

Supplementary Information

Linear–Nonlinear Fusion Neural Operator for Partial Differential Equations

1 Dataset generation and preprocessing

We provide detailed descriptions of the data generation process for each benchmark considered in this work. Specific numerical parameters are detailed below. The core implementation scripts will be made available upon reasonable request.

1.1 Laplace dataset generation

We consider the 2D Laplace equation

$$\Delta u = 0 \quad \text{in } \Omega = [0, 1]^2, \quad u|_{\partial\Omega} = g, \quad (\text{S1})$$

and learn the solution operator mapping the Dirichlet boundary trace g to the solution field u on a fixed grid. This dataset is generated following the Mathematical Artificial Data (MAD) framework [28].

Discretization. We use a uniform $N \times N$ grid on Ω with $N = 51$ and spacing $h = 1/(N - 1)$, resulting in $N_g = N^2 = 2601$ evaluation points. The boundary trace is sampled on $\partial\Omega$ using N points per side with duplicated corner points removed, yielding $N_b = 4(N - 1) = 200$ boundary samples. The boundary points are ordered by concatenating the four edges in a fixed counterclockwise manner.

MAD solution family. For each sample, we synthesize a harmonic function as a linear combination of fundamental solutions with sources located outside the domain. Specifically, we draw $J = 10$ source points $\{c_j\}_{j=1}^J$ outside an expanded box $[-\varepsilon, 1 + \varepsilon]^2$ with $\varepsilon = 10^{-3}$, and random weights $\{w_j\}_{j=1}^J$. The solution is defined by

$$u(x) = \sum_{j=1}^J w_j \cdot \frac{1}{2} \log(\|x - c_j\|^2), \quad (\text{S2})$$

which is harmonic inside Ω because all sources lie outside Ω .

Data format and normalization. Each sample is stored as a single vector

$$[u_b; u] \in \mathbb{R}^{N_b + N_g},$$

where u_b contains boundary values evaluated at the N_b boundary points and u contains the solution values on the N_g grid points. We apply per-sample normalization by dividing the entire vector by its maximum absolute value so that $\max|[u_b; u]| = 1$.

1.2 Burgers dataset generation

We consider the one-dimensional viscous Burgers equation on a periodic domain,

$$\partial_t u(x, t) + u(x, t) \partial_x u(x, t) = \nu \partial_{xx} u(x, t), \quad x \in [0, 2\pi), \quad t \in [0, T], \quad (\text{S3})$$

with periodic boundary conditions and viscosity $\nu = 0.01$. The learning task is to approximate the solution operator mapping the initial condition $u_0(x) = u(x, 0)$ to the solution field $u(x, t)$ for $t > 0$.

Initial condition sampling. Initial conditions are generated as low-frequency random Fourier series,

$$u_0(x) = \sum_{m=1}^K \frac{a_m \cos(mx) + b_m \sin(mx)}{m^\alpha}, \quad (\text{S4})$$

where $a_m, b_m \sim \mathcal{N}(0, 1)$ are independent standard Gaussian variables, $K = 8$ controls the maximum frequency, and $\alpha = 2.0$ controls spectral decay. Each realization is shifted to have zero mean and normalized to unit standard deviation.

Numerical integration. The Burgers equation is integrated on a fine spatial grid ($N_{\text{fine}} = 512$) using a Fourier spectral method combined with an exponential time-differencing fourth-order Runge–Kutta (ETDRK4) scheme. A 2/3 dealiasing rule is applied in Fourier space to avoid aliasing errors. The solution is evolved up to final time $T = 1.0$ with time step $\Delta t = 10^{-4}$.

Downsampling and dataset construction. To obtain training data on a coarse grid, the high-resolution solution is low-pass filtered in Fourier space and then downsampled to $N_x = 64$ spatial points. The solution is recorded at $N_t = 100$ uniformly spaced time snapshots (excluding $t = 0$). Each dataset sample therefore consists of the coarse-grid initial condition $u_0 \in \mathbb{R}^{64}$ and the solution trajectory $u \in \mathbb{R}^{64 \times 100}$. The total dataset size is 2000 samples.

1.3 Darcy flow datasets

We consider two Darcy flow benchmarks with different regularity properties of the permeability field: a discontinuous-coefficient dataset obtained from a publicly released benchmark and a continuous-coefficient dataset generated synthetically for this study.

Discontinuous permeability (public dataset). For the discontinuous-permeability benchmark, we use a publicly released Darcy dataset that has been widely adopted in the neural operator and Fourier Neural Operator (FNO) literature. The dataset consists of paired permeability fields $a(x, y)$ with sharp discontinuities and corresponding solutions $u(x, y)$ of the elliptic problem

$$-\nabla \cdot (a(x, y) \nabla u(x, y)) = 1 \quad \text{in } (0, 1)^2, \quad u = 0 \quad \text{on } \partial(0, 1)^2.$$

The data are provided on a uniform grid of resolution 241×241 and distributed in MATLAB format. We utilize the benchmark dataset originally introduced by [6]. Specifically, we obtain the data files from the public repository `raj-brown/fourier_neural_operator`¹ and use them directly without further modification.

Continuous permeability (synthetic dataset). For the continuous-permeability benchmark, we generate the dataset synthetically using the following procedure.

Coefficient generation. The permeability field $a(x, y)$ is generated by evaluating a randomly initialized sinusoidal multilayer perceptron on a uniform grid. Specifically, we use a fully connected network of architecture $[2, 50, 50, 1]$ with sine activations,

$$a_{\text{raw}}(x, y) = W_3 \sin(W_2 \sin(W_1[x, y]^\top + b_1) + b_2) + b_3,$$

and map it to a strictly positive coefficient via $a(x, y) = 0.1 + \text{softplus}(a_{\text{raw}}(x, y))$. The network weights are sampled independently for each realization.

¹https://github.com/raj-brown/fourier_neural_operator

High-resolution PDE solve. Given $a(x, y)$, we solve the Darcy equation on a high-resolution grid of size 241×241 using a finite-difference discretization with face-averaged coefficients and a sparse direct solver.

Downsampling. Both the coefficient field $a(x, y)$ and the solution $u(x, y)$ are downsampled to a 129×129 grid using bilinear interpolation.

1.4 Poisson–Boltzmann (PB) datasets

All PB datasets follow the following task definition:

$$-\Delta u + k \sinh(u) = f \quad \text{in } \Omega, \quad u|_{\partial\Omega} = g, \quad (\text{S5})$$

where $k > 0$ partially controls the strength of the nonlinearity. The *PB with source* case corresponds to $f \neq 0$, while other PB cases use $f = 0$.

1.4.1 PB on unit square without source

We consider the source-free case $f = 0$ on $\Omega = (0, 1)^2$. The domain is discretized by a uniform $N \times N$ grid ($N = 101$) using the finite difference method (5-point stencil). Boundary values g are sampled on $\partial\Omega$ ($N_b = 400$) and generated as a mixture of low-frequency and high-frequency Gaussian random fields. The resulting discrete nonlinear system is solved using a continuation (homotopy) strategy coupled with a Newton–Raphson method, utilizing a direct sparse solver for the linearized steps. Each accepted realization is stored as a vector $[g; u] \in \mathbb{R}^{N_b + N^2}$.

1.4.2 PB on unit square with source (MAD dataset)

We also consider a source-driven PB dataset following the MAD paradigm [28]. For each sample, we synthesize a smooth field $u(x, y)$ via a randomized sine network and compute the consistent forcing $\tilde{f}(x, y) = \Delta u - k \sinh(u)$. Each sample is stored as $[u_b; \tilde{f}; u]$, where inputs are the boundary trace u_b and the forcing term $\tilde{f}$.

1.4.3 Irregular-domain PB datasets

To evaluate geometric generalization, we generate PB datasets on irregular planar domains using finite-element methods (FEM). Unlike grid-based datasets, these are *node-based*: solutions are stored at FEM mesh nodes.

Geometry and Solver. We generate three benchmarks: (1) a star-shaped polygon, (2) a star with a square hole, and (3) a flower shape with elliptical holes (see Fig. S1). For each geometry, we generate a fixed triangular mesh. Boundary values are sampled from a periodic Gaussian random field and mapped to boundary nodes. The nonlinear system is solved using linear (P1) finite elements on the unstructured mesh. To ensure robust convergence, we employ a homotopy continuation strategy coupled with a damped Newton–Raphson solver. Within each continuation step, the solution is initialized via Picard iterations, and the linearized systems are solved using a direct sparse solver.

1.4.4 3D PB on unit cube

We generate a 3D dataset on $\Omega = [0, 1]^3$ with $k = 1.0$. The domain is discretized into a $33 \times 33 \times 33$ grid ($N_g = 35937$). Dirichlet boundary conditions are sampled on the 6 faces of the cube, resulting

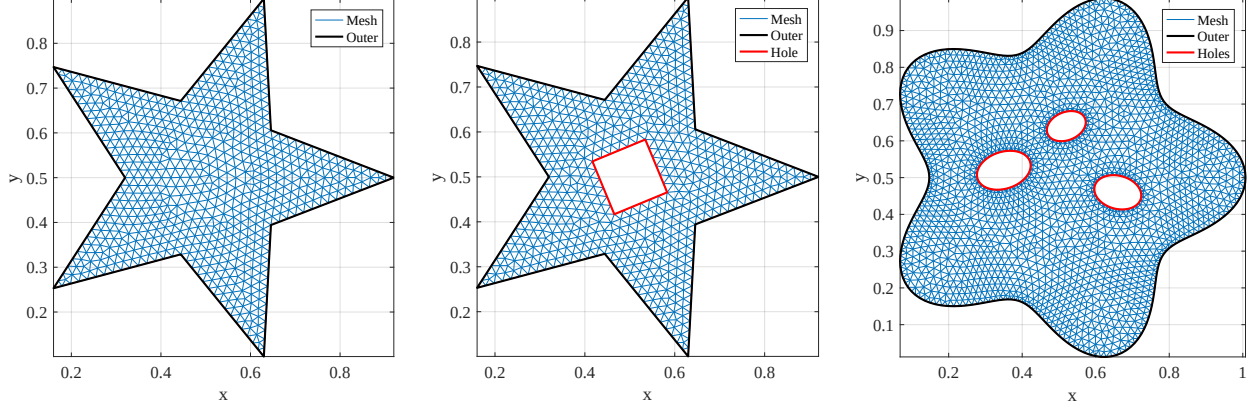


Figure S1: Representative geometries and corresponding finite-element meshes for the irregular-domain Poisson–Boltzmann benchmarks.

in $N_b = 6N^2 - 12N + 8$ unique boundary points. Inputs g are synthesized using random Yukawa potential sources placed outside the domain to ensure smooth but non-trivial boundary patterns. The nonlinear system is solved using a finite-difference discretization combined with a damped Newton–Raphson method, where the linearized systems are solved using a direct sparse solver. The dataset consists of 2000 samples.

1.5 Poisson–Nernst–Planck (PNP) dataset generation

Governing Equations. For operator-learning benchmarking, we consider the dimensionless steady-state Poisson–Nernst–Planck system for a binary symmetric electrolyte. The system describes the coupling between the electrostatic potential ϕ and the ionic concentrations $c_{\pm}$ (see, e.g., [23] for formulation details), governed by the following equations:

$$-\Delta\phi = c_+ - c_-, \quad \nabla \cdot (\nabla c_{\pm} \pm c_{\pm} \nabla \phi) = 0, \quad (\text{S6})$$

where the domain is the unit square $\Omega = (0, 1)^2$. The first equation (Poisson) relates the potential to the charge density difference, while the second pair (Nernst–Planck) describes the conservation of mass under diffusion and electromigration. This formulation captures the essential nonlinear coupling where potential gradients drive ion transport and ion distributions determine the potential field.

Solver & Data. We generate the reference solutions using a Gummel fixed-point iteration scheme. At each iteration step, the Poisson equation and the Nernst–Planck equations are solved sequentially using finite differences on a 129×129 uniform grid until convergence. The input boundary conditions are generated via truncated Fourier series, with strictly enforced positivity ($c_{\pm} > 0$) to ensure physical validity. The final dataset consists of 2000 samples, mapping the three boundary traces ($g_{\phi}, g_{c_+}, g_{c_-}$) to the full interior fields (ϕ, c_+, c_-).

2 Training and evaluation protocol

Hardware and software. Experiments are conducted on a single NVIDIA GeForce RTX 4070 Laptop GPU using PyTorch 2.4.1. Wall-clock training time excludes data generation.

Table S1: Seed stability on the source-free Poisson–Boltzmann benchmark ($k = 1$). We repeat the LNF-NO experiment across five random seeds and report the best test relative ℓ_2 error and total training time.

Seed	Best test rel. ℓ_2	Total training time (s)
0	1.99e-02	671.30
1	1.88e-02	588.14
2	1.88e-02	559.15
3	2.06e-02	680.43
4	2.09e-02	678.09
Mean $\pm$ Std.	$1.98 \times 10^{-2} \pm 8.8 \times 10^{-4}$	635.42 ± 51.35

Data split and normalization. We use a fixed 9:1 train/test split. All inputs and outputs are normalized using global statistics (mean and standard deviation) computed strictly on the training set. Evaluation is always performed on the *physical scale* by decoding predictions back to the original domain.

Loss functions. For scalar fields, we use the relative ℓ_2 error:

$$\text{rel } \ell_2(\hat{u}, u) = \frac{\|\hat{u} - u\|_2}{\|u\|_2 + \varepsilon}, \quad \varepsilon = 10^{-12}.$$

For the coupled PNP task, we use the mean of the relative errors of the three fields (ϕ, c_+, c_-).

Optimization. We use the AdamW optimizer with weight decay 10^{-4} (excluding bias and scale parameters). DeepONet is trained for 5000 epochs, while the other neural operator baselines and LNF-NO are trained for 500 epochs. Learning rates and batch sizes are task- and model-dependent, and are specified in the corresponding experiment scripts.

Reproducibility. Due to computational constraints and to facilitate reproducibility, our main results are reported using a single fixed random seed. To assess sensitivity to initialization, we additionally repeat the source-free PB benchmark ($k = 1$) across five random seeds and report the corresponding results in Table S1.

Code availability. The implementation of LNF-NO and the core baseline scripts will be made available upon reasonable request.

2.1 Seed stability on the source-free PB benchmark

To examine the sensitivity of LNF-NO to random initialization, we repeat the source-free Poisson–Boltzmann benchmark with $k = 1$ across five random seeds. For each run, we report the best test relative ℓ_2 error and the total wall-clock training time.

The results indicate that LNF-NO is reasonably robust to random initialization on this benchmark, with only modest variation in both final accuracy and total training time across the tested seeds.

3 LNF-NO architecture details

We categorize the architectures based on their input-output configurations: **Single-Input Single-Output (SISO)**, mapping a single functional input to a single output field; **Multiple-Input Single-Output (MISO)**, processing multiple functional inputs to predict a single output field; and **Multiple-Input Multiple-Output (MIMO)**, processing multiple functional inputs to predict multiple coupled output fields.

All LNF-NO variants share the same multiplicative fusion structure:

$$\hat{y} = \alpha(\mathcal{B}_L(z) \odot \mathcal{B}_N(z)),$$

where an encoder maps the input to latent features z , and α is a learned scalar. The linear branch $\mathcal{B}_L$ is a pure linear map (affine transformation), while the nonlinear branch $\mathcal{B}_N$ uses GELU activations. Unless otherwise noted, intermediate convolutional layers in the encoders and decoders are also followed by GELU activations.

3.1 Architecture Configurations

(A) Regular-domain SISO (Laplace, PB). Input: Boundary $g \in \mathbb{R}^{N_b}$. Output: Grid $u \in \mathbb{R}^{N^2}$.

- **Encoder (1D convolutional neural network, CNN):** Four layers.
 - Conv1d(1, 96, k=9, s=1, p=4)
 - Three layers of Conv1d(96, 96, k=9, s=2, p=4)
- **Branches (MLP):** Feature dim d to grid dim $D = N^2$.
 - $\mathcal{B}_L$: Linear $[d \rightarrow 256 \rightarrow D]$. Note that while mathematically equivalent to a single linear map, the two-layer structure (without intermediate activation) is maintained to align the latent width with the nonlinear branch for elementwise fusion and to provide additional optimization degrees of freedom.
 - $\mathcal{B}_N$: MLP $[d \rightarrow 256 \rightarrow 256 \rightarrow D]$ with GELU activations.
- **Decoder (2D CNN):**
 - Conv2d(1, 32, k=3, s=1, p=1)
 - Conv2d(32, 32, k=3, s=1, p=1)
 - Conv2d(32, 1, k=3, s=1, p=1)

(B) Regular-domain MISO (PB-Source). Input: Boundary g and Source f . Output: Grid u .

- **Encoders:**
 - Boundary: Same as (A).
 - Source: Four-layer 2D CNN with channels $1 \rightarrow 48$, followed by Adaptive Avg Pool to 8×8 .
- **Branches & Decoder:** Identical to (A).

(C) **Irregular-domain SISO (PB-FEM).** Input: Boundary g . Output: Node vector $u \in \mathbb{R}^{N_p}$.

- **Encoder:** Same 1D boundary encoder as (A).
- **Branches:** Output dimension $D = N_p$ (mesh nodes).
 - $\mathcal{B}_L$: Linear $[d \rightarrow 256 \rightarrow D]$.
 - $\mathcal{B}_N$: MLP $[d \rightarrow 256 \rightarrow 256 \rightarrow D]$ with GELU activations.
- **Decoder:** None (direct node prediction).

(D) **Regular-domain MIMO (PNP).** Input: 3 Boundaries. Output: 3 Fields.

- **Encoders:** Three independent 1D encoders, each with the same architecture as in (A).
- **Branches:** Output dim $D = 3N_xN_y$.
- **Decoder:** 3-channel 2D CNN (3->64->64->3).

(E) **3D Poisson–Boltzmann (Regular-domain SISO).** Input: Boundary $g \in \mathbb{R}^{N_b}$. Output: Grid $u \in \mathbb{R}^{N^3}$ ($N = 33$). Unlike the 2D cases, we omit the explicit 1D encoder for the boundary vector to reduce memory overhead, passing g directly to the branches.

- **Branches (MLP):** Map input dimension N_b directly to grid dimension $D = N^3$.
 - $\mathcal{B}_L$: Linear projection $[N_b \rightarrow 256 \rightarrow D]$.
 - $\mathcal{B}_N$: MLP $[N_b \rightarrow 256 \rightarrow 256 \rightarrow D]$ with GELU activations.
- **Decoder (3D CNN):**
 - Conv3d(1, 32, k=3, s=1, p=1)
 - Conv3d(32, 32, k=3, s=1, p=1)
 - Conv3d(32, 1, k=3, s=1, p=1)

4 Baselines

To benchmark the performance of LNF-NO, we compare it against several representative neural operator baselines, including the Deep Operator Network (DeepONet) [4], the Fourier Neural Operator (FNO) [6], the U-shaped Neural Operator (UNO) [8], and Transolver [9]. The hyperparameters selected for the baselines (e.g., basis dimension p and frequency modes $k_{\max}$) align with standard configurations widely used in the community to ensure a fair comparison. All baseline models are trained using the same general protocol as LNF-NO:

- **Data Split:** A fixed 9:1 training-to-testing ratio is used for all datasets.
- **Normalization:** Inputs and outputs are normalized using global statistics computed strictly on the training set.
- **Metrics:** All reported errors are relative ℓ_2 norms computed on the decoded physical scale after inverse normalization.

- **Optimization:** We use the AdamW optimizer, with bias terms excluded from weight decay. Learning rates, batch sizes, and training schedules are selected according to the model type and task, while remaining consistent with the evaluation protocol described in Section 2.

DeepONet models are trained with a larger epoch budget (5000 epochs), whereas the other baselines are trained for 500 epochs, reflecting their different optimization characteristics.

4.1 Deep Operator Network (DeepONet)

We employ the unstacked DeepONet architecture, which approximates the operator $\mathcal{G}$ by taking an input function u (via a discrete sensor vector) and a query coordinate y to predict the solution value $\mathcal{G}(u)(y)$. The network consists of two sub-networks: a *Branch Net* encoding the input function and a *Trunk Net* encoding the coordinates. The final prediction is given by the inner product of their outputs (plus an optional bias):

$$\hat{u}(y) = \sum_{k=1}^p b_k(u) \cdot t_k(y) + \beta,$$

where $\{b_k\}_{k=1}^p$ and $\{t_k\}_{k=1}^p$ are the p -dimensional outputs of the branch and trunk networks, respectively. For all tasks, we set the basis dimension to $p = 1024$. The Branch and Trunk networks are implemented as 5-layer MLPs with a hidden width of 256 and $\tanh$ activations.

(A) Regular-domain SISO (Laplace, PB). The branch net input is the boundary vector $g \in \mathbb{R}^{N_b}$, and the trunk net input is the coordinate $x \in \mathbb{R}^2$.

- Branch: $\text{MLP}(N_b \rightarrow 1024)$ with hidden width 256.
- Trunk: $\text{MLP}(2 \rightarrow 1024)$ with hidden width 256.

For the 3D PB task, the trunk net takes 3D coordinates (x, y, z) as input. The architecture parameters remain consistent ($p = 1024$, width=256).

(B) Regular-domain MISO (PB-Source). For tasks with multiple inputs (boundary $u_b \in \mathbb{R}^{N_b}$ and source $f \in \mathbb{R}^{N^2}$), we employ a multi-branch fusion strategy (similar to MIONet [29]). Separate branch nets encode u_b and f , and their embeddings are fused via elementwise multiplication before the dot product with the trunk:

$$b(u_b, f) = b^{(u)}(u_b) \odot b^{(f)}(f) \in \mathbb{R}^p.$$

- Branch 1: $\text{MLP}(N_b \rightarrow 1024)$.
- Branch 2: $\text{MLP}(N^2 \rightarrow 1024)$.
- Trunk: $\text{MLP}(2 \rightarrow 1024)$.

(C) Regular-domain MIMO (PNP). For the coupled PNP system, we use a single concatenated branch net. The trunk net is modified to have three separate output heads to predict the three fields (ϕ, c_+, c_-) simultaneously:

$$\hat{u}^{(m)}(x) = \sum_{k=1}^p b_k(\mathbf{u}_{bc}) \cdot t_k^{(m)}(x), \quad m \in \{\phi, c_+, c_-\}.$$

- Branch: $\text{MLP}(3N_b \rightarrow 1024)$.
- Trunk: $\text{MLP}(2 \rightarrow 3 \times 1024)$.

4.2 Fourier Neural Operator (FNO)

We adopt the standard Fourier Neural Operator architecture. The model processes the input $a(x)$ through three main stages:

1. **Encoder (Lifting):** The input is projected pointwise to a high-dimensional channel space $v_0(x) \in \mathbb{R}^{d_h}$ via a linear layer or shallow MLP.
2. **Fourier Integral Layers:** The latent representation is updated through a sequence of L layers. The update rule for the l -th layer is:

$$v_{l+1}(x) = \sigma \left(W_l v_l(x) + (\mathcal{F}^{-1} \circ R_l \circ \mathcal{F})(v_l)(x) \right),$$

where σ is a nonlinear activation function, W_l is a pointwise linear transform (residual connection), $\mathcal{F}$ denotes the FFT, and R_l is a learnable complex weight tensor that mixes the lowest $k_{\max}$ frequency modes.

3. **Decoder (Projection):** The final features $v_L(x)$ are mapped to the target dimension using a pointwise MLP decoder.

Input lifting strategy. Since FNO requires grid-structured inputs, strictly lower-dimensional inputs (e.g., boundary conditions) are mapped to the full spatiotemporal grid prior to the lifting stage. We employ standard techniques consistent with the literature: *domain replication* for scalar/vector inputs, *zero-padding* (placing boundary values on the perimeter with zero-filled interiors) for boundary fields, or learned *MLP projection*. The resulting fields are concatenated with coordinate grids.

Hyperparameters. We set the network depth to $L = 4$ for all tasks. Consistent with our training protocol, we employ ReLU activation for all FNO baselines. Specific structural parameters are:

- **2D steady-state (Laplace, PB, Darcy, PNP):** We use a 2D FNO with $k_{\max} = 16$ modes per direction and width $d_h = 64$. The PNP model uses a 5-channel input (via zero-padding) and predicts 3 fields simultaneously.
- **Burgers (1D time-dependent):** Modeled as a 2D FNO on the (x, t) domain. Configuration: $k_{\max} = 16$, $d_h = 64$.
- **3D Poisson–Boltzmann:** Modeled as a 3D FNO on the (x, y, z) domain. Configuration: $k_{\max} = 8$ modes per direction, width $d_h = 32$.

4.3 U-shaped Neural Operator (UNO)

We adopt the U-shaped Neural Operator (UNO) [8] as a neural operator baseline. UNO is a multi-resolution neural operator architecture that combines global operator learning with a U-shaped encoder–decoder representation. Unlike the standard FNO, which operates at a fixed grid resolution, UNO maps features across multiple grid resolutions, allowing the model to capture both global interactions and fine-scale spatial structures.

The architecture consists of three main components:

1. **Input lifting:** The grid-based input field is concatenated with spatial or spatiotemporal coordinate channels and projected pointwise to a higher-dimensional feature representation.

2. **Multi-resolution operator blocks:** The latent representation is processed by a hierarchy of operator blocks arranged along an encoder–decoder path. The encoder path maps the representation to progressively coarser grids, while the decoder path maps it back to the target output resolution. Each block applies a learned integral operator, implemented in the Fourier domain, together with pointwise channel mixing. The encoder–decoder hierarchy transfers features between different spatial resolutions, and skip connections pass high-resolution representations from the encoder path to the decoder path, mitigating information loss caused by the coarse-grid bottleneck.
3. **Output projection:** After the representation is mapped to the target grid resolution, the final feature map is projected to the target solution field by a pointwise MLP.

Input representation. Following the FNO setting, all inputs are represented as grid-based fields. For coefficient-driven problems such as Darcy flow, the coefficient field is used directly as the input field. For boundary-driven problems such as the Poisson–Boltzmann and Laplace equations, the lower-dimensional boundary data are first embedded into grid-based input fields using the same boundary-to-grid preprocessing strategy as in the corresponding FNO baseline. For source-driven Poisson–Boltzmann problems, the grid-embedded boundary field and the discretized source field are concatenated as input channels. For multi-field systems such as the Poisson–Nernst–Planck equation, multiple grid-embedded boundary fields are used as input channels together with spatial coordinate information, and the model predicts multiple solution fields simultaneously.

Hyperparameters. For each task group, we report the maximum retained Fourier modes and the base channel width. Whenever possible, these settings are kept consistent with the corresponding FNO baseline so that the comparison mainly reflects architectural differences rather than hyperparameter changes. The configurations are summarized as follows:

- **2D steady-state problems (Darcy, Poisson–Boltzmann, Laplace):** We use a 2D UNO with up to $k_{\max} = 16$ retained Fourier modes per spatial dimension and base width $d_h = 64$.
- **2D Poisson–Boltzmann with source:** We use the same 2D UNO configuration with up to $k_{\max} = 16$ retained Fourier modes per spatial dimension and base width $d_h = 64$. The input contains both the grid-embedded boundary information and the discretized source field.
- **2D Poisson–Nernst–Planck (PNP):** We use a 2D UNO with up to $k_{\max} = 16$ retained Fourier modes per spatial dimension and base width $d_h = 64$. The input consists of the grid-embedded boundary fields for ϕ , c_+ , and c_- , together with spatial coordinate information. The model predicts the three solution fields simultaneously.
- **Burgers equation (1D time-dependent):** The problem is represented on the (x, t) domain and modeled using a 2D UNO with up to $k_{\max} = 16$ retained Fourier modes per dimension and base width $d_h = 64$.
- **3D Poisson–Boltzmann:** We use a 3D UNO on the (x, y, z) domain with up to $k_{\max} = 8$ retained Fourier modes per spatial dimension and base width $d_h = 32$.

4.4 Transolver

We adopt Transolver [9] as an attention-based neural operator baseline. Unlike DeepONet, which evaluates the solution at query coordinates through branch–trunk inner products, and unlike FNO,

which relies on spectral convolution on regular grids, Transolver treats the computational domain as a set of tokens and updates token features through physics-attention blocks. For structured-grid problems, the tokens correspond to points on the target Cartesian grid. For irregular-domain finite-element problems, the tokens correspond to FEM mesh nodes.

Backbone. For the two-dimensional structured-grid benchmarks, we use a structured-mesh Transolver backbone. Given token coordinates $x_i \in \mathbb{R}^2$ and optional functional features f_i , the input token feature is first formed by concatenating positional information and functional features, followed by an MLP preprocessing layer. When `unified_pos=True`, the coordinate input is replaced by a unified positional representation based on a reference grid of size `ref` $\times$ `ref`. The resulting token features are passed through stacked Transolver blocks. Each block consists of layer normalization, a physics-attention module, a residual connection, and a feedforward MLP. The last block projects the hidden feature to the output field dimension.

For most two-dimensional regular-grid Transolver baselines, we use the following configuration:

- $n_{\text{hidden}} = 128$, $n_{\text{layers}} = 4$, and $n_{\text{head}} = 8$;
- `slice_num=64`, `mlp_ratio=1`, and `dropout=0`.

We use GELU activations and set the reference-grid parameter to `ref = 8`. The structured-grid models use `unified_pos=True` unless otherwise specified.

(A) Regular-domain SISO benchmarks. For boundary-to-grid tasks such as Laplace and source-free Poisson–Boltzmann equations, the input boundary vector $g \in \mathbb{R}^{N_b}$ is first mapped to grid-wise functional features through a global lifting MLP:

$$g \mapsto F_g \in \mathbb{R}^{N^2 \times d_f}.$$

In our implementation, this lifting module is

$$\text{MLP}(N_b \rightarrow 256 \rightarrow 256 \rightarrow N^2 d_f),$$

with GELU activations and $d_f = 1$ for these SISO tasks. The lifted feature F_g is concatenated with positional features and passed to the structured 2D Transolver backbone, which predicts the grid solution $u \in \mathbb{R}^{N^2}$. The same boundary-lift strategy is used for the Laplace and source-free PB benchmarks, with task-dependent grid sizes.

(B) Burgers equation. For the Burgers benchmark, the input initial condition $u_0(x)$ is represented on the spatial grid and broadcast along the temporal direction to form a full space–time functional field on the (x, t) grid. The resulting field, together with the space–time coordinates, is processed by the structured 2D Transolver backbone. The model outputs the solution trajectory $u(x, t)$ on the full $N_x \times N_t$ space–time grid. This treatment follows the same token-based structured-grid formulation as the steady two-dimensional tasks, but with the two token coordinates corresponding to space and time.

(C) Darcy flow benchmarks. For Darcy flow, the input coefficient field $a(x, y)$ is already defined on the target grid. Therefore, no boundary-to-grid lifting is required. The coefficient field is used as a grid-wise functional input channel and concatenated with positional information before entering the structured 2D Transolver backbone. We use this formulation for both continuous- and discontinuous-coefficient Darcy benchmarks, with grid sizes matching the corresponding datasets.

(D) Regular-domain MISO benchmark: PB with source. For the source-driven Poisson–Boltzmann benchmark, the input consists of a boundary trace $u_b \in \mathbb{R}^{N_b}$ and a source field $f \in \mathbb{R}^{N^2}$. The source field is already a full-grid input and is kept as one functional channel. The boundary trace is mapped to additional grid-wise channels through a global lifting MLP:

$$u_b \mapsto F_{u_b} \in \mathbb{R}^{N^2 \times d_b}.$$

In our implementation, we use $d_b = 4$ and a lifting hidden width of 256. The final functional input to Transolver is the concatenation

$$F = [f, F_{u_b}] \in \mathbb{R}^{N^2 \times (1+d_b)}.$$

This concatenated grid-wise feature is then processed by the structured 2D Transolver backbone to predict the solution field u .

(E) Regular-domain MIMO benchmark: PNP. For the coupled PNP system, the input contains three boundary traces $(g_\phi, g_{c_+}, g_{c_-})$, and the output contains three fields (ϕ, c_+, c_-) . To construct a grid-compatible input for Transolver, we place the three boundary traces on the perimeter of a full grid and fill the interior values with zeros, producing three full-grid input channels. These channels are used as the functional input to the structured 2D Transolver backbone. The final projection layer is modified to output three channels simultaneously, corresponding to the three coupled fields. The training loss is the mean of the decoded physical-scale relative ℓ_2 errors over the three output fields, consistent with the evaluation protocol.

(F) Irregular-domain PB benchmarks. For irregular finite-element PB benchmarks, we use an irregular-mesh Transolver variant. Each FEM node is treated as a token with coordinate $x_i \in \mathbb{R}^2$. The boundary vector $g \in \mathbb{R}^{N_b}$ is lifted to node-wise functional features through a global MLP:

$$g \mapsto F_g \in \mathbb{R}^{N_p \times d_f},$$

where N_p is the number of FEM nodes. The node coordinates and lifted node-wise features are then passed to the irregular Transolver backbone to predict the nodal solution vector $u \in \mathbb{R}^{N_p}$. For these irregular-domain benchmarks, we use the following configuration:

- `n_hidden = 128`, `n_layers = 8`, and `n_head = 8`;
- `slice_num=64`, `ref=8`, and `unified_pos=False`;
- `mlp_ratio=1` and `dropout=0`.

GELU activations are used throughout the Transolver blocks.

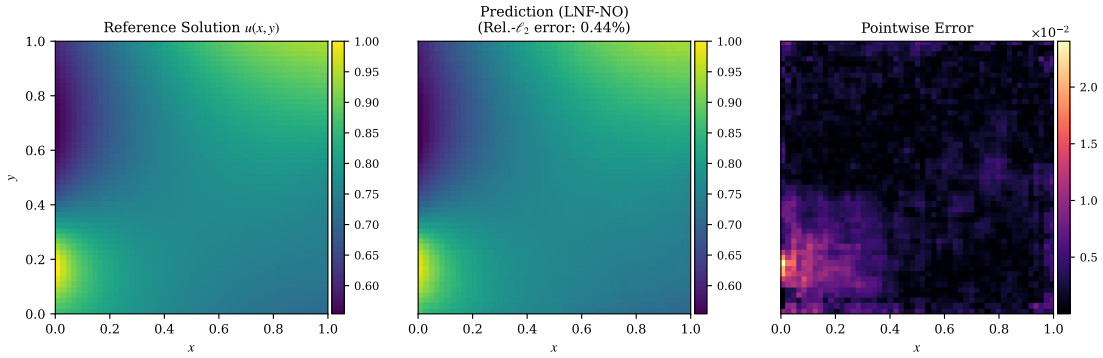
(G) 3D Poisson–Boltzmann benchmark. For the 3D PB benchmark, we use a structured 3D Transolver variant. The unique boundary vector $g \in \mathbb{R}^{N_b}$ is first lifted to a full-grid functional field on the $N \times N \times N$ grid. The lifted feature is then combined with three-dimensional positional information and processed by a 3D structured Transolver backbone. Compared with the 2D structured-grid case, the 3D model uses a lighter attention configuration to control memory usage while keeping the same overall token-based design. The final output is the full volumetric solution field $u \in \mathbb{R}^{N^3}$.

Training setup. All Transolver baselines are trained using the same general evaluation protocol as the other methods: a fixed train/test split, train-only normalization, AdamW optimization, StepLR with $\gamma = 1.0$, and relative ℓ_2 errors computed after decoding predictions back to the physical scale. For multi-field PNP outputs, the reported error is the average relative ℓ_2 error over the three physical fields.

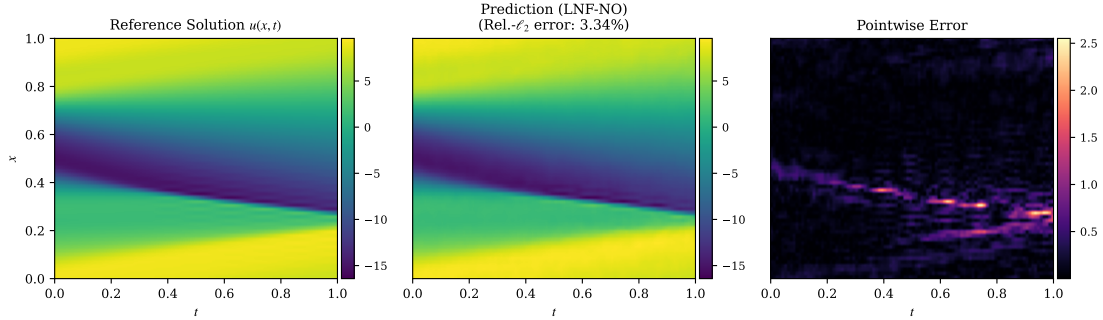
5 Visualization of predictions

In this section, we provide qualitative visualizations of the LNF-NO predictions on unseen test samples. Unless otherwise stated, the columns display the reference solution obtained from the numerical solver, the prediction produced by LNF-NO, and the pointwise absolute error map $|u - \hat{u}|$. For tasks with spatially varying coefficients or source terms, the input field is also visualized in the first column.

Visualizations for the three-dimensional Poisson–Boltzmann benchmark are presented in the main manuscript and are therefore omitted here to avoid redundancy.



(a) **Laplace equation.** Prediction of the harmonic function on a regular grid.



(b) **Burgers equation.** A snapshot of the wave propagation. The model resolves the shock front with minimal oscillation.

Figure S2: Standard benchmark visualizations for the steady-state two-dimensional Laplace equation and the time-dependent one-dimensional Burgers equation.

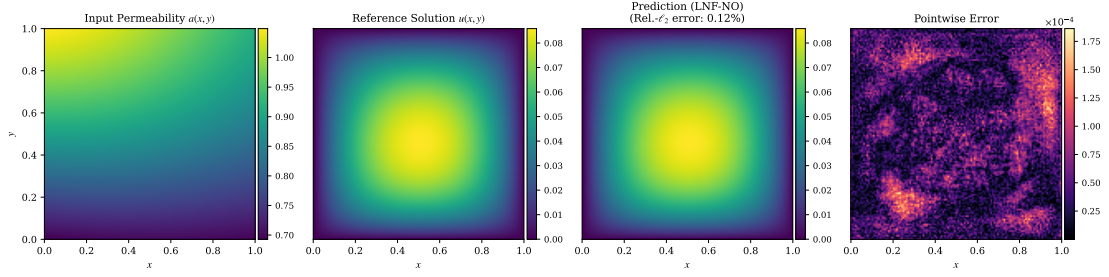


Figure S3: Darcy flow with continuous coefficients. From left to right: input permeability $a(x, y)$, reference solution $u(x, y)$, LNF-NO prediction, and pointwise absolute error.

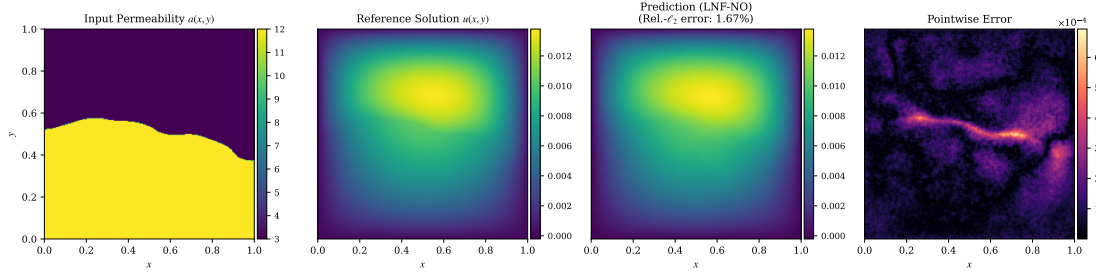


Figure S4: Darcy flow with discontinuous coefficients. The visualization shows the predicted solution and pointwise error under jump coefficients.

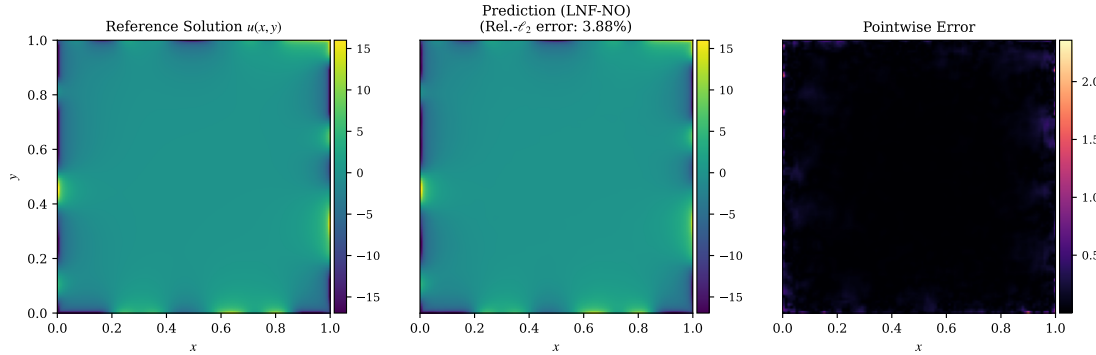


Figure S5: Poisson-Boltzmann equation with high nonlinearity ($k = 100$). The visualization shows the reference solution, LNF-NO prediction, and pointwise absolute error.

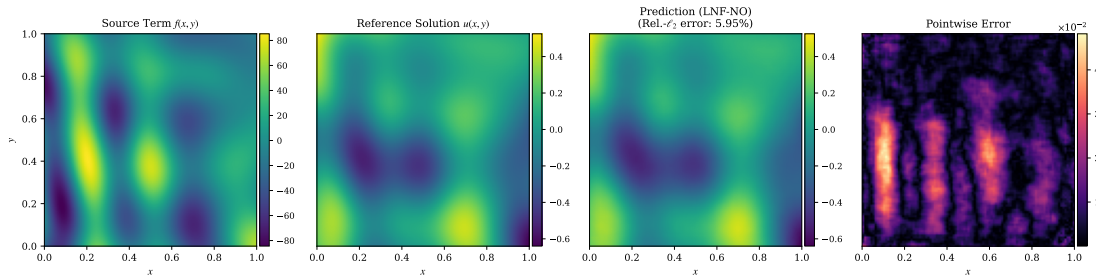


Figure S6: Source-driven Poisson-Boltzmann equation. The variable source term $f(x, y)$ is included as an additional input field.

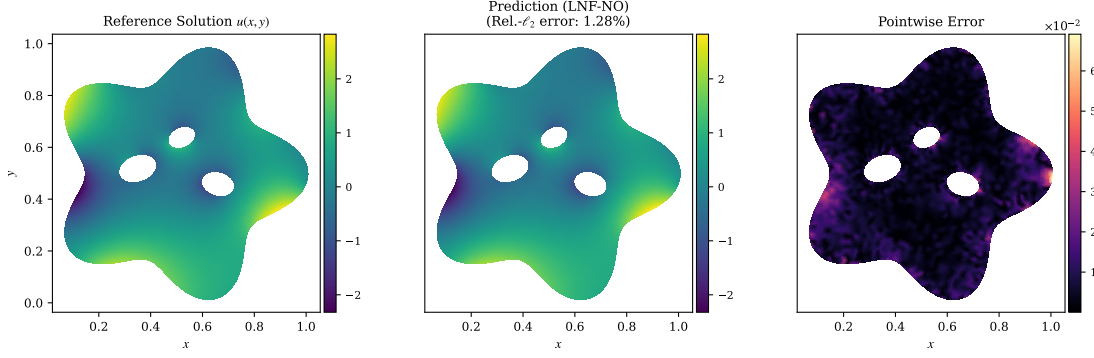


Figure S7: Geometric generalization on irregular Poisson–Boltzmann benchmarks. Predictions are shown on complex non-simply-connected domains, including a flower-shaped domain with elliptical holes. Since this task uses the node-based architecture, results are plotted directly on the finite-element mesh triangulation.

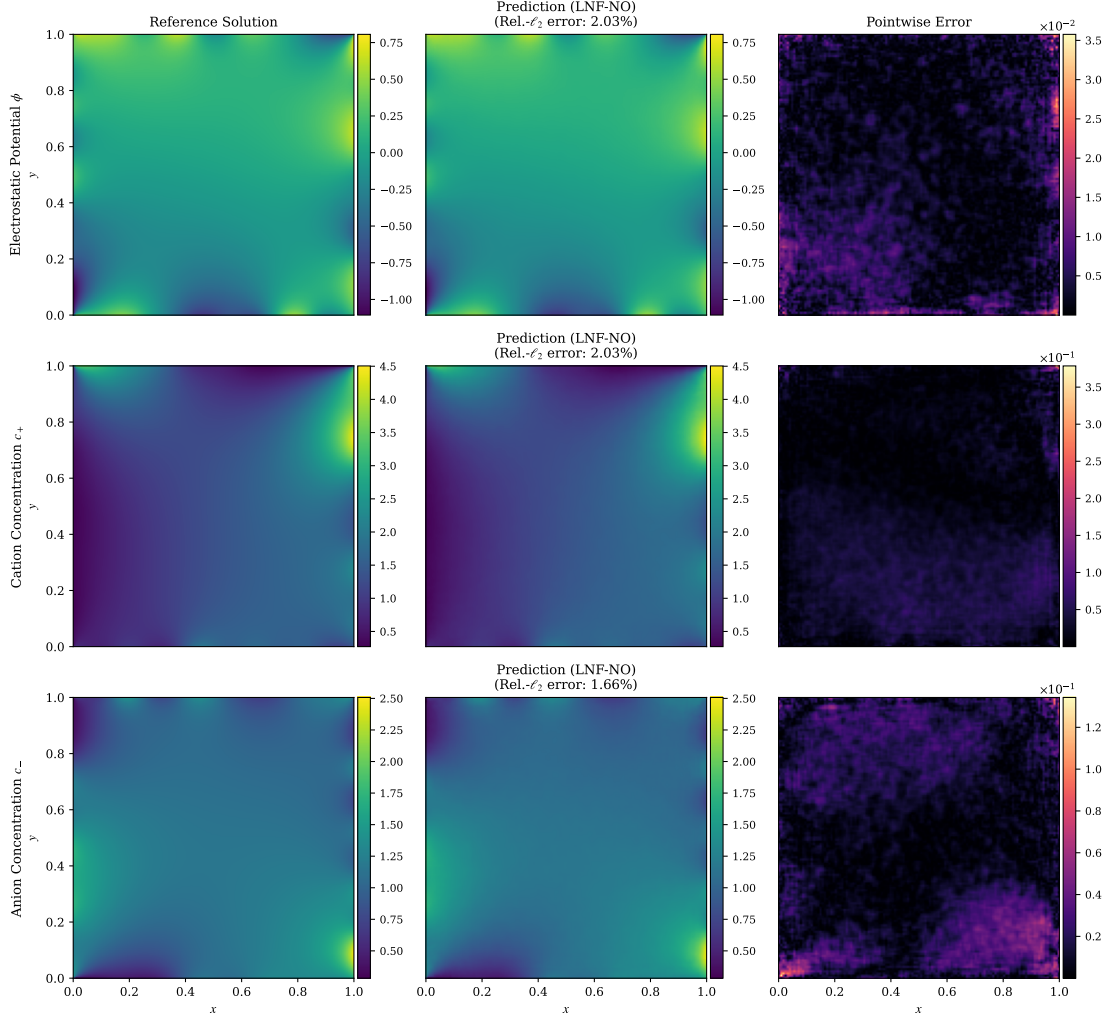


Figure S8: Poisson–Nernst–Planck system. LNF-NO simultaneously predicts three coupled fields: the electrostatic potential ϕ , cation concentration c_+ , and anion concentration c_- . The error maps show the pointwise discrepancies for the three coupled fields.