{\partial t} &= \mathcal{G}(u,v) \equiv D_v \Delta v + bu - u^2 v - g(x,y), \end{aligned} \tag{33}$$

defined in a rectangle $\Omega = [-1,1]^2$ with boundary conditions

$$u(x,y) = a, \quad v(x,y) = b/a, \quad (x,y) \in \partial\Omega, \tag{34}$$

where u and v represent the concentrations of two chemical components, and $D_u = 0.03$ and $D_v = 0.02$ denote diffusion coefficients of u and v, respectively. In addition, $a = 1$ and $b = 3$ are concentration rates. Driving forces are

$$f(x,y) = 0.18\sin(\pi x)\cos(2\pi y) - 0.11\cos(2\pi x)\sin(\pi y) + 0.07\cos(3\pi x)\cos(2\pi y) + 0.05xy, \tag{35}$$

$$g(x,y) = -0.14\cos(\pi x)\cos(\pi y) + 0.09\sin(2\pi x)\sin(\pi y) - 0.06\cos(2\pi y) + 0.04(x^2 - 0.5y^2). \tag{36}$$

To learn the steady-state ($u_t = 0$ and $v_t = 0$) solution using PINNs, we construct the solutions in the form

$$\begin{aligned} \bar{u}(x,y) &= a + (1-x^2)(1-y^2)\mathrm{NN}_u(x,y), \\ \bar{v}(x,y) &= b/a + (1-x^2)(1-y^2)\mathrm{NN}_v(x,y). \end{aligned} \tag{37}$$

where NN_u and NN_v are independent networks for u and v, respectively. With these output forms, the boundary conditions of Eq. (34) are automatically satisfied. This is a standard approach in spectral methods and is also widely adopted in PINNs to alleviate competition among different loss terms. This approach is particularly suitable for OrPoNet, which uses global polynomial representations. Empirically, we found that softly enforcing the boundary conditions through the loss function yields competing training dynamics between the PDE residual and boundary-condition losses and leads to degraded convergence for OrPoNet. For m interior collocation points (x_i, y_i), the loss function is

$$L = \frac{1}{m}\sum_{i=1}^{m}\left[\left|\mathcal{F}(\bar{u},\bar{v})(x_i,y_i)\right|^2 + \left|\mathcal{G}(\bar{u},\bar{v})(x_i,y_i)\right|^2\right]. \tag{38}$$

To generate the training dataset, we randomly sample $2^{11} = 2048$ interior points and $2^9 = 512$ boundary points using Sobol sampling. We apply two two-layer OrPoNets to learn u and v separately, with each layer allocated a single copy of 14th-order univariate Chebyshev polynomials. Because the basis set includes the constant term and all polynomial degrees from 0 to 14, each layer has 16 neurons, yielding $16^2 + 16 = 272$ parameters per network. For comparison, we also generate two three-layer MLPs with 32 neurons per layer (in total 2241 parameters per network) as a baseline. This configuration was selected after testing several alternatives.

We train the networks using the Adam optimizer for 2000 epochs, followed by a refinement phase with L-BFGS for 80 epochs. OrPoNet improves accuracy by a factor of 50 with approximately one-tenth of the trainable parameters (Table 2). A more detailed comparison between the finite-difference ground truth and the predictions by OrPoNet and MLP, together with the corresponding errors, is presented in Fig. 9. While both architectures successfully approximate the solution profile, the error for OrPoNet is more evident near the domain boundaries. This behavior is expected, as polynomial-based approximations often exhibit Runge's phenomenon, i.e., increased sensitivity or oscillations near the domain boundaries.

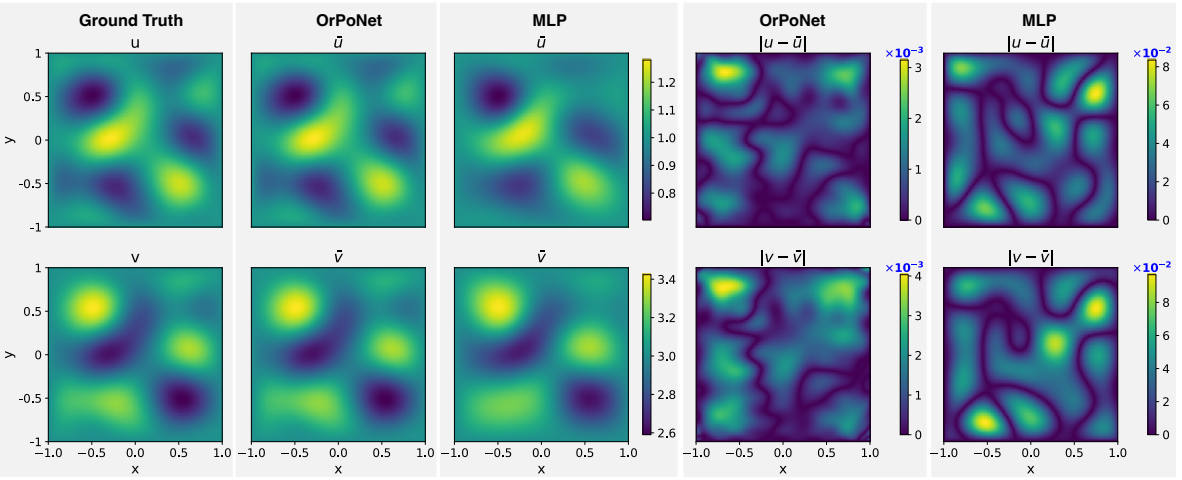


Figure 9: The steady-state solution of the Brusselator obtained by the finite-difference method (ground truth), OrPoNet, and MLP. Notably, OrPoNet yields an order-of-magnitude more accurate result than MLP with approximately one-tenth of the parameter count. The ground truth is generated using a finite-difference method on a 129×129 grid.

Table 3
20D-Poisson final PDE losses and model sizes. OrPoNet includes residual connections and additive branches. Architecture lists neurons per layer; $[6]_{20}$ denotes 20 layers with six neurons each, and $\times 1$ denotes one copy.

Model	Architecture	Params	PDE loss
Residual OrPoNet	$[6]_{20} \times 1$	1,374	4.32×10^{-5}
MLP	$[128, 128, 128, 128]$	52,353	2.19×10^{-5}

4.3.2. Poisson equation in $d = 20$ dimensions

We evaluate the performance of residual OrPoNet in solving high-dimensional PDEs, using as a testbed a 20-dimensional Poisson equation

$$-\Delta u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad f(\boldsymbol{x}) = 2 \sum_{i=1}^{20} \prod_{\substack{j=1 \\ j \neq i}}^{20} (1 - x_j^2), \quad \boldsymbol{x} \in \Omega = [-1, 1]^{20}. \tag{39}$$

with boundary condition $u(\boldsymbol{x}) = 0$ on $\boldsymbol{x} \in \partial\Omega$. We follow the same approach to enforce the boundary conditions to the NN output and formulate the loss function in the form of Eq. (38). We adopt a dataset of $2^{12} = 4096$ interior points and $2^{13} = 8192$ test points. We train residual OrPoNet and MLP using AdamW for 400 and 100 epochs, respectively, with learning rate of 10^{-4} and a mini-batch size of 256, and resample the training points every epoch. Although the PDE loss of OrPoNet is slightly higher than that of MLP, it achieves this accuracy with approximately 40 times fewer parameters (Table 3). Both OrPoNet and MLP capture the overall solution landscape (Fig. 10).

Connection to spectral methods. Forward propagation of OrPoNet recovers the multivariate orthogonal-polynomial representation of a target function. When integrated with PINNs, it can be viewed as a spectral method solved under the machine-learning framework. In practice, how does this approach align with the traditional spectral methods in terms of convergence and accuracy?

To answer this question, we revisit the steady-state solution of the Brusselator and solve it using the traditional spectral method, which is implemented by the collocation (pseudo-spectral) method with Chebyshev polynomials as the basis functions. A similar technique to Eq. (37) is adopted to ensure that the Dirichlet boundary conditions are automatically satisfied for the collocation method. For the OrPoNet, we evaluate the performance of different sampling and optimization schemes, including Sobol sampling, Chebyshev collocation points, single-stage optimization, and an alternating Adam/L-BFGS optimizer, following related practice in the recent PIKANs architecture [48].

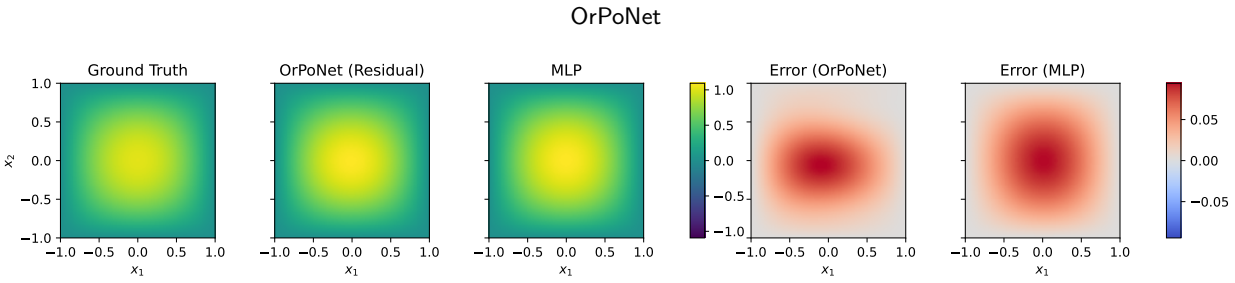


Figure 10: Solution profiles for the 20D Poisson equation on representative two-dimensional slices. The exact solution is generated from the manufactured solution. Residual OrPoNet and MLP both recover the dominant solution structure, while OrPoNet achieves this with substantially fewer trainable parameters.

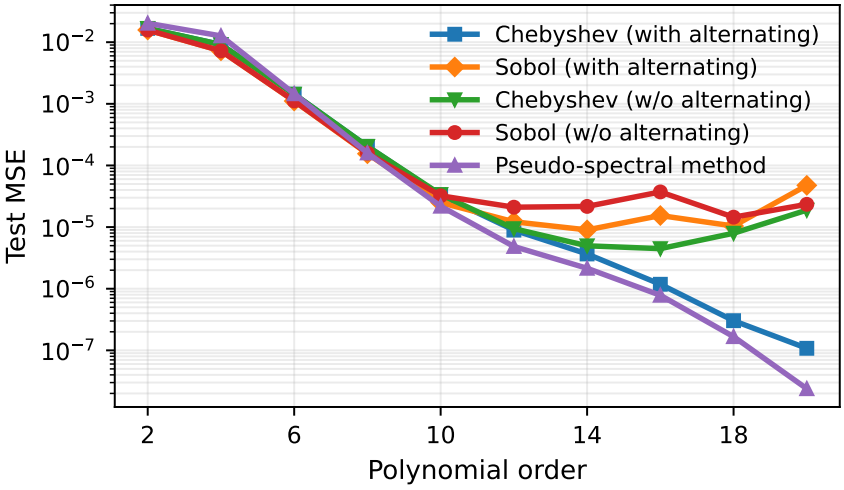


Figure 11: Test PDE residual loss versus polynomial order for the steady Brusselator problem under different sampling and optimization schemes. The pseudo-spectral collocation method provides the reference convergence trend. OrPoNet follows a similar decay when trained with Chebyshev collocation points and the alternating Adam/L-BFGS optimizer, whereas Sobol-sampled or single-stage variants become less accurate at higher orders.

Figure 11 reveals that the pseudo-spectral method exhibits the expected exponential decay in test PDE residual loss as the polynomial order increases. OrPoNet follows a similar convergence trend when trained with Chebyshev collocation points and the alternating Adam/L-BFGS optimizer, reaching a test PDE residual loss of 1.08×10^{-7} at order 20, compared with 2.39×10^{-8} for the pseudo-spectral method. This suggests that OrPoNet exhibits spectral-like error decay for this smooth two-dimensional problem under the tested sampling and optimization scheme. The Sobol-sampled and single-stage variants match with the pseudo-spectral method at low to moderate orders but become less accurate at higher orders, indicating that the observed convergence behavior is sensitive to the sampling and optimization schemes.

5. Conclusion

Orthogonal polynomial expansions are powerful tools for approximating smooth functions in scientific computing, but they suffer from the curse of dimensionality. We addressed this challenge by introducing OrPoNet, a structured feed-forward network representation of orthogonal polynomial expansions. For a fixed copy number, the parameter count of OrPoNet scales linearly with the input dimension and quadratically with the maximum polynomial order. We proved the universal approximation property of OrPoNet and introduced a basis-aware weight initialization scheme together with architectural extensions, including gated residual connections and additive basis branches, to improve expressiveness and scalability of the model. Unlike black-box NNs, OrPoNet preserves polynomial structure, allowing

explicit coefficient reconstruction in moderate dimensions and direct computation of moments and Sobol indices from network weights.

In smooth-function approximation benchmarks, OrPoNet has lower median errors than MLPs after architecture selection. At matched parameter counts, OrPoNet achieves lower mean errors than KAN on selected functions and polynomial orders, particularly in the 10D cases. KAN achieves lower errors in most tested 100D configurations. On the Bratley benchmark, OrPoNet uses substantially fewer trainable parameters than DeepPCE and achieves lower mean test relative MSE at five of the six tested dimensions. In the tested PINN problems, OrPoNet achieves comparable or lower PDE losses than MLPs with substantially fewer trainable parameters.

The proposed architecture should be viewed as a structured alternative that connects spectral approximation with modern machine learning, rather than as a replacement for black-box NNs such as MLPs. Its advantages are most relevant when the target problem benefits from smooth global polynomial representations; conversely, it inherits limitations of global polynomial methods, such as reduced efficiency for non-smooth functions, complex geometries, or intricate boundary conditions. Instead of aiming to cover all problem classes, OrPoNet provides a route for embedding spectral representations into differentiable learning pipelines, including stochastic optimization, automatic differentiation, and physics-informed training.

OrPoNet has several limitations. Similar to KAN, the computational time of this architecture can be considerably higher than that of MLPs, especially in high-dimensional problems. This overhead arises from two major sources: the iterative evaluation of orthogonal basis functions and the longer computational chains used in backpropagation. In addition, as stated previously, the performance of OrPoNet can deteriorate for non-smooth functions due to the known limitations of polynomial approximations near discontinuities or sharp features; illustrative nonsmooth examples are provided in Appendix D.4. Finally, as in classical spectral approximations, the extrapolation capability of OrPoNet is limited by the analyticity domain of the target function or solution manifold, which is generally unknown a priori. Although residual and additive branches improve trainability in high-dimensional settings, the current soft-gate mechanism does not yet reliably enforce sparsity in the induced polynomial expansion.

Future work may explore more efficient basis evaluation, such as precomputing explicit polynomial coefficients for low-order bases, although coefficient-based evaluation can become numerically unstable at high orders. Further architectural innovations may also be needed to shorten or parallelize the propagation paths in high-dimensional settings. Another promising direction is to apply OrPoNet to ultra-high-dimensional systems after projection onto reduced-order coordinates, where tens to hundreds of modal coefficients may be sufficient to capture the dominant dynamics. This setting may require integration with arbitrary polynomial bases constructed from data [49, 50]. However, application of the moment recursions in Appendix C to reduced-order coordinates requires statistical independence. One possible approach is to regularize an encoder to encourage independent latent coordinates and construct univariate aPC bases orthonormal with respect to their marginal distributions. The effect of any remaining dependence on the accuracy of the propagated moments would need to be assessed.

More broadly, OrPoNet suggests a route toward neuralized spectral representations, where classical approximation bases are embedded into trainable network architectures to combine interpretability, structured inference, and optimization.

Code availability

The core PyTorch implementation of OrPoNet and selected examples from this study are available at `https://github.com/siyuan-xing/OrPoNet`. An archived copy is available on Zenodo at `https://doi.org/10.5281/zenodo.24326168`.

Declaration of generative AI and AI-assisted technologies in the writing process

The first author used Google's Gemini and OpenAI's ChatGPT to assist with language editing. The authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Acknowledgments

This research was supported in part by the Air Force Office of Scientific Research under grants FA9550-21-1-0381 and FA9550-24-1-0237, and by the Office of Advanced Scientific Computing Research (ASCR) within the Department

of Energy Office of Science under award number DE-SC0023163. The first author gratefully acknowledges Energy Science and Engineering Department at Stanford University for hospitality and support during his sabbatical visit. He also thanks Stathis Caralampidis for recommending materials on spectral methods and Brendon Anderson for useful discussions. The authors thank the anonymous reviewers for their constructive comments and helpful suggestions.

CRediT authorship contribution statement

Siyuan Xing: Conceptualization, Methodology, Research Design, Investigation, Software, Validation, Formal analysis, Writing - original draft, Writing – review and editing. **Daniel M. Tartakovsky:** Conceptualization, Research Design, Writing - review and editing, Resources.

References

[1] A. I. J. Forrester, A. Sóbester, and A. J. Keane. *Engineering Design via Surrogate Modelling: A Practical Guide*. John Wiley & Sons, Chichester, UK, 2008.

[2] D. Xiu. *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press, Princeton, NJ, 2010.

[3] M. Raissi, P. Perdikaris, and G. E. Karniadakis. 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, 2019.

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

[5] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[6] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[7] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

[8] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105, 2012.

[10] L. Novák, H. Sharma, and M. D. Shields. Physics-informed polynomial chaos expansions. *Journal of Computational Physics*, 506:112926, 2024.

[11] J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, Mineola, NY, second edition, 2001.

[12] D. Xiu and G. E. Karniadakis. The wiener–askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing*, 24(2):619–644, 2002.

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

[14] J. Hampton and A. Doostan. Compressive sampling of polynomial chaos expansions: Convergence analysis and sampling strategies. *Journal of Computational Physics*, 280:363–386, 2015.

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

[16] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[17] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–451, 2004.

[18] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 40–44, 1993.

[19] N. Lüthen, S. Marelli, and B. Sudret. Automatic selection of basis-adaptive sparse polynomial chaos expansions for engineering applications. *International Journal for Uncertainty Quantification*, 12:49–74, 2022.

[20] A. Doostan, A. Validi, and G. Iaccarino. Non-intrusive low-rank separated approximation of high-dimensional stochastic models. *Computer Methods in Applied Mechanics and Engineering*, 263:42–55, 2013.

[21] K. Konakli and B. Sudret. Polynomial metamodels with canonical low-rank approximations: Numerical insights and comparison to sparse polynomial chaos expansions. *Journal of Computational Physics*, 321:1144–1169, 2016.

[22] M. Chevreuil, R. Lebrun, A. Nouy, and P. Rai. A least-squares method for sparse low rank approximation of multivariate functions. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):897–921, 2015.

[23] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[24] I. V. Oseledets. Constructive representation of functions in low-rank tensor formats. *Constructive Approximation*, 37(1):1–18, 2013.

[25] Daniele Bigoni, Allan P. Engsig-Karup, and Youssef M. Marzouk. Spectral tensor-train decomposition. *SIAM Journal on Scientific Computing*, 38(4):A2405–A2439, 2016.

[26] Alex Gorodetsky, Sertac Karaman, and Youssef Marzouk. A continuous analogue of the tensor-train decomposition. *Computer Methods in Applied Mechanics and Engineering*, 347:59–84, 2019.

[27] K. D. Kantarakias and G. Papadakis. Sensitivity-enhanced generalized polynomial chaos for efficient uncertainty quantification. *Journal of Computational Physics*, 491:112377, 2023.

[28] S. Oladyshkin, T. Praditia, I. Kroeker, F. Mohammadi, W. Nowak, and S. Otte. 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, 2023.
[29] H. Sharma, L. Novák, and M. D. Shields. Physics-constrained polynomial chaos expansion for scientific machine learning and uncertainty quantification. *Computer Methods in Applied Mechanics and Engineering*, 431:117314, 2024.
[30] B. Bahmani, I. G. Kevrekidis, and M. D. Shields. Neural chaos: A spectral stochastic neural operator. *Journal of Computational Physics*, 539:114233, 2025.
[31] H. Mhaskar, Q. Liao, and T. Poggio. Learning functions: When is deep better than shallow? In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 2343–2349, 2017.
[32] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacic, T. Y. Hou, and M. Tegmark. KAN: Kolmogorov–Arnold networks. In *The Thirteenth International Conference on Learning Representations*, 2025.
[33] J. Exenberger, S. Ranftl, and R. Peharz. Deep polynomial chaos expansion. In *International Conference on Artificial Intelligence and Statistics*, 2026.
[34] G. G. Chrysos, S. Moschoglou, G. Bouritsas, Y. Panagakis, J. Deng, and S. Zafeiriou. P-nets: Deep polynomial neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7325–7335, 2020.
[35] S. S. Sahoo, C. H. Lampert, and G. Martius. Learning equations for extrapolation and control. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4442–4450. PMLR, 2018.
[36] S. Xing, Q. Han, E.G. Charalampidis, and Y.C. Lai. Data-driven discovery of high-dimensional dynamical systems with sparse interpretable neural networks. *PRX Intelligence*, 1:013024, 2026.
[37] M. H. Stone. The generalized Weierstrass approximation theorem. *Mathematics Magazine*, 21(4):167–184, 1948.
[38] M. H. Stone. The generalized Weierstrass approximation theorem (continued). *Mathematics Magazine*, 21(5):237–254, 1948.
[39] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
[40] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
[41] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
[42] B. Lim, S. O. Arik, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.
[43] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 933–941. PMLR, 2017.
[44] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.
[45] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved March 27, 2026, from `http://www.sfu.ca/~ssurjano`.
[46] A. Vergari, Y. Choi, A. Liu, S. Teso, and G. Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 13189–13201. Curran Associates, Inc., 2021.
[47] D. G. Patsatzis, N. Kazantzis, I. G. Kevrekidis, and C. Siettos. A hybrid neural network–polynomial series scheme for learning invariant manifolds of discrete dynamical systems, 2025.
[48] J. Wan, L. Wen, Z. Jian, J. Wu, J. Li, M. Lian, and K. Wang. PIKANs: Physics-informed Kolmogorov–Arnold networks for landslide time-to-failure prediction. *Computers & Geosciences*, 208:106094, 2026.
[49] C. Soize and R. Ghanem. Physical systems with random uncertainties: Chaos representations with arbitrary probability measure. *SIAM Journal on Scientific Computing*, 26(2):395–410, 2004.
[50] S. Oladyshkin and W. Nowak. Data-driven uncertainty quantification using the arbitrary polynomial chaos expansion. *Reliability Engineering & System Safety*, 106:179–190, oct 2012.
[51] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.
[52] P. Tichavský and O. Straka. Optimizing the order of modes in tensor train decomposition. *IEEE Signal Processing Letters*, 32:1361–1365, 2025.

Appendix

A. Parameter count of OrPoNet

We summarize the trainable parameter count of OrPoNet using the notation in Sec. 3.1. Let $L = n$ be the number of polynomial layers, C the number of copies, and m the output dimension. Let p_l denote the maximum polynomial order in layer l. Then the number of neurons per layer becomes

$$r_l = p_l + 2, \tag{A.1}$$

where the extra two neurons correspond to the explicit constant and the zeroth-order polynomial basis. The scaling layer and the first polynomial layer do not introduce trainable weights. For the vanilla OrPoNet, each subsequent hidden

layer has a projection tensor of size $r_{l-1} \times r_l \times C$, and the final output layer has size $m \times r_L \times C$. Therefore,

$$N_{\text{vanilla}} = C \sum_{l=2}^{n} r_{l-1} r_l + mC r_L. \tag{A.2}$$

For a single-output model with the same polynomial order p in all layers, $r_l = r = p + 2$, and Eq. (A.2) reduces to

$$N_{\text{vanilla}} = C \left[(n-1) r^2 + r \right], \tag{A.3}$$

which gives the scaling $\mathcal{O}(Cnp^2)$.

For residual OrPoNet and its variants, the multiplicative polynomial branch has the same parameter count as Eq. (A.2). Additional parameters depend on which architectural extensions are enabled. If adjacent layers have different widths, the residual branch uses a skip projection with $Cr_{l-1}r_l$ parameters; otherwise, the skip connection is the identity and introduces no trainable parameters. When enabled, an additive basis branch contributes Cr_l^2 parameters in layer l. When enabled, the residual gate contributes Cr_l gate logits in layer l. Thus, for layers $l = 2, \dots, n$, the total parameter count is

$$N_{\text{res}} = N_{\text{vanilla}} + C \sum_{l=2}^{n} \mathbb{I}(r_{l-1} \neq r_l) r_{l-1} r_l + C \sum_{l=2}^{n} \mathbb{I}_{\text{add}} r_l^2 + C \sum_{l=2}^{n} \mathbb{I}_{\text{gate}} r_l, \tag{A.4}$$

where $\mathbb{I}_{\text{add}}$ and $\mathbb{I}_{\text{gate}}$ indicate whether the additive basis branch and residual gate are used, respectively. In the common case where all layers use the same order and the additive branch is enabled without gates, Eq. (A.4) becomes

$$N_{\text{res}} = C \left[2(n-1) r^2 + mr \right], \tag{A.5}$$

which gives the scaling $\mathcal{O}(Cnp^2)$ as well.

B. Functional tensor-train representation of multi-copy OrPoNet

Consider a vanilla OrPoNet with $n \geq 2$ coordinates, C independent copies, and layer widths r_i per copy. For an internal layer $i = 2, \dots, n-1$, we define a third-order array $\mathcal{G}^{[i]}(x_i) \in \mathbb{R}^{r_i \times r_{i-1} \times C}$ by its copy slices,

$$\mathcal{G}^{[i]}_{:,:,c}(x_i) = \boldsymbol{G}^{[i]}_{\text{net},c}(x_i) = \boldsymbol{D}^{[i]}(x_i) \boldsymbol{W}_c^{[i-1]}, \qquad c = 1, \dots, C. \tag{B.1}$$

The boundary arrays have sizes $r_1 \times 1 \times C$ and $1 \times r_{n-1} \times C$, with slices

$$\begin{aligned} \mathcal{G}^{[1]}_{:,:,c}(x_1) &= \boldsymbol{G}^{[1]}_{\text{net},c}(x_1) = \boldsymbol{\Psi}^{[1]}(x_1), \\ \mathcal{G}^{[n]}_{:,:,c}(x_n) &= \boldsymbol{G}^{[n]}_{\text{net},c}(x_n) = \boldsymbol{W}_c^{[n]} \boldsymbol{D}^{[n]}(x_n) \boldsymbol{W}_c^{[n-1]}. \end{aligned} \tag{B.2}$$

The output layer sums the contributions from all copies and yields:

$$\bar{f}(\boldsymbol{x}) = \sum_{c=1}^{C} \mathcal{G}^{[n]}_{:,:,c}(x_n) \mathcal{G}^{[n-1]}_{:,:,c}(x_{n-1}) \cdots \mathcal{G}^{[1]}_{:,:,c}(x_1), \tag{B.3}$$

with the products evaluated from right to left. To express Eq. (B.3) as a single matrix-product chain, we combine the copy index with the bond indices and define the internal matrix-valued functional cores

$$\widehat{\boldsymbol{G}}^{[i]} = \operatorname{diag}\left(\boldsymbol{G}^{[i]}_{\text{net},1}, \dots, \boldsymbol{G}^{[i]}_{\text{net},C} \right), \quad i = 2, \dots, n-1. \tag{B.4}$$

The first core $\widehat{\boldsymbol{G}}^{[1]}$ is formed by vertically concatenating $\boldsymbol{G}^{[1]}_{\text{net},c}$ over all copies, while the last core $\widehat{\boldsymbol{G}}^{[n]}$ is formed by horizontally concatenating $\boldsymbol{G}^{[n]}_{\text{net},c}$. Thus, we have

$$\bar{f}(\boldsymbol{x}) = \widehat{\boldsymbol{G}}^{[n]}(x_n) \widehat{\boldsymbol{G}}^{[n-1]}(x_{n-1}) \cdots \widehat{\boldsymbol{G}}^{[1]}(x_1). \tag{B.5}$$

Since the output is a scalar, transposing this product gives the FT form defined in Eq. (11). This representation has FT ranks $\gamma_i = Cr_i$ that need not be minimal. Consequently, the minimal FT rank $\gamma_i^{\min}$ associated with the partition $(x_1, \dots, x_i) \mid (x_{i+1}, \dots, x_n)$ satisfies

$$\gamma_i^{\min} \leq Cr_i, \qquad i = 1, \dots, n-1. \tag{B.6}$$

This means that increasing the number of copies C by one will increase the upper bound of the minimal FT rank by r_i.

C. Sobol indices from OrPoNet weights

For the orthonormal PCE in Eq. (16), the mean and variance are

$$\mathbb{E}\left[f(\boldsymbol{\xi})\right] = c_{\mathbf{0}}, \qquad \mathrm{Var}(f(\boldsymbol{\xi})) = \sum_{\boldsymbol{\alpha}\in\mathcal{A},\boldsymbol{\alpha}\neq\mathbf{0}} c_{\boldsymbol{\alpha}}^2. \tag{C.1}$$

In addition, the variance of the conditional expectation can also be obtained from PCE coefficients. Let $\mathcal{D} = \{1,\dots,n\}$ be an index set of dimensions, and let $\mathcal{K} \subseteq \mathcal{D}$ denote a subset of interest. Then the variance of the corresponding conditional expectation is

$$\mathrm{Var}_{\xi_{\mathcal{K}}}\left(\mathbb{E}_{\xi_{\mathcal{D}\setminus\mathcal{K}}}[Y \mid \boldsymbol{\xi}_{\mathcal{K}}]\right) = \sum_{\boldsymbol{\alpha}\in\mathcal{A}_{\mathcal{K}}} c_{\boldsymbol{\alpha}}^2. \tag{C.2}$$

Thus the first-order and total-effect Sobol indices in Eq. (18) is written as

$$S_i = \frac{\sum_{\boldsymbol{\alpha}\in\mathcal{A}_i} c_{\boldsymbol{\alpha}}^2}{\sum_{\boldsymbol{\alpha}\in\mathcal{A},\boldsymbol{\alpha}\neq\mathbf{0}} c_{\boldsymbol{\alpha}}^2}, \qquad \mathcal{A}_i = \{\boldsymbol{\alpha}\in\mathcal{A} : \alpha_i > 0,\ \alpha_j = 0 \text{ for all } j \neq i\}, \tag{C.3}$$

and

$$S_i^T = \frac{\sum_{\boldsymbol{\alpha}\in\mathcal{A}_i^T} c_{\boldsymbol{\alpha}}^2}{\sum_{\boldsymbol{\alpha}\in\mathcal{A},\boldsymbol{\alpha}\neq\mathbf{0}} c_{\boldsymbol{\alpha}}^2}, \qquad \mathcal{A}_i^T = \{\boldsymbol{\alpha}\in\mathcal{A} : \alpha_i > 0\}. \tag{C.4}$$

For moderate-dimensional problems, the coefficients $c_{\boldsymbol{\alpha}}$ can be recovered by expanding active paths through the trained network and aggregating all path products that produce the same multi-index. In high dimensions, we avoid explicitly constructing this coefficient table and instead compute the same quantities by propagating first and second moments through the network weights. For notational clarity, the below derivation is written for a single copy.

For a standard multiplicative layer of OrPoNet, with $\ell = 2,\dots,n$, the layer update can be recast as

$$\boldsymbol{y}^{[\ell]} = D_{\boldsymbol{\Psi}}^{[\ell]} W^{[\ell-1]} \boldsymbol{y}^{[\ell-1]}, \quad \text{with} \quad D_{\boldsymbol{\Psi}}^{[\ell]} = \mathrm{diag}\left(\boldsymbol{\Psi}^{[\ell]}(\xi_\ell)\right), \tag{C.5}$$

where $\boldsymbol{y}^{[\ell]}$ denotes the output of layer ℓ, and $\boldsymbol{\Psi}^{[\ell]}$ is the basis vector for coordinate ξ_ℓ. With a slight abuse of notation, we use ξ_ℓ to denote the scaled input. We denote the first and second moments of the layer output as

$$\boldsymbol{m}^{[\ell]} = \mathbb{E}[\boldsymbol{y}^{[\ell]}], \qquad Q^{[\ell]} = \mathbb{E}\left[\boldsymbol{y}^{[\ell]}\left(\boldsymbol{y}^{[\ell]}\right)^{\mathsf{T}}\right]. \tag{C.6}$$

Substituting Eq. (C.5) into the definition of $m^{[\ell]}$ yields

$$\boldsymbol{m}^{[\ell]} = W^{[\ell-1]}\boldsymbol{m}^{[\ell-1]} \odot \mathbb{E}[\boldsymbol{\Psi}^{[\ell]}]. \tag{C.7}$$

Similarly, the second moment becomes

$$\begin{aligned} Q^{[\ell]} &= \mathbb{E}\left[D_{\boldsymbol{\Psi}}^{[\ell]} W^{[\ell-1]} \boldsymbol{y}^{[\ell-1]}\left(\boldsymbol{y}^{[\ell-1]}\right)^{\mathsf{T}}\left(W^{[\ell-1]}\right)^{\mathsf{T}} D_{\boldsymbol{\Psi}}^{[\ell]}\right] \\ &= W^{[\ell-1]} Q^{[\ell-1]}\left(W^{[\ell-1]}\right)^{\mathsf{T}} \odot B^{[\ell]}, \end{aligned} \tag{C.8}$$

where $B^{[\ell]} = \mathbb{E}[\boldsymbol{\Psi}^{[\ell]}\left(\boldsymbol{\Psi}^{[\ell]}\right)^{\mathsf{T}}]$. This indicates that we can calculate $Q^{[\ell]}$ recursively by propagating it from previous layers. Consequently, $Q^{[\ell]}$ compactly stores all matching multi-index path-pair contributions accumulated up to layer ℓ.

Residual OrPoNet can be treated using the same moment-propagation principle by expanding the layer update in Eq. (28) into three branches:

$$\boldsymbol{y}^{[\ell]} = \underbrace{\boldsymbol{\Psi}^{[\ell]}(\xi_\ell) \odot \left(W^{[\ell-1]}\boldsymbol{y}^{[\ell-1]}\right)}_{\boldsymbol{a}^{[\ell]}} + \underbrace{\boldsymbol{g}^{[\ell]} \odot \left(R^{[\ell]}\boldsymbol{y}^{[\ell-1]}\right)}_{\boldsymbol{s}^{[\ell]}} + \underbrace{P^{[\ell]}\boldsymbol{\Psi}^{[\ell]}(\xi_\ell)}_{\boldsymbol{h}^{[\ell]}}, \tag{C.9}$$

where $R^{[\ell]}$ is either the identity map or the learned skip projection, $P^{[\ell]}$ is the additive-basis projection, and $\boldsymbol{g}^{[\ell]}$ is the residual gate. Assuming independent input coordinates, $\boldsymbol{\Psi}^{[\ell]}(\xi_\ell)$ is independent of $\boldsymbol{y}^{[\ell-1]}$, which depends only on previous coordinates. The first moment is therefore

$$\boldsymbol{m}^{[\ell]} = \mathbb{E}[\boldsymbol{\Psi}^{[\ell]}] \odot \left(W^{[\ell-1]}\boldsymbol{m}^{[\ell-1]}\right) + \boldsymbol{g}^{[\ell]} \odot \left(R^{[\ell]}\boldsymbol{m}^{[\ell-1]}\right) + P^{[\ell]}\mathbb{E}[\boldsymbol{\Psi}^{[\ell]}]. \tag{C.10}$$

The second moment is obtained by expanding the branch-pair products,

$$Q^{[\ell]} = \mathbb{E}\left[\left(\boldsymbol{a}^{[\ell]} + \boldsymbol{s}^{[\ell]} + \boldsymbol{h}^{[\ell]}\right)\left(\boldsymbol{a}^{[\ell]} + \boldsymbol{s}^{[\ell]} + \boldsymbol{h}^{[\ell]}\right)^{\mathsf{T}}\right], \tag{C.11}$$

which consists of the multiplicative–multiplicative, skip–skip, additive–additive, and corresponding cross terms. In the implementation, these terms are accumulated directly in weight space. For example, the main diagonal branch terms are

$$\mathbb{E}\left[\boldsymbol{a}^{[\ell]}(\boldsymbol{a}^{[\ell]})^{\mathsf{T}}\right] = W^{[\ell-1]}Q^{[\ell-1]}\left(W^{[\ell-1]}\right)^{\mathsf{T}} \odot B^{[\ell]}, \tag{C.12}$$

$$\mathbb{E}\left[\boldsymbol{s}^{[\ell]}(\boldsymbol{s}^{[\ell]})^{\mathsf{T}}\right] = \mathrm{diag}(\boldsymbol{g}^{[\ell]})R^{[\ell]}Q^{[\ell-1]}\left(R^{[\ell]}\right)^{\mathsf{T}}\mathrm{diag}(\boldsymbol{g}^{[\ell]}), \tag{C.13}$$

$$\mathbb{E}\left[\boldsymbol{h}^{[\ell]}(\boldsymbol{h}^{[\ell]})^{\mathsf{T}}\right] = P^{[\ell]}B^{[\ell]}\left(P^{[\ell]}\right)^{\mathsf{T}}. \tag{C.14}$$

The cross terms are computed analogously from the first and second moments of $\boldsymbol{y}^{[\ell-1]}$ and $\boldsymbol{\Psi}^{[\ell]}$. If the residual or additive branch is disabled, the corresponding branch terms are set to zero.

After propagating $\boldsymbol{m}^{[\ell]}$ and $Q^{[\ell]}$ to the final hidden layer, the moments of the output for the single-copy case are obtained from the linear output layer:

$$\mu = W^{[n]}\boldsymbol{m}^{[n]}, \qquad M = W^{[n]}Q^{[n]}(W^{[n]})^T. \tag{C.15}$$

The variance used in the Sobol indices is obtained by centering,

$$\mathrm{Var}(f(\boldsymbol{\xi})) = M - \mu^2. \tag{C.16}$$

When multiple copies are used, the hidden states are propagated through separate copy channels and are combined only at the output layer. These copy channels do not exchange hidden states during forward propagation. However, their outputs are functions of the same input variables and are therefore not statistically independent in general. Therefore, the output second moment is computed by retaining both within-copy and cross-copy second-moment contributions before summing the copies at the output layer.

The calculation of conditional-expectation variances of Eq. (C.2) for Sobol indices is similar, except that the basis moments are masked to retain only the constant block for marginalized dimensions. In our implementation, each coordinate uses a local basis vector

$$\boldsymbol{\psi}^{[i]}(\xi_i) = \left[1,\ \psi_0^{[i]}(\xi_i),\ \psi_1^{[i]}(\xi_i),\ \ldots,\ \psi_{p_i}^{[i]}(\xi_i)\right]^{\mathsf{T}}, \tag{C.17}$$

where $\psi_0^{[i]}(\xi_i)$ is the zeroth order basis function (i.e., the constant term). When dimension i is marginalized, the nonconstant second moments are masked while the constant block is retained:

$$\boldsymbol{\eta}^{[i]} = [1,\ 1,\ 0,\ \ldots,\ 0]^{\mathsf{T}}, \qquad \widetilde{\boldsymbol{\psi}}^{[i]}(\xi_i) = \boldsymbol{\eta}^{[i]} \odot \boldsymbol{\psi}^{[i]}(\xi_i). \tag{C.18}$$

For an active set $\mathcal{K}$, this marginalization is applied to all $i \notin \mathcal{K}$, while the full basis moment matrix is retained for $i \in \mathcal{K}$.

D. OrPoNet performance on function approximation

This section contains simulation results for section 4.1.

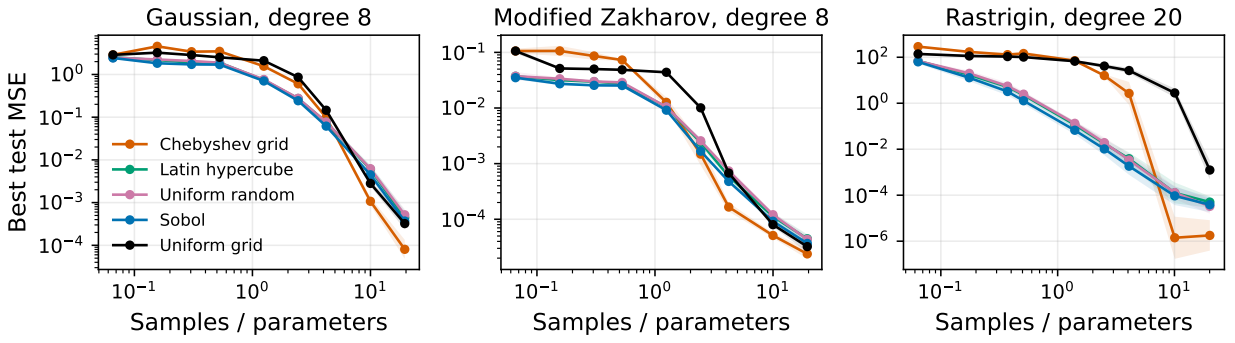


Figure D.1: Comparison of different sampling strategies for training residual OrPoNet. The test MSE is plotted against the number of training samples for each sampling method. The results indicate that Sobol sampling is more favorable with low sample/parameter ratios and Chebyshev collocation points become more effective as the number of samples increases. The results are obtained by averaging runs with five random seeds.

D.1. Sampling Strategies

We evaluate the effect of training-point sampling on residual OrPoNet using three 3D benchmark functions: Gaussian, Modified Zakharov, and Rastrigin. The comparison is performed in 3D because grids become prohibitively expensive in higher dimensions, while 3D still allows direct comparison between grid-based and non-grid sampling strategies. Gaussian and Modified Zakharov are approximated with polynomial degree 8 while Rastrigin is approximated with degree 20. To make the comparison across these settings meaningful, the horizontal axis is the sample-to-parameter ratio, N/P, where N is the number of training samples and P is the number of trainable parameters.

We compare five sampling strategies: uniform grid, Chebyshev grid, independent uniform random sampling, Latin hypercube sampling, and Sobol sampling. Each configuration is repeated with five seeds, and all methods are evaluated on the same fixed Sobol test set of $20,000$ points. The model is trained for 100 epochs with AdamW, and performance is reported as the best raw-scale test MSE over training. For plotting, repeated runs are aggregated using the geometric mean of the best test MSE, with uncertainty bands computed as one standard deviation. As shown in Fig. D.1, the results indicate that Sobol sampling is more efficient with low sample/parameter ratios, while Chebyshev collocation points become more effective as the number of samples increases.

D.2. Effect of polynomial orders

We demonstrate the effect of polynomial order on the convergence of OrPoNet in approximating smooth functions in Fig. D.2. OrPoNet exhibits approximately exponential error decay over a range of polynomial orders for the tested functions. However, we also observe that beyond a certain polynomial order, the convergence trend slows down, which may reflect optimization difficulties. This highlights the importance of selecting an appropriate polynomial order based on the complexity of the target function and the optimization landscape.

D.3. Number of copies

Figure D.3 exhibits the effect of the number of copies on the approximation accuracy of OrPoNet. Increasing the number of copies improves the accuracy of the vanilla architecture by providing additional parallel polynomial paths. In contrast, residual OrPoNet achieves substantially lower errors with fewer copies. This means that these architectural extensions increase the model's expressive capacity and, therefore, reduce the need for multiple repeated copy channels.

D.4. Approximating nonsmooth functions

We demonstrate the performance of OrPoNet in approximating non-smooth functions $f_1(x, y) = |x| + |y|$ and $f_2(x, y) = \text{sign}(x) + 0.25y$. The first of these has a corner singularity at the origin, and the second contains a jump discontinuity across $x = 0$. For each target function, we train a Chebyshev-basis OrPoNet with polynomial orders $4, 8, 16, 32$ in each input direction. The training set contains $2,048$ uniformly sampled points from $[-1, 1]^2$, and the test set contains $4,096$ independent uniformly sampled points. The same test set is used across all orders to make the test

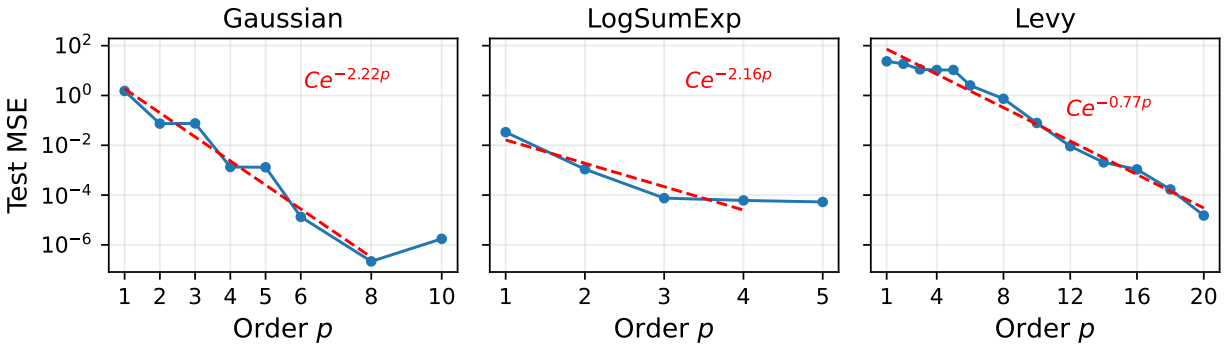


Figure D.2: Approximately exponential error decay of OrPoNet over a range of polynomial orders for the tested smooth functions. The network is able to maintain the exponential convergence trend up to a certain order, beyond which the convergence slows down. We speculate that this is due to optimization challenges. The red dashed line indicates the fitted exponential trend.

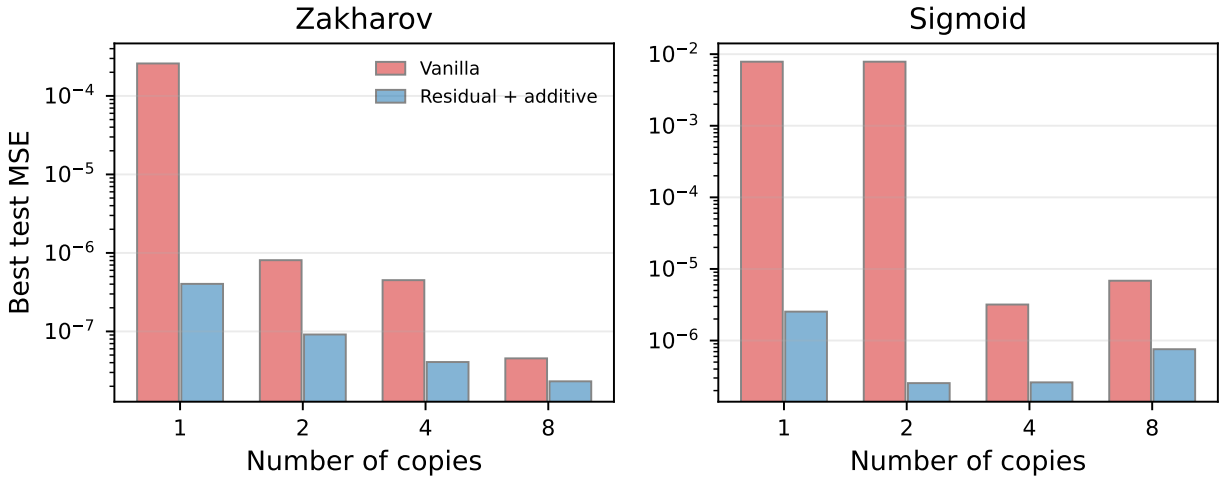


Figure D.3: Effect of the number of copies on the best test MSE for the vanilla and residual OrPoNet architectures. Residual OrPoNet achieves lower errors with fewer copies, while the vanilla architecture benefits more directly from increasing the number of copies.

errors directly comparable. Each model is optimized in two stages: we first run AdamW for 800 epochs and then apply full-batch L-BFGS for 25 refinement steps.

Figure D.4 shows that OrPoNet captures the overall shape of the target functions, but the approximation quality is limited by the non-smoothness. Increasing the polynomial order improves the fit initially, but the test MSE flattens or even worsens as the polynomial network grows. This behavior is expected, as polynomial approximations are known to struggle with non-smooth features due to such phenomena as Gibbs oscillations near discontinuities and slow convergence rates for functions with singularities [11]. This suggests that OrPoNet is less effective in approximating non-smooth functions.

D.5. Function approximation benchmark configurations

Data generation and training. We use Sobol sequences to generate training, validation, and test sets, each containing $200d$ points (Table D.5). For each experiment, we train the three networks (OrPoNet, MLP, KAN) using AdamW for 500 epochs, followed by 20 steps of L-BFGS. We set AdamW with a learning rate of 10^{-3} at 10D and 20D and 10^{-4}

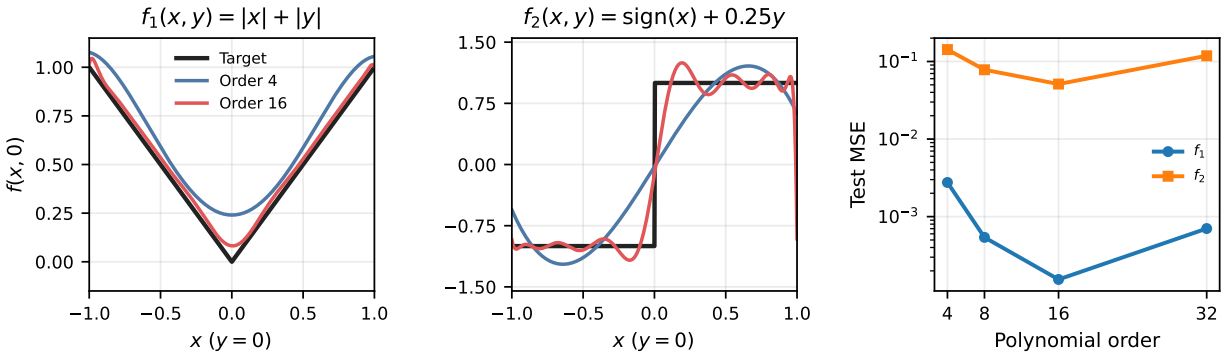


Figure D.4: Examples of nonsmooth functions approximated by OrPoNet. The first two panels show the fitted functions on the slice $y = 0$ using Chebyshev orders 4 and 16, and the right panel illustrates the test MSE for all tested orders (4, 8, 16, 32). Increasing the order improves the fit initially, but the nonsmooth cases can flatten or worsen as the polynomial network grows, indicating that OrPoNet is less economical for these boundary cases than for smooth polynomial-like functions.

Table D.1
Hyperparameter search spaces for Residual OrPoNet, MLP and KAN in Fig. 5 at $d = 10, 20, 50, 100$.

Model	Width / polynomial order space	Depth	Other settings
Residual OrPoNet	$p \in \{2, 4, 6, 8, 10, 12, 16\}$	d	Chebyshev basis; one copy; additive branch; no gate
MLP	Width $\in \{8, 16, 32, 64, 128\}$	$\{1, 2, 3, 4\}$	GELU, tanh, or sigmoid activation
KAN	Width $\in \{1, 2, 3, 5, 8, 10\}$	$\{1, 2\}$	Cubic splines, SiLU base function; final grid $\in \{3, 5, 10, 20\}$

at 50D and 100D, weight decay 10^{-5}, and batch sizes 64, 128, 256, and 512, respectively. For KAN, we adopt the grid-refinement strategy described in [32], where the grid is refined and AdamW is restarted after evenly spaced epochs. At last, KAN receives the same 20-step L-BFGS phase.

Benchmark configurations for Fig. 5. We use one-copy, Chebyshev-basis residual OrPoNet throughout the experiments. The residual gate is disabled as it only marginally improves performance in the benchmark functions. As the depth of OrPoNet is fixed and equivalent to the number of input dimensions, we modulate its width by varying the polynomial order $p \in \{2, 4, 6, 8, 10, 12, 16\}$ to produce seven candidate configurations, as listed in Table D.1. In comparison, MLP and KAN have more hyperparameters to tune, including depth, width, and other network-specific settings. For MLP, we vary network depth from 1 to 4, width is sampled from $\{8, 16, 32, 64, 128\}$, and the activation function is chosen from $\{\tanh, \mathrm{sigmoid}, \mathrm{GELU}\}$. For KAN, we choose depth from 1 to 2, width from $\{1, 2, 3, 5, 8, 10\}$, and final grid size from $\{3, 5, 10, 20\}$.

To control the tuning budget, we use seven configurations for each method. For MLP and KAN, we independently draw seven distinct joint configurations uniformly without replacement from the respective spaces. OrPoNet instead enumerates its seven prescribed polynomial orders. This approach is efficient for hyperparameter tuning, because NNs are often sensitive to only a subset of hyperparameters [51]. The candidate lists are fixed before training and shared across all functions and dimensions. For each function, dimension, and method, we first train each of the seven candidates once. The two candidates with the lowest validation errors are each evaluated in two additional training runs. We then select the configuration with the lowest mean validation error across the three runs and report its test errors. This gives 672 screening runs and 384 additional training runs, for a total of 1056 runs. The final selected configurations for each function and dimension are provided in Table D.2.

Benchmark configurations for Fig. 6. We use the same training settings as in Fig. 5, but construct candidate architectures to approximately match five parameter-count targets defined by OrPoNet orders $p \in \{2, 4, 6, 8, 12\}$ at

Table D.2
Selected architectures for Fig. 5. A dash in the d column indicates the same dimension as the preceding row.

Function	d	OrPoNet order p	MLP (depth, width)	MLP activation	KAN (depth, width, grid)
Gaussian	10	6	$(3, 16)$	GELU	$(2, 1, 5)$
Modified Rosenbrock	-	6	$(3, 32)$	tanh	$(1, 10, 3)$
Modified Zakharov	-	2	$(3, 16)$	GELU	$(2, 8, 3)$
Log-Sum-Exp	-	4	$(3, 16)$	GELU	$(2, 1, 5)$
Sigmoid	-	4	$(2, 32)$	sigmoid	$(1, 10, 3)$
Griewank	-	6	$(3, 16)$	GELU	$(2, 5, 5)$
Rastrigin	-	10	$(2, 8)$	GELU	$(1, 10, 20)$
Levy	-	8	$(3, 16)$	GELU	$(2, 1, 5)$
Gaussian	20	6	$(4, 32)$	sigmoid	$(2, 8, 3)$
Modified Rosenbrock	-	6	$(3, 32)$	tanh	$(1, 10, 3)$
Modified Zakharov	-	2	$(3, 32)$	tanh	$(2, 8, 3)$
Log-Sum-Exp	-	6	$(3, 32)$	tanh	$(2, 1, 5)$
Sigmoid	-	4	$(3, 16)$	GELU	$(2, 1, 5)$
Griewank	-	6	$(2, 32)$	sigmoid	$(2, 5, 5)$
Rastrigin	-	10	$(3, 16)$	GELU	$(1, 10, 20)$
Levy	-	8	$(4, 32)$	sigmoid	$(2, 1, 5)$
Gaussian	50	4	$(3, 32)$	tanh	$(2, 1, 5)$
Modified Rosenbrock	-	4	$(2, 32)$	sigmoid	$(2, 8, 3)$
Modified Zakharov	-	2	$(3, 32)$	tanh	$(2, 8, 3)$
Log-Sum-Exp	-	6	$(2, 32)$	sigmoid	$(2, 1, 5)$
Sigmoid	-	2	$(2, 8)$	GELU	$(2, 1, 5)$
Griewank	-	6	$(3, 16)$	GELU	$(2, 5, 5)$
Rastrigin	-	10	$(3, 32)$	tanh	$(1, 5, 20)$
Levy	-	6	$(2, 32)$	sigmoid	$(2, 1, 5)$
Gaussian	100	4	$(3, 16)$	GELU	$(2, 1, 5)$
Modified Rosenbrock	-	2	$(3, 32)$	tanh	$(2, 1, 5)$
Modified Zakharov	-	4	$(3, 16)$	GELU	$(2, 8, 3)$
Log-Sum-Exp	-	6	$(3, 32)$	tanh	$(1, 10, 3)$
Sigmoid	-	2	$(2, 8)$	GELU	$(2, 1, 5)$
Griewank	-	4	$(3, 16)$	GELU	$(2, 5, 5)$
Rastrigin	-	10	$(4, 32)$	sigmoid	$(1, 10, 20)$
Levy	-	6	$(3, 32)$	tanh	$(2, 1, 5)$

each of $d = 10$ and $d = 100$. OrPoNet uses one copy, the same order in every layer, residual connections, and additive branches, with no residual gate.

MLP depth varies in $\{1, 2, 3, 4\}$ and KAN depth in $\{1, 2\}$. MLP activation functions are chosen from $\{\text{GELU}, \tanh, \text{sigmoid}\}$, and the final KAN grid size varies in $\{3, 5, 10, 20\}$. Widths are chosen so that each baseline's trainable parameter count is within $\pm 5\%$ of the OrPoNet target. Each target is shared across all eight functions at a given dimension. We enumerate the MLP depths and KAN depth-grid combinations in the search space and compute the corresponding widths that yield parameter counts within the tolerance. We select up to three candidate architectures per baseline and evaluate each MLP architecture with GELU, tanh, and sigmoid activations, giving nine MLP candidate configurations per target. For each function, dimension, and target, we select the MLP and KAN configurations with the lowest validation errors in the initial run and evaluate them in two additional training runs. Tables D.3 and D.4 list the configurations and trainable parameter counts for 10D and 100D, respectively.

Table D.3
OrPoNet configurations and selected MLP and KAN configurations for the parameter-matched comparison in Fig. 6 at $d = 10$. Here, p denotes the OrPoNet polynomial order and N the trainable parameter count.

Function	OrPoNet		MLP			KAN	
	p	N	Depth, width	Activation	N	Depth, width, grid	N
Gaussian	2	292	$(2, 12)$	GELU	301	$(2, 1, 20)$	300
	4	654	$(3, 15)$	GELU	661	$(1, 4, 10)$	660
	6	1,160	$(3, 21)$	GELU	1,177	$(2, 8, 3)$	1,216
	8	1,810	$(4, 22)$	GELU	1,783	$(2, 9, 5)$	1,800
	12	3,542	$(4, 32)$	GELU	3,553	$(2, 16, 3)$	3,456
Modified Rosenbrock	2	292	$(1, 24)$	GELU	289	$(2, 1, 20)$	300
	4	654	$(1, 54)$	GELU	649	$(1, 6, 5)$	660
	6	1,160	$(1, 97)$	GELU	1,165	$(1, 11, 5)$	1,210
	8	1,810	$(1, 151)$	GELU	1,813	$(1, 21, 3)$	1,848
	12	3,542	$(1, 295)$	GELU	3,541	$(2, 16, 3)$	3,456
Modified Zakharov	2	292	$(2, 12)$	GELU	301	$(2, 1, 20)$	300
	4	654	$(3, 15)$	GELU	661	$(1, 6, 5)$	660
	6	1,160	$(3, 21)$	GELU	1,177	$(2, 8, 3)$	1,216
	8	1,810	$(4, 22)$	GELU	1,783	$(1, 21, 3)$	1,848
	12	3,542	$(4, 32)$	GELU	3,553	$(1, 32, 5)$	3,520
Log-Sum-Exp	2	292	$(2, 12)$	GELU	301	$(2, 1, 20)$	300
	4	654	$(3, 15)$	GELU	661	$(1, 6, 5)$	660
	6	1,160	$(2, 28)$	GELU	1,149	$(2, 8, 3)$	1,216
	8	1,810	$(4, 22)$	GELU	1,783	$(2, 9, 5)$	1,800
	12	3,542	$(4, 32)$	GELU	3,553	$(1, 32, 5)$	3,520
Sigmoid	2	292	$(1, 24)$	tanh	289	$(2, 1, 20)$	300
	4	654	$(1, 54)$	tanh	649	$(1, 6, 5)$	660
	6	1,160	$(2, 28)$	sigmoid	1,149	$(2, 8, 3)$	1,216
	8	1,810	$(4, 22)$	sigmoid	1,783	$(1, 21, 3)$	1,848
	12	3,542	$(2, 53)$	sigmoid	3,499	$(2, 16, 3)$	3,456
Griewank	2	292	$(2, 12)$	GELU	301	$(2, 1, 20)$	300
	4	654	$(3, 15)$	GELU	661	$(1, 4, 10)$	660
	6	1,160	$(2, 28)$	GELU	1,149	$(1, 7, 10)$	1,155
	8	1,810	$(4, 22)$	GELU	1,783	$(2, 9, 5)$	1,800
	12	3,542	$(4, 32)$	GELU	3,553	$(2, 16, 3)$	3,456
Rastrigin	2	292	$(1, 24)$	GELU	289	$(2, 1, 20)$	300
	4	654	$(1, 54)$	GELU	649	$(1, 4, 10)$	660
	6	1,160	$(1, 97)$	GELU	1,165	$(1, 7, 10)$	1,155
	8	1,810	$(1, 151)$	GELU	1,813	$(1, 11, 10)$	1,815
	12	3,542	$(1, 295)$	GELU	3,541	$(1, 13, 20)$	3,575
Levy	2	292	$(1, 24)$	GELU	289	$(2, 1, 20)$	300
	4	654	$(1, 54)$	GELU	649	$(1, 4, 10)$	660
	6	1,160	$(1, 97)$	GELU	1,165	$(1, 11, 5)$	1,210
	8	1,810	$(1, 151)$	GELU	1,813	$(1, 11, 10)$	1,815
	12	3,542	$(1, 295)$	GELU	3,541	$(1, 32, 5)$	3,520

Table D.4
OrPoNet configurations and selected MLP and KAN configurations for the parameter-matched comparison in Fig. 6 at $d = 100$. Here, p denotes the OrPoNet polynomial order and N the trainable parameter count.

Function	OrPoNet		MLP			KAN	
	p	N	Depth, width	Activation	N	Depth, width, grid	N
Gaussian	2	3,172	$(2, 25)$	GELU	3,201	$(1, 4, 3)$	3,232
	4	7,134	$(1, 70)$	GELU	7,141	$(2, 8, 3)$	6,976
	6	12,680	$(1, 124)$	GELU	12,649	$(2, 14, 3)$	12,880
	8	19,810	$(1, 194)$	GELU	19,789	$(2, 17, 5)$	20,060
	12	38,822	$(3, 116)$	GELU	38,977	$(2, 21, 10)$	38,430
Modified Rosenbrock	2	3,172	$(1, 31)$	GELU	3,163	$(2, 3, 5)$	3,120
	4	7,134	$(1, 70)$	GELU	7,141	$(1, 7, 5)$	7,070
	6	12,680	$(1, 124)$	GELU	12,649	$(2, 11, 5)$	12,320
	8	19,810	$(1, 194)$	GELU	19,789	$(2, 17, 5)$	20,060
	12	38,822	$(2, 152)$	GELU	38,761	$(1, 38, 5)$	38,380
Modified Zakharov	2	3,172	$(2, 25)$	GELU	3,201	$(1, 4, 3)$	3,232
	4	7,134	$(1, 70)$	GELU	7,141	$(2, 8, 3)$	6,976
	6	12,680	$(1, 124)$	GELU	12,649	$(2, 11, 5)$	12,320
	8	19,810	$(1, 194)$	GELU	19,789	$(2, 17, 5)$	20,060
	12	38,822	$(3, 116)$	GELU	38,977	$(1, 38, 5)$	38,380
Log-Sum-Exp	2	3,172	$(1, 31)$	GELU	3,163	$(1, 4, 3)$	3,232
	4	7,134	$(1, 70)$	GELU	7,141	$(1, 7, 5)$	7,070
	6	12,680	$(1, 124)$	GELU	12,649	$(2, 14, 3)$	12,880
	8	19,810	$(1, 194)$	GELU	19,789	$(1, 13, 10)$	19,695
	12	38,822	$(2, 152)$	GELU	38,761	$(1, 38, 5)$	38,380
Sigmoid	2	3,172	$(2, 25)$	GELU	3,201	$(1, 4, 3)$	3,232
	4	7,134	$(3, 39)$	GELU	7,099	$(2, 8, 3)$	6,976
	6	12,680	$(4, 50)$	GELU	12,751	$(2, 14, 3)$	12,880
	8	19,810	$(3, 77)$	GELU	19,867	$(2, 17, 5)$	20,060
	12	38,822	$(3, 116)$	GELU	38,977	$(1, 48, 3)$	38,784
Griewank	2	3,172	$(3, 22)$	GELU	3,257	$(2, 3, 5)$	3,120
	4	7,134	$(3, 39)$	GELU	7,099	$(1, 7, 5)$	7,070
	6	12,680	$(4, 50)$	GELU	12,751	$(2, 11, 5)$	12,320
	8	19,810	$(3, 77)$	GELU	19,867	$(2, 17, 5)$	20,060
	12	38,822	$(3, 116)$	GELU	38,977	$(2, 21, 10)$	38,430
Rastrigin	2	3,172	$(3, 22)$	GELU	3,257	$(2, 2, 10)$	3,090
	4	7,134	$(3, 39)$	GELU	7,099	$(1, 7, 5)$	7,070
	6	12,680	$(2, 72)$	GELU	12,601	$(1, 5, 20)$	12,625
	8	19,810	$(2, 98)$	GELU	19,699	$(1, 13, 10)$	19,695
	12	38,822	$(3, 116)$	GELU	38,977	$(2, 21, 10)$	38,430
Levy	2	3,172	$(1, 31)$	GELU	3,163	$(2, 3, 5)$	3,120
	4	7,134	$(1, 70)$	GELU	7,141	$(1, 7, 5)$	7,070
	6	12,680	$(1, 124)$	GELU	12,649	$(2, 11, 5)$	12,320
	8	19,810	$(1, 194)$	GELU	19,789	$(2, 17, 5)$	20,060
	12	38,822	$(2, 152)$	GELU	38,761	$(1, 38, 5)$	38,380

Table D.5
Numbers of points in each independent Sobol split. The supplementary parameter-matched experiment uses the 10D and 100D rows, with a separate fixed test scramble.

Dimension	Training size	Validation size	Test size
10D	2,000	2,000	2,000
20D	4,000	4,000	4,000
50D	10,000	10,000	10,000
100D	20,000	20,000	20,000

D.6. Bratley Sum benchmark

We use the dataset sizes and polynomial basis from [33]. Specifically, we sample 8,000 training points, 2,000 validation points, and 2,000 test points for each input dimension $d \in [100, 250, 500, 750, 1000, 2000]$. Following the DeepPCE benchmark setup, we use degree-6 Hermite polynomial features with one copy for OrPoNet so that the comparison uses the same input basis as the reported DeepPCE results. We train the one-copy residual OrPoNets using mini-batch AdamW for up to 500 epochs in each of 10 runs with distinct random seeds. The learning rate is fixed at 10^{-3} for all dimensions in the full sweep, and batch size is 256. We train the network using early stopping with a 20-epoch period and a patience setting of 10. Table D.6 lists the test relative MSE values of the 10 repeated OrPoNet runs used in Fig. 7.

Table D.6
Test relative MSE values for the 10 repeated OrPoNet runs on the Bratley Sum benchmark.

Run	$d = 100$	$d = 250$	$d = 500$	$d = 750$	$d = 1000$	$d = 2000$
1	1.85×10^{-4}	3.21×10^{-4}	1.49×10^{-3}	1.04×10^{-2}	1.01×10^{-2}	1.65×10^{-2}
2	1.52×10^{-4}	9.69×10^{-4}	1.43×10^{-3}	1.52×10^{-2}	1.14×10^{-2}	1.49×10^{-2}
3	1.73×10^{-4}	3.31×10^{-3}	2.62×10^{-3}	1.23×10^{-2}	6.06×10^{-3}	2.44×10^{-2}
4	1.53×10^{-4}	1.21×10^{-3}	1.05×10^{-3}	1.12×10^{-2}	1.11×10^{-2}	1.53×10^{-2}
5	1.69×10^{-4}	3.08×10^{-3}	5.94×10^{-3}	1.09×10^{-2}	1.23×10^{-2}	1.74×10^{-2}
6	1.49×10^{-4}	6.82×10^{-4}	1.91×10^{-3}	9.86×10^{-3}	9.66×10^{-3}	3.00×10^{-2}
7	3.57×10^{-4}	1.89×10^{-3}	2.97×10^{-3}	1.10×10^{-2}	1.39×10^{-2}	1.77×10^{-2}
8	1.50×10^{-4}	4.02×10^{-3}	5.24×10^{-4}	1.37×10^{-2}	8.41×10^{-3}	1.48×10^{-2}
9	1.89×10^{-4}	1.01×10^{-3}	7.17×10^{-4}	1.18×10^{-2}	9.56×10^{-3}	2.16×10^{-2}
10	4.05×10^{-4}	1.05×10^{-3}	3.87×10^{-3}	1.40×10^{-2}	1.28×10^{-2}	1.34×10^{-2}

D.7. Interpretability of OrPoNet

For testing purposes, we uniformly sample $2^{12} = 4,096$ points from the domain $[-1, 1]^3$ and use them to generate output data. We train an OrPoNet with fourth-order Legendre polynomials using the AdamW optimizer with a learning rate of $5 \cdot 10^{-3}$ and weight decay of 10^{-6} to encourage sparsity. After training, we reconstruct the Legendre expansion and prune negligible terms with magnitudes less than 10^{-3}. As presented in Fig. D.5, the recovered coefficients precisely match their ground-truth counterparts, indicating that OrPoNet can be used to discover the series expansion form of input-output relations. This can be particularly useful when the exact symbolic form is too complex to be obtained from symbolic regression. This example focuses on a low-dimensional vanilla OrPoNet, where the induced polynomial expansion can be reconstructed directly. Scaling such coefficient-level discovery to high-dimensional settings remains challenging: although residual branches improve trainability, they can also introduce redundant expansion terms, and the current soft-gate training does not reliably remove them. We therefore leave high-dimensional sparse polynomial discovery with improved sparsity-promoting gates or regularization for future work.

D.8. Ablation study

We tabulate the results of ablation study for the basis-aware initialization and architectural extensions in Tab. D.7 and Tab. D.8, respectively.

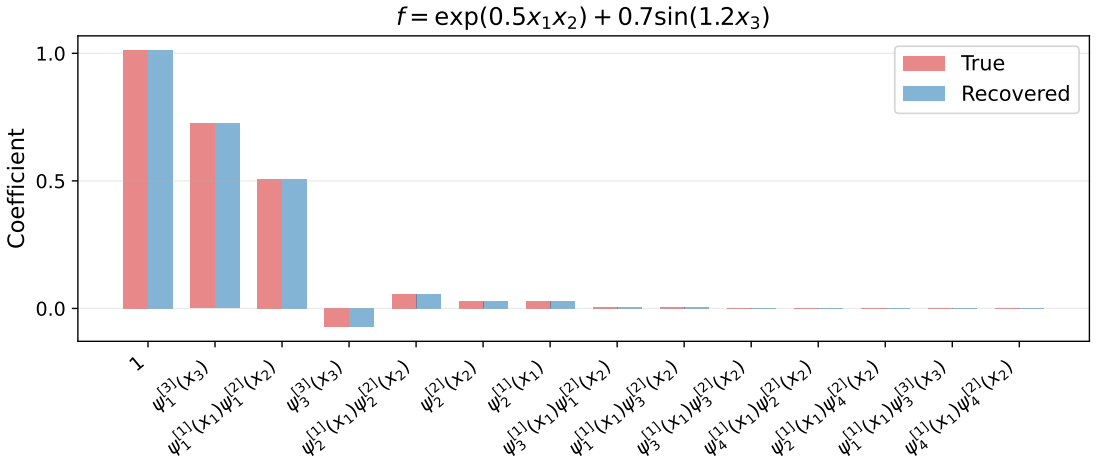


Figure D.5: OrPoNet precisely recovers the dominant Legendre polynomial terms of the approximated functions from the trained network.

Table D.7
Effect of basis-aware initialization on initial output and gradient scales. The test is performed on the modified Zakharov function. Random initialization uses zero-mean uniform weights with standard deviation 0.01, and basis-aware initialization uses gain 1.0. The basis-aware initialization avoids the vanishing output and gradient problem at initialization.

	Initial output RMS		Initial minimum gradient norm	
Dimension	Random init	Basis-aware init	Random init	Basis-aware init
5	3.03×10^{-9}	4.61×10^{-1}	3.25×10^{-8}	1.93×10^{-1}
10	8.22×10^{-18}	8.31×10^{-1}	5.33×10^{-17}	4.27×10^{-1}
20	0.00	8.77×10^{-1}	0.00	8.82×10^{-1}

Table D.8
Architecture ablation on the 20D modified Zakharov function. The OrPoNet adopts Chebyshev bases with one copy and up to degree 4. The combination of basis-aware initialization, residual connection, and additive branch achieves the best performance. On this example, the gated residual connection provides only marginal improvement over the standard residual connection.

Architecture	Best relative L_2	Best MSE
Basis-aware Init	1.98×10^{-1}	4.69×10^{-3}
Basis-aware Init + Residual connection	1.58×10^{-2}	3.01×10^{-5}
Basis-aware Init + Gated residual connection	1.61×10^{-2}	3.12×10^{-5}
Basis-aware Init + Residual + additive branch	1.75×10^{-3}	3.67×10^{-7}
Basis-aware Init + Gated residual + additive branch	1.37×10^{-3}	2.26×10^{-7}

D.9. Sensitivity to input coordinate ordering

To examine sensitivity to input coordinate ordering, we use the 10D modified Rosenbrock function defined in Table 1, as Rosenbrock functions have also been used to investigate mode ordering in a recent study [52].

We use residual OrPoNet with degree-four Chebyshev polynomials and one copy. We follow the data-generation and per-configuration training protocol for the 10D experiments in Fig. 5 (see Appendix D.5 for detail). We use the same network and training configurations for two experiments with reordered inputs. The training, validation, and test sets each contain 2,000 samples. Table D.9 shows that separating interacting variables substantially increases the approximation error, indicating sensitivity to input coordinate ordering at fixed model capacity.

Table D.9
Effect of input coordinate ordering on the 10D modified Rosenbrock function.

Ordering	Input sequence	Test MSE
Adjacent	$(x_1, x_2, \dots, x_9, x_{10})$	1.65×10^{-2}
Separated	$(x_1, x_3, \dots, x_9, x_2, x_4, \dots, x_{10})$	2.04

We further examine whether additional constant basis channels can mitigate this sensitivity. We add $k \in \{0, 1, 2, 4, 8\}$ constant entries per layer while keeping the polynomial degree and copy number fixed at four and one. Table D.10 shows that adding constant channels substantially reduces the test MSE for both the adjacent and separated orderings. With eight additional constants, the test MSE is 1.33×10^{-5} for the adjacent ordering and 5.72×10^{-6} for the separated ordering.

Table D.10
Effect of additional constant basis channels on the test MSE for the 10D modified Rosenbrock function with different input orderings.

Added channels (k)	Adjacent	Separated
0	1.65×10^{-2}	2.04
1	4.46×10^{-3}	1.45
2	4.58×10^{-4}	8.31×10^{-1}
4	2.37×10^{-4}	2.53×10^{-2}
8	1.33×10^{-5}	5.72×10^{-6}