GRAPH SPECTRAL NEURAL OPERATORS: LEARNING SPACE-TIME PDE SOLUTIONS ON ARBITRARY GEOMETRIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning solution operators for partial differential equations (PDEs) on arbitrary geometries remains a major challenge. Traditional spectral methods are limited to regular domains, while existing neural approaches often struggle to capture global spatiotemporal structures efficiently. We introduce the Graph Spectral Neural Operator (GSNO), a geometry-adaptive framework that combines graph spectral decompositions for spatial learning with real-valued Fourier transforms for temporal modeling. By learning a joint space–time spectral kernel, GSNO enables globally coherent and mesh-invariant operator learning without domain warping or heavy graph convolutional overhead. Across a variety of steady and time-dependent PDE problems, GSNO demonstrates improved accuracy compared to well-known neural operators on irregular geometries, along with reduced runtimes. These results suggest GSNO as a scalable and resolution-robust spectral operator, capable of generalizing to higher resolutions on complex geometries and contributing to scientific machine learning for physical systems.

1 INTRODUCTION

Many problems in science and engineering involve solving complex partial differential equations (PDEs) repeatedly for varying parameters. This is common in applications such as fluid dynamics, structural analysis, and geophysical modeling. These systems often require fine spatial and temporal resolution to capture multiscale dynamics, leading to extremely high computational costs. For instance, simulating unsteady flow in fractured porous media or tracking pollutant transport in irregular terrain may require solving forward PDE models thousands of times, which becomes infeasible using classical solvers (Palais & Palais, 2009).

Conventional solvers vs. data-driven approaches. Classical numerical techniques such as the finite difference method (FDM), finite volume method (FVM), and finite element method (FEM) rely on discretizing the domain and solving resulting algebraic equations to approximate PDE solutions (LeVeque, 2007; Quarteroni et al., 2010). Although these methods are known for their precision, they are often computationally intensive, especially as the resolution increases. Using coarse grids can reduce cost but typically sacrifices accuracy, while finer grids deliver better results at the expense of speed and scalability (Blechschmidt & Ernst, 2021). In contrast, data-driven models take a different route by learning a direct mapping from input parameters to solutions, using training data (Rudy et al., 2017; Grady et al., 2023; Xiao et al., 2024). Once trained, these models can produce predictions for new inputs with significantly less computational effort, often achieving speedups of several orders of magnitude over traditional solvers (Raissi et al., 2019; Kovachki et al., 2023; Raissi et al., 2017). Recent advances in machine learning have accelerated this trend by introducing *Neural Operators* (NOs)—models designed to learn mappings between infinite-dimensional function spaces. Unlike classical neural networks, which operate on fixed grids and struggle to generalize across resolutions, neural operators are mesh-invariant and can generalize to unseen discretizations. Popular frameworks such as DeepONets (Lu et al., 2021; Wang et al., 2021), Fourier Neural Operators (FNO) (Li et al., 2021), and Wavelet Neural Operators (WNO) (Tripura & Chakraborty, 2023) demonstrate the potential of this approach. FNO, in particular, enables efficient spectral learning via the Fast Fourier Transform and achieves state-of-the-art accuracy on several benchmark PDEs.

Limitations on irregular domains. Despite these successes, most neural operator architectures are designed for structured, grid-based domains. This significantly limits their applicability to real-world problems involving irregular geometries or unstructured meshes. Many engineering applications fundamentally require unstructured meshes to accurately represent complex geometries. Examples include modeling propagating cracks in structural analysis, capturing precise airfoil contours in aerodynamics, and simulating patient-specific anatomies in biomedical applications - cases where regular grid approximations would fail to capture critical physical details.

Related work for irregular geometries. Several recent methods have extended the neural operator framework to accommodate PDEs on irregular domains and unstructured meshes. The Multipole Graph Kernel Network (**MGKN**) (Li et al., 2020) introduces a graph-based operator learning framework that generalizes across irregular spatial domains. By constructing graphs over unstructured point clouds and applying a learned multipole kernel, MGKN captures both local and global interactions without relying on fixed meshes or uniform discretization.

However, MGKN is designed primarily for spatial operators and does not include an explicit temporal modeling component. The Geometry-aware Fourier Neural Operator (**Geo-FNO**) (Li et al., 2023) learns a geometric mapping to warp irregular domains into a structured latent grid, enabling conventional Fourier layers to operate in the latent space. This allows spectral learning on unstructured geometries. However, the method assumes the existence of a smooth and globally consistent mapping (a diffeomorphism) between the physical and latent domains. In domains with sharp boundaries, holes, or complex topologies, this assumption fails, leading to distortions and loss of critical geometric information. The Coordinate-based Radial-basis Latent operator (**CORAL**) (Serrano et al., 2023) proposes a mesh-free operator learning framework that represents functions using coordinate-based implicit neural networks. By embedding input–output mappings in a continuous latent space, CORAL approximates function values without requiring structured meshes, offering strong flexibility and resolution generalization. However, by abstracting away the underlying spatial structure, CORAL cannot leverage geometry-aware priors such as graph Laplacians or spectral bases. This limits its ability to capture long-range spatial dependencies and reduces generalization to unseen mesh topologies. The General Neural Operator Transformer (**GNOT**) (Hao et al., 2023) is a transformer-based framework targeting three key challenges: irregular meshes, multiple input functions, and multi-scale solution behavior. It introduces heterogeneous normalized attention (HNA) to provide a unified interface for encoding diverse inputs such as boundary conditions, source terms, and global parameters. In addition, GNOT employs a geometric gating mechanism inspired by domain decomposition, which softly partitions the domain into subregions and routes information through specialized subnetworks. While these mechanisms enhance flexibility and multi-scale modeling capacity, the approach relies heavily on learned gating and attention, increasing computational cost and reducing interpretability. Furthermore, GNOT does not employ an explicit spectral formulation in either space or time, instead depending on large-scale attention blocks to approximate operator mappings. This limits the integration of physics-informed priors and makes performance sensitive to training scale and data availability. The Spatio-Spectral Graph Neural Operator (**Sp^{2}GNO**) (Sarkar & Chakraborty, 2025) combines graph neural networks with Laplacian-based spectral filtering to capture both local and global spatial dependencies on irregular meshes. Its spatial representation is built on learnable GNN layers operating on a k-NN graph, making it heavily dependent on a complex, trainable architecture rather than a principled geometric prior. More critically, Sp^{2}GNO does not include a temporal spectral module. Instead, time is handled implicitly through stacked or recurrent GNN iterations, which restricts the model's ability to capture long-range temporal correlations and global frequency-domain structure. This autoregressive design also introduces error accumulation and prevents direct modeling of spatiotemporal interactions in the spectral domain. The Transformer-based solver (**Transolver**) (Wu et al., 2024) introduces Physics-Attention, which replaces pointwise attention with a grouping mechanism that clusters mesh points into learnable "slices" and encodes them as physics-aware tokens. This design improves efficiency by capturing global correlations without the quadratic cost of standard attention. However, the representation depends entirely on the learned slicing procedure, which enforces that similarity in the feature space corresponds to physical states. This dependence makes the method sensitive to design choices and obscures fine-scale structures when grouping is misaligned. Moreover, Transolver does not incorporate an explicit spectral operator in space or time, relying solely on token attention to approximate long-range dependencies. This reliance restricts robustness in multi-scale PDEs and excludes the physics-informed guarantees provided by spectral methods. The **AMG** framework (Li et al., 2025) introduces a multi-graph neural operator designed for PDEs on arbitrary geometries. It constructs three complementary graph types: a local graph to capture fine-scale details, a global graph for long-range interactions, and a physics graph to encode domain-specific physical attributes. These are integrated through a Graph Former architecture with dynamic graph attention, enabling adaptive multi-scale and physics-aware modeling. While effective across structured and unstructured meshes, AMG depends heavily on attention across multiple learned graph types, which increases computational complexity and reduces interpretability relative to parameter-free spectral operators. Moreover, although the physics graph introduces a useful inductive bias, it is not derived from explicit spectral principles, limiting the framework's ability to provide the physics-grounded guarantees offered by spectral-domain methods.

Our contributions. We propose the *Graph Spectral Neural Operator (GSNO)*, a unified neural operator architecture that performs explicit spectral learning across both space and time on irregular domains. In contrast to prior methods that rely on heavy graph neural networks, learned embeddings, or coordinate transformations, GSNO employs a parameter-free graph Laplacian basis constructed from Delaunay-triangulated point clouds. This yields a geometry-adaptive representation without additional learnable complexity. GSNO operates directly in the spectral domain by combining Laplacian eigenvectors for space with a real-valued FFT for time, learning a single complex-valued kernel to capture global spatiotemporal dependencies. A lightweight residual branch complements this design to refine local interactions, achieving a balance between expressivity and efficiency. This architecture provides three major advances. First, it achieves **efficiency and scalability** by removing recurrent temporal modules and learned spatial embeddings, leading to higher accuracy with fewer parameters and reduced training cost. Second, it delivers **superior performance** across a wide range of PDE benchmarks, consistently reaching state-of-the-art results while remaining robust to irregular geometries and multi-scale dynamics. Third, it ensures **zero-shot generalization**, naturally transferring across unseen meshes, resolutions, and discretizations without the need for retraining. Collectively, these contributions establish GSNO as the first operator framework to unify space–time spectral learning on arbitrary geometries, setting a new standard for principled, efficient, and mesh-invariant PDE

modeling. A detailed comparison of the designs and key differences of GSNO from other state-of-the-art models for problems on irregular, non-rectangular domains are presented in Appendix G.

2 METHODOLOGY

We present the Graph Spectral Neural Operator (GSNO), a neural operator architecture for learning solution operators of parametric partial differential equations (PDEs) on irregular domains. GSNO builds upon the general neural operator framework by incorporating spectral representations in both space and time: a graph Fourier transform for spatial decomposition on unstructured meshes, and a classical Fourier transform for temporal dynamics. Notably, the spatial graph structure is constructed from a Delaunay triangulation of the input mesh, and its associated spectral basis is precomputed and reused throughout training and inference for computational efficiency.

2.1 NEURAL OPERATOR FRAMEWORK

Let $D \subset \mathbb{R}^d$ be a bounded spatial domain. We define input and output function spaces as separable Banach spaces $\mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a})$ and $\mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$. The goal is to learn a nonlinear operator $G^\dagger : \mathcal{A} \to \mathcal{U}$, such as the solution operator of a parametric PDE. Given training pairs $\{(a_j, u_j)\}_{j=1}^N$ with $a_j \sim \mu$ and $u_j = G^\dagger(a_j)$, we learn a parametric model

$$G_\theta : \mathcal{A} \to \mathcal{U}, \quad \theta \in \Theta, \tag{1}$$

by minimizing an empirical loss over the data.

Neural operators are typically formulated as iterative architectures of the form:

$$v_{t+1}(x) = \sigma\left(W v_t(x) + (\mathcal{K}_\phi v_t)(x)\right), \quad t = 0, \ldots, T-1, \tag{2}$$

where $v_t : D \to \mathbb{R}^{d_v}$ is a hidden representation, $W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ is a pointwise linear map, σ is a nonlinear activation function (e.g., GELU), and $\mathcal{K}_\phi$ is a learnable global operator defined over function spaces. The input $a(x)$ is lifted to $v_0(x) = P(a(x))$, and the final output is $u(x) = Q(v_T(x))$, where P and Q are neural networks Kovachki et al. (2023); Behroozi et al. (2025).

2.2 GRAPH SPECTRAL NEURAL OPERATOR (GSNO)

GSNO instantiates the general neural operator framework by defining $\mathcal{K}_\phi$ through a spectral convolution over the joint spatial and temporal frequency domains. It is designed to handle both time-dependent and time-independent PDEs on irregular domains, where discretization points are nonuniform or unstructured. Let the spatial domain $D \subset \mathbb{R}^d$ be discretized as a point cloud $\{x_i\}_{i=1}^N$. For time-dependent problems, we consider input fields observed over an initial time window of length T_{in}, with the goal of predicting the solution over a future horizon of length T_{out}. The input signal is a spatiotemporal function $a : \{x_i\} \times [0, T_{\text{in}}] \to \mathbb{R}^{d_a}$, and the target output is $u : \{x_i\} \times [T_{\text{in}}, T_{\text{in}} + T_{\text{out}}] \to \mathbb{R}^{d_u}$. For time-independent PDEs, the formulation reduces to a purely spatial mapping, where the input field is $a : \{x_i\} \to \mathbb{R}^{d_a}$ and the output is $u : \{x_i\} \to \mathbb{R}^{d_u}$. In both settings, GSNO learns a parametric operator that maps the input field a to the corresponding solution u, leveraging spectral representations over both space and time. The spatial structure is encoded via a fixed graph Laplacian constructed from a Delaunay triangulation of the input mesh, enabling mesh-invariant modeling across irregular domains.

2.2.1 GRAPH CONSTRUCTION AND LAPLACIAN SPECTRAL BASIS

We represent the point cloud as an undirected geometric graph $G = (V, E)$ with $V = \{x_i\}$ and edges E constructed from a Delaunay triangulation. Compared to k-nearest neighbor graphs, Delaunay triangulation produces well-shaped, isotropic edge connectivity and avoids arbitrary metric-based thresholds, leading to better geometric fidelity in the learned graph Laplacian.

We define a symmetric weight matrix $A \in \mathbb{R}^{N_s \times N_s}$ using a Gaussian kernel:

$$A_{ij} = \begin{cases} \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma^2}\right), & \text{if } (x_i, x_j) \in E, \\ 0, & \text{otherwise}, \end{cases} \tag{3}$$

and compute the normalized graph Laplacian:

$$\tilde{L} = I - D^{-1/2} A D^{-1/2}, \quad D_{ii} = \sum_j A_{ij}. \tag{4}$$

Here, N_s is the number of spatial nodes. The eigendecomposition of $\tilde{L}$ gives:

$$\tilde{L} = \Phi \Lambda \Phi^\top, \tag{5}$$

where $\Phi \in \mathbb{R}^{N_s \times N_s}$ is orthonormal and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_{N_s})$ are the eigenvalues. We truncate to the first k_s eigenvectors $\Phi_{k_s} \in \mathbb{R}^{N_s \times k_s}$ to obtain the spatial Fourier basis. This basis is precomputed once and reused throughout training and inference.

The graph Fourier transform of a spatial function $f \in \mathbb{R}^{N_s \times d_v}$ is:

$$\hat{f} = \Phi_{k_s}^\top f, \quad f \approx \Phi_{k_s} \hat{f}. \tag{6}$$

A key advantage of GSNO is that this construction naturally extends across dimensions. In 2D, we rely on Delaunay triangulation, while in 3D we use Delaunay tetrahedralization to form the graph and compute the Laplacian. The resulting eigenbasis provides a geometry-aware spectral representation in both cases. Thus, GSNO avoids the added complexity of volumetric meshing and 3D boundary handling, while retaining the same spectral decomposition framework. This enables a unified design that scales from irregular 2D surfaces to complex 3D volumes without architectural changes.

2.2.2 Spectral Operator over Graph Space and Temporal Frequencies

At the heart of GSNO lies the operator $\mathcal{K}_\phi$, which performs learned convolution in the joint space-time frequency domain. This construction enables the model to capture long-range dependencies in both space and time while operating on irregular geometries. The operator works by projecting features into a spectral domain defined by the eigenbasis of a graph Laplacian (for space) and the discrete Fourier basis (for time), applying a learnable kernel in that domain, and mapping the result back to physical space-time.

Let $v_t \in \mathbb{R}^{N_s \times T \times d_v}$ denote the latent representation at a GSNO layer, where T is the number of time steps and, d_v is the number of channels.

The full process of applying the joint spectral operator $\mathcal{K}_\phi$ is outlined below.

1. Graph Fourier Transform (Spatial Projection): We begin by projecting v_t into the spatial frequency domain using a truncated eigenbasis $\Phi_{k_s} \in \mathbb{R}^{N_s \times k_s}$, obtained from the eigendecomposition of the normalized graph Laplacian. This gives:

$$\hat{v}_s(k_s, t, c) = \sum_{i=1}^{N_s} \Phi_{k_s}^\top(i, k_s) \cdot v_t(i, t, c) \qquad \text{which yields} \qquad \hat{v}_s = \Phi_{k_s}^\top v_t \in \mathbb{R}^{k_s \times T \times d_v} \tag{7}$$

2. Temporal Fourier Transform: We then apply the real-valued Discrete Fourier Transform (DFT) along the temporal axis:

$$\hat{v}_{st}(k_s, k_t, c) = \sum_{t=0}^{T-1} \hat{v}_s(k_s, t, c) \cdot e^{-2\pi i \cdot t \cdot k_t / T} \qquad \hat{v}_{st} = \mathcal{F}_t(\hat{v}_s) \in \mathbb{C}^{k_s \times k_t \times d_v} \tag{8}$$

where $k_t = \lfloor T/2 \rfloor + 1$ denotes the number of retained temporal frequency modes.

3. Spectral Convolution: In the joint space-time frequency domain, a learnable spectral kernel $R_\phi \in \mathbb{C}^{k_s \times k_t \times d_v \times d_v}$ is applied to mix channels via a linear transformation at each frequency pair:

$$\hat{v}'_{st}(k_s, k_t, l) = \sum_{j=1}^{d_v} R_\phi(k_s, k_t, j, l) \cdot \hat{v}_{st}(k_s, k_t, j), \tag{9}$$

resulting in the transformed tensor $\hat{v}'_{st} \in \mathbb{C}^{k_s \times k_t \times d_v}$.

4. Inverse Temporal Fourier Transform: We return to the spatial spectral domain by applying the inverse Fourier transform along time:

$$\hat{v}'_s(k_s, t, l) = \frac{1}{T} \sum_{k_t=0}^{k_t-1} \hat{v}'_{st}(k_s, k_t, l) \cdot e^{2\pi i \cdot t \cdot k_t / T} \qquad \text{or compactly} \qquad \hat{v}'_s = \mathcal{F}_t^{-1}(\hat{v}'_{st}) \in \mathbb{R}^{k_s \times T \times d_v} \tag{10}$$

5. Inverse Graph Fourier Transform (Spatial Reconstruction): Finally, the result is projected back to spatial domain using the truncated eigenbasis, reconstructing the updated latent representation:

$$v'(x_i, t, l) = \sum_{k_s=1}^{k_s} \Phi_{k_s}(i, k_s) \cdot \hat{v}'_s(k_s, t, l), \quad v' = \Phi_{k_s} \cdot \hat{v}'_s \in \mathbb{R}^{N_s \times T \times d_v}. \tag{11}$$

The spectral operator $\mathcal{K}_\phi$ combines these operations:

$$(\mathcal{K}_\phi v_t)(x) := \Phi_{k_s} \cdot \mathcal{F}_t^{-1}\left(R_\phi \cdot \mathcal{F}_t(\Phi_{k_s}^\top v_t)\right), \tag{12}$$

where R_ϕ governs interactions in the compressed joint frequency domain, forming the backbone of GSNO's spatiotemporal modeling on irregular geometries.

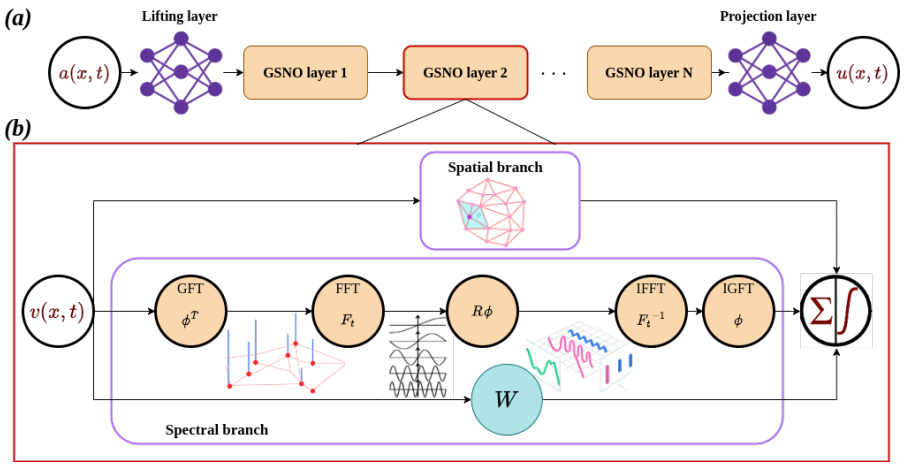


Figure 1: Overview of the GSNO architecture. **(a)** The input coefficient $a(x,t)$ is lifted into a high-dimensional space and passed through N GSNO layers. Each layer combines spatial graph Fourier transforms and temporal FFTs to model global dynamics, followed by a projection to obtain $u(x,t)$. **(b)** Inside each GSNO layer: input $v(x,t)$ is projected into space-time spectral domains, modulated by a learnable kernel R_ϕ, then reconstructed via inverse transforms. A local branch W captures fine-scale features before merging.

2.2.3 GSNO Layer Update Rule

Each GSNO layer updates latent features via two complementary paths:

- A **spectral path** that models non-local dependencies through joint graph–temporal frequency convolution.
- A **spatial path** that applies a localized residual map using a 1×1 convolution.

Given $v_\ell \in \mathbb{R}^{N\times T\times d_v}$, the next representation is

$$v_{\ell+1} = \sigma\left(\underbrace{W(v_\ell)}_{\text{Local Residual}} + \underbrace{\Phi_{k_s}\,\mathcal{F}_t^{-1}\big(R_\phi \cdot \mathcal{F}_t(\Phi_{k_s}^\top v_\ell)\big)}_{\text{Spectral Operator } \mathcal{K}_\phi v_\ell}\right), \tag{13}$$

where W is a learnable 1×1 convolution, σ a GELU activation, $\Phi_{k_s} \in \mathbb{R}^{N\times k_s}$ the truncated Laplacian eigenbasis, $\mathcal{F}_t$ and $\mathcal{F}_t^{-1}$ temporal Fourier transforms, and $R_\phi \in \mathbb{C}^{k_s\times k_t\times d_v\times d_v}$ the learnable spectral kernel.

This architecture captures **global interactions** through the low-rank operator $\mathcal{K}_\phi$, while the residual branch $W(v_\ell)$ ensures **local adaptivity** on irregular meshes. As visualized in Figure 1, the two branches are fused and passed through a nonlinear activation, balancing spectral expressivity with spatial detail. Spectral decomposition is a *one-time, offline pre-processing step*, performed once per mesh. We build the Laplacian from a Delaunay triangulation and compute the first k_s eigenvectors with iterative solvers such as LOBPCG, avoiding full diagonalization. The basis is saved and reused across training and inference, unlike learnable graphs that recompute edge weights every iteration. Hence, per-epoch cost scales only with k_s and k_t, not the full resolution N_s. This design yields high efficiency and scalability, as confirmed by GSNO's superior runtimes across benchmarks.

3 Numerical Experiments

We evaluate GSNO on six PDE systems spanning steady-state and time-dependent regimes: (i) steady-state Darcy flow; (ii) Euler equations over a 2D Airfoil, posed here as a single-step temporal forecast (one input step predicts the next); (iii) steady-state flow around the Shape-Net 3D car; (iv) unsteady Burgers' equation; (v) unsteady Navier–Stokes in vorticity form; and (vi) unsteady Shallow Water equations. Full descriptions of each PDE—including domain geometry, mesh specifications, initial-condition sampling, and data preprocessing—are provided in Appendices B.1–B.6. We benchmark GSNO against state-of-the-art neural-operator architectures; Classical numerical solvers (FEM/FDM), as well as publicly available datasets from the literature, are used solely to provide ground-truth training and evaluation data and are not treated as competing methods. Inputs and outputs are normalized using min–max scaling. Spectral configurations, channel widths, batch sizes, and other training hyperparameters—along with sensitivity analyses of key settings—are detailed in Appendix F. All experiments were run on a single NVIDIA V100 GPU (32 GB).

Benchmarks. We evaluate GSNO against seven neural-operator baselines—**DeepONet**, **MGKN**, **CORAL**, **Geo-FNO**, **AMG**, **Sp^{2}GNO**, **GNOT**, and **Transolver**—spanning diverse operator-learning strategies on irregular domains. All baselines are retrained on our datasets using identical training splits, mesh configurations, and

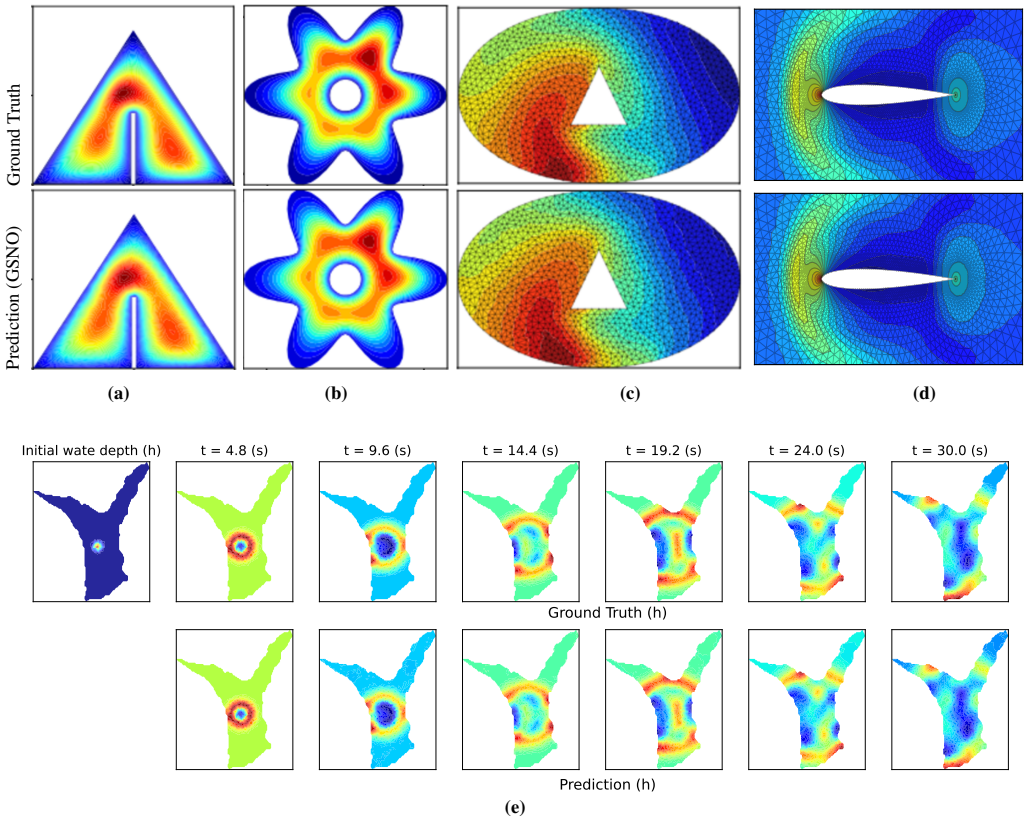


Figure 2: Selected results from PDE benchmarks solved using GSNO. Ground truth (top) and GSNO predictions (bottom) are shown for each case. Additional samples and temporal generalization results are provided in Appendix C. **(a) Darcy flow:** steady-state pressure field $u(x, y)$ on a mesh with $N_s = 1184$. **(b) Burgers' equation:** velocity field $u(x, y)$ at final time step $t = 1.00$, evaluated on a mesh with $N_s = 1168$. **(c) Navier–Stokes:** stream function $\psi(x, y)$ at $t = 10.0$; GSNO was trained on a mesh with $N_s = 972$ and evaluated zero-shot on a finer mesh with $N_s = 1903$. **(d) 2D Airfoil:** Pressure field on a mesh with $N_s = 5233$ **(e) Shallow water:** predicted evolution of water height $h(x, y, t)$ over 30 seconds in a realistic lake basin; GSNO was trained on $N_s = 1832$ and evaluated zero-shot on $N_s = 3663$.

optimizer settings. Hyperparameters follow the original papers, with light tuning for a fair comparison on our irregular benchmarks. Hyperparameters and training settings for the baseline models are provided in Appendix G.2.

3.1 Forward PDE Benchmarks

Tables 1 and 2 provide a comprehensive summary of relative L_2 errors across all benchmark settings: the former reports steady-state PDEs, and the latter summarizes time-dependent PDEs. For clarity, the best result is shown in **bold** and the second best is underlined. *Promotion* denotes the relative error reduction with respect to the second-best model, $1 - \frac{E_{\text{GSNO}}}{E_{\text{2nd-best}}}$ (reported as a percentage where indicated). Figure 2 presents selected visual comparisons of GSNO predictions against ground truth to be discussed below, with more comprehensive set of results provided in Appendix C.

Table 1: Relative L_2 error of models on steady-state PDEs at fixed resolution. In these steady-state cases, the model directly predicts the solution from the input field. (For more error metrics, see Appendix C.)

Model	(a) Darcy Flow ($N_s = 1184$)	(b) 2D-Airfoil ($N_s = 5233$)			(c) Shape-Net 3D Car ($N_s = 32186$)	
	Hydraulic head	Density	Pressure	Velocity_x	Velocity_y	Pressure	Velocity magnitude
CORAL	0.0664	0.0650	0.0610	0.0365	0.0410	0.1680	0.1750
Geo-FNO	0.0548	0.0580	0.0550	0.0320	0.0360	0.1560	0.1620
MGKN	0.0242	0.0500	0.0480	0.0215	0.0260	0.1350	0.1420
DeepONet	0.0312	0.0400	0.0370	0.0290	0.0310	0.1400	0.1480
AMG	0.0172	0.0021	0.0020	0.0014	0.0018	0.0878	0.0919
SP^{2}GNO	0.0150	0.0030	0.0028	0.0022	0.0025	0.1005	0.1102
GNOT	0.0118	0.0054	0.0049	0.0040	0.0040	0.1199	0.1206
Transolver	0.0142	0.0036	0.0032	0.0028	0.0035	0.0993	0.1208
GSNO	**0.0083**	**0.0012**	**0.0012**	**0.0009**	**0.0008**	**0.0712**	**0.0759**
Promotion (vs 2nd-best)	↓29.66%	↓42.86%	↓40.00%	↓35.71%	↓55.56%	↓18.91%	↓17.41%

Darcy Flow. We evaluate steady-state Darcy flow on a triangular domain with a notch (Figure B.4a), discretized via a Delaunay triangular mesh with N_s nodes. The task is to learn the solution operator mapping the diffusion coefficient field $a \in \mathbb{R}^{N_s}$ to the hydraulic head $u \in \mathbb{R}^{N_s}$, i.e., $G_\theta : \mathbb{R}^{N_s} \to \mathbb{R}^{N_s}$. GSNO achieves the lowest relative L_2 error (**0.0083**), representing a threefold reduction over the next best model, GNOT (0.0118), and a 29.6% improvement in error reduction (Table 1a). Qualitatively, GSNO captures the elevated hydraulic head around

Table 2: Relative L_2 error of models on time-dependent PDEs at fixed resolution. For these cases, the model takes T_{in} input steps to predict the next T_{out} steps. (For more error metrics, see Appendix C.)

Model	(a) Burgers' Equation ($N_s = 1168$)				(b) Navier–Stokes Equation ($N_s = 1244$) Temporal Config: $T_{\text{in}} \rightarrow T_{\text{out}}$				(c) Shallow Water Equation ($N_s = 1830$)			
	1→50	3→48	5→46	10→41	1→50	3→48	5→46	10→41	1→50	3→48	5→46	10→41
	Velocity magnitude				Vorticity (ω)				Water height (h)			
CORAL	0.1542	0.1308	0.1052	0.0966	0.1654	0.1412	0.1148	0.1070	0.1784	0.1534	0.1238	0.1162
Geo-FNO	0.1968	0.1718	0.1564	0.1410	0.2056	0.1790	0.1614	0.1512	0.2112	0.1848	0.1650	0.1526
MGKN	0.0876	<u>0.0686</u>	0.0612	0.0562	<u>0.0964</u>	<u>0.0770</u>	0.0654	0.0606	<u>0.1128</u>	<u>0.0876</u>	0.0724	0.0668
DeepONet	0.1146	0.1004	0.0894	0.0846	0.1250	0.1096	0.0958	0.0916	0.1284	0.1128	0.0982	0.0904
AMG	0.0812	0.0694	0.0570	0.0540	0.1060	0.0780	0.0580	0.0540	0.1210	0.0890	0.0692	0.0630
GNOT	0.1301	0.1232	0.0907	0.0855	0.1876	0.1326	0.0912	0.0844	0.2150	0.1534	0.1093	0.0995
SP^{2}GNO	0.1054	0.0998	0.0736	0.0695	0.1533	0.1081	0.0744	0.0689	0.1752	0.1250	0.0891	0.0810
Transolver	<u>0.0806</u>	0.0763	<u>0.0564</u>	<u>0.0534</u>	0.1189	0.0836	<u>0.0575</u>	<u>0.0534</u>	0.1354	0.0965	<u>0.0689</u>	<u>0.0625</u>
GSNO	**0.0221**	**0.0213**	**0.0156**	**0.0148**	**0.0336**	**0.0237**	**0.0164**	**0.0152**	**0.0375**	**0.0268**	**0.0193**	**0.0174**
Promotion (vs 2nd-best)	↓72.58%	↓68.95%	↓72.34%	↓72.28%	↓65.15%	↓69.22%	↓71.48%	↓71.54%	↓66.76%	↓69.41%	↓71.99%	↓72.16%

the notch and the no-flow effect along irregular boundaries (Figures 2a and C.5). Across mesh resolutions, GSNO maintains consistent gains, with error reductions of up to $8\times$ compared to competing methods (Figure C.6). Finally, GSNO also provides the fastest per-epoch training on steady-state PDEs, with runtime speedups of up to $6\times$ (Figure C.7). Resource usage for this Darcy setup—*inference time per batch* and *peak GPU memory during training and inference*—is summarized in Table H.25, where GSNO attains the fastest inference while keeping memory comparable to the most efficient baselines.

2D Airfoil. We evaluate GSNO on unsteady compressible Euler flow around a 2D airfoil (Figure B.4b). The irregular domain is discretized with an unstructured mesh containing $N_s = 5233$ nodes. The task is one-step prediction: given the flow state at time t, including density, velocity, and pressure fields, the operator predicts the next state at $t{+}1$. Formally, $G_\theta : \mathbb{R}^{N_s} \rightarrow \mathbb{R}^{N_s}$. As summarized in Table 1b, GSNO achieves the lowest relative L_2 errors across all state variables: **0.0012** for density, **0.0012** for pressure, **0.0009** for u_x, and **0.0008** for u_y. Compared to AMG model, the second-best baseline, this corresponds to error reductions of 42.9%, 40.0%, 35.7%, and 55.6%, respectively. Qualitatively, GSNO accurately reconstructs the near-field flow features, including pressure distribution along the airfoil surface and velocity separation in the wake (Figurs 2d and C.8). These results highlight GSNO's ability to generalize to highly irregular meshes and to simultaneously recover multiple flow variables with high fidelity. In addition to accuracy, GSNO demonstrates superior efficiency: it records the fastest per-epoch training time of **24s**, outperforming Transolver (27s) and GNOT (36s), and delivering speedups of up to $7.5\times$ relative to AMG (181s) (Table C.5).

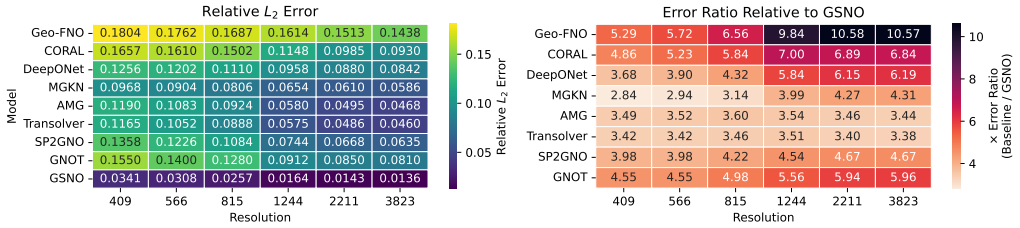
Shape-Net 3D Car. On ShapeNet car meshes ($\sim$32k unstructured points), the model maps geometry (coordinates, signed distance, normals) to *time-averaged* velocity and pressure. Figure C.9 illustrates streamlines of the velocity field from both the ground truth and GSNO prediction. The close alignment of flow patterns indicates that GSNO accurately captures the aerodynamic behavior around the car, showing strong qualitative agreement with the reference solution. As shown in Table 1c, GSNO delivers the best accuracy on ShapeNet Car, with relative L_2 errors of **0.0712** for pressure and **0.0759** for velocity magnitude. This marks improvements of 18.9% and 17.4% over the second-best baseline, AMG. Beyond accuracy, GSNO is also the most efficient: it has the fastest per-epoch training time of **24s**, against Transolver (27s) and GNOT (36s), and achieves up to a $7.5\times$ speedup over AMG (181s) (Table C.6).

Burgers' Equation. We evaluate the two-dimensional, time-dependent Burgers' equation on a flower-shaped domain with a central hole (Figure B.4c), simulated over 51 time steps. The velocity field $[u, v] \in \mathbb{R}^{N_s \times T \times 2}$ is predicted jointly, where the operator $G_\theta : \mathbb{R}^{N_s \times T_{\text{in}} \times 2} \rightarrow \mathbb{R}^{N_s \times T_{\text{out}} \times 2}$ maps the first T_{in} sequences to the next T_{out} snapshots. GSNO consistently outperforms all baselines across temporal configurations. In the most challenging split, $T_{\text{in}}{=}1 \rightarrow T_{\text{out}}{=}50$, GSNO attains 0.0221 versus 0.0806 for the second-best model, Transolver (Table 2a), i.e., over a 72% reduction. For $T_{\text{in}}{=}5 \rightarrow T_{\text{out}}{=}46$, GSNO further lowers the error to **0.0156**, maintaining error rates below 0.022 across all splits. Qualitatively, GSNO captures nonlinear transport and dissipation of both velocity components within the complex hollowed-out domain (Figures 2b and C.10). In multi-resolution tests, it achieves up to $10\times$ lower error than competing methods (Figure C.11). Moreover, GSNO provides the fastest per-epoch training despite temporal complexity, with runtime speedups of up to $3.5\times$ (Figure C.12).

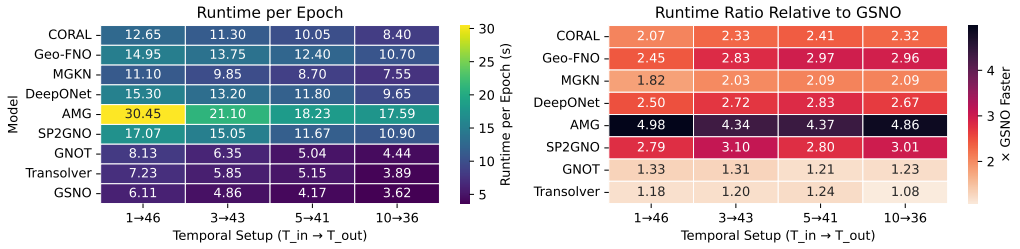
Navier-Stokes Equations. For the incompressible 2D Navier-Stokes equations, we consider an elliptical domain with a triangular cutout (Figure B.4e). The task is to predict the vorticity field $\omega \in \mathbb{R}^{N_s \times T \times 1}$, where the operator $G_\theta : \mathbb{R}^{N_s \times T_{\text{in}} \times 1} \rightarrow \mathbb{R}^{N_s \times T_{\text{out}} \times 1}$ transforms the first T_{in} input snapshots into forecasts of the following T_{out} snapshots. Across all temporal splits, GSNO yields the most accurate predictions (Table 2b). For instance, in the long-horizon case of $T_{\text{in}}{=}10 \rightarrow T_{\text{out}}{=}41$, GSNO achieves an error of **0.0152**, whereas the next best method, Transolver, records 0.0534—over 71% higher. These quantitative gains are matched by qualitative fidelity: GSNO reconstructs the roll-up of vortical structures and the onset of flow separation with high accuracy (Figures 2c and C.13). Its resolution generalization is also notable, delivering errors up to an order of magnitude lower than baselines on finer meshes (Figure 3a). Finally, in terms of efficiency, GSNO completes each training epoch up to $5\times$ faster than competing operators (Figure 3b). The computational profile for this NSE setup, summarized in

Table H.26, shows that GSNO achieves the fastest inference while maintaining a memory footprint on par with the most efficient baselines.

Shallow Water Equations. We test GSNO on the 2D Shallow Water Equations in conservative form, a standard model for flood inundation, applied to an irregular mesh extracted from Lake Union (Figure B.4f). The mesh contains $N_s = 3663$ nodes, and the simulation covers 30 seconds. The operator is tasked with advancing the water height field from initial conditions $h \in \mathbb{R}^{N_s \times T_{\text{in}}}$ to future states $h \in \mathbb{R}^{N_s \times T_{\text{out}}}$, i.e., $G_\theta : \mathbb{R}^{N_s \times T_{\text{in}}} \to \mathbb{R}^{N_s \times T_{\text{out}}}$. Across all temporal settings, GSNO delivers the most accurate results (Table 2c). In the long-horizon case of T_{in}=10 $\to$ T_{out}=41, it achieves a relative error of **0.0174**, whereas the second-best model, Transolver, records 0.0625—corresponding to a 72% improvement. Beyond raw numbers, GSNO reliably captures the propagation of wavefronts and their reflections against the irregular shoreline geometry (Figure 2e). Its advantage persists under mesh refinement, with errors up to 9$\times$ lower than baselines (Figure C.16). In terms of efficiency, GSNO also trains substantially faster, providing per-epoch speedups of up to 3.5$\times$ (Figure C.17).



(a) Accuracy vs. spatial resolution. All models are trained and evaluated on the same temporal setup $T_{\text{in}} = 5 \to T_{\text{out}} = 46$ and tested across increasing mesh resolutions. Left: relative L_2 error. Right: model-wise error ratio with respect to GSNO.



(b) Training runtime per epoch. All models are trained and evaluated on the same mesh resolution ($N_s = 1244$) and tested across different temporal configurations. Left: runtime in seconds. Right: slowdown factor relative to GSNO.

Figure 3: Performance comparison of neural operator models on the 2D Navier–Stokes equation using GSNO and baselines.

3.2 Zero-Shot Super-Resolution

GSNO supports zero-shot generalization across spatial resolutions through its spectral formulation. The key lies in using a geometry-aware Fourier basis $\Phi_{k_s} \in \mathbb{R}^{N_s \times k_s}$, derived from the eigendecomposition of the graph Laplacian. During training, latent features $v_t \in \mathbb{R}^{N_s \times T \times d_v}$ are projected into this basis via $\hat{v}_s = \Phi_{k_s}^\top v_t$, allowing GSNO to operate in the frequency domain. The spectral operator acts on this compressed representation as $(\mathcal{K}_\phi v_t)(x) = \Phi_{k_s} \mathcal{F}_t^{-1}(R_\phi \cdot \mathcal{F}_t(\Phi_{k_s}^\top v_t))$, where R_ϕ is a learnable spectral kernel and $\mathcal{F}_t$ is the temporal Fourier transform. Since this formulation depends only on the spectral representation, not on explicit coordinates, GSNO naturally generalizes to new meshes. At inference, we recompute the Laplacian and its eigenbasis $\Phi_{k_s}^{\text{test}}$, reusing the trained kernel R_ϕ without retraining. This makes GSNO inherently mesh-invariant and resolution-adaptive. Unlike methods that rely on coordinate encodings or grid mappings, GSNO ensures consistent behavior across discretizations. Figures 2(e) and C.13 demonstrate that GSNO trained on a coarse mesh with $N_s = 1832$ for the shallow water equations and $N_s = 972$ for the Navier–Stokes equations generalizes effectively to finer meshes with $N_s = 3663$ and $N_s = 1903$, respectively, without re-training. These results confirm GSNO's capability for zero-shot super-resolution across mesh resolutions.

3.3 Bayesian Inverse Problem for GSNO

We address the inverse problem of recovering the unknown Darcy coefficient field $a(x, y)$ from a single observed solution u_{obs}. To this end, we employ a function-space Markov Chain Monte Carlo (MCMC) method Geyer (1992); Geyer & Thompson (1995), specifically the Metropolis–Hastings algorithm Chib & Jeliazkov (2001), to sample from the posterior distribution over admissible coefficient fields. The forward map $a \mapsto u$ is approximated by a trained GSNO model, enabling fast and differentiable surrogate evaluations. The posterior is constructed using a Gaussian prior on a and a data misfit term based on the squared error between GSNO predictions and the observed solution. We perform 5,000 MCMC iterations, discarding the first 500 as burn-in, resulting in 5,000 forward passes

through GSNO—completed within minutes on GPU. Appendix D provides the full explanation and visualization of the posterior distribution.

3.4 Component-wise Ablation Analysis

To assess the contribution of each architectural component, we perform ablation experiments on steady-state (Darcy Flow) and time-dependent (Burgers') PDEs. Results show that removing the spectral kernel, residual path, or temporal FFT increases error and runtime. The largest degradation occurs when replacing the Laplacian eigenbasis with a random orthonormal basis, resulting in more than 5× increase in error on both steady and dynamic PDEs. This confirms that GSNO's use of a geometry-aware spectral basis is not just a design choice, but a critical enabler of generalization on irregular domains. It validates our hypothesis that spectral locality and mesh fidelity are essential for robust operator learning beyond Euclidean settings. Full ablation Results are reported in Appendix E.

4 Further Discussion and Conclusion

The performance of GSNO across a diverse set of PDE benchmarks highlights the benefits of leveraging problem structure through spectral representations. Rather than relying on mesh-specific encodings or densely parameterized architectures, GSNO operates in a compact space-time frequency domain aligned with the underlying discretization. This formulation enables not only high accuracy and scalability, but also flexible deployment across resolutions and geometries—without requiring model reconfiguration. In the following, we interpret these outcomes through the lens of spectral learning, compression-based efficiency, and resolution-adaptive generalization.

Spectral Learning Enables Accurate Operator Approximation. GSNO consistently demonstrates high predictive accuracy across all evaluated benchmarks, validating the strength of its joint space-time spectral formulation. Unlike coordinate-based multilayer perceptrons or graph message-passing architectures, GSNO projects spatial inputs onto a truncated graph Laplacian basis Φ_{k_s}, and applies a real-valued temporal Fourier transform to decompose dynamic behavior. This results in a compact and structured representation over which the learnable spectral kernel R_ϕ performs coherent filtering. The approach captures long-range spatial and temporal dependencies while preserving the geometric and physical characteristics of the solution. As demonstrated in Figures 2, C.5, C.10, and C.13, GSNO produces smooth and physically consistent predictions on complex and irregular domains, across both steady and unsteady PDE types.

Efficient Training via Low-Rank Spectral Compression.

GSNO's computational efficiency stems from its spectral operating domain and low-rank design. Instead of repeatedly working on the full-resolution mesh of size N_s, GSNO performs a *one-time, offline pre-processing step* to construct the graph Laplacian and compute only the first k_s low-frequency eigenvectors using the Locally Optimal Block Preconditioned Conjugate Gradient method. This truncated spectral basis is then stored and reused throughout training and inference, ensuring that the per-epoch cost depends only on k_s and k_t—not on the full input resolution N_s. By projecting inputs into a compact spectral subspace, GSNO filters a reduced set of spatial and temporal modes, avoiding the burden of high-resolution spatial convolutions and the depth cost of stacked message-passing layers. This spectral compression substantially lowers memory usage and compute requirements, while still retaining the dominant physical modes needed for accurate predictions. Empirical benchmarks (Figures C.7, C.12, and C.15) confirm that GSNO achieves faster per-epoch runtimes—up to 2–3× speedups over state-of-the-art baselines—while matching or surpassing their accuracy. This combination of low-rank design, one-time preprocessing, and mode compression makes GSNO both scalable and practical, enabling deployment to large-scale or resource-constrained PDE learning tasks without sacrificing fidelity.

Generalization Across Mesh Resolutions. A key strength of GSNO lies in its inherent capacity for resolution-independent inference. Rather than relying on coordinate encodings or mesh-specific graph constructions, GSNO learns directly in a spectral basis derived from the normalized graph Laplacian. During inference, the graph structure and its spectral basis can be recomputed for new spatial discretizations, while the learned spectral kernel R_ϕ remains fixed. This decoupling enables zero-shot generalization to unseen meshes without any retraining or finetuning. As discussed in Section 3.2 and illustrated in Figures C.6, C.11, C.14, and C.16, GSNO maintains strong accuracy across a wide range of mesh resolutions, demonstrating robustness and adaptability in both linear and nonlinear PDE regimes.

Conclusion. GSNO illustrates how joint space-time spectral modeling can offer a unified and scalable framework for learning solution operators across a wide range of PDE systems. By operating in a frequency domain that decouples spatial and temporal structure, the model avoids the limitations of coordinate-dependent and mesh-specific methods. This enables consistent performance without architecture-specific modifications, making GSNO well-suited for deployment in scientific workflows involving hybrid solvers, resolution-varying simulations, or surrogate-based acceleration. The results across accuracy, training efficiency, and generalization indicate that space-time spectral operators provide a principled foundation for reliable and adaptable operator learning in physics-driven applications.

REFERENCES

Abdolmehdi Behroozi, Chaopeng Shen, and Daniel Kifer. Sensitivity-constrained fourier neural operators for forward and inverse problems in parametric differential equations. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=DPzQ5n3mNm.

Jan Blechschmidt and Oliver G Ernst. Three ways to solve partial differential equations with neural networks—a review. *GAMM-Mitteilungen*, 44(2):e202100006, 2021.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Siddhartha Chib and Ivan Jeliazkov. Marginal likelihood from the metropolis–hastings output. *Journal of the American statistical association*, 96(453):270–281, 2001.

Charles J Geyer. Practical markov chain monte carlo. *Statistical science*, pp. 473–483, 1992.

Charles J Geyer and Elizabeth A Thompson. Annealing markov chain monte carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90(431):909–920, 1995.

Thomas J Grady, Ravi Khan, Mathias Louboutin, Ziyi Yin, Philipp A Witte, Ravi Chandra, et al. Model-parallel fourier neural operators as learned surrogates for large-scale parametric pdes. *Computational Geosciences*, 178: 105402, 2023. doi: 10.1016/j.cageo.2023.105402.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pp. 12556–12569. PMLR, 2023.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

Zhihao Li, Haoze Song, Di Xiao, Zhilu Lai, and Wei Wang. Harnessing scale and physics: A multi-graph neural operator framework for pdes on arbitrary geometries. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 729–740, 2025.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations, May 2021.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3): 218–229, 2021.

Richard S Palais and Robert Andrew Palais. *Differential equations, mechanics, and computation*, volume 51. American Mathematical Soc., 2009.

Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

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. ISSN 0021-9991. doi: 10/gfzbvx.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.

Subhankar Sarkar and Souvik Chakraborty. Spatio-spectral graph neural operator for solving computational mechanics problems on irregular domain and unstructured grid. *Computer Methods in Applied Mechanics and Engineering*, 435:117659, 2025.

Louis Serrano, Lise Le Boudec, Armand Kassaï Koupaï, Thomas X Wang, Yuan Yin, Jean-Noël Vittaut, and Patrick Gallinari. Operator learning with neural fields: Tackling pdes on general geometries. *Advances in Neural Information Processing Systems*, 36:70581–70611, 2023.

Tapas Tripura and Souvik Chakraborty. Wavelet Neural Operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, February 2023. ISSN 0045-7825. doi: 10.1016/j.cma.2022.115783.

Nobuyuki Umetani and Bernd Bickel. Learning three-dimensional flow for interactive aerodynamic design. *ACM Transactions on Graphics (TOG)*, 37(4):1–10, 2018.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605, 2021.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for pdes on general geometries. *arXiv preprint arXiv:2402.02366*, 2024.

Wang Xiao, Ting Gao, Kai Liu, Jinqiao Duan, and Meng Zhao. Fourier neural operator based fluid–structure interaction for predicting the vesicle dynamics. *Physica D: Nonlinear Phenomena*, 463:134145, 2024.

APPENDICES

A TABLE OF NOTATIONS

The definitions of key mathematical symbols used throughout this work are summarized in Table A.3.

Table A.3: Summary of notations used in the GSNO methodology.

Notation	Meaning
$D \subset \mathbb{R}^d$	Spatial domain
$\mathcal{A}, \mathcal{U}$	Input/output function spaces
$a \in \mathcal{A}$	Input field (e.g., coefficients, initial conditions)
$u \in \mathcal{U}$	Output field (e.g., PDE solution)
$G^\dagger$	True PDE solution operator
G_θ	Learnable neural operator with parameters θ
$v_t \in \mathbb{R}^{N_s \times T \times d_v}$	Latent representation at layer t
W	Learnable pointwise (local) linear operator
$\mathcal{K}_\phi$	Learnable global spectral operator
σ	Nonlinear activation function (e.g., GELU)
N_s	Number of spatial nodes
T	Number of temporal steps
$T_{\text{in}}, T_{\text{out}}$	Number of input and output time steps
$\Phi_{k_s} \in \mathbb{R}^{N_s \times k_s}$	Truncated graph Laplacian eigenbasis
$\hat{v}_s \in \mathbb{R}^{k_s \times T \times d_v}$	Spatial graph Fourier transform of latent features
$\hat{v}_{st} \in \mathbb{C}^{k_s \times k_t \times d_v}$	Joint spatiotemporal Fourier representation
k_s	Number of retained spatial frequency modes
k_t	Number of retained temporal frequency modes
$\mathcal{F}_t, \mathcal{F}_t^{-1}$	Temporal Fourier transform and its inverse
$R_\phi \in \mathbb{C}^{k_s \times k_t \times d_v \times d_v}$	Learnable spectral convolution kernel
$A \in \mathbb{R}^{N_s \times N_s}$	Graph adjacency matrix (Gaussian-weighted)
$D \in \mathbb{R}^{N_s \times N_s}$	Degree matrix
$\tilde{L} \in \mathbb{R}^{N_s \times N_s}$	Normalized graph Laplacian
$\Lambda \in \mathbb{R}^{N_s \times N_s}$	Diagonal matrix of Laplacian eigenvalues
$\hat{f}$	Graph Fourier coefficients of a signal f

B PDE SETUP AND DATA GENERATION

B.1 2D DARCY FLOW

We consider the steady-state Darcy flow equation defined on an irregular domain shaped like a triangle with a single notch (see Figure B.4a). The PDE is given by:

$$\begin{aligned} -\nabla \cdot (a(\mathbf{x}) \nabla u(\mathbf{x})) &= f(\mathbf{x}), && \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= 0, && \mathbf{x} \in \partial\Omega, \end{aligned}$$

where $u(\mathbf{x})$ denotes the hydraulic head (solution field), and the forcing term is fixed as $f(\mathbf{x}) = 1$.

The spatially-varying diffusion coefficient $a(\mathbf{x})$ is drawn from a pushforward Gaussian random field distribution, $a \sim \psi_{\#}\mathcal{N}(0,\ (-\Delta + 9I)^{-2})$, where ψ is a nonlinear transformation to ensure positivity, and the Laplacian is defined with zero Neumann boundary conditions. The GRF is discretized on unstructured triangular meshes generated using PyMesh. We solve the equation using a generalized finite difference method (GFDM) on multiple mesh resolutions. This setup enables us to assess the model's performance under complex geometries and varying spatial discretization scales. We curated 1,000 samples, allocated 600/200/200 to train/validation/test, and trained for 1,000 epochs.

B.2 EULER EQUATIONS OVER A 2D AIRFOIL

We model subsonic/transonic flow past a two–dimensional airfoil (see Figure B.4b) using the compressible Euler equations on an unstructured mesh (Li et al., 2023). The governing system is

$$\begin{aligned} \partial_t \rho + \nabla \cdot (\rho\, \mathbf{u}) &= 0, \\ \partial_t (\rho \mathbf{u}) + \nabla \cdot (\rho\, \mathbf{u} \otimes \mathbf{u} + p\, \mathbf{I}) &= 0, \end{aligned}$$

where ρ is density, $\mathbf{u} \in \mathbb{R}^2$ is velocity, and p is pressure.

The dataset is generated on irregular, unstructured meshes and evolved for 10 time steps. It contains $10{,}000$ training samples and $1{,}000$ samples each for validation and testing. For each sample, the targets are the solution fields $\{\rho,\ p,\ u_x,\ u_y\}$ defined on the provided mesh.

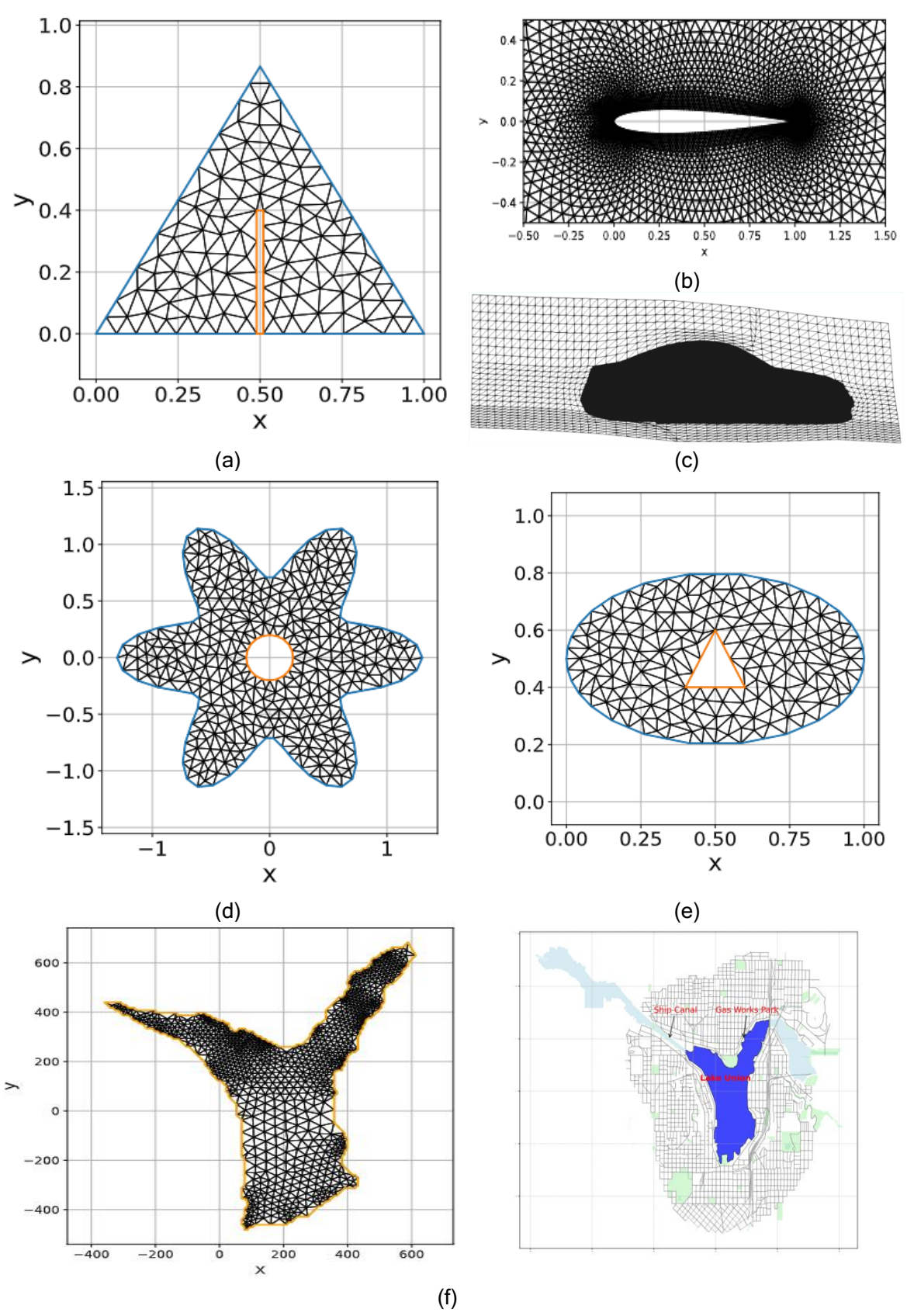


Figure B.4: Geometries used in the PDE benchmarks. Layout: (a) Darcy—triangle with a notch; (b) 2D Airfoil; (c) Shape-Net 3D Car (3D case; shown as a representative 2D cross-sectional slice for visualization); (d) Burgers—six-petal flower with a circular hole; (e) Navier–Stokes—ellipse with a triangular hole; (f) Shallow Water—Lake Union domain with a zoomed view of the southern inlet. All domains feature irregular boundaries and are discretized using unstructured triangular meshes.

B.3 SHAPE-NET 3D CAR

We utilize the Car dataset (see Figure B.4c) introduced by Umetani & Bickel (2018), which sources its base geometries from the ShapeNet Car category (Chang et al., 2015). Consistent with the procedure in Umetani & Bickel (2018), these geometries were manually modified to remove tires, spoilers, and side mirrors. The dataset was generated by obtaining time-averaged pressure and velocity fields from simulations of the Reynolds-Averaged Navier-Stokes (RANS) equations. These simulations incorporated a k–ϵ turbulence model, were stabilized using SUPG, and solved with a finite element method. A fixed inlet velocity of 20 m/s (72 km/h) was used, resulting in an approximate Reynolds number of 5×10^6. Each individual simulation required roughly 50 minutes to complete, with car surfaces discretized into 3.7k mesh points. From an initial pool of 889 instances, we selected the 611 water-tight shapes. This final dataset was then divided into 500 instances for training and 111 for validation.

B.4 2D BURGERS' EQUATION

We study the two-dimensional vector-valued Burgers' equation defined on an irregular domain shaped like a six-petal flower with a circular hole at its center (see Figure B.4d). The domain is embedded in the unit square and subject to no-slip boundary conditions, enforcing both velocity components to vanish along the boundary:

$$\begin{aligned} \partial_t \mathbf{u}(\mathbf{x},t) + \mathbf{u}(\mathbf{x},t)\cdot\nabla\mathbf{u}(\mathbf{x},t) &= \nu\Delta\mathbf{u}(\mathbf{x},t), && \mathbf{x}\in\Omega,\ t\in(0,T], \\ \mathbf{u}(\mathbf{x},0) &= \mathbf{u}_0(\mathbf{x}), && \mathbf{x}\in\Omega, \\ \mathbf{u}(\mathbf{x},t) &= \mathbf{0}, && \mathbf{x}\in\partial\Omega, \end{aligned}$$

where $\mathbf{x} = (x, y)$ denotes spatial coordinates and $\mathbf{u}(\mathbf{x}, t) = (u(\mathbf{x}, t), v(\mathbf{x}, t))$ is the velocity vector field. The viscosity is set to $\nu = 0.2$. Spatial discretization is carried out using a generalized finite difference method (GFDM) over unstructured triangular meshes generated with PyMesh. Temporal integration is performed using a fourth-order adaptive Runge–Kutta scheme implemented through the `torchdiffeq` package. The initial condition $\mathbf{u}_0(\mathbf{x})$ is sampled componentwise from a Gaussian random field with distribution $\mu = \mathcal{N}(0,\ 625(-\Delta + 25I)^{-2})$, where the Laplacian is defined with zero Neumann boundary conditions. The PDE is solved across multiple mesh resolutions to evaluate the model's generalization performance under varying discretization levels.

The Burgers' system was simulated for $T = 1.0s$ physical minutes, generating 51 temporal snapshots to capture nonlinear transport and dissipation. We prepared a dataset of 1,000 samples, divided into 600 for training, 200 for validation, and 200 for testing, and trained all models for 1,000 epochs.

B.5 2D NAVIER–STOKES EQUATION

We study the two-dimensional incompressible Navier–Stokes equations in vorticity–stream function formulation, defined on an irregular domain shaped like an ellipse with a triangular hole (see Figure B.4e). The governing equations are:

$$\begin{aligned} \partial_t w(\mathbf{x},t) + \mathbf{u}(\mathbf{x},t)\cdot\nabla w(\mathbf{x},t) &= \nu\Delta w(\mathbf{x},t) + f(\mathbf{x}), && \mathbf{x}\in\Omega,\ t\in(0,T], \\ -\Delta\psi(\mathbf{x},t) &= w(\mathbf{x},t), && \mathbf{x}\in\Omega, \\ \mathbf{u}(\mathbf{x},t) &= \nabla^{\perp}\psi(\mathbf{x},t) = \left(\frac{\partial\psi}{\partial y},\ -\frac{\partial\psi}{\partial x}\right), && \mathbf{x}\in\Omega, \\ w(\mathbf{x},0) &= w_0(\mathbf{x}), && \mathbf{x}\in\Omega. \end{aligned}$$

ν here is the viscosity. No-slip boundary conditions are imposed by enforcing $\mathbf{u} = 0$ and $\frac{\partial w}{\partial n} = 0$ on $\partial\Omega$. The initial vorticity $w_0(\mathbf{x})$ is sampled from a Gaussian random field with law $\mathcal{N}(0,\ (-\Delta + 49I)^{-2.5})$, where the Laplacian is equipped with zero Neumann boundary conditions. Spatial discretization is performed using a generalized finite difference method (GFDM) on unstructured triangular meshes generated via PyMesh. Time integration is carried out using a fourth-order adaptive Runge–Kutta scheme via the `torchdiffeq` package, consistent with the Burgers experiment.

The external forcing term is defined as:

$$f(\mathbf{x}) = 0.1\left(\sin(2\pi(x+y)) + \cos(2\pi(x-y))\right).$$

The Navier–Stokes solver was run for $T = 10s$ minutes of physical time, with 51 solution snapshots saved to resolve vortical dynamics and flow separation. From this, we generated a dataset of 1,000 samples, partitioned into 600 for training, 200 for validation, and 200 for testing, and trained all models for 1,000 epochs.

B.6 2D SHALLOW WATER EQUATIONS

We consider the two-dimensional nonlinear Shallow Water Equations (SWE) in conservative form, defined over an irregular domain shaped like the Lake Union (see Figure B.4f). The SWE system models the evolution of

water surface height and horizontal momentum under gravity and is widely used for simulating wave propagation, including tsunami and flood inundation scenarios. The governing equations are:

$$\begin{aligned}
\partial_t h(\mathbf{x},t) + \nabla \cdot (h\mathbf{v})(\mathbf{x},t) &= 0, && \mathbf{x} \in \Omega,\ t \in (0,T],\\
\partial_t (hv_x)(\mathbf{x},t) + \nabla \cdot \left(hv_x^2 + \frac{1}{2}gh^2,\ hv_x v_y \right)(\mathbf{x},t) &= 0, && \mathbf{x} \in \Omega,\ t \in (0,T],\\
\partial_t (hv_y)(\mathbf{x},t) + \nabla \cdot \left(hv_x v_y,\ hv_y^2 + \frac{1}{2}gh^2 \right)(\mathbf{x},t) &= 0, && \mathbf{x} \in \Omega,\ t \in (0,T],
\end{aligned}$$

where h denotes the fluid height, $\mathbf{v} = (v_x, v_y)$ is the velocity field, and $g = 8.81$ is the gravitational acceleration. The state variables are collectively represented as $\mathbf{u} = [h,\ hu,\ hv]^\top$, and the system is solved using a finite volume method with Rusanov flux.

The shallow water simulation setup mimics the propagation of surface gravity waves initiated by a localized disturbance—an abstraction often used to model real-world scenarios such as tsunami generation from undersea earthquakes or landslides. The simulation begins with a quiescent water column and introduces a spatially localized Gaussian perturbation in the height field:

$$\begin{aligned}
h(\mathbf{x},0) &= h_{\text{base}} + \text{max_field} \cdot \exp\left(-\frac{(x-x_c)^2 + (y-y_c)^2}{2\sigma^2} \right),\\
hu(\mathbf{x},0) &= 0, \quad hv(\mathbf{x},0) = 0, \quad \mathbf{x} \in \Omega,
\end{aligned}$$

where $(x_c, y_c) \in \Omega$ denotes the center of the perturbation—randomly sampled for each instance—and σ controls the spread of the Gaussian. We use fixed values $h_{\text{base}} = 5.0$, max_field $= 1.0$, and $\sigma = 20$. This setup leads to outward-propagating circular wavefronts, reminiscent of the early stages of tsunami evolution in enclosed basins or coastal zones.

Reflective (wall) boundary conditions are applied on $\partial\Omega$, enforcing zero normal velocity at the domain boundary:

$$\mathbf{v}_{\text{inv}} = \mathbf{v} - 2(\mathbf{v} \cdot \mathbf{n})\mathbf{n},$$

where $\mathbf{n}$ is the outward unit normal vector. This condition ensures zero penetration and free-slip behavior, making it appropriate for modeling wave reflection against rigid coastal boundaries or natural terrain features. The SWE system is discretized on an unstructured triangular mesh containing 3,663 nodes, representing the Lake Union geometry (see Figure B.4f). A finite volume method is used to solve the conservative form of the equations, with Rusanov flux applied at each face. Reflective wall boundary conditions are enforced on all domain boundaries. Temporal integration is performed using a fourth-order adaptive Runge–Kutta scheme implemented via the `torchdiffeq` package. The simulation spans over 30 physical minutes, during which 51 solution snapshots are saved to capture the wave propagation dynamics. This configuration allows us to simulate wave propagation and reflection in a closed, irregular lake. The dataset includes multiple Gaussian-perturbed initial conditions sampled on varying mesh resolutions, enabling mesh-invariant surrogate modeling and resolution-aware prediction benchmarks. For this problem, we generated a dataset of 1,000 samples, divided into 600 for training, 200 for validation, and 200 for testing, and trained all models for 1,000 epochs.

C Additional results

C.1 Additional results for Darcy Flow

Figure C.5 shows a sample prediction for the steady-state Darcy flow problem, comparing the input field, ground truth, and GSNO output. Figure C.6 reports the resolution-based generalization performance, highlighting GSNO's accuracy gains over baselines. Figure C.7 presents the training efficiency of all models across varying mesh resolutions.

Table C.4: Comparison of neural operator models on Darcy Flow ($N_s = 1184$).

Model	Relative L_2 Error	RMSE	MAE
CORAL	0.0664	4.82×10^{-4}	3.38×10^{-4}
Geo-FNO	0.0548	3.97×10^{-4}	2.91×10^{-4}
MGKN	0.0242	1.74×10^{-4}	1.43×10^{-4}
DeepONet	0.0312	2.60×10^{-4}	2.04×10^{-4}
AMG	0.0172	1.29×10^{-4}	9.83×10^{-5}
SP^2GNO	0.0150	1.12×10^{-4}	8.92×10^{-5}
GNOT	0.0118	8.86×10^{-5}	6.74×10^{-5}
Transolver	0.0142	1.07×10^{-4}	8.11×10^{-5}
GSNO (Ours)	**0.0083**	$\mathbf{1.84 \times 10^{-5}}$	$\mathbf{1.62 \times 10^{-5}}$

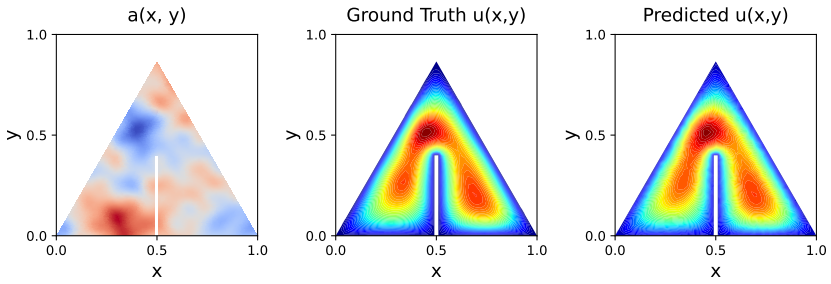


Figure C.5: Steady-state Darcy flow simulation on an irregular triangular domain with a notch. The first column shows the input diffusion field $a(x, y)$, the second column shows the ground truth hydraulic head u, and the third column presents GSNO predictions. The model is trained and tested on a mesh with $N_s = 1184$ points, highlighting GSNO's ability to accurately recover the solution from heterogeneous input fields.

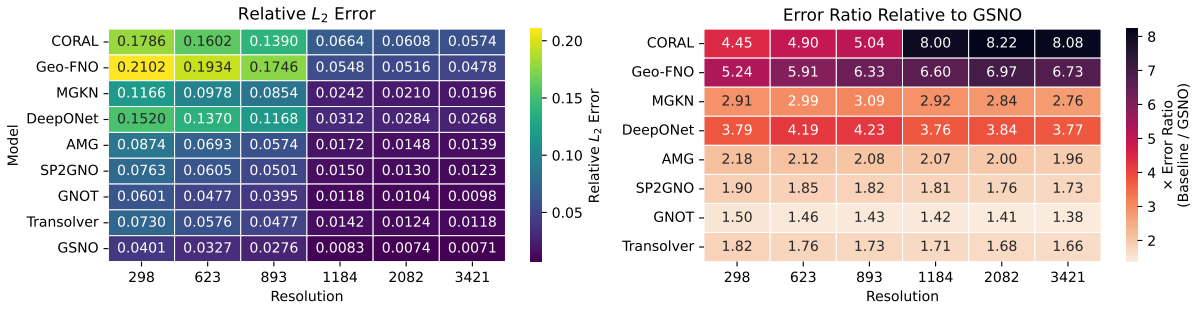


Figure C.6: Resolution-based generalization comparison for the steady-state Darcy flow problem. **Left:** Relative L_2 error across increasing mesh resolutions for all models. **Right:** Performance gap with respect to GSNO, shown as the ratio of each model's error to GSNO's at the same resolution. All models are trained and evaluated on identical unstructured meshes. GSNO demonstrates superior predictive accuracy across all resolutions, with error reductions of up to $8\times$ over competing methods.

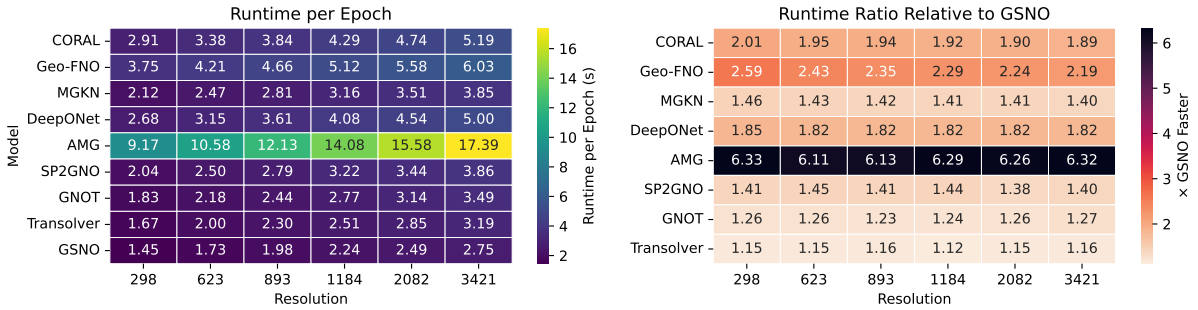


Figure C.7: Training efficiency of neural operator models on Darcy Flow across increasing spatial resolutions. **Left:** Average runtime per epoch (in seconds) across six mesh resolutions from 298 to 3421 nodes. **Right:** Slowdown relative to GSNO, computed as the ratio of each model's runtime to GSNO's at the same resolution. GSNO consistently achieves the fastest per-epoch training, showcasing its efficiency on steady-state PDEs.

C.2 Additional results for Euler equations over a 2D Airfoil

Figure C.8 shows a qualitative one-step example for the 2D Airfoil. Table C.5 reports the corresponding quantitative metrics—training time (s/epoch) and errors (Relative L_2, RMSE, MAE) for fluid quantities across all models.

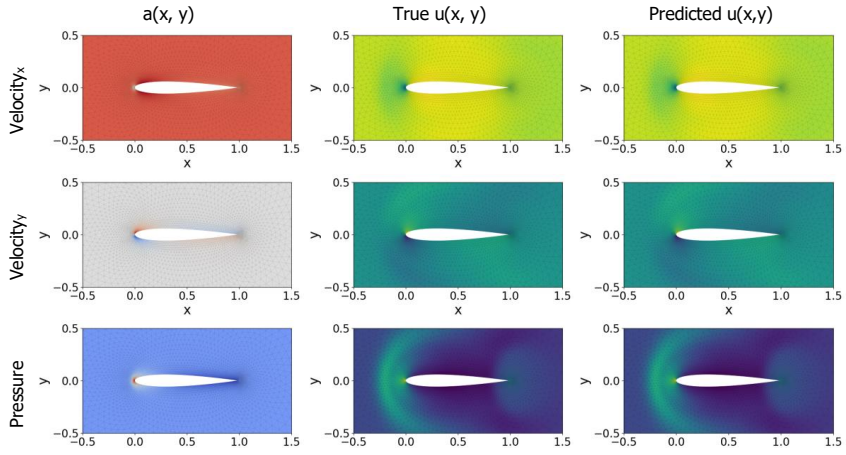


Figure C.8: 2D Airfoil (compressible Euler) — one-step prediction. Columns: (left) input state at time t, $a(x, y)$; (middle) ground truth at $t+1$, $u(x, y)$; (right) GSNO prediction at $t+1$. Rows: velocity u_x (top), velocity u_y (middle), and pressure p (bottom). All fields are on the same unstructured mesh with $N_s = 5233$; the white region indicates the airfoil.

Table C.5: Comparison of neural operator models on **2D-Airfoil** ($N_s = 5233$)

Model	Train (s/epoch)	Density Rel.L_2	Density RMSE	Density MAE	Pressure Rel.L_2	Pressure RMSE	Pressure MAE	Velocity_x Rel.L_2	Velocity_x RMSE	Velocity_x MAE	Velocity_y Rel.L_2	Velocity_y RMSE	Velocity_y MAE
CORAL	252	0.0650	1.49×10^{-2}	1.27×10^{-2}	0.0610	1.34×10^{-2}	1.13×10^{-2}	0.0365	4.93×10^{-3}	4.20×10^{-3}	0.0410	6.15×10^{-3}	5.12×10^{-3}
Geo-FNO	298	0.0580	1.33×10^{-2}	1.13×10^{-2}	0.0550	1.21×10^{-2}	1.02×10^{-2}	0.0320	4.32×10^{-3}	3.68×10^{-3}	0.0360	5.40×10^{-3}	4.50×10^{-3}
MGKN	264	0.0500	1.15×10^{-2}	0.975×10^{-2}	0.0480	1.06×10^{-2}	8.88×10^{-3}	0.0215	2.90×10^{-3}	2.47×10^{-3}	0.0260	3.90×10^{-3}	3.25×10^{-3}
DeepONet	312	0.0400	9.20×10^{-3}	7.80×10^{-3}	0.0370	8.14×10^{-3}	6.85×10^{-3}	0.0290	3.92×10^{-3}	3.34×10^{-3}	0.0310	4.65×10^{-3}	3.88×10^{-3}
AMG	672	0.0021	4.83×10^{-4}	4.09×10^{-4}	0.0020	4.40×10^{-4}	3.70×10^{-4}	0.0014	1.89×10^{-4}	1.61×10^{-4}	0.0018	2.70×10^{-4}	2.25×10^{-4}
SP^{2}GNO	189	0.0030	6.90×10^{-4}	5.80×10^{-4}	0.0028	6.20×10^{-4}	5.20×10^{-4}	0.0022	3.00×10^{-4}	2.50×10^{-4}	0.0025	3.80×10^{-4}	3.20×10^{-4}
GNOT	121	0.0054	1.24×10^{-3}	1.05×10^{-3}	0.0049	1.08×10^{-3}	8.97×10^{-4}	0.0040	5.40×10^{-4}	4.60×10^{-4}	0.0040	6.00×10^{-4}	5.00×10^{-4}
Transolver	108	0.0036	8.28×10^{-4}	7.02×10^{-4}	0.0032	7.04×10^{-4}	5.92×10^{-4}	0.0028	3.78×10^{-4}	3.22×10^{-4}	0.0035	5.25×10^{-4}	4.38×10^{-4}
GSNO (Ours)	96	**0.0012**	$\mathbf{2.76 \times 10^{-4}}$	$\mathbf{2.34 \times 10^{-4}}$	**0.0012**	$\mathbf{2.64 \times 10^{-4}}$	$\mathbf{2.22 \times 10^{-4}}$	**0.0009**	$\mathbf{1.21 \times 10^{-4}}$	$\mathbf{1.04 \times 10^{-4}}$	**0.0008**	$\mathbf{1.20 \times 10^{-4}}$	$\mathbf{1.00 \times 10^{-4}}$

C.3 Additional results for Shape-Net 3D Car

Figure C.9 illustrates a representative prediction from GSNO on the Car3D dataset, highlighting the reconstructed velocity streamlines and pressure distribution. The corresponding quantitative comparison is provided in Table C.6, which reports per-epoch training cost (s/epoch) together with error measures (Relative L_2, RMSE, MAE) for velocity and pressure across all competing models.

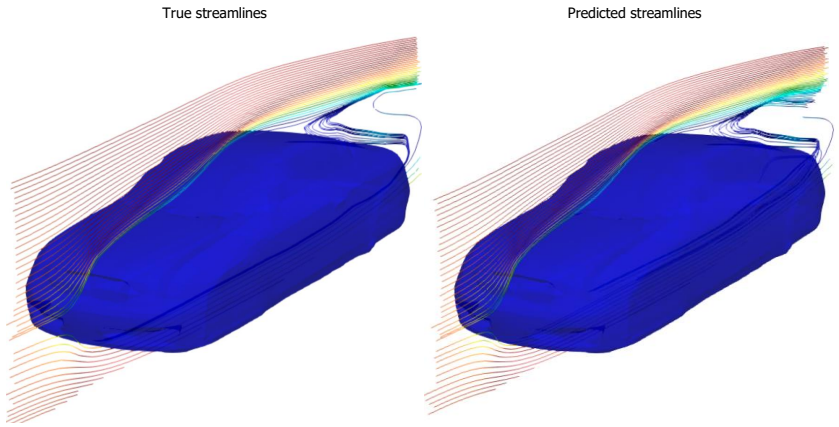


Figure C.9: Shape-Net 3D Car — flow streamlines. Left: reference simulation; right: GSNO prediction.

Table C.6: Comparison of neural operator models on **Shape-Net 3D Car** ($N_s = 32186$)

Model	Train (s/epoch)	Pressure Rel.L_2	Pressure RMSE	Pressure MAE	Velocity Rel.L_2	Velocity RMSE	Velocity MAE
CORAL	62	0.1680	4.20×10^{-2}	3.57×10^{-2}	0.1750	4.72×10^{-2}	4.02×10^{-2}
Geo-FNO	58	0.1560	3.90×10^{-2}	3.31×10^{-2}	0.1620	4.37×10^{-2}	3.72×10^{-2}
MGKN	67	0.1350	3.38×10^{-2}	2.87×10^{-2}	0.1420	3.83×10^{-2}	3.26×10^{-2}
DeepONet	87	0.1400	3.50×10^{-2}	2.98×10^{-2}	0.1480	4.00×10^{-2}	3.40×10^{-2}
AMG	181	0.0878	2.20×10^{-2}	1.87×10^{-2}	0.0919	2.48×10^{-2}	2.11×10^{-2}
SP^{2}GNO	48	0.1005	2.50×10^{-2}	2.13×10^{-2}	0.1102	2.95×10^{-2}	2.50×10^{-2}
GNOT	36	0.1199	3.00×10^{-2}	2.55×10^{-2}	0.1206	3.26×10^{-2}	2.77×10^{-2}
Transolver	27	0.0993	2.48×10^{-2}	2.11×10^{-2}	0.1208	3.26×10^{-2}	2.77×10^{-2}
GSNO (Ours)	24	**0.0712**	$\mathbf{1.78 \times 10^{-2}}$	$\mathbf{1.51 \times 10^{-2}}$	**0.0759**	$\mathbf{2.05 \times 10^{-2}}$	$\mathbf{1.74 \times 10^{-2}}$

C.4 Additional results for 2D Burgers' Equation

Figure C.10 presents GSNO predictions for the 2D Burgers' equation benchmark, showcasing both horizontal and vertical velocity components over time. Figures C.11 and C.12 provide a comparative analysis of generalization accuracy across resolutions and training efficiency under varying temporal settings.

Table C.7: Comparison of neural operator models on Burgers' Equation ($N_s = 1168$).

Model	Temporal Config: 1→50			Temporal Config: 3→48			Temporal Config: 5→46			Temporal Config: 10→41		
	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE
CORAL	0.1542	1.30×10^{-1}	9.47×10^{-2}	0.1308	1.05×10^{-1}	7.97×10^{-2}	0.1052	7.76×10^{-2}	6.15×10^{-2}	0.0966	6.88×10^{-2}	5.52×10^{-2}
Geo-FNO	0.1968	1.59×10^{-1}	1.10×10^{-1}	0.1718	1.32×10^{-1}	9.58×10^{-2}	0.1564	1.17×10^{-1}	8.71×10^{-2}	0.1410	1.02×10^{-1}	7.77×10^{-2}
MGKN	0.0876	5.86×10^{-2}	4.78×10^{-2}	0.0686	3.68×10^{-2}	3.10×10^{-2}	0.0612	3.09×10^{-2}	2.62×10^{-2}	0.0562	2.76×10^{-2}	2.35×10^{-2}
DeepONet	0.1146	8.56×10^{-2}	6.70×10^{-2}	0.1004	6.98×10^{-2}	5.59×10^{-2}	0.0894	5.79×10^{-2}	4.72×10^{-2}	0.0846	5.25×10^{-2}	4.32×10^{-2}
AMG	0.0812	5.45×10^{-2}	4.46×10^{-2}	0.0694	3.74×10^{-2}	3.15×10^{-2}	0.0570	2.88×10^{-2}	2.48×10^{-2}	0.0540	2.63×10^{-2}	2.25×10^{-2}
GNOT	0.1301	9.60×10^{-2}	7.60×10^{-2}	0.1232	8.90×10^{-2}	7.10×10^{-2}	0.0907	6.30×10^{-2}	5.05×10^{-2}	0.0855	5.75×10^{-2}	4.60×10^{-2}
SP^{2}GNO	0.1054	7.20×10^{-2}	5.85×10^{-2}	0.0998	6.60×10^{-2}	5.48×10^{-2}	0.0736	3.95×10^{-2}	3.30×10^{-2}	0.0695	3.50×10^{-2}	3.02×10^{-2}
Transolver	0.0806	5.40×10^{-2}	4.42×10^{-2}	0.0763	4.10×10^{-2}	3.44×10^{-2}	0.0564	2.85×10^{-2}	2.45×10^{-2}	0.0534	2.58×10^{-2}	2.20×10^{-2}
GSNO	**0.0221**	$\mathbf{1.21 \times 10^{-2}}$	$\mathbf{1.05 \times 10^{-2}}$	**0.0213**	$\mathbf{1.16 \times 10^{-2}}$	$\mathbf{1.01 \times 10^{-2}}$	**0.0156**	$\mathbf{7.60 \times 10^{-3}}$	$\mathbf{6.67 \times 10^{-3}}$	**0.0148**	$\mathbf{7.20 \times 10^{-3}}$	$\mathbf{6.32 \times 10^{-3}}$

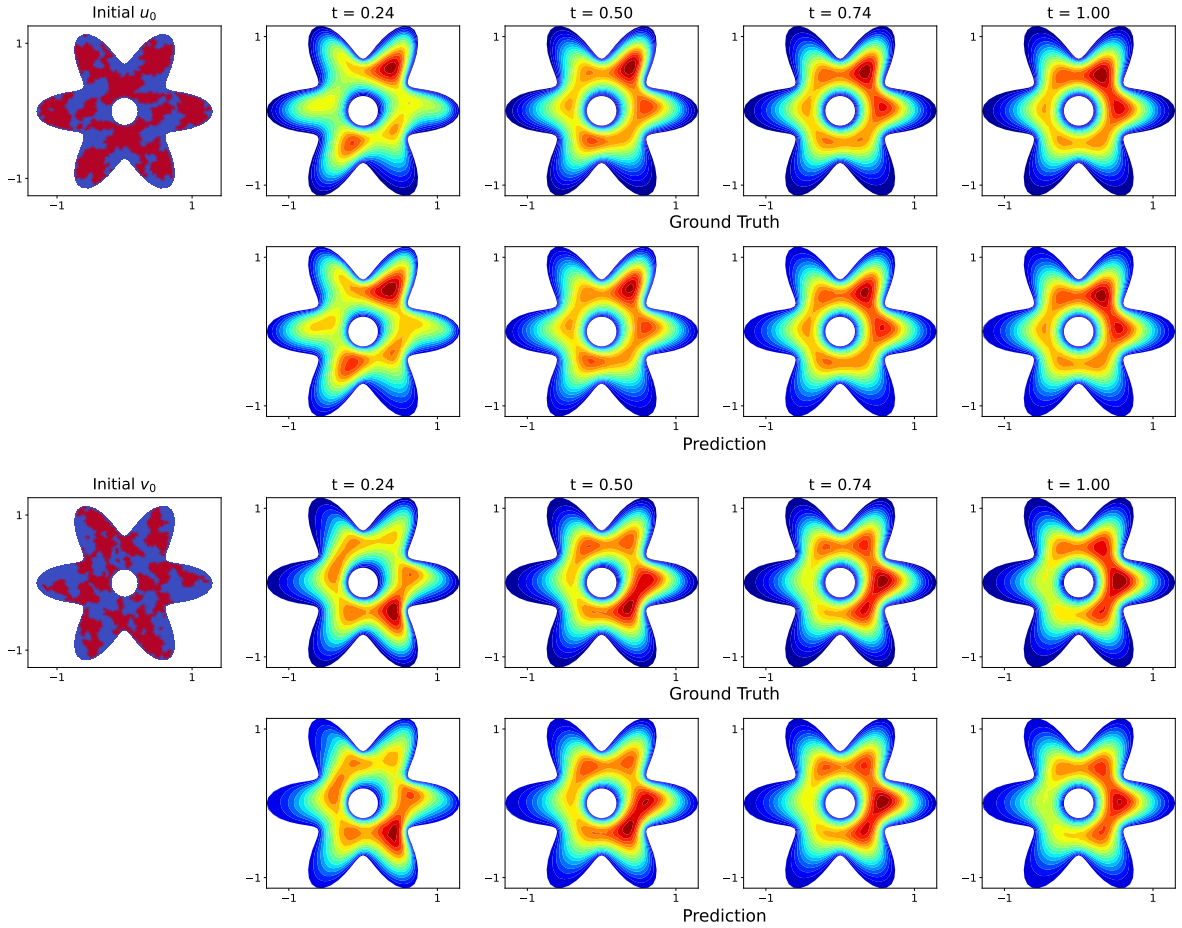


Figure C.10: 2D Burgers' equation on an irregular flower-shaped domain with a circular hole. **Top row:** Ground truth evolution of the horizontal velocity component u. **Bottom row:** GSNO inference predictions of the vertical component v. The model is trained and evaluated on the same resolution mesh with $N_s = 1168$ nodes and 51 temporal snapshots. Inference is performed using $T_{\text{in}} = 5$ input steps to forecast $T_{\text{out}} = 46$ future steps.

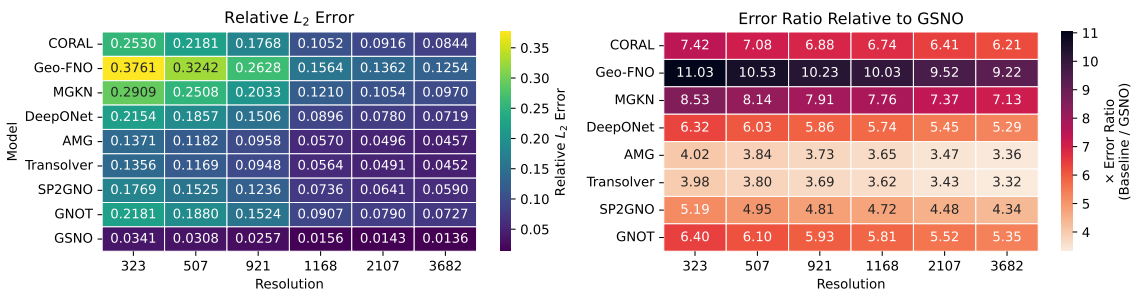


Figure C.11: Resolution-wise generalization results on the 2D Burgers' equation benchmark. **Left:** Relative L_2 error on the test set across increasing spatial resolutions. **Right:** Error degradation relative to GSNO, computed as the ratio of each model's error to GSNO's at the same resolution. All models are trained and evaluated on matching meshes using $T_{\text{in}} = 5$ input snapshots to predict $T_{\text{out}} = 46$ future steps. GSNO consistently outperforms baselines across all scales, achieving up to $10\times$ lower error.

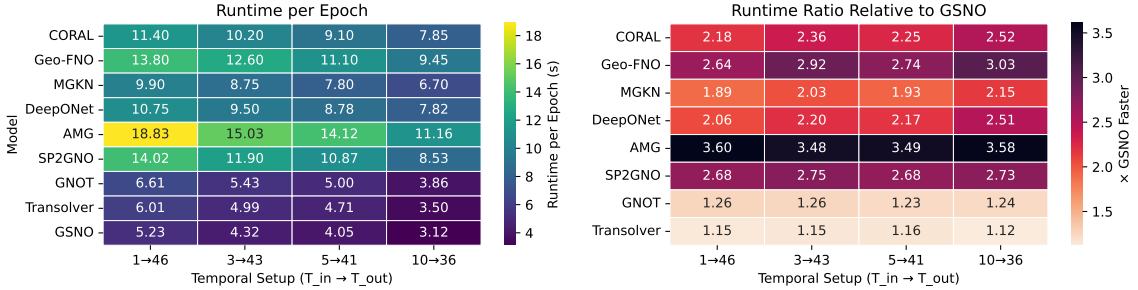


Figure C.12: **Training efficiency of neural operator models on Burgers' equation under varying temporal configurations at resolution $N_s = 1168$. Left:** Average runtime per epoch (in seconds) across different temporal input-output setups ($T_{\text{in}} \rightarrow T_{\text{out}}$). **Right:** Slowdown relative to GSNO, computed as the ratio of each model's runtime to GSNO's at the same setting. GSNO consistently achieves the lowest per-epoch runtime, highlighting its computational efficiency.

C.5 Additional results for 2D Navier–Stokes Equation Equation

Figure C.13 demonstrates GSNO's ability to perform zero-shot super-resolution on the 2D Navier–Stokes equation benchmark. Figures C.14 and C.15 further present generalization accuracy across resolutions and training efficiency under varying temporal input-output settings.

Table C.8: Comparison of neural operator models on Navier–Stokes Equation ($N_s = 1244$).

Model	Temporal Config: 1→50			Temporal Config: 3→48			Temporal Config: 5→46			Temporal Config: 10→41		
	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE
CORAL	0.1654	8.91×10^{-1}	6.59×10^{-1}	0.1412	7.61×10^{-1}	5.79×10^{-1}	0.1148	6.19×10^{-1}	4.86×10^{-1}	0.1070	5.77×10^{-1}	4.57×10^{-1}
Geo-FNO	0.2056	$1.11 \times 10^{+0}$	7.81×10^{-1}	0.1790	9.65×10^{-1}	7.02×10^{-1}	0.1614	8.70×10^{-1}	6.47×10^{-1}	0.1512	8.15×10^{-1}	6.13×10^{-1}
MGKN	0.0964	5.20×10^{-1}	4.16×10^{-1}	0.0770	4.15×10^{-1}	3.39×10^{-1}	0.0654	3.52×10^{-1}	2.92×10^{-1}	0.0606	3.27×10^{-1}	2.72×10^{-1}
DeepONet	0.1250	6.74×10^{-1}	5.23×10^{-1}	0.1096	5.91×10^{-1}	4.66×10^{-1}	0.0958	5.16×10^{-1}	4.14×10^{-1}	0.0916	4.94×10^{-1}	3.98×10^{-1}
AMG	0.1060	5.71×10^{-1}	4.61×10^{-1}	0.0780	4.20×10^{-1}	3.39×10^{-1}	0.0580	3.13×10^{-1}	2.52×10^{-1}	0.0540	2.91×10^{-1}	2.35×10^{-1}
GNOT	0.1876	$1.01 \times 10^{+0}$	8.16×10^{-1}	0.1326	7.15×10^{-1}	5.77×10^{-1}	0.0912	4.91×10^{-1}	3.97×10^{-1}	0.0844	4.55×10^{-1}	3.67×10^{-1}
SP^2GNO	0.1533	7.35×10^{-1}	5.90×10^{-1}	0.1081	5.80×10^{-1}	4.65×10^{-1}	0.0744	3.65×10^{-1}	2.95×10^{-1}	0.0689	3.35×10^{-1}	2.70×10^{-1}
Transolver	0.1189	6.41×10^{-1}	5.17×10^{-1}	0.0836	4.51×10^{-1}	3.64×10^{-1}	0.0575	3.10×10^{-1}	2.50×10^{-1}	0.0534	2.88×10^{-1}	2.32×10^{-1}
GSNO	**0.0336**	$\mathbf{1.81 \times 10^{-1}}$	$\mathbf{1.55 \times 10^{-1}}$	**0.0237**	$\mathbf{1.28 \times 10^{-1}}$	$\mathbf{1.11 \times 10^{-1}}$	**0.0164**	$\mathbf{8.84 \times 10^{-2}}$	$\mathbf{7.71 \times 10^{-2}}$	**0.0152**	$\mathbf{8.19 \times 10^{-2}}$	$\mathbf{7.15 \times 10^{-2}}$

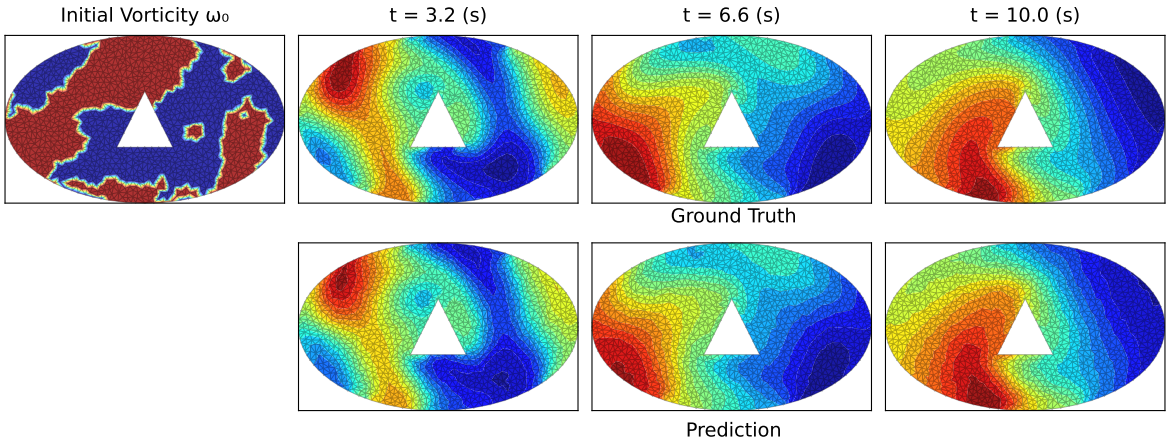


Figure C.13: Navier–Stokes simulation with viscosity $\nu = 10^{-3}$, demonstrating the model's ability to perform zero-shot super-resolution. The GSNO is trained on a coarse point cloud with $N_s = 972$ nodes and evaluated on a finer mesh with $N_s = 1903$ nodes without retraining. Ground truth results are shown on the top row, and GSNO predictions are shown on the bottom row. (See Section 3.2 for further details.)

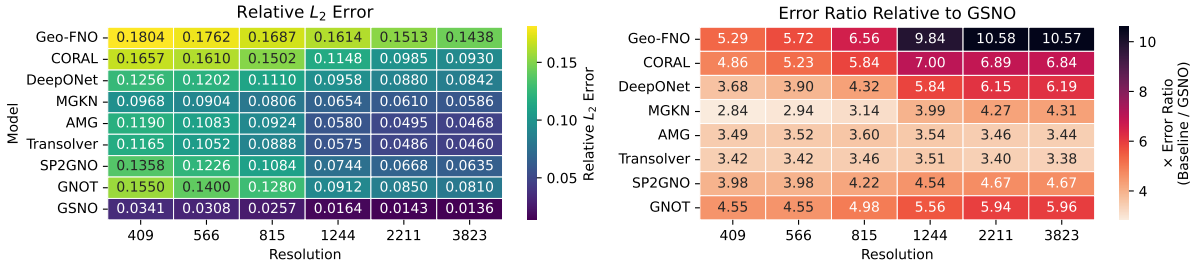


Figure C.14: Resolution-based generalization comparison for the 2D Navier–Stokes equation in vorticity form. **Left:** Relative L_2 error across increasing mesh resolutions for all models. **Right:** Accuracy gap with respect to GSNO, computed as the ratio of each model's error to GSNO's error at each resolution. All models are trained and tested on matching unstructured meshes using $T_{\text{in}} = 5 \rightarrow T_{\text{out}} = 46$. GSNO consistently achieves the lowest error across all resolutions, outperforming others by margins of up to $10\times$.

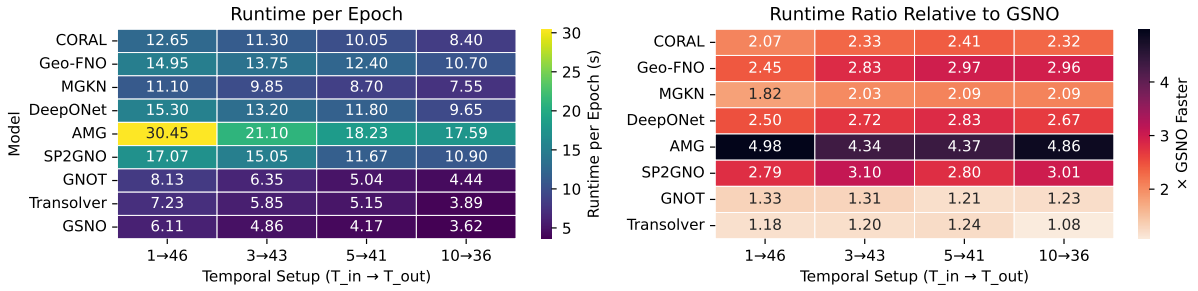


Figure C.15: **Training efficiency of neural operator models on the 2D Navier–Stokes equation under varying temporal configurations at resolution $N_s = 1244$. Left:** Average runtime per epoch (in seconds) for different input-output lengths ($T_{\text{in}} \rightarrow T_{\text{out}}$). **Right:** Runtime overhead relative to GSNO, computed as the ratio of each model's epoch runtime to GSNO's at the same setting. GSNO remains the most computationally efficient, outperforming all baselines in training speed across temporal configurations.

C.6 Additional results for 2D Shallow Water Equations Equation

Figure C.16 reports the generalization accuracy of GSNO on the 2D Shallow Water Equations benchmark across different mesh resolutions. Figure C.17 further compares the training efficiency of all models under varying temporal input-output settings, highlighting GSNO's computational advantages.

Table C.9: Comparison of neural operator models on Shallow Water Equation ($N_s = 1830$).

Model	Temporal Config: 1→50			Temporal Config: 3→48			Temporal Config: 5→46			Temporal Config: 10→41		
	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE	Rel L_2	RMSE	MAE
CORAL	0.1784	1.78×10^{-1}	1.52×10^{-1}	0.1534	1.53×10^{-1}	1.31×10^{-1}	0.1238	1.24×10^{-1}	1.07×10^{-1}	0.1162	1.16×10^{-1}	1.00×10^{-1}
Geo-FNO	0.2112	2.11×10^{-1}	1.79×10^{-1}	0.1848	1.85×10^{-1}	1.58×10^{-1}	0.1650	1.65×10^{-1}	1.41×10^{-1}	0.1526	1.53×10^{-1}	1.31×10^{-1}
MGKN	0.1128	1.13×10^{-1}	9.79×10^{-2}	0.0876	8.76×10^{-2}	7.63×10^{-2}	0.0724	7.24×10^{-2}	6.32×10^{-2}	0.0668	6.68×10^{-2}	5.84×10^{-2}
DeepONet	0.1284	1.28×10^{-1}	1.11×10^{-1}	0.1128	1.13×10^{-1}	9.79×10^{-2}	0.0982	9.82×10^{-2}	8.53×10^{-2}	0.0904	9.04×10^{-2}	7.87×10^{-2}
AMG	0.1210	1.21×10^{-1}	1.04×10^{-1}	0.0890	8.90×10^{-2}	7.65×10^{-2}	0.0692	6.92×10^{-2}	5.95×10^{-2}	0.0630	6.30×10^{-2}	5.42×10^{-2}
GNOT	0.2150	2.15×10^{-1}	1.85×10^{-1}	0.1534	1.53×10^{-1}	1.32×10^{-1}	0.1093	1.09×10^{-1}	9.40×10^{-2}	0.0995	9.95×10^{-2}	8.56×10^{-2}
SP^{2}GNO	0.1752	1.75×10^{-1}	1.51×10^{-1}	0.1250	1.25×10^{-1}	1.08×10^{-1}	0.0891	8.91×10^{-2}	7.66×10^{-2}	0.0810	8.10×10^{-2}	6.97×10^{-2}
Transolver	0.1354	1.35×10^{-1}	1.16×10^{-1}	0.0965	9.65×10^{-2}	8.40×10^{-2}	0.0689	6.89×10^{-2}	5.93×10^{-2}	0.0625	6.25×10^{-2}	5.38×10^{-2}
GSNO	**0.0375**	$\mathbf{3.75 \times 10^{-2}}$	$\mathbf{3.30 \times 10^{-2}}$	**0.0268**	$\mathbf{2.68 \times 10^{-2}}$	$\mathbf{2.36 \times 10^{-2}}$	**0.0193**	$\mathbf{1.93 \times 10^{-2}}$	$\mathbf{1.70 \times 10^{-2}}$	**0.0174**	$\mathbf{1.74 \times 10^{-2}}$	$\mathbf{1.54 \times 10^{-2}}$

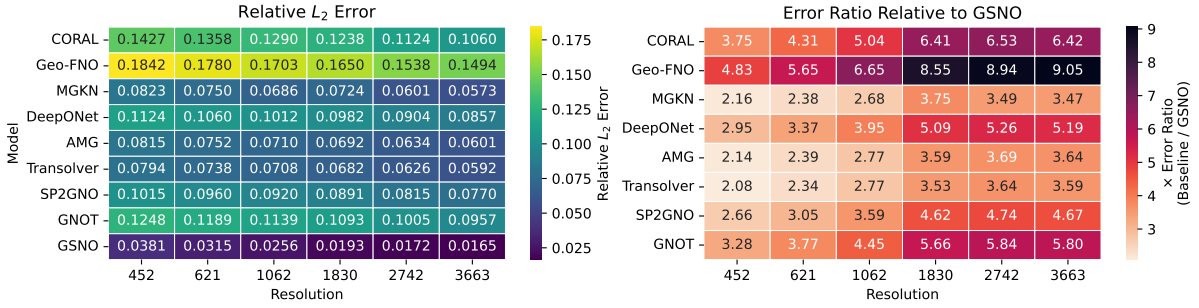


Figure C.16: Resolution-based generalization comparison for the 2D Shallow Water Equation (SWE). **Left:** Relative L_2 error across increasing mesh resolutions for all models. **Right:** Accuracy gap with respect to GSNO, computed as the ratio of each model's error to GSNO's error at each resolution. All models are trained and tested on matching unstructured meshes using $T_{\text{in}} = 5 \rightarrow T_{\text{out}} = 46$. GSNO consistently delivers the highest predictive accuracy across spatial scales, with improvements of up to 7× over baseline methods.

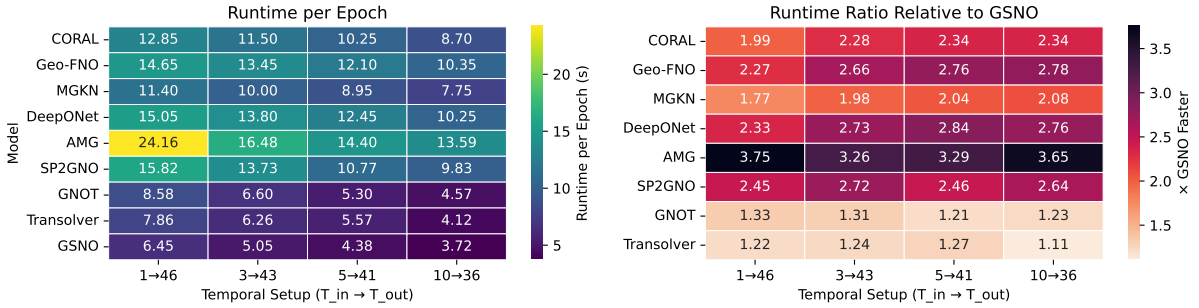


Figure C.17: **Training efficiency of neural operator models on the 2D Shallow Water Equations at resolution $N_s = 3663$. Left:** Average runtime per epoch (in seconds) for various temporal configurations ($T_{\text{in}} \rightarrow T_{\text{out}}$). **Right:** Relative runtime overhead, calculated as the ratio of each model's epoch runtime to GSNO's. GSNO achieves the best training efficiency across all settings, offering significant speedups over baseline models in time-dependent simulations.

D Bayesian Inversion Result for Darcy Flow

This appendix presents the results of the Bayesian inverse problem described in Section 3.3, where we aim to recover the coefficient field $a(x, y) \in \mathbb{R}^{1184\times1}$ in Darcy flow from a single output solution $u_{\text{obs}} \in \mathbb{R}^{1184\times1}$. We use the Metropolis–Hastings algorithm to sample from the posterior distribution over a, leveraging GSNO as the surrogate forward model. We assume a Gaussian prior on $a \sim \mathcal{N}(0, \sigma^2_{\text{prior}} I)$, and define the posterior using a squared-error misfit between the GSNO prediction and the observed output. Since the observations are noise-free, the unnormalized log-posterior becomes:

$$\log p(a \mid u_{\text{obs}}) \propto -\frac{1}{2}\|\mathsf{GSNO}(a) - u_{\text{obs}}\|^2 - \frac{1}{2\sigma^2_{\text{prior}}}\|a\|^2.$$

A total of 5,000 samples are drawn from the posterior distribution, with the first 500 discarded as burn-in. Each iteration involves a single forward evaluation of the GSNO model, enabling efficient sampling due to its mesh-invariant spectral formulation and GPU-accelerated execution. Figure D.18 presents a comparison between the true coefficient field and the posterior mean inferred from the sampled distribution. The close agreement illustrates GSNO's effectiveness in enabling fast and accurate Bayesian inversion under noise-free conditions.

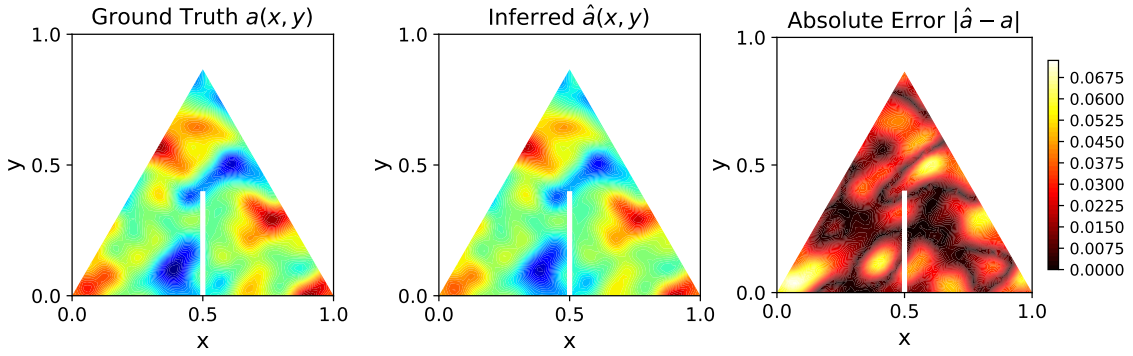


Figure D.18: Posterior mean of the coefficient field inferred from noise-free observations $u_{\text{obs}} \in \mathbb{R}^{1184\times 1}$, using GSNO as the surrogate model and 5,000 Metropolis–Hastings samples. The reconstructed field closely matches the true coefficient.

E Ablation Results and Analysis

This section evaluates the contribution of key components in the GSNO architecture through a series of ablation experiments. Four modifications are considered: (1) removing the spectral kernel R_ϕ, (2) disabling the local residual path W, (3) replacing the temporal Fourier transform (FFT) with a multilayer perceptron, and (4) using a random orthonormal basis instead of the Laplacian eigenbasis Φ_{k_s}. Experiments are conducted on the steady-state Darcy flow and the time-dependent 2D Burgers' equation. The training setup and mesh resolution match those used in the main experiments. For Burgers', we use the configuration $T_{\text{in}} = 5 \rightarrow T_{\text{out}} = 46$; Darcy is treated as a static mapping from input field $a(x)$ to output solution $u(x)$. Results are reported in Table E.10.

Table E.10: Ablation results on GSNO architecture. Each variant is evaluated on Darcy Flow (steady-state) and Burgers' Equation (time-dependent, with $T_{\text{in}} = 5 \rightarrow T_{\text{out}} = 46$). Removing or altering core components increases error and/or runtime.

Model Variant	**(a) Darcy Flow** ($N_s = 1184$)		**(b) Burgers' Equation** ($N_s = 1168$)	
	Relative L_2 Error	Runtime (s/epoch)	Relative L_2 Error	Runtime (s/epoch)
Full GSNO (ours)	**0.0083**	2.24	**0.0156**	3.12
No Spectral Kernel ($R_\phi = I$)	0.0126	2.11	0.0243	3.05
No Local Path ($W = 0$)	0.0111	2.21	0.0278	3.08
No Temporal FFT (MLP instead)	–	–	0.0315	2.47
Random Spatial Basis	0.0472	2.31	0.0982	3.15

The results show that removing the spectral kernel reduces accuracy on both problems. While spectral projection alone provides a useful basis, learning to mix frequencies improves global expressivity. Disabling the local residual path leads to a similar drop, particularly on Darcy flow, indicating that local updates are important for correcting and refining the spectral output.

Replacing the temporal FFT with an MLP has the most significant effect. Without access to global temporal frequencies, the model struggles with dynamic tasks like Burgers' and becomes slower to train. The largest degradation occurs when the Laplacian eigenbasis is replaced with a random basis. This confirms that spatial generalization strongly depends on the geometry-aware structure encoded in the Laplacian modes.

These results demonstrate that all components of GSNO contribute meaningfully to its performance. Their combined effect supports efficient learning of PDE solutions across space and time, even on irregular domains.

F GSNO HYPERPARAMETERS

F.1 GSNO ARCHITECTURE, TRAINING SETUP, AND SENSITIVITY TO THE NUMBER OF SPATIAL AND TEMPORAL MODES

Two important hyperparameters in GSNO are the number of spatial modes k_s and the number of temporal modes k_t. Together they control the trade-off between accuracy, efficiency, and generalization. Larger values capture finer details but increase computation and can lead to overfitting, while smaller values act as a regularizer but may lose important information. The parameter k_s determines how many graph Laplacian eigenvectors are used to form the spatial spectral basis. We fix $k_t = 8$ and evaluate $k_s \in \{4, 6, 8, 16, 32\}$ on the 2D Navier–Stokes case ($N_s = 1244$, Temporal Config: $5 \to 46$). We report Relative L^2 error (lower is better) and CPU time per epoch.

Table F.11: **Sensitivity of GSNO to spatial modes** k_s on 2D Navier–Stokes (N_s=1244, k_t=8).

Spatial Modes k_s	**Relative L^2 Error**	**CPU Time / epoch (s)**
4	0.0412	2.11
6	0.0351	3.85
8	**0.0164**	**4.17**
16	0.0218	7.12
32	0.0275	10.98

As shown in Table F.11, performance follows a U–shaped curve: very small k_s underfits, very large k_s increases cost and slightly hurts generalization. The sweet spot is $k_s = 8$, which minimizes error with moderate runtime. In practice, values between 6 and 10 work well. Next, we fix $k_s = 8$ (the optimal setting above) and vary k_t to see its impact. The parameter k_t controls how many Fourier modes are used along the temporal dimension. Larger k_t can improve long-term dynamics but at the expense of slower training.

Table F.12: **Sensitivity of GSNO to temporal modes** k_t on 2D Navier–Stokes (N_s=1244, k_s=8).

Temporal Modes k_t	**Relative L^2 Error**	**CPU Time / epoch (s)**
4	0.0287	3.02
6	0.0219	3.65
8	**0.0164**	**4.17**
12	0.0182	5.46
16	0.0235	6.88

Table F.12 shows a similar trend: too few temporal modes limit accuracy, while too many slow training and slightly degrade generalization. The best balance occurs at $k_t = 8$.

Overall, GSNO achieves the best trade-off when both k_s and k_t are chosen in the mid-range. Too few modes limit expressivity, while too many increase cost and risk overfitting. Based on our experiments, $k_s = 8$ and $k_t = 8$ provide a strong default setting for 2D Navier–Stokes and related PDE benchmarks. We therefore used this configuration in our main experiments.

Table F.13 summarizes the architecture and training hyperparameters used for GSNO across all benchmark PDEs. The columns "k_s" and "k_t" represent the number of retained spectral modes in the spatial (graph Laplacian) and temporal (FFT) domains, respectively. The "Width" column denotes the latent feature dimensionality throughout the GSNO blocks. Each block includes a learnable 4D spectral kernel and a residual 1×1 convolution branch, followed by a GELU nonlinearity. Inputs and outputs are min-max normalized per dataset. All GSNO models are trained using the Adam optimizer with a batch size of 32.

Table F.13: GSNO architecture and training hyperparameters.

PDE Case	k_s	k_t	Width	Lifting MLP	GSNO Layers	Projection MLP	Spatial Branch	Nonlinearity	#Params	LR	Epochs
Darcy Flow	8	–	20	1-layer	4	2-layer: (20 × 128), (128 × 1)	Conv 1×1	GELU	20,817	0.001	1000
2D Airfoil	8	–	20	1-layer	4	2-layer: (20 × 128), (128 × 1)	Conv 1×1	GELU	20,817	0.001	1000
Shape Net 3d Car	8	-	40	1-layer	4	2-layer: (20 × 128), (128 × 1)	Conv 1×1	GELU	42,516	0.001	1000
Burgers' Equation	8	8	40	1-layer	4	2-layer: (20 × 128), (128 × 2)	Conv 1×1	GELU	431,108	0.001	1000
Navier–Stokes	8	8	40	1-layer	4	2-layer: (20 × 128), (128 × 1)	Conv 1×1	GELU	430,857	0.001	1000
Shallow water	8	8	40	1-layer	4	2-layer: (20 × 128), (128 × 1)	Conv 1×1	GELU	430,857	0.001	1000

G BASELINE MODELS OVERVIEW, KEY DIFFERENCES, AND HYPERPARAMETERS

We compare GSNO against a diverse set of state-of-the-art neural operator models: **DeepONet**, **MGKN**, **CORAL**, **Geo-FNO**, **GNOT**, **Transolver**, and **AMG**. Each baseline embodies a distinct philosophy for operator learning on irregular or multi-scale domains, ranging from dual-network architectures and kernel-based graph operators to mesh-free latent encodings, Fourier-based domain warping, transformer-driven attention mechanisms, and multi-graph constructions. We briefly summarize their architectures below before presenting a detailed comparison.

DeepONet Lu et al. (2021) employs a dual-network structure: a branch network encodes the input function (e.g., coefficients or initial conditions), while a trunk network processes spatial or spatiotemporal coordinates. The outputs of both networks are combined via an inner product to yield the final prediction. This design enables flexible, pointwise evaluation of the solution operator but does not incorporate mesh structure or explicit spectral modeling. As a result, DeepONet's generalization can be sensitive to the distribution and quality of sampled input points, especially on highly irregular domains.

MGKN Li et al. (2020) extends the kernel integral operator framework using learned multipole kernels over graphs built from unstructured meshes. It models spatial interactions by encoding inputs and applying graph convolutions based on Delaunay connectivity. MGKN effectively captures spatial dependencies but does not exploit temporal structure spectrally. Temporal dynamics are typically modeled through standard sequence processing methods, limiting their capability to globally capture long-range temporal dependencies.

CORAL Serrano et al. (2023) proposes a mesh-free, coordinate-based neural operator framework. It encodes input data into a latent space using MLPs and reconstructs outputs through coordinate queries. While this design allows CORAL to flexibly generalize across different geometries, it lacks structured spatial priors such as Laplacian smoothness or graph connectivity, which can limit its ability to capture long-range or multi-scale spatial correlations efficiently.

Geo-FNO Li et al. (2023) adapts Fourier Neural Operators to irregular domains by learning a mapping from the physical domain to a latent uniform grid using a transformer encoder. Standard Fourier convolutions are then applied in the latent space. Although Geo-FNO preserves the advantages of global receptive fields inherent to Fourier methods, its performance depends critically on the smoothness and quality of the learned domain warping. In highly complex domains with topological irregularities or sharp features, this warping may introduce distortions, reducing model accuracy and stability.

GNOT (Hao et al., 2023) introduces a transformer-based neural operator designed to jointly address three core challenges in operator learning: irregular meshes, multiple input functions, and multi-scale physical dynamics. Its architecture is built around a *heterogeneous normalized attention (HNA)* mechanism, which encodes arbitrary types of inputs (e.g., boundary shapes, parameters, or distributed functions) into a unified representation and applies efficient cross- and self-attention with linear complexity. This enables flexible handling of irregular discretizations and diverse inputs. In addition, GNOT incorporates a *geometric gating mechanism*, inspired by domain decomposition, which adaptively assigns different expert subnetworks to regions of the domain. This soft domain decomposition allows the model to capture multi-scale phenomena more effectively.

SP^{2}GNO (Sarkar & Chakraborty, 2025) adopts a hybrid design that couples truncated spectral graph convolutions with a spatial message-passing branch gated by Lipschitz positional embeddings. While this dual-path strategy balances local and global modeling, it introduces architectural complexity and runtime overhead due to dynamic gating and stacked GNN layers. More critically, SP^{2}GNO treats the temporal dimension implicitly through autoregressive rollout of spatial layers, which limits its ability to capture long-range correlations and global frequency structure. In contrast, **GSNO** follows a simpler and more principled approach: it directly leverages the graph Laplacian eigenbasis for spatial spectral learning and augments it with real Fourier transforms along the temporal dimension, forming a joint space–time spectral kernel without auxiliary gating. This unified treatment eliminates error accumulation from autoregression, avoids over-smoothing, and reduces computation. As a result, GSNO achieves higher efficiency and scalability, with faster runtimes, lower memory footprints, and stronger mesh-invariant generalization, while SP^{2}GNO remains sensitive to graph construction choices. Empirical results confirm that GSNO consistently surpasses SP^{2}GNO in both accuracy and efficiency across steady-state and time-dependent PDE benchmarks.

Transolver (Wu et al., 2024) is a transformer-based neural operator specifically designed for solving PDEs across diverse geometries and boundary conditions. Unlike models that rely on fixed grids or handcrafted kernels, Transolver treats operator learning as a sequence-to-sequence problem. It encodes input functions and geometric features through a transformer encoder, then reconstructs solution fields using a decoder equipped with spectral attention. A key design is its ability to incorporate positional and geometric encodings that allow it to directly handle irregular domains without requiring domain warping. By leveraging long-range self-attention, Transolver captures global dependencies in both space and time, which improves its robustness on PDE benchmarks with complex dynamics.

AMG (Li et al., 2025) introduces a *multi-graph neural operator* framework designed to solve PDEs on arbitrary geometries. Its key innovation is the use of three complementary graphs: a *local graph* that captures fine-scale, high-frequency interactions, a *global graph* that encodes broad spatial dependencies, and a *physics graph* that incorporates physical priors into the representation. These graphs are processed through a novel **GraphFormer** block with dynamic graph attention, which generalizes attention as a learnable integral operator over irregular domains. This design allows AMG to balance local detail and global coherence, while explicitly grounding predictions in physical attributes. Unlike purely spectral or kernel-based models, AMG can adapt to highly complex geometries and dynamically changing meshes.

G.1 Key Differences Compared to GSNO.

Unlike **Geo-FNO**, GSNO does not rely on learned domain warping to a latent grid. Instead, it operates directly on the physical mesh using Delaunay-based graphs and fixed Laplacian eigenvectors, preserving native geometry without distortion. Compared to **DeepONet** and **CORAL**, which lack explicit spectral structure, GSNO projects features into a spatial spectral basis, capturing global correlations across irregular domains. Relative to **MGKN**, which learns multipole graph kernels but does not address temporal dynamics spectrally, GSNO introduces a real-valued Fourier transform in the temporal dimension, enabling a joint space–time spectral kernel for coherent dynamical modeling. Compared to transformer-based operators, GSNO follows a lighter but more structured approach. Unlike **GNOT** and **Transolver**, which rely on heavy multi-head attention, GSNO avoids quadratic attention costs by restricting spectral learning to graph Laplacians and Fourier modes, while still capturing global dependencies. Unlike the **AMG** multi-graph strategy that aggregates local, global, and physics graphs, GSNO emphasizes a single spectral basis with lightweight 1×1 convolutional residual paths, reducing computational complexity while retaining generalization. Finally, while the **SP^{2}GNO** (Sarkar & Chakraborty, 2025) framework combines truncated Laplacian eigenbasis filtering with gated spatial GNN layers, this hybrid design introduces architectural complexity and depends on k-NN graph construction and message passing. More importantly, SP^{2}GNO lacks an explicit temporal spectral module, instead modeling time implicitly through stacked GNN updates, which limits its ability to capture long-range spatiotemporal correlations. In contrast, GSNO integrates both graph-based spatial spectra and Fourier temporal spectra into a unified space–time kernel, achieving superior accuracy and efficiency without relying on recurrent or autoregressive iterations. These design choices allow GSNO to maintain mesh-invariant generalization (via Laplacian recomputation), minimize overhead, and deliver robust accuracy across steady-state and time-dependent PDEs. As our experiments demonstrate, GSNO achieves higher predictive accuracy and efficiency compared to all baselines, while requiring fewer architectural components. The architectural and functional differences between GSNO and the baselines are summarized in Tables G.14 and G.15.

Table G.14: Comparison of GSNO with prior neural operator methods for irregular domains: **Core Modeling Features**.

Feature	DeepONet	MGKN	CORAL	Geo-FNO	GNOT	Transolver	AMG	SP^{2}GNO	GSNO
Space spectral learning	No	Yes (multipole kernels)	No	Yes (after warping)	No (attention-based)	No (spectral attention, not graph-based)	Yes (multi-graph basis)	Yes (truncated Laplacian + spectral kernel)	Yes (graph Laplacian eigenbasis)
Time spectral learning	No	No	No	No	No	Yes (spectral attention)	No	No (time implicit via stacked GNNs)	Yes (real FFT)
Mesh invariance	Partial (fixed sampling)	Partial	Full	Limited (warping quality)	Full (HNA encoder)	Full (geometric encoding)	Full (multi-graph)	Partial (depends on k-NN graph construction)	Full (via Laplacian recomputation)
GNN stacking required	No	Yes (spatial GNN layers)	No	No	No (transformer layers)	No (transformer layers)	Yes (GraphFormer blocks)	Yes (gated spatial GNN layers)	No (1×1 convolution only)

Table G.15: Comparison of GSNO with prior neural operator methods for irregular domains: **Advanced Mechanisms**.

Feature	DeepONet	MGKN	CORAL	Geo-FNO	GNOT	Transolver	AMG	SP^{2}GNO	GSNO
Attention mechanism	No	No	No	No	HNA + geometric gating	Multi-head + spectral attention	Dynamic graph attention	No (uses gated spatial convolution)	No (spectral convolution only)
Multi-graph / gating	No	No	No	No	Yes (soft domain decomposition)	No	Yes (local, global, physics graphs)	Yes (edge gating via Lipschitz embeddings)	No
Global space-time convolution	No	No	No	No	No	Yes (global self-attention)	Partial (via multi-graph aggregation)	No (temporal dynamics implicit, not spectral)	Yes (joint space-time spectral kernel)

G.2 Hyperparameters for Baseline Models

All baseline models are retrained under identical data splits and training settings as GSNO to ensure a fair and consistent comparison. Detailed hyperparameter configurations for each model are provided in Tables G.16–G.19.

Table G.16: Hyperparameters for **DeepONet**.

Component	Configuration
Trunk Network	3-layer MLP, 100 hidden units, ReLU
Branch Network	2-layer MLP, 100 hidden units, ReLU
Input	Coordinates on a grid
Output	Pointwise function values

Table G.17: Hyperparameters for **MGKN**.

Component	Configuration
Input Encoder	3-layer MLP, 64 units, GELU
Decoder	2-layer MLP, 64 units, GELU
Graph Kernel	Multipole (Gaussian RBF)
RBF Width (γ)	1.0

Table G.18: Hyperparameters for **CORAL**.

Component	Configuration
Encoder	4-layer SIREN, width 128, $\omega_0 = 10$
Latent Code	128-dimensional vector
Decoder	3-layer MLP, width 64
Training Strategy	Meta-learning (outer/inner loops)
Input Representation	Coordinate-based (mesh-free)

Table G.19: Hyperparameters for **Geo-FNO**.

Component	Configuration
Input Encoder	3-layer MLP, width 32, sinusoidal encoding
Latent Mapping	Learned warp to regular grid
Latent Grid	Uniform FFT grid (2D for static, 3D for temporal PDEs)
Fourier Layers	4 layers, 8 retained modes, width 32
Spectral Operation	Complex-valued FFT on latent grid

Table G.20: SP^{2}GNO (Steady-state)

Component	Configuration
Blocks (L)	6
Hidden Width (d)	32
Graph Construction	k-NN (k=16)
Laplacian Basis	First m=32 eigenvectors (LOBPCG)
Spectral Kernel	$K \in \mathbb{R}^{m \times d \times d}$ (learnable)
Spatial Branch	Gated GCN-style conv
Positional Encoding	Lipschitz anchor embeddings

Table G.21: SP^{2}GNO (Time-dependent)

Component	Configuration
Blocks (L)	6
Hidden Width (d)	32
Graph Construction	k-NN (k=16), fixed per frame
Laplacian Basis	Reuse first m=32 eigenvectors
Temporal Handling	Train $1 \to 1$; autoregressive rollout for multi-step
Rollout Settings	Eval: $1 \to K$ via iterative $1 \to 1$ predictions

Table G.22: Hyperparameters for **GNOT**.

Component	Configuration
Attention Layers	4
Hidden Size (Attention)	256
Embedding Dimension	256
MLP Depth / Width	4 layers, 256 units
Attention Heads	8
Experts (Geometric Gating)	3

Table G.23: Hyperparameters for **Transolver**.

Component	Configuration
Transformer Layers	6
Embedding Dimension	256
MLP Depth / Width	2 layers, 256 units
Attention Heads	8
Spectral Attention Modes	16
Positional Encoding	Sinusoidal + geometric features

Table G.24: Hyperparameters for **AMG**.

Component	Configuration
Graph Types	Local, Global, Physics
Processor Depth	3 GraphFormer layers
Local Node Number	1024
Global Sample Ratio	75% of nodes
Physics Nodes	32
Attention Heads	8
Hidden Size	256

H MEMORY FOOTPRINTS

This appendix summarizes the computational footprint of all models. For the NSE and Darcy Flow setups, we report *inference time per batch* and *peak GPU memory* during training and inference at fixed N_s and batch size, as summarized in Tables H.26 and H.25.

Table H.25: Inference time and memory footprint of models for the Darcy Flow case (N_s=3421, batch size 32).

Model	Inference Time (s/batch)	Peak Training GPU Mem	Inference GPU Mem
CORAL	∼0.025	∼1.5 GB	∼320 MB
MGKN	∼0.023	∼1.8 GB	∼360 MB
Geo-FNO	∼0.030	∼2.1 GB	∼370 MB
GNOT	∼0.027	∼2.0 GB	∼380 MB
Transolver	∼0.025	∼1.9 GB	∼380 MB
AMG	∼0.104	∼2.6 GB	∼420 MB
SP^{2}GNO	∼0.023	∼2.1 GB	∼400 MB
GSNO	∼0.022	∼1.8 GB	∼360 MB

Table H.26: Inference time and memory footprint of models for the NSE case (N_s=1244, 5→46, batch size 32).

Model	Inference Time (s/batch)	Peak Training GPU Mem	Inference GPU Mem
CORAL	∼0.31	∼5.1 GB	∼1.0 GB
MGKN	∼0.27	∼7.3 GB	∼1.4 GB
Geo-FNO	∼0.32	∼6.4 GB	∼1.1 GB
GNOT	∼0.261	∼6.2 GB	∼1.1 GB
Transolver	∼0.239	∼6.0 GB	∼1.1 GB
AMG	∼0.992	∼8.5 GB	∼1.2 GB
SP^{2}GNO	∼0.285	∼6.8 GB	∼1.2 GB
GSNO	∼0.21	∼5.8 GB	∼1.0 GB