IKNO: Infinite-order Kernel Neural Operators

Pengyuan Zhu [1 2] **Ivor W. Tsang** [1 2] **Yueming Lyu** [† 1]

Abstract

Neural operators have achieved significant success in modern scientific computing due to their flexibility and strong generalization capabilities. Existing models, however, primarily rely on first-order kernel integral approximations, which severely limit their expressivity. To address this, we propose the Infinite-order Kernel Neural Operator (IKNO), which constructs neural operators via infinite-order kernel integrals and admits an elegant closed-form finite approximation. We develop two complementary infinite-order neural operator constructions: IKNO-Vanilla, which applies the full-kernel resolvent on the product grid via Kronecker eigendecomposition, and IKNO-TP, an alternative tensor-product operator that composes per-axis resolvents. Furthermore, we develop fast computation schemes for both variants of IKNO, which achieve outstanding global information aggregation while maintaining high computational efficiency. Empirically, we evaluate our IKNO on both time-dependent and time-independent benchmarks with arbitrary input shapes, including large-scale industrial datasets. Extensive experiments demonstrate that the IKNO method consistently achieves the SOTA accuracy with significant improvements on nearly all benchmark datasets while maintaining scalability to very large point clouds.

1. Introduction

Partial differential equations serve as fundamental tools for describing how properties evolve over time and space, finding applications across physics, finance, and numerous other fields(Evans, 2022). For instance, weather and sea condition forecasts rely on solving the NS equations(Vallis, 2017). In finance, pricing many financial products requires solving the Black-Scholes equation(Black & Scholes, 1973). In medicine, partial differential equations are used to describe drug diffusion within tissues(Murray, 2007).

Traditional methods for solving partial differential equations involve numerical simulation techniques such as FEM(Hughes, 2003), FDM(Thomas, 2013), and others. The issue with these methods lies in their relatively slow computational speed. Particularly in industrial design, computational complexity increases exponentially(Hughes, 2003) as mesh resolution is refined, making high-fidelity simulations prohibitively expensive. Furthermore, these approaches can only determine performance metrics after a shape is designed. However, the goal of industrial design is to derive shapes with better performance. Traditional PDE solvers fall far short of meeting the demands of the industrial design field.

In recent years, to overcome the limitations of traditional numerical methods in computational efficiency and generalization capability, data-driven deep learning PDE solvers have emerged. Within this field, Neural Operators(Li et al., 2020a) stand out as a highly promising paradigm. Unlike conventional deep learning models, neural operators aim to learn nonlinear mappings between infinite-dimensional function spaces, thereby offering a novel approach to achieving efficient and high-precision PDE solutions.

Although existing neural operator models achieve considerable progress in solving PDEs on arbitrary point clouds, they primarily rely on first-order kernel integral approximations(Li et al., 2020b)(Li et al., 2023b)(Wen et al., 2025)(Brandstetter et al., 2022), which severely limit their expressivity. Our preliminary experiment on *Poisson-C-Sines* shows a clear monotonic decrease in error as the finite-order kernel propagation order increases from 1 to 4 (Table 1). This preliminary experiment demonstrates that higher-order kernel integral propagation can aggregate and summarize global information more effectively, leading to better representations and enhanced model performance. These insights suggest that neural operators may be significantly enhanced by incorporating higher-order kernel integrals. Consequently, we investigate a pivotal question: *can we formulate an infinite-order kernel operator to achieve*

[1]Centre for Frontier AI Research(CFAR), and Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore [2]Nanyang Technological University. Correspondence to: Yueming Lyu <Lyu_Yueming@a-star.edu.sg>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

comprehensive information aggregation over all propagation orders? To answer this question, we propose a novel neural operator model, the Infinite-order Kernel Neural Operator (IKNO), which improves expressivity via infinite-order kernel integrals that benefit from superior global information aggregation and propagation.

Our main contributions are summarized as follows:

- We propose a novel Infinite-order Kernel Neural Operator (IKNO), which constructs neural operators via infinite-order kernel integrals that benefit from superior global information aggregation and propagation. Moreover, we derive an elegant closed-form finite approximation for the infinite-order kernel integral operators.
- We introduce two complementary multi-dimensional infinite-order neural operator constructions: IKNO-Vanilla, which applies the full-kernel resolvent on the product grid via Kronecker eigendecomposition, and IKNO-TP, an alternative tensor-product operator that composes per-axis resolvents and thereby induces an implicit functional regularization. These two constructions are not algebraically identical, but both admit Kronecker-structured fast computation that reduces the dominant preprocessing cost from $O(N^{3d})$ to $O(d \cdot N^3)$ with essentially identical wall-clock cost, enabling near-instantaneous computation of global dependencies on high-resolution grids.
- Empirically, our IKNO method consistently achieves state-of-the-art accuracy with significant improvements over the previous SOTA (GAOT (Wen et al., 2025)) on the large majority of PDE benchmarks while maintaining scalability to very large point clouds. Across 15 benchmarks, the two variants exhibit complementary strengths, with IKNO-TP providing the strongest overall performance and IKNO-Vanilla offering a competitive non-tensorized alternative across diverse PDE settings.

2. Related work

Early studies on neural operator learning primarily focused on regular grids. The most representative among them is the Fourier Neural Operator(Li et al., 2020a), which achieves exceptional computational efficiency by parameterizing integral kernels in the frequency domain and utilizing FFT. Furthermore, to better capture features across different spatial scales in physical fields, U-NO(Rahman et al., 2022) with multi-scale collaborative structure of U-Net was proposed. However, due to their underlying design, these models are highly dependent on regular grids. Consequently, they are only applicable to uniformly discretized regular grid datasets, which greatly limits their application in complex geometric domains.

Subsequently, researchers sought to extend the advantages of neural operator to non-regular spaces and larger-scale datasets to achieve better generalization and practicality. However, when addressing practical engineering problems, the input data mesh often comprises a vast number of points. Learning Neural Operator directly in the original high-resolution space would bring memory overhead and computational complexity. To address this challenge, recent neural operator models typically adopt a strategy of mapping large-scale unstructured data into a compact latent space via an encoder(Wang & Wang, 2024; Wu et al., 2024; Luo et al., 2025). By decoupling inputs' resolution from model's internal resolution, the trained neural operators can maintain strong generalization capabilities across varying resolutions and sampling methods. Based on the different methods of latent space representation, two mainstream approaches have emerged in research area.

2.1. Latent grid-based mapping

The core idea of this approach is to encode unstructured input data onto a regular latent grid. The regular grid inherently preserves geometric information, enabling the model to implicitly encode spatial structure. Operators dependent on the grid can also be directly applied in the latent space, such as convolutions, Fourier transforms, and multi-scale operators. Geo-FNO(Li et al., 2023a) maps input points into a regular grid via a reversible transformation, successfully extending the FNO(Li et al., 2020a) model to irregular domains. GINO(Li et al., 2023b) achieves mapping from physical coordinates to a latent grid space using a learnable first-order kernel function, then processes this grid space using the FNO(Li et al., 2020a) model architecture. Li et al. (Li et al., 2025) employ optimal transport to map inputs and outputs onto a latent mesh space, enabling efficient neural operator learning within this latent domain. GAOT(Wen et al., 2025) utilizes a multi-scale encoder from GINO(Li et al., 2023b) with learnable weights, supplemented by artificially designed geometric embeddings, to map inputs onto the latent grid space. Within this space, a ViT(Dosovitskiy, 2020) model is employed to efficiently learn the latent neural operator.

2.2. Latent token-based mapping

Latent token-based methods no longer explicitly map input functions to a regular grid. Instead, they employ learnable encoding mechanisms to compress raw inputs into a finite set of implicit token representations. These tokens serve as low-dimensional summaries of the input physics information or key structural features, regarded as fundamental units for subsequent neural operator computations. The encoder component has few constraints on the mapping, resulting in strong expressive capabilities, and it can be compatible with various geometric input formats such as point clouds,

SDF(Park et al., 2019), parametric curves, and meshes.

LNO(Wang & Wang, 2024) maps input coordinates and physical quantities to latent tokens via an encoder centered on a cross-attention(Vaswani et al., 2017) module. Transolver(Wu et al., 2024) weights all information through an MLP(Rumelhart et al., 1986) to map onto latent slices, incorporating a PhysicsAttention mechanism designed to aggregate information from multiple slices. Transolver++(Luo et al., 2025) further enhances this by adding constraints to the weighted mapping, enabling each slice to carry more diverse information and boosting the encoder's expressive ability. Both OFermer(Li et al., 2022) and GNOT(Hao et al., 2023) directly map input physical quantities to tokens via MLP. The former employs a special attention module resembling kernel functions, offering both Fourier-type and Galerkin-type(Galerkin, 1968) variants. The latter utilizes a specially designed Heterogeneous Normalized Cross-Attention module to encode and incorporate diverse input forms, such as shape and edge information, into the latent neural operator.

However, current latent grid-based methods lack sufficient expressive power, often requiring manual design of kernel function summation ranges(Wen et al., 2025). This leads to the loss of information at different scales and over long distances. Meanwhile, latent token-based methods struggle to learn geometric structural information. Too few constraints also make them overly sensitive to hyperparameters, prone to overfitting(Luo et al., 2025). To deal with these problems, inspired by the theory of integral equations, we introduce the infinite-order kernel integral. Instead of a single aggregation step, we consider the multiplier of all higher-order interactions. This approach allows the model to aggregate information across the entire input domain more effectively while simultaneously capturing geometric structure features.

3. Methodology

Problem setup. We study neural operator learning for PDE solution maps on arbitrary point clouds. For time-independent problems, the model learns an operator $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ mapping input conditions a to solutions u on Ω. For time-dependent problems, it learns a trajectory operator from the initial state and time-varying conditions to the solution trajectory over $\Omega_T = [0, T]$. In all cases, inputs and outputs are observed on point clouds; the detailed formulation is provided in Appendix A.

3.1. Motivation: from finite-order to infinite-order kernel propagation

Many recent neural operators for irregular or large-scale domains follow an encoder–processor–decoder structure (Li et al., 2023b; Wu et al., 2024; Wen et al., 2025; Mousavi et al., 2025): the encoder maps observations from the physical input domain into a latent representation, the processor performs global mixing in the latent space, and the decoder maps the processed latent representation back to arbitrary query locations. In a preliminary finite-order study on the *Poisson-C-Sines* benchmark, we instantiate this paradigm with a fixed latent grid $G = G_1 \times \cdots \times G_d$ that has the same spatial dimensionality as the input domain. Each axis contains L grid points, so the latent grid contains L^d points independent of the input point-cloud resolution.

We replace the conventional first-order kernel aggregation with a truncated kernel propagation operator on the latent grid,

$$\boldsymbol{R}_p(G) = \boldsymbol{I} + \alpha \boldsymbol{K}_{GG} + \cdots + \alpha^p \boldsymbol{K}_{GG}^p, \tag{1}$$

where $\boldsymbol{K}_{GG} \in \mathbb{R}^{L^d \times L^d}$ is the kernel Gram matrix over latent grid points and p is the propagation order. In this preliminary study, we use a fixed linear-window kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \max(1 - \|\boldsymbol{x} - \boldsymbol{y}\|/r, 0)$ with radius $r = 0.2$ and unit scale. Given input points $P = \{\boldsymbol{x}^i\}_{i=1}^{\bar{N}}$ with token matrix $\boldsymbol{V}_P$ and query points $Q = \{\boldsymbol{q}^r\}_{r=1}^{N_q}$, the finite-order encoder–processor–decoder pipeline is

$$\boldsymbol{V}_G = \boldsymbol{R}_p(G) \boldsymbol{K}_{GP} \boldsymbol{V}_P, \tag{2}$$

$$\boldsymbol{V}'_G = \mathrm{TransformerEncoder}(\boldsymbol{V}_G), \tag{3}$$

$$\boldsymbol{V}_Q = \boldsymbol{K}_{QG} \boldsymbol{R}_p(G) \boldsymbol{V}'_G, \tag{4}$$

where $\boldsymbol{K}_{GP}$ maps input points to the latent grid and $\boldsymbol{K}_{QG}$ maps latent-grid features to query points. We fix the propagation coefficient to $\alpha = -0.15$ and keep both the kernel scale and α non-learnable; additional notation details are provided in Appendix B.1.

The quantitative results are summarized in Table 1, and the corresponding error-field visualizations are shown in Fig. 1. The aggregate error decreases monotonically as the propagation order increases from $p = 1$ to $p = 4$, dropping from 2.49 to 2.13. The visualizations further show that higher orders not only reduce the error magnitude, but also shrink the spatial regions with large relative error. This controlled trend provides direct empirical evidence that higher-order kernel propagation improves global information aggregation. Rather than treating the truncation order as a manually selected hyperparameter, the infinite-order formulation below aggregates all propagation orders through a closed-form operator.

3.2. Infinite-order kernel integral

3.2.1. Definitions and discretization

One fundamental component of a Neural Operator is the first-order kernel integral operator, which computes embed-

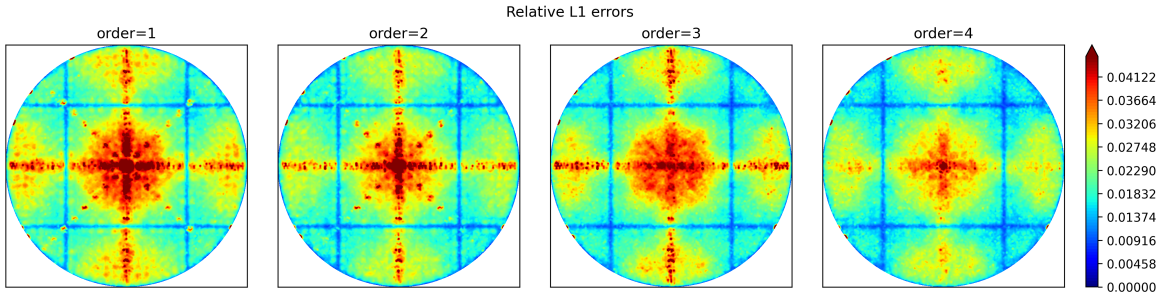

Figure 1. Visualization of relative L_1 error fields for finite-order kernel propagation on the *Poisson-C-Sines* benchmark. As the propagation order increases from $p = 1$ to $p = 4$, both the error magnitude and the spatial extent of high-error regions decrease.

Table 1. Preliminary finite-order kernel results on the *Poisson-C-Sines* benchmark using median relative L_1 error (%), with a fixed linear-window kernel ($r = 0.2$, unit scale, $\alpha = -0.15$).

Order p	**Median Relative** L_1 **Error** (%)
1	2.49
2	2.32
3	2.25
4	2.13

ding aggregation by considering global information:

$$\mathcal{I}[v](\boldsymbol{x}) = \int_{\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{y}) v(\boldsymbol{y}) \mathrm{d}\boldsymbol{y} \tag{5}$$

where $k(\boldsymbol{x}, \boldsymbol{y})$ denotes the kernel function with input $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X} \subseteq \mathbb{R}^d$, and $v(\cdot) : \mathcal{X} \to \mathbb{R}^h$ denotes feature map function.

In practical applications, the continuous integral in Eq.(5) can be computed via a discrete approximation over N samples:

$$\hat{\mathcal{I}}_1[v](\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} k(\boldsymbol{x}, \boldsymbol{y}^i) v(\boldsymbol{y}^i) \tag{6}$$

where $\boldsymbol{y}^i$ denotes the i-th sample of variable $\boldsymbol{y}$.

To achieve superior global information aggregation and propagation, we introduce the n-order kernel integral (with $n \geq 2$) as follows:

$$\begin{aligned} \mathcal{I}_n[v](\boldsymbol{x}) \;=\; & \int_{\mathcal{X}} \cdots \int_{\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{y}_1) \times \ldots \\ & \times k(\boldsymbol{y}_{n-1}, \boldsymbol{y}_n) \times v(\boldsymbol{y}_n) \mathrm{d}\boldsymbol{y}_1 \ldots \mathrm{d}\boldsymbol{y}_n \end{aligned} \tag{7}$$

where $\boldsymbol{y}_m \in \mathbb{R}^d$ denotes the m-th vector variable for integration. The kernel $k(\cdot, \cdot)$ is chosen to ensure the multi-integral exists. For example, given a bounded $v(\cdot)$ and $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}, |k(\boldsymbol{x}, \boldsymbol{y})| < 1$, the $\mathcal{I}_n[v](\boldsymbol{x}) < \infty$.

The discrete approximation of the n-order kernel integral over N^n samples (with N samples for each integral variable $\boldsymbol{y}_m$) is given by:

$$\begin{aligned} \hat{\mathcal{I}}_n[v](\boldsymbol{x}) = \frac{1}{N^n} \sum_{i_1=1}^{N} \cdots \sum_{i_n=1}^{N} & \left[\prod_{m=1}^{n-1} k(\boldsymbol{y}_m^{i_m}, \boldsymbol{y}_{m+1}^{i_{m+1}}) \right] \\ & \times k(\boldsymbol{x}, \boldsymbol{y}_1^{i_1}) v(\boldsymbol{y}_n^{i_n}) \end{aligned} \tag{8}$$

where $\boldsymbol{y}_m^{i_m}$ denotes the i_m-th sample of the m-th integral variable $\boldsymbol{y}_m$.

With the n-order kernel integral operator $\mathcal{I}_n[v](\cdot)$, we propose the infinite-order kernel integral operator as Eq. (9)

$$\mathcal{I}_\infty[v](\cdot) := \mathcal{I}_1[v](\cdot) + \alpha \mathcal{I}_2[v](\cdot) + \alpha^2 \mathcal{I}_3[v](\cdot) + \cdots \tag{9}$$

with proper α to ensure series convergence.

3.2.2. Closed-form finite approximation

Defining feature maps over N samples as $\boldsymbol{V}(Y) := [v(\boldsymbol{y}^1), \ldots, v(\boldsymbol{y}^N)]^\top \in \mathbb{R}^{N \times h}$ and the kernel vector $\boldsymbol{k}(\boldsymbol{x}) := \frac{1}{N}[k(\boldsymbol{x}, \boldsymbol{y}^1), \ldots, k(\boldsymbol{x}, \boldsymbol{y}^N)]^\top \in \mathbb{R}^{N \times 1}$, we can formulate the discrete approximations of the nth-order kernel integral $\mathcal{I}_n[v](\boldsymbol{x})$ in a matrix form:

$$\hat{\mathcal{I}}_n[v](x) = \boldsymbol{k}^\top(x) \boldsymbol{K}^{n-1} \boldsymbol{V}(Y) \tag{10}$$

where $\boldsymbol{K}$ denotes the kernel Gram matrix $\boldsymbol{K}(Y, Y)$ with its ij^{th}-element as $K_{ij} = \frac{1}{N} k(\boldsymbol{y}^i, \boldsymbol{y}^j)$, and $\boldsymbol{K}^{n-1}$ denotes $(n-1)$ order of $\boldsymbol{K}$, i.e., $\boldsymbol{K}^{n-1} = \underbrace{\boldsymbol{K}\boldsymbol{K} \cdots \boldsymbol{K}}_{n-1 \text{ times}}$, and $\boldsymbol{K}^0 = \boldsymbol{I}$.

We provide the finite approximation of the proposed infinite-order kernel integral operator as follows

$$\hat{\mathcal{I}}_\infty[v](\boldsymbol{x}) := \hat{\mathcal{I}}_1[v](\boldsymbol{x}) + \alpha \hat{\mathcal{I}}_2[v](\boldsymbol{x}) + \alpha^2 \hat{\mathcal{I}}_3[v](\boldsymbol{x}) + \cdots \tag{11}$$

Substituting matrix forms yields the geometric series:

$$\hat{\mathcal{I}}_\infty[v](\boldsymbol{x}) = \boldsymbol{k}^\top(\boldsymbol{x})(\boldsymbol{I} + \alpha \boldsymbol{K} + \alpha^2 \boldsymbol{K}^2 + \cdots) \boldsymbol{V}(Y) \tag{12}$$

When $\rho(\alpha \boldsymbol{K}) < 1$, where $\rho(\cdot)$ denotes the spectral radius, the positive-power Neumann series in Eq. (12) converges and gives the closed-form resolvent:

$$\hat{\mathcal{I}}_\infty[v](\boldsymbol{x}) = \boldsymbol{k}^\top(x)(\boldsymbol{I} - \alpha \boldsymbol{K})^{-1} \boldsymbol{V}(Y) \tag{13}$$

The same resolvent also admits a convergent inverse-power expansion when $\alpha\boldsymbol{K}$ is invertible and $\rho((\alpha\boldsymbol{K})^{-1}) < 1$:

$$(\boldsymbol{I} - \alpha\boldsymbol{K})^{-1} = -(\alpha\boldsymbol{K})^{-1}\left(\boldsymbol{I} - (\alpha\boldsymbol{K})^{-1}\right)^{-1} \tag{14}$$

$$= -\sum_{n=1}^{\infty}(\alpha\boldsymbol{K})^{-n}. \tag{15}$$

For a strictly positive-definite kernel evaluated on distinct sample points, $\boldsymbol{K}$ is symmetric positive definite and hence invertible; in this case the inverse-power series converges whenever the spectral norm of matrix $(\alpha\boldsymbol{K})^{-1}$ is less than one, i.e., $\|(\alpha\boldsymbol{K})^{-1}\|_2 = \frac{1}{|\alpha|}\lambda_{max}(\boldsymbol{K}^{-1}) = \frac{1}{|\alpha|\lambda_{min}(\boldsymbol{K})} < 1$. Equivalently $|\alpha|\lambda_{\min}(\boldsymbol{K}) > 1$.

Remark: The parameter α can either be positive or negative. By choosing $\alpha < 0$, the operator $\boldsymbol{I} - \alpha\boldsymbol{K}$ avoids ill-posed problems and makes the training process significantly more stable.

3.2.3. Two infinite-order neural operator constructions

While the closed-form approximation in Eq. (13) is derived in the matrix form on a single set of N samples, extending the infinite-order idea to a d-dimensional product grid admits two natural neural operator constructions. We adopt the standard dimension-wise product base kernel,

$$k(\boldsymbol{x}, \boldsymbol{y}) = k_1(x_1, y_1) \times k_2(x_2, y_2) \times \cdots \times k_d(x_d, y_d), \tag{16}$$

which induces a kernel Gram matrix $\boldsymbol{K} \in \mathbb{R}^{N^d \times N^d}$ over the product grid that factorizes as the Kronecker product

$$\boldsymbol{K} = \boldsymbol{K}_1 \otimes \boldsymbol{K}_2 \otimes \cdots \otimes \boldsymbol{K}_d, \tag{17}$$

where $\boldsymbol{K}_j \in \mathbb{R}^{N\times N}$ is the one-dimensional kernel matrix on axis j with elements $(\boldsymbol{K}_j)_{pq} = k_j(y_j^p, y_j^q)$, and y_j^p, y_j^q denote the pth and q^{th} grid points in the j-th dimension. Based on this factorization, we define two alternative discrete infinite-order operators. They should be viewed as two distinct infinite-order neural operator constructions.

IKNO-Vanilla. The first construction applies Eq. (13) directly to the full kernel matrix on the product grid:

$$\mathcal{K}_\infty^{\text{Van}} := (\boldsymbol{I}_M - \alpha\boldsymbol{K})^{-1}, \quad M = N^d. \tag{18}$$

This is a faithful discretization of the infinite-order geometric series in Eq. (12) on the d-dimensional grid: every cross-dimensional coupling term $\boldsymbol{K}^n$ is retained, so $\mathcal{K}_\infty^{\text{Van}}$ inherits the full expressive power of the continuous operator $\mathcal{I}_\infty$.

IKNO-TP. The second construction is motivated by the equivalence between tensor products in function spaces and Kronecker products in matrix spaces. For Hilbert spaces $\mathcal{H}_j = L^2(\mathcal{X}_j)$ $(j = 1, \ldots, d)$, the tensor product space satisfies $\mathcal{H}_1 \otimes \cdots \otimes \mathcal{H}_d \cong L^2(\mathcal{X}_1 \times \cdots \times \mathcal{X}_d)$ and linear operators compose via

$$(A_1 \otimes \cdots \otimes A_d)(f_1 \otimes \cdots \otimes f_d) = (A_1 f_1) \otimes \cdots \otimes (A_d f_d). \tag{19}$$

Interpreting each one-dimensional resolvent $(\boldsymbol{I}_N - \alpha\boldsymbol{K}_j)^{-1}$ as the infinite-order kernel operator acting on its own axis, we *define* the multi-dimensional operator as the tensor product of these per-axis operators:

$$\mathcal{K}_\infty^{\text{TP}} := (\boldsymbol{I}_N - \alpha\boldsymbol{K}_1)^{-1} \otimes (\boldsymbol{I}_N - \alpha\boldsymbol{K}_2)^{-1} \otimes \cdots \otimes (\boldsymbol{I}_N - \alpha\boldsymbol{K}_d)^{-1}. \tag{20}$$

We emphasize that $\mathcal{K}_\infty^{\text{TP}}$ is *not* algebraically identical to $\mathcal{K}_\infty^{\text{Van}}$; rather, it is an alternative operator defined by tensor composition of per-axis resolvents. Geometrically, $\mathcal{K}_\infty^{\text{TP}}$ lifts the input into a structured latent space spanned by the tensor-product basis induced by the per-dimension kernels. This structured hypothesis class implicitly constrains the functional form of the learned operator and introduces a form of *functional regularization*, which can be beneficial on problems with strong multi-scale or high-frequency components.

We refer to the neural operators built with $\mathcal{K}_\infty^{\text{Van}}$ and $\mathcal{K}_\infty^{\text{TP}}$ as **IKNO-Vanilla** and **IKNO-TP**, respectively. They share an identical end-to-end architecture (Sec. 3.3) and differ only in the choice of discrete infinite-order operator.

3.2.4. Fast computation

Directly forming and inverting the $M \times M$ resolvent in either Eq. (18) or Eq. (20) would incur $O(M^3) = O(N^{3d})$ operations, which is prohibitive even for moderate N and d. For both variants, the Kronecker structure admits a scheme with $O(d \cdot N^3)$ preprocessing and $O(d \cdot N \cdot M)$ cost per forward application, without ever materializing the $M \times M$ operator.

Fast computation for IKNO-Vanilla. For the full-kernel resolvent in Eq. (18), we diagonalize each one-dimensional kernel matrix via a symmetric eigendecomposition $\boldsymbol{K}_j = \boldsymbol{U}_j\boldsymbol{\Lambda}_j\boldsymbol{U}_j^\top$. Using the Kronecker product property $(A_1 \otimes \cdots \otimes A_d)(B_1 \otimes \cdots \otimes B_d) = (A_1B_1) \otimes \cdots \otimes (A_dB_d)$, the full resolvent admits the closed form

$$(\boldsymbol{I}_M - \alpha\,\boldsymbol{K}_1 \otimes \cdots \otimes \boldsymbol{K}_d)^{-1} = (\boldsymbol{U}_1 \otimes \cdots \otimes \boldsymbol{U}_d)\left[\boldsymbol{I}_M - \alpha\,\boldsymbol{\Lambda}_1 \otimes \cdots \otimes \boldsymbol{\Lambda}_d\right]^{-1} \times (\boldsymbol{U}_1 \otimes \cdots \otimes \boldsymbol{U}_d)^\top, \tag{21}$$

where the middle term is diagonal and given by

$$\left[\boldsymbol{I}_M - \alpha\, \boldsymbol{\Lambda}_1 \otimes \cdots \otimes \boldsymbol{\Lambda}_d\right]^{-1} = \operatorname{diag}\left(\frac{1}{1 - \alpha\, \lambda_{i_1}^{(1)} \lambda_{i_2}^{(2)} \cdots \lambda_{i_d}^{(d)}}\right)_{(i_1,\ldots,i_d)}, \tag{22}$$

with $\lambda_{i_j}^{(j)}$ the i_j-th eigenvalue of $\boldsymbol{K}_j$. Preprocessing thus consists of d eigendecompositions at $O(d \cdot N^3)$ plus construction of the M diagonal entries at $O(M)$. Each forward application applies $(\boldsymbol{U}_1 \otimes \cdots \otimes \boldsymbol{U}_d)^\top$, the diagonal reweighting, and $(\boldsymbol{U}_1 \otimes \cdots \otimes \boldsymbol{U}_d)$ through per-dimensional tensor-matrix multiplications at total cost $O(d \cdot N \cdot M)$.

Fast computation for IKNO-TP. By definition, $\mathcal{K}_\infty^{\text{TP}}$ is already a Kronecker product of per-axis resolvents, so it reduces to inverting d independent $N \times N$ matrices:

$$\mathcal{K}_\infty^{\text{TP}} = \bigotimes_{j=1}^{d} (\boldsymbol{I}_N - \alpha \boldsymbol{K}_j)^{-1}. \tag{23}$$

This yields a preprocessing cost of $O(d \cdot N^3)$. Applying $\mathcal{K}_\infty^{\text{TP}}$ to any feature tensor $\boldsymbol{V} \in \mathbb{R}^{M \times h}$ is then carried out as d successive per-dimensional $N \times N$ matrix multiplications at total cost $O(d \cdot N \cdot M)$.

Complexity comparison. Both IKNO-Vanilla and IKNO-TP reduce the naive $O(N^{3d})$ matrix-inverse cost to $O(d{\cdot}N^3)$ at preprocessing and $O(d{\cdot}N{\cdot}M)$ per forward pass. For both variants, applying the operator to features is dominated by a similar sequence of tensor-matrix multiplications along each dimension. IKNO-Vanilla incurs only a minor additional overhead from the symmetric eigendecomposition relative to direct inversion. In practice, the wall-clock runtimes of the two variants are nearly indistinguishable on all tested grids, so the choice between them is driven entirely by accuracy considerations (Sec. 4.3).

3.2.5. Learnable multi-scale base kernel

To capture both smooth global structures and sharp local gradients, we design a learnable base kernel for each dimension j. We introduce learnable parameters c_j and β_j and γ_j that rescale the coordinate distance, effectively reconfiguring the metric space for Gaussian and Laplace components respectively:

$$k_j(x_j, y_j) = c_j \Big(\exp\left(-|\beta_j (x_j - y_j)|^2\right) + \exp\left(-|\gamma_j (x_j - y_j)|\right) \Big) \tag{24}$$

where x_j and y_j denote the element in j^{th} dimension of input vector $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. Since both Gaussian and Laplacian kernels are strictly positive-definite, their nonnegative linear combination remains strictly positive-definite when the coefficients are non-degenerate; in our parameterization, this corresponds to $c_j > 0$ and nonzero distance scales. Therefore, for distinct grid points, the induced Gram matrix is real symmetric positive definite.

By jointly learning $\{c_j, \beta_j, \gamma_j\}$ for $j \in \{1, \cdots, d\}$, the model reconfigures its receptive field for each dimension, providing a flexible representation for complex anisotropic operators.

3.3. IKNO architecture

IKNO follows an encoder–processor–decoder architecture on a structured latent grid. Given point-cloud observations, an MLP first tokenizes geometry-enhanced input features. The infinite-order kernel encoder maps the point tokens to the latent grid; a Transformer processor performs global mixing on this grid; and the infinite-order kernel decoder maps the processed latent representation back to arbitrary query coordinates. Both IKNO-Vanilla and IKNO-TP use this architecture and differ only in the discrete infinite-order operator defined in Sec. 3.2.3. Detailed tokenization, encoding, decoding, and multi-scale kernel-branch designs are provided in Appendix C.1.

$$\boldsymbol{V}_G = \mathcal{K}_\infty \boldsymbol{K}_{GP} \boldsymbol{V}_P, \tag{25}$$

$$\boldsymbol{V}'_G = \text{TransformerEncoder}(\boldsymbol{V}_G), \tag{26}$$

$$\hat{u}(\boldsymbol{x}) = \text{MLP}_{\text{head}}\left(\boldsymbol{k}^\top(\boldsymbol{x}) \mathcal{K}_\infty \boldsymbol{V}'_G\right). \tag{27}$$

4. Experiments

In this section, we evaluate the performance of IKNO across a diverse range of benchmarks, spanning from classical physical systems to complex industrial-grade aerodynamic simulations.

4.1. Experimental setup

Datasets. We consider three categories of datasets to comprehensively assess the generalization and scalability of the proposed IKNO model: **Time-Independent Benchmarks**, **Time-Dependent Benchmarks** and **Industrial-scale Benchmarks**. More details about the benchmarks can be found in the Appendix B.

Baselines and Protocol. To ensure a fair and rigorous comparison, we strictly follow the experimental protocols established by the current state-of-the-art model, GAOT(Wen et al., 2025). We directly adopt the reported results from the GAOT study for all baseline models, including Transolver(Wu et al., 2024), GINO(Li et al., 2023b), GNOT(Hao et al., 2023) and RIGNO(Mousavi et al., 2025). All models are evaluated using the same training and testing splits and identical evaluation metrics.

Table 2. Comparison of median relative L_1 error (%) on time-independent benchmarks. Bold and underline indicate the best and second-best results across all methods, respectively.

	Ours		Baselines					
Dataset	IKNO-Vanilla	IKNO-TP	GAOT	RIGNO-18	Transolver	GNOT	UPT	GINO
Poisson C-Sines	<u>1.42</u>	**1.00**	3.10	6.83	77.3	100	100	20.0
Poisson Gauss	<u>0.34</u>	**0.26**	0.83	2.26	2.02	88.9	48.4	7.57
Elasticity	**0.93**	<u>0.96</u>	1.34	4.31	4.92	10.4	12.6	4.38
NACA0012	**3.76**	<u>4.22</u>	6.81	5.30	8.69	6.89	16.1	9.01
NACA2412	<u>4.99</u>	**4.23**	6.66	6.72	8.51	8.82	17.9	9.39
RAE2822	4.97	**4.70**	6.61	5.06	<u>4.82</u>	7.15	16.1	8.61

Table 3. Comparison of median relative L_1 error (%) on time-dependent datasets. Bold and underline indicate the best and second-best results across all methods, respectively.

	Ours		Baselines					
Dataset	IKNO-Vanilla	IKNO-TP	GAOT	RIGNO-18	GeoFNO	FNO DSE	UPT	GINO
NS-Gauss	**1.48**	<u>1.69</u>	2.91	2.29	41.1	38.4	92.5	13.1
NS-PwC	<u>0.83</u>	**0.63**	1.50	1.58	26.0	56.7	100	5.85
NS-SL	1.31	**1.12**	<u>1.21</u>	1.28	13.7	22.6	51.5	4.48
NS-SVS	<u>0.38</u>	**0.34**	0.46	0.56	9.75	26.0	4.2	1.19
CE-Gauss	7.30	**6.22**	<u>6.40</u>	6.90	42.1	30.8	64.2	25.1
CE-RP	7.93	<u>5.85</u>	5.97	**3.98**	18.4	27.7	26.8	12.3
Wave-Layer	5.84	**3.17**	<u>5.78</u>	6.77	11.1	28.3	19.6	19.2
Wave-C-Sines	<u>2.86</u>	**1.70**	4.65	5.35	13.1	5.52	12.7	5.82

4.2. Implementation details

We provide the detailed architectural configuration of our proposed IKNO model. Almost all parameters are kept consistent across all experiments to demonstrate the robustness of the architecture, with the exception of the grid resolution which is adapted to the spatial dimensionality and complexity of specific datasets. More details can be found in the Appendix C.

4.3. Experimental results

Results on Non-Industrial Datasets.

The experimental results for the median relative L_1 error on time-independent benchmarks are summarized in Table 2. We compare the two variants introduced in Sec. 3.2.3: IKNO-Vanilla, which applies the full-kernel resolvent on the product grid, and IKNO-TP, which composes the per-axis resolvents via tensor product. On these time-independent datasets, IKNO-TP shows a clear performance advantage, achieving the best result on four of the six benchmarks and the second-best result on the two benchmarks where IKNO-Vanilla performs best.

The two variants exhibit complementary strengths, with IKNO-TP providing the strongest overall performance across the non-industrial benchmarks. Across the 14 time-independent and time-dependent datasets in Tables 2 and 3, IKNO-TP achieves the best result on ten benchmarks and the second-best result on the remaining four. IKNO-Vanilla remains a competitive non-tensorized alternative and attains the best result on several cases, including *Elasticity*, *NACA0012*, and *NS-Gauss*. This pattern suggests that the tensor-product construction often provides a more robust inductive bias, while the full-coupling construction can still be advantageous on selected datasets.

These comprehensive results empirically validate that the infinite-order kernel integral – under either construction – yields a highly expressive neural operator, and that the two variants together cover a broad spectrum of PDE behaviors.

Industrial Dataset Scalability.

To evaluate the robustness and scalability of IKNO in handling complex, industrial-grade aerodynamic simulations, we conduct experiments on the high-fidelity *NASA CRM* benchmark (Rivers, 2019). Unlike simplified 2D airfoils, this dataset involves 3D geometries with intricate topological features and high-Reynolds-number turbulent flows, posing a significant challenge for neural operators. Furthermore, each individual mesh in this dataset contains 440k points, presenting additional hurdles for GPU memory capacity and computational speed during model training.

We evaluate the two IKNO variants against the GAOT baseline on the large-scale industrial dataset *NASA CRM*. The quantitative results, including Mean Squared Error and Mean Absolute Error for two groups of physical quantities (surface pressure P and skin friction coefficient C_f), are summarized in Table 4.

All IKNO variants outperform GAOT by a substantial margin on every metric. IKNO-TP achieves the strongest perfor-

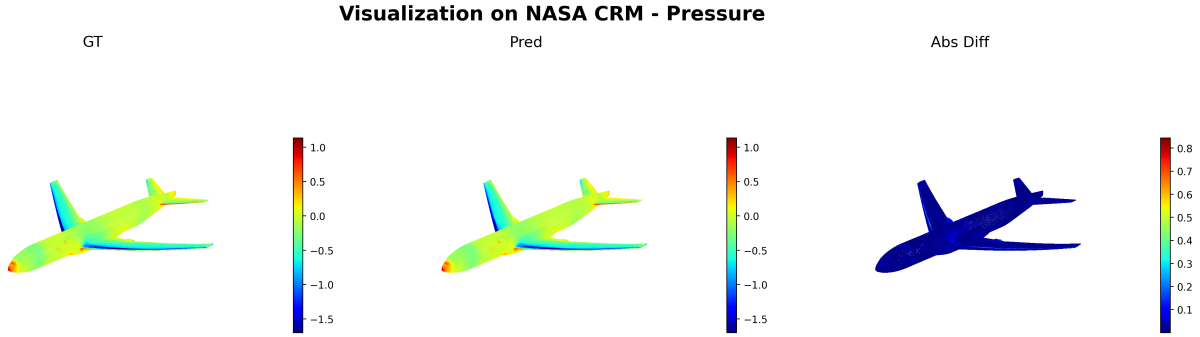

Figure 2. Visualization of IKNO predictions on the *NASA CRM* industrial benchmark.

Table 4. Comparison of GAOT and IKNO variants on the NASA CRM benchmark with MSE ($\times 10^{-2}$) and mean AE ($\times 10^{-1}$). Bold marks the best result per column. **P**: pressure. **Cf**: skin friction coefficient. **CRM**: NASA CRM dataset.

Model	P (CRM)		Cf (CRM)	
	MSE	Mean AE	MSE	Mean AE
GAOT	7.7170	1.6014	16.1091	2.2305
IKNO-Vanilla	2.5108	0.8768	7.3943	1.4202
IKNO-TP	**2.1682**	**0.7801**	**6.0754**	**1.2372**

mance, reducing the pressure MSE from 7.72 to 2.17 and the skin-friction MSE from 16.1 to 6.08, which demonstrates a remarkable performance improvement. IKNO-Vanilla also substantially improves over GAOT, showing that the full-coupling construction remains effective in this industrial regime. The further gain of IKNO-TP is particularly informative for *NASA CRM*, which combines extremely high-resolution meshes ($\sim$440k points per sample) with a very small training set (105 samples). Under such a low-data, high-dimensional setting, the structured tensor-product hypothesis class of IKNO-TP provides a more suitable representational bias and yields the best accuracy across both pressure and skin-friction fields.

Fig. 2 further shows that IKNO produces coherent full-aircraft predictions on the high-resolution geometry, with visually small error patterns that indicate an accurate neural-operator approximation of the aerodynamic fields.

Table 5. Computational efficiency on *Wave-C-Sines* (L40S GPU, identical training settings). "IKNO (direct inverse)" denotes the ablation in which the Kronecker-structured fast computation is disabled and $(\boldsymbol{I}_M - \alpha \boldsymbol{K})^{-1}$ is formed and inverted explicitly on the full product grid. Bold marks the best result per column among deployable methods (top three rows).

Model	**Memory (GB)**	**Time / Epoch (s)**
IKNO-TP	**25.06**	**281**
GAOT	33.14	473
Transolver	25.89	324
IKNO (direct inverse)	24.84	1539

Computational efficiency. Table 5 confirms that the Kronecker-structured fast computation preserves the practical scalability of IKNO: on *Wave-C-Sines*, IKNO uses less memory and runs faster than GAOT and Transolver, while direct full-grid inversion causes a large runtime slowdown. Additional implementation details and analysis are provided in Appendix G.

Ablation summary. Additional ablation studies in Appendix F show that the derivative prediction target is consistently the strongest temporal formulation, time-dependent datasets benefit substantially from increased latent grid resolution, and replacing the infinite-order kernel encoder with Cross-Attention severely degrades performance on challenging time-dependent benchmarks.

5. Conclusions and future work

In this paper, we have presented the Infinite-order Kernel Neural Operator (IKNO), a novel neural operator designed to address the expressive limitations of first-order kernel integral approximations in current neural operators. By introducing the infinite-order kernel integral to the encoder, IKNO achieves superior global information aggregation and propagation compared to traditional methods. We further introduce two complementary multi-dimensional infinite-order neural operator constructions, IKNO-Vanilla and IKNO-TP, together with Kronecker-structured fast computation schemes that make both constructions scalable on high-resolution latent grids.

Our empirical evaluation across 15 benchmarks demonstrates that IKNO achieves the best or highly competitive accuracy on nearly all time-independent and highly time-dependent PDE systems. Across these benchmarks, IKNO-TP is consistently the more robust variant, with particularly clear advantages on multi-scale, high-frequency, and low-data high-resolution industrial benchmarks. On the low-data, high-resolution *NASA CRM* benchmark, both IKNO variants substantially improve over GAOT, with IKNO-TP achieving the strongest accuracy across all pressure and skin-friction metrics. These results also reveal practical limitations: IKNO depends on latent grid resolution, representation capacity can vary with the chosen infinite-order construction, and the initial kernel scales still require validation-based search rather than being automatically optimized to their best setting. Taken together, these findings position IKNO, especially IKNO-TP, as a scalable operator-learning framework for high-fidelity physics simulations.

Impact Statement

This paper presents work whose goal is to advance scientific machine learning and neural operator methods for efficient PDE surrogate modeling. Potential societal impacts are primarily tied to the downstream scientific and engineering domains in which such models are deployed; we do not identify additional ethical concerns that must be specifically highlighted beyond standard model validation, reliability, and responsible use requirements for high-stakes simulations.

Acknowledgements

This research was supported by Career Development Fund (CDF) of the Agency for Science, Technology and Research (A*STAR) (No: H26-KSR0056).

References

Altman, D. G. and Bland, J. M. Standard deviations and standard errors. *Bmj*, 331(7521):903, 2005.

Black, F. and Scholes, M. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3): 637–654, 1973.

Brandstetter, J., Worrall, D., and Welling, M. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Evans, L. C. *Partial differential equations*, volume 19. American mathematical society, 2022.

Galerkin, B. G. *Rods and plates: series in some questions of elastic equilibrium of rods and plates*. National Technical Information Service Springfield, VA, USA, 1968.

Hao, Z., Wang, Z., Su, H., Ying, C., Dong, Y., Liu, S., Cheng, Z., Song, J., and Zhu, J. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pp. 12556–12569. PMLR, 2023.

Herde, M., Raonic, B., Rohner, T., Käppeli, R., Molinaro, R., de Bézenac, E., and Mishra, S. Poseidon: Efficient foundation models for pdes. *Advances in Neural Information Processing Systems*, 37:72525–72624, 2024.

Hughes, T. J. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2003.

Li, X., Li, Z., Kovachki, N., and Anandkumar, A. Geometric operator learning with optimal transport. *arXiv preprint arXiv:2507.20065*, 2025.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Li, Z., Meidani, K., and Farimani, A. B. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022.

Li, Z., Huang, D. Z., Liu, B., and Anandkumar, A. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023a.

Li, Z., Kovachki, N., Choy, C., Li, B., Kossaifi, J., Otta, S., Nabian, M. A., Stadler, M., Hundt, C., Azizzadenesheli, K., et al. Geometry-informed neural operator for large-scale 3d pdes. *Advances in Neural Information Processing Systems*, 36:35836–35854, 2023b.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Luo, H., Wu, H., Zhou, H., Xing, L., Di, Y., Wang, J., and Long, M. Transolver++: An accurate neural solver for pdes on million-scale geometries. *arXiv preprint arXiv:2502.02414*, 2025.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

Mousavi, S., Wen, S., Lingsch, L., Herde, M., Raonić, B., and Mishra, S. Rigno: A graph-based framework for robust and accurate operator learning for pdes on arbitrary domains. *arXiv preprint arXiv:2501.19205*, 2025.

Murray, J. D. *Mathematical biology: I. An introduction*, volume 17. Springer Science & Business Media, 2007.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 165–174, 2019.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International*

conference on machine learning, pp. 1310–1318. Pmlr, 2013.

Rahman, M. A., Ross, Z. E., and Azizzadenesheli, K. U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.

Rivers, M. B. Nasa common research model: a history and future plans. In *AIAA Aviation 2019 Forum*, pp. 3725, 2019.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Thomas, J. W. *Numerical partial differential equations: finite difference methods*, volume 22. Springer Science & Business Media, 2013.

Vallis, G. K. *Atmospheric and oceanic fluid dynamics*. Cambridge University Press, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, T. and Wang, C. Latent neural operator for solving forward and inverse pde problems. *Advances in Neural Information Processing Systems*, 37:33085–33107, 2024.

Wen, S., Kumbhat, A., Lingsch, L., Mousavi, S., Zhao, Y., Chandrashekar, P., and Mishra, S. Geometry aware operator transformer as an efficient and accurate neural surrogate for pdes on arbitrary domains. *arXiv preprint arXiv:2505.18781*, 2025.

Wu, H., Luo, H., Wang, H., Wang, J., and Long, M. Transolver: A fast transformer solver for pdes on general geometries. *arXiv preprint arXiv:2402.02366*, 2024.

A. Problem formulation

We consider the problem of learning the neural operator of a Partial Differential Equation from data. Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. We aim to learn a mapping between infinite-dimensional function spaces defined over Ω and with potentially a temporal domain $\Omega_T = [0, T]$.

A.1. Time-independent case

In the time-independent setting, the PDE describes a steady state where the solution u depends on a given input condition a. Let $\mathcal{A}$ and $\mathcal{U}$ be two Banach spaces of functions. The time-independent PDE can be abstractly defined as:

$$\begin{cases} \mathcal{L}(u(\boldsymbol{x}); a(\boldsymbol{x})) = 0, & \boldsymbol{x} \in \Omega \\ \mathcal{B}(u(\boldsymbol{x}); a(\boldsymbol{x})) = 0, & \boldsymbol{x} \in \partial\Omega \end{cases} \tag{28}$$

where $\mathcal{L}$ is a differential operator representing the corresponding PDE, $\mathcal{B}$ denotes the boundary operator, $a \in \mathcal{A}$ represents the input condition including coefficients, geometry parameters and source terms, and $u \in \mathcal{U}$ is the corresponding solution.

Our goal is to learn the solution operator $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ such that:

$$u(\boldsymbol{x}) = \mathcal{G}(a)(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega \tag{29}$$

In our implementation, both a and u are observed on an arbitrary point cloud $\mathcal{P} = \{\boldsymbol{x}^i\}_{i=1}^{\bar{N}} \subset \Omega$. The input for each point $\boldsymbol{x}^i$ is a feature set $\{\boldsymbol{x}^i, a(\boldsymbol{x}^i)\}$.

A.2. Time-dependent case

For the time-dependent setting, we consider the evolution of the system over a time interval $\Omega_T = [0, T]$. Let $u(\boldsymbol{x}, t)$ denote the solution at position $\boldsymbol{x} \in \Omega$ and time $t \in \Omega_T$. The evolution is governed by the following PDE:

$$\begin{cases} \frac{\partial u(\boldsymbol{x},t)}{\partial t} = \mathcal{F}(u(\boldsymbol{x}, t); a(\boldsymbol{x}, t)), & (\boldsymbol{x}, t) \in \Omega \times \Omega_T \\ u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \end{cases} \tag{30}$$

where $\mathcal{F}$ is a spatial differential operator representing the corresponding PDE, while the time-varying condition $a(\boldsymbol{x}, t)$ encapsulates all external forcing, source terms, and time-dependent parameters. $u_0(\boldsymbol{x})$ is the initial state of the system (Evans, 2022).

In this scenario, we aim to learn a trajectory operator $\mathcal{G}_{\text{td}}$ that maps the initial state and the entire history of conditions to the full solution trajectory:

$$u(\cdot, t)_{t \in \Omega_T} = \mathcal{G}_{\text{td}}(u_0, \{a(\cdot, t)\}_{t \in \Omega_T}) \tag{31}$$

The dataset consists of multiple trajectories sampled on arbitrary point clouds. For a single trajectory, given the discrete point set $\{\boldsymbol{x}^i\}_{i=1}^{\bar{N}}$ and time steps $\{t_j\}_{j=0}^{\bar{M}}$, the model is trained to predict $\{u(\boldsymbol{x}^i, t_j)\}$ for all i, j based on the initial state $\{u_0(\boldsymbol{x}^i)\}$ and the sequence of conditions $\{a(\boldsymbol{x}^i, t_j)\}$.

For experiments on time-dependent benchmarks, we follow the unified lead-time prediction protocol of RIGNO (Mousavi et al., 2025), which is also adopted by GAOT (Wen et al., 2025). Details are provided in Appendix C.2.

B. Datasets

Datasets. We consider three categories of datasets to comprehensively assess the generalization and scalability of the proposed IKNO model:

- **Time-Independent Benchmarks:** We select six time-independent datasets defined on complex, non-grid domains. These include *Elasticity* from GeoFNO(Li et al., 2023a), which involves stresses on an irregular mesh with a central hole; *Poisson-Gauss* from RIGNO(Mousavi et al., 2025), solving the Poisson equation with Gaussian sources; and four datasets from GAOT(Wen et al., 2025) —*NACA0012*, *NACA2412*, *RAE2822*, and *Poisson-C-Sines*. The former three are airfoil datasets represent challenging transonic flow conditions with significant geometric variations, while the *Poisson-C-Sines* dataset features multi-scale solutions on a circular domain, testing the model's ability to handle unstructured inputs and high-frequency components.

- **Time-Dependent Benchmarks:** We choose eight time-dependent datasets introduced by RIGNO(Mousavi et al., 2025), covering the *Compressible Euler equations*, *incompressible Navier-Stokes equations*, and *acoustic Wave equations*. These benchmarks involve complex physical phenomena such as shocks, sharp traveling waves, and diffraction. The non-linear interactions and reflections in these systems make them significantly harder to learn than time-independent cases.

- **Industrial-scale Benchmarks:** To evaluate the scalability of the IKNO model, we utilize the *NASA CRM* (Common Research Model)(Rivers, 2019). This dataset serves as a rigorous industrial benchmark for civil transport aircraft, characterized by high-fidelity complexity: each individual sample contains approximately 440,000 spatial points. Despite this high dimensionality, the dataset provides a constrained set of only 105 training samples and 44 test samples. This large-scale data with small-scale samples nature poses a significant challenge for neural operators to generalize effectively. The model is required to predict pressure P and skin friction coefficients C_f based on aircraft geometry and simulation-specific parameters, such as the Mach number.

A detailed summary of these datasets is provided in Table B.1.

Dataset Licenses and Access. The RIGNO datasets used in this work are released on Zenodo under the Creative Commons Attribution 4.0 International (CC-BY-4.0) license. The GAOT datasets, Geo-FNO/Elasticity dataset, and NASA CRM dataset are publicly available through links provided by their original authors or maintainers, and we cite the corresponding original sources. However, we did not find explicit license statements for these datasets on their public access pages.

Table B.1. Summary of datasets used in this work.

Abbreviation	Characteristic	Grid domain
Poisson-C-Sines	Circular domain with sines	F
Poisson-Gauss	Gaussian source	T
Elasticity	Hole boundary distance	F
NACA0012	Flow past NACA0012 airfoil	F
NACA2412	Flow past NACA2412 airfoil	F
RAE2822	Flow past RAE2822 airfoil	F
NASA-CRM	Surface pressure / friction coefficient	F
NS-Gauss	Gaussian vorticity IC	T
NS-PwC	Piecewise constant IC	T
NS-SL	Shear layer IC	T
NS-SVS	Sinusoidal vortex sheet IC	T
CE-Gauss	Gaussian vorticity IC	T
CE-RP	4-quadrant RP	T
Wave-Layer	Layered wave medium	T
Wave-C-Sines	Circular domain with sines	F

B.1. Preliminary finite-order kernel study

This appendix provides additional notation details for the preliminary study in Sec. 3.1. The latent grid is denoted by $G = \{\boldsymbol{y}^j\}_{j=1}^{L^d}$, and the point-cloud input and query sets are denoted by $P = \{\boldsymbol{x}^i\}_{i=1}^{\bar{N}}$ and $Q = \{\boldsymbol{q}^r\}_{r=1}^{N_q}$, respectively. The matrix $\boldsymbol{K}_{GP} \in \mathbb{R}^{L^d \times \bar{N}}$ maps input points to latent grid points with entries $k(\boldsymbol{y}^j, \boldsymbol{x}^i)$, while $\boldsymbol{K}_{QG} \in \mathbb{R}^{N_q \times L^d}$ maps latent grid points to query locations with entries $k(\boldsymbol{q}^r, \boldsymbol{y}^j)$.

The token matrix over input points is $\boldsymbol{V}_P \in \mathbb{R}^{\bar{N} \times h}$. The matrices $\boldsymbol{V}_G$ and $\boldsymbol{V}'_G \in \mathbb{R}^{L^d \times h}$ denote latent-grid features before and after Transformer processing, respectively, and $\boldsymbol{V}_Q$ denotes the output tokens at query points before the final prediction head. All finite-order runs use a fixed linear-window kernel with radius $r = 0.2$ and unit scale, together with a fixed propagation coefficient $\alpha = -0.15$; neither the kernel scale nor α is learned. The same architecture, training protocol, and dataset split are used across all finite orders; only the truncation order p in $\boldsymbol{R}_p(G)$ is varied.

C. Implementation details

We provide the detailed architectural configuration of our proposed IKNO model. Almost all parameters are kept consistent across all experiments to demonstrate the robustness of the architecture, with the exception of the grid resolution which is adapted to the spatial dimensionality and complexity of specific datasets.

C.1. Detailed IKNO architecture

C.1.1. Architectural overview

IKNO is designed to learn the solution operator of PDEs on complex, non-standard geometries. Unlike traditional operators that rely on local message passing or fixed-grid spectral methods, IKNO establishes a continuous mapping between unstructured point clouds and a latent grid through the infinite-order kernel integral. The architecture consists of four primary stages: (i) Feature Tokenization with geometry enhancement, (ii) Infinite Kernel Encoding from points to a latent grid, (iii) a Latent Processor for global feature mixing, and (iv) Infinite Kernel Decoding back to the physical quantities at query coordinates.

C.1.2. Tokenization with geometry enhancement

To ensure the model well captures the information of the underlying manifold of the PDE domain, we employ a coordinate-based encoding strategy inspired by NeRF(Mildenhall et al., 2021). For an input point $\boldsymbol{x}^i \in \Omega$ with associated condition $a(\boldsymbol{x}^i)$, we apply a positional encoding feature map with $l = 1$ frequency levels to capture fundamental spatial variations:

$$\psi(\boldsymbol{x}^i) = \big(\boldsymbol{x}^i, \cos(\boldsymbol{x}^i), \sin(\boldsymbol{x}^i)\big) \in \mathbb{R}^{3d} \tag{32}$$

where $\cos(\cdot)$ and $\sin(\cdot)$ denotes element-wise operation.

The augmented features are concatenated and projected into a high-dimensional latent space via an embedding MLP to produce the initial tokens $v(\boldsymbol{x}^i)$:

$$v(\boldsymbol{x}^i) = \text{MLP}(\text{Concat}(\psi(\boldsymbol{x}^i), a(\boldsymbol{x}^i))) \tag{33}$$

C.1.3. Infinite kernel encoding

The encoding stage maps the scattered point cloud tokens $\{v(\boldsymbol{x}^i)\}_{i=1}^{\bar{N}}$ onto a structured latent grid G with grid points $\{\boldsymbol{y}^j\}_{j=1}^{M}$. To capture infinite-order global dependencies, we utilize the closed-form operator given in Eq. (13). The latent grid representation $\boldsymbol{V}_G \in \mathbb{R}^{M\times h}$ is generated as follows:

$$\boldsymbol{V}_G = (\boldsymbol{I}_M - \alpha\boldsymbol{K})^{-1}\boldsymbol{K}_{GP}\boldsymbol{V}_P \tag{34}$$

where $\boldsymbol{K}_{GP} \in \mathbb{R}^{M\times\bar{N}}$ denotes kernel matrix such that its element in j^{th}-row and i^{th}-column is $k(\boldsymbol{y}^j, \boldsymbol{x}^i)$, and $\boldsymbol{V}_P = [v(\boldsymbol{x}^1), \cdots, v(\boldsymbol{x}^{\bar{N}})]^\top \in \mathbb{R}^{\bar{N}\times h}$ denotes the embedding feature matrix over input data.

By utilizing the infinite-order formulation, each grid point $\boldsymbol{y}^j$ adaptively aggregates information from all input data points, implicitly considering high-order physical correlations beyond simple linear kernels. In practice, multiple kernel results are fused through concatenation followed by an encoder fusion MLP to obtain the unified latent grid representation.

C.1.4. Latent processor

On the latent grid G, we employ a series of Transformer Encoder blocks to perform non-linear mixing of the grid features. This stage refines the global physical states in a latent space where the topology is regular and computationally efficient. Let $\boldsymbol{V}'_G$ denote the processed latent features:

$$\boldsymbol{V}'_G = \text{TransformerEncoder}(\boldsymbol{V}_G) \tag{35}$$

$\boldsymbol{V}'_G$ is constructed by our TransformerEncoder.

C.1.5. Infinite kernel decoding

The decoding stage reconstructs the solution at arbitrary query coordinates $\boldsymbol{x}$ by projecting information from the latent grid G back to the continuous domain. This process utilizes the same infinite-order kernel logic to maintain consistency in the

functional representation:

$$v_{\text{out}}(\boldsymbol{x}) = \hat{\mathcal{I}}_\infty[v'](\boldsymbol{x}) = \boldsymbol{k}^\top(\boldsymbol{x})(\boldsymbol{I}_M - \alpha\boldsymbol{K})^{-1}\boldsymbol{V}'_G \tag{36}$$

where $\boldsymbol{V}'_G \in \mathbb{R}^{M\times h}$ denotes the embedding feature matrix over the latent grids.

Similar to the encoding stage, the output tokens will pass through an output head MLP to yield the predicted physical quantities $\hat{u}(\boldsymbol{x})$:

$$\hat{u}(\boldsymbol{x}) = \text{MLP}_{\text{head}}(v_{\text{out}}(\boldsymbol{x})) \tag{37}$$

C.1.6. Multi-scale kernel design

To capture anisotropic physical gradients and various interaction scales, IKNO instantiates $Q = 3$ independent kernel branches. Each branch $q \in \{1, \dots, Q\}$ utilizes the base kernel described in Sec. 3.2.5. Crucially, both the coefficients $c_{q,j}$ and the scaling parameters $\beta_{q,j}$, $\gamma_{q,j}$ are learnable for each branch q and each spatial dimension j. The dimension-wise $c_{q,j}$ allows the model to adaptively control the propagation density of physical information along different axes, while $\beta_{q,j}$ and $\gamma_{q,j}$ reconfigure the metric space to capture both sharp local gradients and smooth global structures. The multi-scale features are concatenated and integrated using fusion MLPs during both the encoding and decoding phases, enabling a comprehensive representation of complex multiphysics interactions.

C.2. Unified operator framework for temporal evolution

Following RIGNO (Mousavi et al., 2025), and consistent with the GAOT (Wen et al., 2025) setting, we reformulate time-dependent trajectory prediction as a unified lead-time operator learning problem. This transition allows the neural operator to process time-dependent data as a collection of generalized snapshots, effectively treating the lead-time prediction as a time-independent mapping task.

Specifically, for a discrete trajectory sampled at $\{t_j\}_{j=0}^{\bar{M}}$, we consider any pair of time instances (t_i, t_j) where $t_j = t_i + \tau$ and $\tau > 0$ denotes the lead time. To transform the evolution into a time-independent form, we augment the input condition $a(\boldsymbol{x})$ by incorporating the state at the current time t_i and the temporal coordinates. The augmented input feature $\tilde{a}$ is defined as:

$$\tilde{a}(\boldsymbol{x}) \coloneqq \{u(\boldsymbol{x}, t_i), a(\boldsymbol{x}, t_i), t_i, \tau\}, \quad \boldsymbol{x} \in \Omega \tag{38}$$

where $u(\boldsymbol{x}, t_i)$ serves as the initial condition for the subsequent evolution over the interval τ. Under this formulation, the trajectory operator $\mathcal{G}_{\text{td}}$ is simplified to a unified operator $\mathcal{G} : \tilde{\mathcal{A}} \to \mathcal{U}$, where $\tilde{\mathcal{A}}$ is the augmented input space.

We define three distinct prediction modes for the target quantity $y(\boldsymbol{x}) = \mathcal{G}(\tilde{a})(\boldsymbol{x})$:

- **Direct Prediction:** The model directly maps the augmented input to the future state:

$$\mathcal{G}(\tilde{a})(\boldsymbol{x}) = u(\boldsymbol{x}, t_i + \tau) \tag{39}$$

- **Residual Prediction:** The model learns the incremental change between time steps:

$$\mathcal{G}(\tilde{a})(\boldsymbol{x}) = u(\boldsymbol{x}, t_i + \tau) - u(\boldsymbol{x}, t_i) \tag{40}$$

- **Derivative Prediction:** The model predicts the average rate of change over time:

$$\mathcal{G}(\tilde{a})(\boldsymbol{x}) = \frac{u(\boldsymbol{x}, t_i + \tau) - u(\boldsymbol{x}, t_i)}{\tau} \tag{41}$$

By adopting this unified framework, the neural operator is capable of handling irregularly sampled temporal data and zero-shot super-resolution in the time domain, utilizing the same architecture as the time-independent solver.

C.3. Model parameter configuration

For the latent space processor, we utilize a Transformer-based backbone with $d_{model} = 256$, consisting of 12 layers and 8 attention heads. The feed-forward expansion factor is set to 4. For geometry enhancement, we employ a NeRF-inspired positional encoding with $l = 1$ frequency levels, which includes the original input coordinates to preserve linear spatial relationships.

The multi-scale kernel mechanism consists of $Q = 3$ independent branches. The decay coefficients α and scaling parameters are initialized as shown in Table C.2 and are optimized during training, allowing the model to adaptively learn the physical correlation scales.

C.4. Model parameters

Table C.2. IKNO implementation parameters and default settings used in all experiments.

Parameter	Default Value	Description
d_model	256	Transformer Hidden dimension
num_heads	8	Transformer Attention heads
num_layers	12	Transformer layers
ff_mult	4	Transformer FFN expansion ratio
dropout	0.0	Transformer Dropout rate
nerf_frequencies	1	Positional encoding frequencies
nerf_include_input	True	Include raw coordinates
nerf_log_sampling	True	Log-spaced frequencies
nerf_scale	1.0	Positional encoding scale
grid_size	Task-dependent	Latent grid resolution
grid_min	-1.0	Latent grid lower bound
grid_max	1.0	Latent grid upper bound
num_kernels	3	Number of kernels
kernel_alpha	$[-1, -1, -1]$	Initial decay coefficients
kernel_alpha_grad	True	Learn decay coefficients
init_scales_l1	Base $[1, 2, 4]$	Laplacian kernel scale base
init_scales_l2	Base $[1, 2, 4]$	Gaussian kernel scale base
scales_grad	True	Learn kernel scales

C.5. Latent grid settings

While the majority of model parameters remain fixed, we adjust the resolution of the latent grid G to accommodate the varying complexity and dimensionality of the benchmarks:

- **Time-independent Datasets:** We employ a 2D grid with a resolution of $[24, 24]$.
- **Time-dependent Datasets:** To capture temporal evolution, the grid resolution is increased to $[32, 32]$.
- **Industrial-scale Datasets:**For the *NASA CRM* dataset, the grid resolution is set to $[24, 24, 24]$.

D. Training details

D.1. Loss function

To train the neural operator within the unified framework, we utilize the **relative** L_2 **loss** as our optimization objective. Given a set of points $\mathcal{P} = \{\boldsymbol{x}^i\}_{i=1}^n$, let $\hat{y}(\boldsymbol{x}^i)$ be the model prediction and $y(\boldsymbol{x}^i)$ be the ground truth defined according to the prediction modes. The loss for a single sample is defined as:

$$\mathcal{L}(y, \hat{y}) = \frac{\sqrt{\sum_{i=1}^n \|y(\boldsymbol{x}^i) - \hat{y}(\boldsymbol{x}^i)\|_2^2}}{\sqrt{\sum_{i=1}^n \|y(\boldsymbol{x}^i)\|_2^2}} \tag{42}$$

For time-dependent trajectories, the total loss is averaged over all sampled time pairs (t_i, t_j) within the batch.

D.2. Training parameters and hardware

All models are optimized using the AdamW(Loshchilov & Hutter, 2017) optimizer with a cosine annealing learning rate scheduler. We apply gradient clipping(Pascanu et al., 2013) to ensure training stability, especially for complex time-

Table D.3. Training, validation, and test sizes of datasets used in this work.

Abbreviation	Train size	Val size	Test size
Poisson-C-Sines	2048	256	256
Poisson-Gauss	2048	256	256
Elasticity	1024	256	256
NACA0012	5000	256	256
NACA2412	5000	256	256
RAE2822	5000	256	256
NASA-CRM	105	0	44
NS-Gauss	1024	256	256
NS-PwC	1024	256	256
NS-SL	1024	256	256
NS-SVS	1024	256	256
CE-Gauss	1024	256	256
CE-RP	1024	256	256
Wave-Layer	1024	256	256
Wave-C-Sines	1024	256	256

dependent systems. All experiments are conducted on a single GPU. The time-dependent benchmarks are trained on a single NVIDIA L40S GPU, the time-independent benchmarks are trained on a single NVIDIA A100 Tensor Core GPU, and the *NASA CRM* benchmark is trained on a single NVIDIA H200 Tensor Core GPU.

D.3. Data partitioning and preprocessing

Dataset Splitting. Following the experimental setting of GAOT (Wen et al., 2025), we use the same dataset partitioning strategy for all shared benchmarks except *NASA CRM*. Specifically, the first N_{train} samples are used for training, the subsequent N_{val} samples are reserved for validation and hyperparameter search, and the final N_{test} samples, counted backward from the end of the dataset, are held out exclusively for final metric computation. Since the *NASA CRM* benchmark does not provide a validation split, only on this dataset do we use the training loss for hyperparameter selection.

Hyperparameter Search. We only search the initial kernel scale hyperparameter. Starting from the base scale vector $[1, 2, 4]$, we evaluate multiplicative factors in $\{1, 2, 4\}$, corresponding to candidate initial scale vectors $[1, 2, 4]$, $[2, 4, 8]$ and $[4, 8, 16]$. Each candidate is trained for 20% of the total training epochs, after which we select the candidate with the lowest validation loss and continue training it for the remaining epochs. Since *NASA CRM* has no validation split, we use the training loss for this selection step.

Data Pre-processing and Sampling. For datasets labeled as *grid domain* in Table D.3, we transform them into an *arbitrary domain* format to test the discretization-invariance of our operator. We follow the point cloud sampling strategy proposed in GAOT (Wen et al., 2025). Specifically, since the datasets provided by RIGNO (Mousavi et al., 2025) already contain shuffled physical coordinates, we directly sample the first 9,216 points for each instance to construct the arbitrary point cloud input.

Normalization. To stabilize the training process, we apply Z-score normalization(Altman & Bland, 2005) to both the input features and target outputs. For any scalar or vector field v, the normalized field $\hat{v}$ is computed as:

$$\hat{v}(\boldsymbol{x}) = \frac{v(\boldsymbol{x}) - \mu_v}{\sigma_v + \epsilon} \tag{43}$$

where μ_v and σ_v are the mean and standard deviation calculated across the training set, respectively, and $\epsilon = 10^{-10}$ is a small constant for numerical stability.

Temporal Data Handling. For time-dependent PDEs, we adopt the *all2all* training strategy introduced in Poseidon(Herde et al., 2024). For a given trajectory with time steps $\{t_0, t_1, \ldots, t_m\}$, we generate all possible pairs (t_i, t_j) where $t_j = t_i + \tau$ and $\tau > 0$. Each pair constitutes a single training sample with current time t_i and lead time τ as additional input scalars. To ensure consistent feature scaling, these temporal coordinates (t_i and τ) also undergo independent Z-score normalization

based on their distributions in the training set.

Furthermore, when employing the Residual or Derivative prediction modes, the target quantity $y(\boldsymbol{x})$ is fundamentally transformed from the original state distribution. Consequently, we recompute the normalization statistics specifically for these Residual-based or Derivative-based targets. This ensures that the loss function remains well-conditioned and that the model's output range is appropriately scaled to the transformed label space.

E. Statistical robustness

To evaluate the sensitivity of IKNO-TP to random initialization and stochastic training effects, we repeat representative experiments on four datasets with five random seeds. Table E.4 reports the mean and sample standard deviation of the median relative L_1 error (%).

Table E.4. Statistical robustness of IKNO-TP over five random seeds. We report the mean and sample standard deviation of the median relative L_1 error (%).

DATASET	MEAN $\pm$ STD.
POISSON-C-SINES	1.01 ± 0.02
ELASTICITY	0.97 ± 0.02
POISSON-GAUSS	0.26 ± 0.01
NS-SL	1.12 ± 0.03

The standard deviations are small across all four datasets, indicating that IKNO-TP is stable with respect to random seed variation and that the reported performance is not driven by a single favorable run.

F. Additional ablation studies

In this section, we conduct a series of ablation experiments to evaluate the impact of the training objectives, latent grid configurations, and the proposed infinite kernel mechanism on the performance of the IKNO model.

F.1. Impact of temporal prediction modes

For time-dependent datasets, we investigate three distinct training formulations: *Direct*, *Residual*, and *Derivative* prediction. While previous studies in neural operators have explored these variations, we re-examine their efficacy within the IKNO framework. As summarized in Table F.5, the **Derivative** mode consistently yields the lowest relative error across all benchmarks, whereas the other two modes exhibit varying degrees of performance.

Table F.5. Comparison of temporal prediction modes. We report the **median relative L_1 error** (%) for two time-dependent datasets. The Derivative mode demonstrates the highest precision.

DATASET	DIRECT	RESIDUAL	DERIVATIVE
WAVE-LAYER	8.40	8.66	**8.16**
CE-GAUSS	9.77	9.73	**9.52**

We hypothesize that the superiority of the derivative mode stems from the explicit normalization by the time interval τ. By predicting $\frac{u(t+\tau)-u(t)}{\tau}$, the model effectively amplifies the learning signal for small τ, preventing the gradients from vanishing and ensuring numerical stability during training. However, it is worth noting that the performance gap between these modes remains relatively narrow, suggesting that IKNO is robust to the choice of optimization target.

F.2. Scalability of latent grid resolution

We analyze the sensitivity of the model to the latent grid size L, defined as the per-axis resolution of the latent grid G, so that the grid contains L^d points in total, where d is the spatial dimension. Across four representative datasets, Figures 4 and 3 illustrate the performance trends for time-independent and time-dependent PDEs, respectively. In time-independent tasks, increasing the latent grid resolution leads to marginal performance gains. In contrast, for time-dependent systems, the model exhibits a critical dependence on the latent capacity. At low resolutions, the model fails to converge or produces high

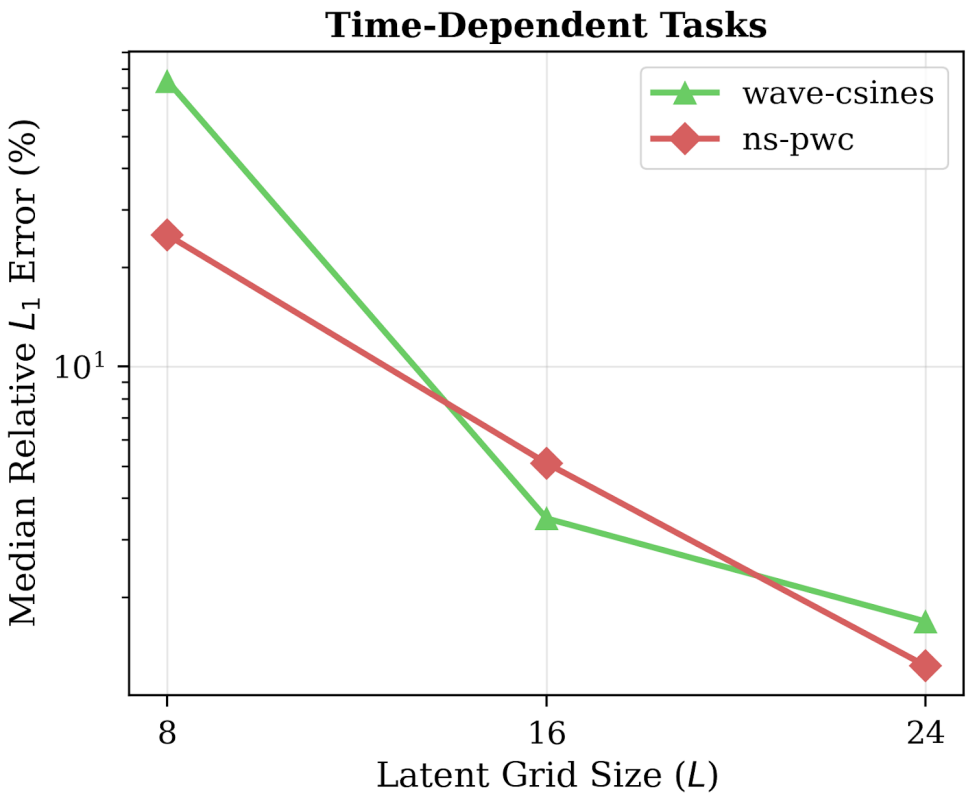

Figure 3. Ablation study on the latent grid size L (per-axis resolution of the latent grid G) on time-dependent datasets. For instance, $L = 32$ corresponds to a 32×32 latent grid.

errors, whereas increasing the grid size results in a dramatic improvement in accuracy. This suggests that time-dependent trajectories contain higher-frequency components that necessitate a denser latent representation to be captured effectively.

F.3. Necessity of the infinite kernel mechanism

To verify the importance of the infinite-order kernel formulation, we compare our IKNO encoder against two degraded alternatives that ablate different aspects of it.

First-order Kernel. The first alternative replaces the infinite-order resolvent in the encoder with its p-th order truncation of the geometric series in Eq. (12). In our ablation we set $p = 1$, which, mirroring the per-latent-point form of the Cross-Attention baseline below, yields the encoding

$$h(\boldsymbol{y}^j) = \sum_{i=1}^{\bar{N}} k(\boldsymbol{y}^j, \boldsymbol{x}^i)\, v(\boldsymbol{x}^i) + \alpha \sum_{l=1}^{M} \sum_{i=1}^{\bar{N}} k(\boldsymbol{y}^j, \boldsymbol{y}^l)\, k(\boldsymbol{y}^l, \boldsymbol{x}^i)\, v(\boldsymbol{x}^i), \tag{44}$$

where $\{\boldsymbol{y}^l\}_{l=1}^{M}$ are the latent grid points and $\{\boldsymbol{x}^i\}_{i=1}^{\bar{N}}$ the input points. Equivalently, in matrix form this reads $\boldsymbol{V}_G^{(1)} = (\boldsymbol{I} + \alpha \boldsymbol{K}_{GG})\, \boldsymbol{K}_{GP} \boldsymbol{V}_P$, with $\boldsymbol{K}_{GG}$ the kernel Gram matrix on the latent grid; for general p this generalizes to $\boldsymbol{V}_G^{(p)} = (\boldsymbol{I} + \alpha \boldsymbol{K}_{GG} + \cdots + \alpha^p \boldsymbol{K}_{GG}^p)\, \boldsymbol{K}_{GP} \boldsymbol{V}_P$. Thus the $p = 1$ setting combines the single-hop kernel aggregation used in prior neural operators with a single extra hop of propagation on the latent grid, and discards all higher-order propagation enabled by the infinite-order resolvent.

Cross-Attention. The second alternative replaces the kernel-based encoder with a standard Cross-Attention mechanism. In this setup, we treat the input coordinates $\boldsymbol{x} \in \Omega$ as queries and the latent grid coordinates $\boldsymbol{z} \in \mathcal{Z}$ as keys; the encoding for a

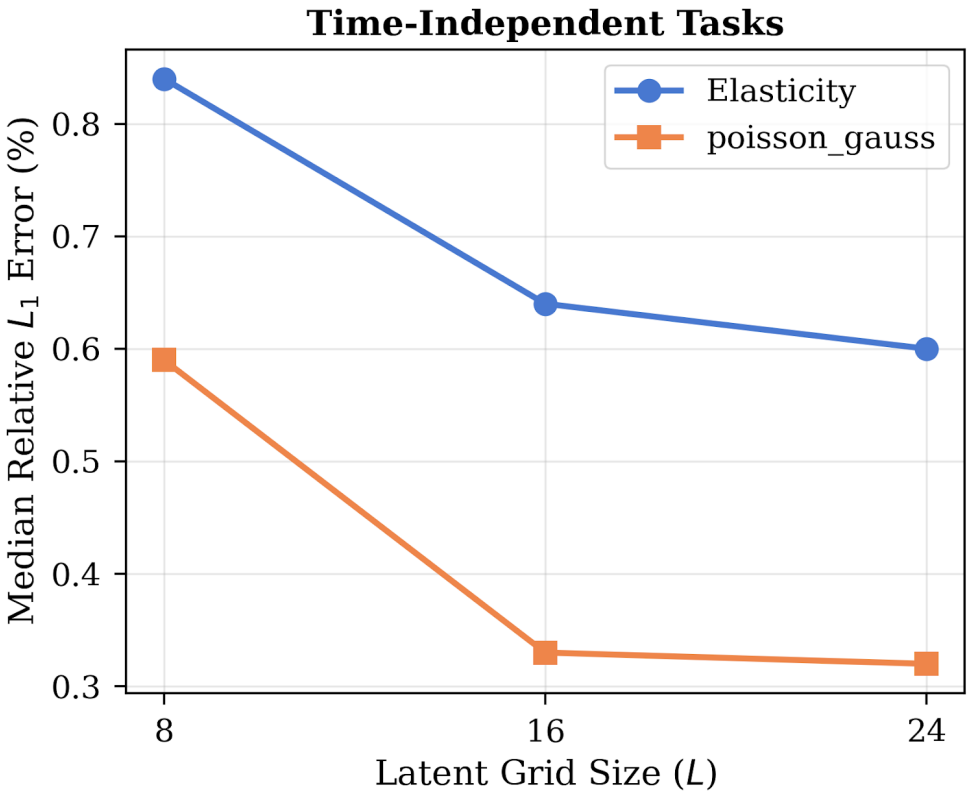

Figure 4. Ablation study on the latent grid size L (per-axis resolution of the latent grid G) on time-independent datasets. For instance, $L = 24$ corresponds to a 24×24 latent grid.

latent point $\boldsymbol{z}_k$ is

$$h(\boldsymbol{z}_k) = \sum_{i=1}^{N} \text{Softmax}\left(\frac{\phi_Q(\boldsymbol{x}_i)\phi_K(\boldsymbol{z}_k)^\top}{\sqrt{d}}\right)\phi_V(u(\boldsymbol{x}_i)), \tag{45}$$

where ϕ_Q, ϕ_K, ϕ_V are shallow MLP projections. This replacement eliminates the continuous kernel integration entirely, reducing the mapping to a discrete attention-based interpolation.

Table F.6. Ablation on the encoder mechanism: median relative L_1 error (%) of IKNO-Vanilla, IKNO-TP, the first-order kernel variant ($p = 1$), and cross-attention. Bold marks the best result per column.

Model	Elasticity	Poisson-Gauss	Wave-C-Sines	NS-PwC
IKNO-Vanilla	0.93	0.34	2.86	0.83
IKNO-TP	0.96	**0.26**	**1.70**	**0.63**
First-order kernel	1.09	0.76	29.38	1.12
Cross-Attention	**0.78**	1.24	85.72	62.87

The results in Table F.6 show that replacing the kernel encoder with Cross-Attention can be competitive on the simpler *Elasticity* benchmark, but it does not provide a robust encoder across tasks. On *Poisson-Gauss* and the two time-dependent benchmarks, IKNO-TP substantially improves over Cross-Attention; the gap is especially large on *Wave-C-Sines* and *NS-PwC*, where Cross-Attention yields much higher errors. The first-order kernel variant further isolates the contribution of the infinite-order expansion: although the $p = 1$ truncation preserves a kernel-based latent-grid mapping, it discards higher-order propagation and therefore remains markedly worse than IKNO-TP on the challenging time-dependent problems. Together, these comparisons indicate that the infinite-order kernel mechanism is not merely an architectural detail, but a key component for robustly capturing complex, multi-scale operators.

G. Computational efficiency

We further investigate the computational cost of IKNO and verify that the Kronecker-structured fast-computation scheme developed in Sec. 3.2.4 preserves the expressivity of the infinite-order formulation without introducing runtime overhead. All measurements in Table 5 are taken on a single NVIDIA L40S GPU, using the *Wave-C-Sines* benchmark with identical training settings across all methods. We report peak GPU memory and wall-clock time per epoch, and compare against two representative baselines (GAOT and Transolver), as well as an ablation in which the infinite-order resolvent of IKNO is evaluated by explicit $M \times M$ matrix inversion instead of the Kronecker-structured scheme.

Table 5 shows that IKNO not only matches but outperforms both baselines on efficiency: relative to GAOT, IKNO consumes roughly 24% less GPU memory and runs about $1.7\times$ faster per epoch, while against Transolver it is on par in memory and $\sim 15\%$ faster. The contrast with the IKNO (direct inverse) ablation is particularly revealing: replacing the Kronecker-structured scheme of Sec. 3.2.4 with direct matrix inversion keeps memory usage roughly unchanged but inflates the per-epoch wall-clock time from 281s to 1539s – a $5.5\times$ slowdown. This empirically confirms the $O(N^{3d}) \to O(d \cdot N^3)$ complexity reduction predicted in Sec. 3.2.4, and demonstrates that the practical appeal of the infinite-order kernel formulation relies critically on the Kronecker-based fast computation. The two IKNO variants (Vanilla and TP) exhibit statistically indistinguishable memory and time profiles; we therefore report a single representative IKNO row in the main table and omit a dedicated column per variant.

H. Evaluation metrics

To evaluate the performance of our proposed neural operator, we employ different evaluation protocols for non-industrial and industrial datasets to ensure a comprehensive assessment of accuracy and robustness.

H.1. Non-industrial datasets

For non-industrial benchmarks, we follow the evaluation protocol of GAOT (Wen et al., 2025) and report the median relative L_1 error as a percentage. This metric is computed on normalized data to ensure scale-invariance across different physical fields.

First, let $u_{i,c}$ and $\hat{u}_{i,c}$ denote the ground-truth and predicted fields for test sample i and solution component c, respectively. The relative L_1 error for that sample-component pair is defined as:

$$e_{i,c} = \frac{\|u_{i,c} - \hat{u}_{i,c}\|_1}{\|u_{i,c}\|_1}. \tag{46}$$

Consistent with GAOT, we take the median of the relative errors to reduce the sensitivity to large outliers. For PDEs with multiple output components, such as velocity and pressure fields, we first compute the median error for each component and then average these component-wise medians, which yields a single scalar score.

H.2. Temporal evaluation and iterative strategies

For time-dependent PDEs, we specifically report the performance at the **final time step** T. Since these systems often require iterative rollouts to reach the terminal state, errors inevitably accumulate over time. Consequently, the final step metric serves as the most demanding and representative indicator of a model's long-term stability and accuracy.

To reach the final state, we implement three distinct inference strategies:

- **Direct:** The model predicts the state at $t = T$ directly from the initial state u_0 in a single forward pass (i.e., $\tau = T$).
- **Auto-regressive (AR):** The model evolves the system using the smallest available time step Δt_{min}. While each individual step may be highly accurate, the large number of iterations required to reach T can lead to significant error accumulation.
- **Star (Strided):** We employ a hybrid approach using a larger stride (the second smallest time step Δt_{next}). The model iterates using this larger stride until the penultimate state, then determines whether to use a final Δt_{min} or Δt_{next} step to reach T. By reducing the total number of model calls while maintaining temporal resolution, this strategy often achieves the optimal balance between computational efficiency and cumulative accuracy.

We evaluate all three inference strategies separately and report the best performance, defined as the minimum error across the methods.

H.3. Industrial datasets

For industrial datasets, we adhere to standard regression metrics include Mean Squared Error and Mean Average Error computed on the normalized feature space.

These metrics provide a direct measure of the absolute error magnitude, which is often prioritized in engineering applications for safety and tolerance analysis.

I. Visualization of results

To provide additional qualitative insight into the predictions produced by our IKNO model, we include representative visualizations on the *Poisson-Gauss* and *Wave-C-Sines* benchmarks. The *NASA CRM* visualization is shown in Fig. 2.

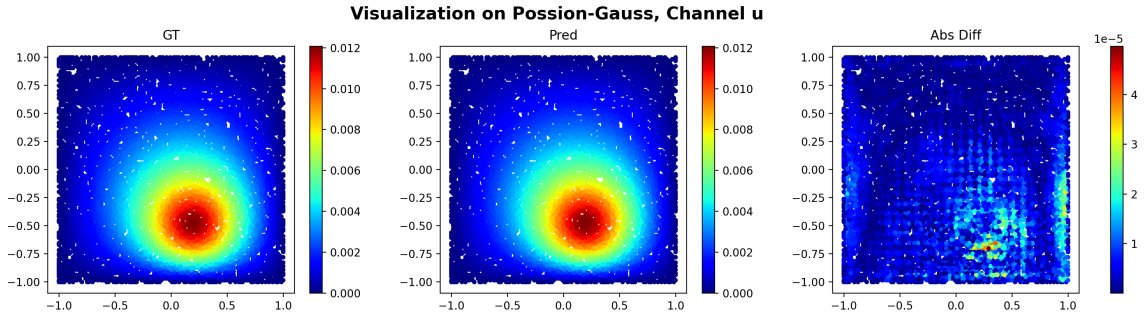

Figure 5. Visualization of IKNO predictions on the *Poisson-Gauss* benchmark.

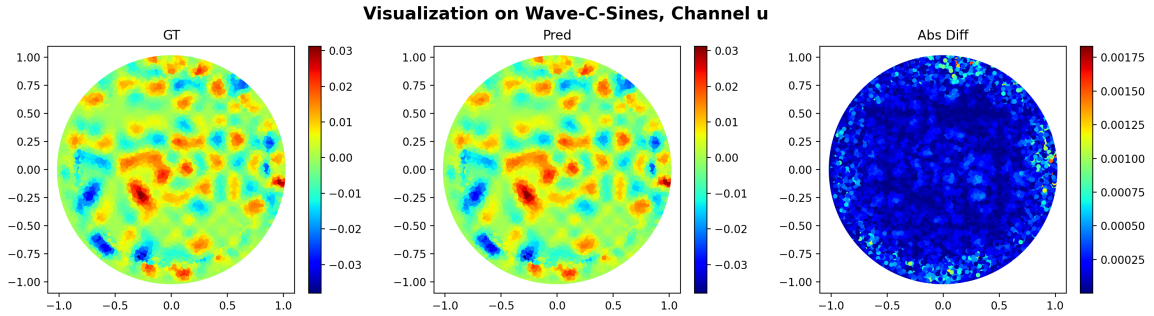

Figure 6. Visualization of IKNO predictions on the *Wave-C-Sines* benchmark.