Invertible Fourier Neural Operators for Tackling Both Forward and Inverse Problems

Da Long **Zhitong Xu** **Qiwei Yuan** **Yin Yang** **Shandian Zhe**
Kahlert School of Computing, University of Utah
{da.long, u1502956, joshua.yuan, yin.yang}@utah.edu, zhe@cs.utah.edu

Abstract

Fourier Neural Operator (FNO) is a powerful and popular operator learning method. However, FNO is mainly used in forward prediction, yet a great many applications rely on solving inverse problems. In this paper, we propose an invertible Fourier Neural Operator (iFNO) for jointly tackling the forward and inverse problems. We developed a series of invertible Fourier blocks in the latent channel space to share the model parameters, exchange the information, and mutually regularize the learning for the bi-directional tasks. We integrated a variational auto-encoder to capture the intrinsic structures within the input space and to enable posterior inference so as to mitigate challenges of illposedness, data shortage, noises that are common in inverse problems. We proposed a three-step process to combine the invertible blocks and the VAE component for effective training. The evaluations on seven benchmark forward and inverse tasks have demonstrated the advantages of our approach. The code is available at https://github.com/BayesianAIGroup/iFNO.

1 INTRODUCTION

Operator learning (OL) is currently at the forefront of AI for science. It seeks to estimate function-to-function mappings from data and can be used as a valuable surrogate in various applications related to scientific simulation. Among the notable approaches in this domain is the Fourier neural operator (FNO) (Li et al., 2020). Leveraging the convolution theorem and fast Fourier transform (FFT), FNO executes a sequence of global linear transform and nonlinear activation within the functional space to capture complex function-to-function mappings. FNO is computationally efficient, and has shown excellent performance across many OL tasks.

Despite many success stories (Pathak et al., 2022; Kovachki et al., 2023; Kashefi and Mukerji, 2024), FNO is primarily applied for solving forward problems. Typically, it is used to predict solution functions based on the input sources, parameters of partial differential equations (PDEs), and/or initial conditions. However, practical applications often involve another crucial category of tasks, namely inverse problems (Stuart, 2010). For instance, given the solution measurements, how to deduce the unknown sources, how to determine the PDE parameters, or identify the initial conditions? Inverse problems tend to be more challenging due to issues such as ill-posedness (Tikhonov, 1963; Engl and Groetsch, 2014), noisy measurements, and limited data quantity. Even if one attempts to directly train an FNO to map the measured solution function back to the input of interest, the aforementioned challenges persist.

To bridge this gap, we propose iFNO, a novel invertible Fourier neural operator jointly addressing both forward and inverse problems. Due to the sharing of the model parameters, the co-learning for the bi-directional tasks enables efficient information exchange and mutual regularization, enhancing performance on both fronts and mitigating challenges especially for inverse problems. Specifically, we first develop a series of invertible Fourier blocks in the lifted channel space, capturing rich representation information. Each block takes a pair of inputs with an equal number of channels, generating outputs through the Fourier layer of FNO, softplus transform, and element-wise multiplication. This ensures a rigorous bijection pair between the inputs and outputs during expressive functional transforms in the latent channel space. To enable inverse prediction in the original space, we incorporate a pair of multi-layer perceptions (MLPs) that lift the final prediction's channel back to the latent space and project the latent channels to the input space. Second, to capture intrinsic structures within the input space and further mitigate challenges like ill-posedness and data shortage, we introduce a low-dimensional representation and integrate a variational auto-encoder (VAE) to reconstruct the input function values. The VAE component

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258. Copyright 2025 by the author(s).

also enables generation of posterior samples for prediction. Third, for effective training, we developed a three-step process: We first train the invertible blocks and the VAE component separately, then combine them to continue training, resulting in the final model.

For evaluation, we examined our method in seven benchmark problems, including scenarios based on Darcy flow, second-order wave propagation, diffusion reaction, and Naiver-Stoke (NS) equations. In addition to forward solution prediction, our method was tested for inverse inference, specifically deducing permeability, square slowness, initial conditions, earlier system states from (noisy) solution or system state measurements. We compared with the recent inverse neural operator (Molinaro et al., 2023) designed explicitly for solving inverse problems, and invertible deep operator net (Kaltenbach et al., 2022). Comparisons were also made with FNO and another recent state-of-the-part attention-based neural operator, both of which were trained *separately* for forward and inverse prediction. In addition, we tested with adapting FNO to the classical framework for solving inverse problem. That is, one optimizes the input to the forward model to match the prediction with observations. Across all the tasks, iFNO consistently delivers the best or near-best performance, often outperforming the competing methods by a large margin. In the vast majority of cases, iFNO significantly improves upon the standard FNO in both forward and inverse predictions. Visualization of the predictive variance for the inverse problems reveals intriguing and reasonable uncertainty calibration results.

2 PRELIMINARIES

Operator Learning. Consider learning a mapping between two function spaces (*e.g.,* Banach spaces) $\psi : \mathcal{H} \rightarrow \mathcal{U}$. We collect a training dataset that consists of pairs of discretized input and output functions, denoted by $\mathcal{D} = \{(\mathbf{f}_n, \mathbf{u}_n)\}_{n=1}^N$. Each $\mathbf{f}_n$ and $\mathbf{u}_n$ represents samples from an input function $f_n \in \mathcal{H}$ and the corresponding output function $u_n = \psi(f_n) \in \mathcal{U}$, respectively. Both f_n and u_n are sampled at evenly-spaced locations, *e.g.,* over a 64×64 grid in $[0, 1]^2$.

Fourier Neural Operator (FNO) first lifts the sampled input function values $\mathbf{f}$ into a higher-dimensional feature space (via an MLP) to enrich the representation. Then it uses a series of Fourier layers to alternatingly conduct linear transforms and nonlinear activation,

$$v_{t+1}(\mathbf{x}) \leftarrow \sigma\left(\mathcal{W}v_t(\mathbf{x}) + \int \kappa(\mathbf{x}-\mathbf{x}')v_t(\mathbf{x}')\mathrm{d}\mathbf{x}'\right), \quad (1)$$

where v_t is the input to the t-th layer, v_{t+1} the output, κ the integration kernel, and σ the activation. Utilizing the convolution theorem expressed as

$$\int \kappa(\mathbf{x}-\mathbf{x}')v_t(\mathbf{x}')\mathrm{d}\mathbf{x}' = \mathcal{F}^{-1}\left[\mathcal{F}[\kappa]\cdot\mathcal{F}[v_t]\right](\mathbf{x}),$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier and inverse Fourier transforms, respectively, the Fourier layer first executes a fast Fourier transform (FFT) over v_t, then multiplies the result with the discretized representation of κ in the frequency domain, *i.e.,* $\mathcal{F}[\kappa]$, and executes inverse FFT; $\mathcal{W}v_t(\mathbf{x})$ perform a location-wise linear transform. Owning to FFT, the Fourier layer is computationally efficient. The storage of $\mathcal{F}[\kappa]$ can be memory intensive. In practice, one often truncates the high frequency modes to save the memory cost. After several Fourier layers, another MLP is used to project back from the latent space to generate the final prediction. The training is typically carried out by minimizing a relative L_2 loss.

3 INVERTIBLE FOURIER NEURAL OPERATORS

Neural operators, including FNO, are commonly employed to address forward problems. Taking PDE systems as an example, the input typically comprises external sources or forces, system parameters, and/or initial conditions, with the output representing the corresponding solution function. However, many applications require deducing unknown causes from the observed effect or measurement data, a scenario known as inverse problems. For instance, this involves inferring unknown sources, system parameters, or initial conditions from solution measurements. Inverse problems have broad significance in scientific and engineering domains, *e.g.,* (Tanaka and Dulikravich, 1998; Yilmaz, 2001; Nashed and Scherzer, 2002). However, tackling inverse problems is notably challenging due to their inherently ill-posed nature (Tikhonov, 1963; Stuart, 2010; Engl and Groetsch, 2014). In contrast to forward problems, inverse problems often lack a unique solution and are highly sensitive to variations in data. These challenges are further compounded in complex applications where measurement data is often limited, noisy, and/or inaccurate.

To better address these challenges, we propose iFNO, an invertible Fourier neural operator jointly tackling both the forward and inverse problems. Gven the training dataset $\mathcal{D} = \{\mathbf{f}_n, \mathbf{u}_n\}_{n=1}^N$, our goal is to jointly learn the forward and inverse mappings via a *unified* neural operator. By sharing the model parameters across both tasks, we conduct co-learning for the bi-directional tasks to enable efficient information exchange and mutual regularization, so as to improve performance on both fronts and alleviate aforementioned challenges like ill-posedness and data noise. The details of iFNO are specified as follows.

3.1 Invertible Fourier Blocks in Latent Space

To enable invertible computing and prediction while preserving the expressiveness of the original FNO, we first design and stack a series of invertible Fourier blocks in the latent channel space. Specifically, following the standard FNO, we apply a multi-layer preceptron (MLP) $\mathcal{P}$ over each element

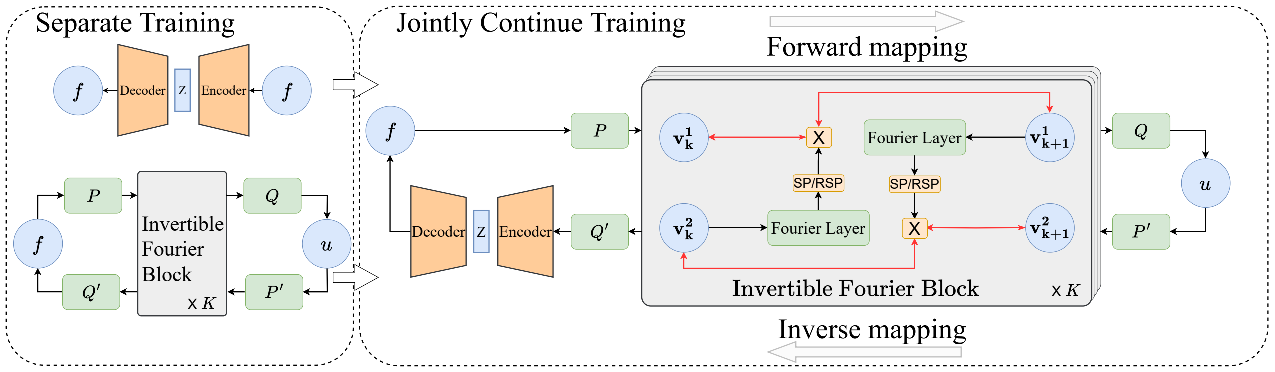


Figure 1: An Overview of iFNO. Left panel: the β-VAE module and the invertible Fourier blocks. Right panel: the entire architecture. “SP” and “RSP” denote softplus and the reciprocal of the softplus, respectively. We first train separately the β-VAE module and invertible Fourier blocks as shown in the first panel. Then we combine them to continue training the entire model as depicted in the right panel.

of the discretized input function $\mathbf{f}$ and the sampling locations to map $\mathbf{f}$ into a higher-dimensional latent space with $2d$ channels. That means, at each sampling location, we have a $2d$-dimensional feature representation. We split the channels of $\mathcal{P}(\mathbf{f})$ into halves, each with d channels, and feed them into a series of invertible Fourier blocks $\mathcal{IF}_1, \ldots, \mathcal{IF}_K$. Each block $\mathcal{IF}_k$ receives a pair of inputs $\mathbf{v}_k^1$ and $\mathbf{v}_k^2$, and produces a pair of outputs $\mathbf{v}_{k+1}^1$ and $\mathbf{v}_{k+1}^2$, which are then fed into $\mathcal{IF}_{k+1}$. All the input and outputs possess the same number of channels. We use the framework in (Dinh et al., 2016) to design each block $\mathcal{IF}_k$,

$$\begin{aligned} \mathbf{v}_{k+1}^1 &\leftarrow \mathbf{v}_k^1 \odot S(\mathcal{L}(\mathbf{v}_k^2)), \\ \mathbf{v}_{k+1}^2 &\leftarrow \mathbf{v}_k^2 \odot S(\mathcal{L}(\mathbf{v}_{k+1}^1)), \end{aligned} \tag{2}$$

where $\odot$ is the element-wise multiplication, $\mathcal{L}$ is the Fourier layer of the standard FNO that fulfills the linear transform and nonlinear activation as expressed in (1), and S is the element-wise softplus transform[1], $S(x) = \tau^{-1} \log(1 + \exp(\tau x))$, where τ is a hyperparameter to adjust the shape. In this way, $\{\mathbf{v}_k^1, \mathbf{v}_k^2\}$ and $\{\mathbf{v}_{k+1}^1, \mathbf{v}_{k+1}^2\}$ form a bijection pair. Given the outputs $\{\mathbf{v}_{k+1}^1, \mathbf{v}_{k+1}^2\}$, the inputs can be inversely computed via

$$\begin{aligned} \mathbf{v}_k^2 &\leftarrow \mathbf{v}_{k+1}^2 \odot \left[S(\mathcal{L}(\mathbf{v}_{k+1}^1))\right]^{-1}, \\ \mathbf{v}_k^1 &\leftarrow \mathbf{v}_{k+1}^1 \odot \left[S(\mathcal{L}(\mathbf{v}_k^2))\right]^{-1}, \end{aligned} \tag{3}$$

where $[\cdot]^{-1}$ denotes the element-wise reciprocal.

The outputs of the last invertible Fourier block, $\mathbf{v}_K^1$ and $\mathbf{v}_K^2$, are concatenated and fed into a second MLP $\mathcal{Q}$, which projects back to generate the prediction of the output function at the sampling locations,

$$\psi_{\text{iFNO}}(\mathbf{f}) = \mathcal{Q}(\mathbf{v}_K^1, \mathbf{v}_K^2). \tag{4}$$

[1]We did not use the exponential transform as suggested in (Dinh et al., 2016). We empirically found that the softplus transform is numerically more stable and consistently achieved better performance.

Next, to fulfill the inverse prediction from the (discretized) output function $\mathbf{u}$ back to the input function, we introduce another pairs of MLPs, $\mathcal{P}'$ and $\mathcal{Q}'$, for which, $\mathcal{P}'$ first maps $\mathbf{u}$ back to the latent space to predict the outputs of the last invertible Fourier block,

$$\mathbf{v}_K^1, \mathbf{v}_K^2 = \mathcal{P}'(\mathbf{u}),$$

then inversion (3) is sequentially executed to predict the inputs to each block until the first one, and finally $\mathbf{Q}'$ projects the inputs to the first invertible Fourier block back to generate the prediction of the original input function,

$$\psi_{\text{iFNO}}^{-1}(\mathbf{u}) = \mathbf{Q}'(\mathbf{v}_1^1, \mathbf{v}_1^2). \tag{5}$$

One might question why not position the invertible Fourier blocks directly in the original space, eliminating channel lifting and projection. Adopting this approach will limit the representation power and miss the rich information in the higher-dimensional latent space. Additionally, deciding whether to split the original input function samples into halves or duplicate them into two copies becomes a challenge. The former can potentially disrupt the internal structures of the input function, while the latter introduces additional training issues, such as how to enforce the prediction of the two inputs to be identical. We found that empirically in both our model and standard FNO, channel lifting is crucial to achieve promising performance.

3.2 Embedding Intrinsic Structures

To extract the intrinsic structures within the input space so as to further mitigate challenges for solving inverse problems, we integrate a β-variational auto-encoder (β-VAE) (Higgins et al., 2016) into our model. Specifically, after projecting $\{\mathbf{v}_1^1, \mathbf{v}_1^2\}$ back to the input space — see (5) — we feed the results to the an encoder network to obtain a stochastic latent embedding $\mathbf{z}$. Then through a decoder network, we produce the prediction of the input function values. We henceforth modify (5) as the follows,

$$\mathbf{z} = \text{Encoder}\left(\mathcal{Q}'(\mathbf{v}_1^1, \mathbf{v}_2^1), \boldsymbol{\epsilon}\right), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$
$$\mathbf{f} = \psi_{\text{iFNO}}^{-1}(\mathbf{u}) = \text{Decoder}(\mathbf{z}). \tag{6}$$

We leverage the auto-encoding variational Bayes framework (Kingma and Welling, 2013) to estimate the posterior distribution of $\mathbf{z}$. This allows us to generate the posterior samples of the prediction for $\mathbf{f}$, enabling the evaluation of the uncertainty. Our overall model is illustrated in Fig. 1.

3.3 Three-Step Training

For effective learning, we perform three steps. We first train the invertible Fourier blocks to fully capture the supervised information (*i.e.,* using (5); without VAE). The loss is:

$$J_{\text{IFB}} = J_{\text{FWD}} + J_{\text{INV}} + J_{\mathcal{P},\mathcal{Q}'} + J_{\mathcal{P}',\mathcal{Q}}, \tag{7}$$

where

$$J_{\text{FWD}} = \frac{1}{N}\sum\nolimits_{n=1}^{N} \frac{\|\mathbf{u}_n - \psi_{\text{iFNO}}(\mathbf{f}_n)\|}{\|\mathbf{u}_n\|},$$
$$J_{\text{INV}} = \frac{1}{N}\sum\nolimits_{n=1}^{N} \frac{\|\mathbf{f}_n - \psi_{\text{iFNO}}^{-1}(\mathbf{u}_n)\|}{\|\mathbf{f}_n\|} \tag{8}$$

measure the data fitness for forward and inverse predictions, $\|\cdot\|$ is the Frobenius norm, and

$$J_{\mathcal{P},\mathcal{Q}'} = \frac{1}{N}\sum_{n=1}^{N} \frac{\|\mathbf{f}_n - \mathcal{Q}'(\mathcal{P}(\mathbf{f}_n))\|}{\|\mathbf{f}_n\|},$$
$$J_{\mathcal{P}',\mathcal{Q}} = \frac{1}{N}\sum_{n=1}^{N} \frac{\|\mathbf{u}_n - \mathcal{Q}(\mathcal{P}'(\mathbf{u}_n))\|}{\|\mathbf{u}_n\|} \tag{9}$$

are two additional reconstruction loss terms that encourage the invariance of the original information after channel lifting and projection at both the input and output ends.

Next, we train the β-VAE component to fully capture the hidden structures wthin the input function values. We minimize a variational free energy,

$$J_{\beta-\text{VAE}} = \frac{1}{N}\sum_{n=1}^{N} J_n, \tag{10}$$
$$J_n = \beta \text{KL}(q(\mathbf{z}_n)\|p(\mathbf{z}_n)) + \mathbb{E}_q\left[\frac{\|\mathbf{f}_n - \text{Decoder}(\mathbf{z}_n)\|}{\|\mathbf{f}_n\|}\right],$$

where KL is the Kullback-Leibler divergence, β is a hyperparameter, $\mathbf{z}_n$ is the stochastic embedding representation of $\mathbf{f}_n$, and $q(\mathbf{z}_n)$ is the posterior distribution. The samples from $q(\mathbf{z}_n)$ is generated by: $\mathbf{z}_n = \text{Encoder}(\mathbf{f}_n, \boldsymbol{\epsilon}),\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Finally, we combine the two trained models, start with their current parameters, and train the entire model by minimizing a joint loss,

$$J = J_{\text{FWD}} + J_{P,Q'} + J_{P',Q} + J_{\beta-\text{VAE}}, \tag{11}$$

where according to (6), the inverse prediction is now $\psi_{\text{iFNO}}^{-1}(\mathbf{u}_n) = \text{Decoder}(\mathbf{z}_n)$, and the input to the encoder network for each $\mathbf{z}_n$ is generated by lifting $\mathbf{u}_n$ with $\mathcal{P}'$, going through invertible Fourier blocks, $\mathcal{IF}_K, \ldots, \mathcal{IF}_1$, and then applying projection MLP $\mathcal{Q}'$. See Figure 1. One might consider directly minimizing the joint loss (11) instead, but this could introduce complications and reduce the efficiency in extracting both supervised and structural knowledge from data. Empirically, we found that our tree-step training process results in more stable predictive performance.

4 RELATED WORK

Operator learning is a rapidly advancing research field, with various methods falling under the category of neural operators, primarily based on neural networks. Alongside FNO, other notable approaches have been proposed, such as low-rank neural operator (LNO) (Li et al., 2020), multiwavelet-based NO (Gupta et al., 2021), and convolutional NOs (CNO) (Raonic et al., 2023). Recently, Li et al. (2024) proposed active learning methods for multi-resolution FNO. Deep Operator Net (DON) (Lu et al., 2021) is another popular approach, which consists of a branch net and a trunk net. The branch net is applied over the input function values while the trunk net over the sampling locations. The prediction is generated by the dot product between the outputs of the two nets. To improve the stability and efficiency, Lu et al. (2022) replaced the trunk net by the POD (PCA) bases. Another line of research uses transformers to build NOs (Cao, 2021; Li et al., 2022; Hao et al., 2023). Recent works have also explored kernel operator learning approaches (Long et al., 2022; Batlle et al., 2023).

Molinaro et al. (2023) proposed neural inverse operator (NIO), explicitly designed for address inverse problems. NIO sequentially combines DeepONet and FNO, where the DeepONet takes observations and querying locations as the input, and the outputs are subsequently passed to FNO to obtain the prediction for the inverse problems. NIO does not offer uncertainty quantification. Kaltenbach et al. (2023) designed an invertible version of DeepONet. The key idea is to modify the branch net to be invertible following the framework of (Dinh et al., 2016). Given the observations, the approach starts by solving a least-squares problem to restore the output of the branch net, and then proceeds to back-predict the input function values. The least-squares problem is further casted into a Bayesian inference task and a Gaussian mixture prior is assigned for the unknown output of the branch net. One potential constraint of this approach is that the input and output dimensions of the branch net must be identical for invertibility. When the dimensionality or resolution is high, it can substantially raise the model size and the computational costs, posing learning challenges (see Sec 5).

The classical methods for solving inverse problem is based on the assumption that the forward system is known, *e.g.,* a particular PDE system (Stuart, 2010). Then one optimizes the input to the forward system to match the system output and measurement data. Markov-Chain Monte-Carlo (MCMC) sampling is often used to import prior knowledge and to quantify uncertainty. However, the classical methods require extensively simulating the forward system, such as running a numerical solver, which is very expensive in computation. The recent work (Zhao et al., 2022) inherits the classical framework, but use a data-driven surrogate model, such as MeshGraphNet (Pfaff et al., 2020) and U-Net (Ronneberger et al., 2015) to replace the forward simulator, so as to accelerate the optimization procedure. Though effective, these approaches need massive simulation examples to train an enough accurate surrogate. Moreover, one has to conduct numerical optimization from scratch for every prediction, which is still quite expensive. See run time comparison in Appendix Table 3.

5 NUMERICAL EXPERIMENTS

We evaluated our method on seven benchmark problems, each covering both the forward and inverse scenarios. These problems are grounded in Darcy flow, wave propagation, diffusion-reaction, and Navier-Stokes (NS) equations, which are commonly used in the literature of operator learning and inverse problems (Li et al., 2020; Lu et al., 2022; Takamoto et al., 2022; Iglesias et al., 2016; Chada et al., 2018; Zhang et al., 2018). We summarize the benchmark problems as follows.

- **D-LINE**: We considered a single-phase 2D Darcy Flow equation. We are interested in predicting from the permeability field to the fluid pressure field (forward) and from pressure field recovering the permeability (inverse). The permeability field is piece-wise constant with a linear interface.
- **D-CURV**: Similar to D-LINE, but the permeability is piece-wise constant with a curved interface.
- **W-OVAL**: We considered a seismic survey based on an acoustic seismic wave equation (Zhang et al., 2018). Given an external wave source, the forward task is to predict from the square slowness of the physical media to the wave measurements at the signal receivers. The inverse task is to recover the square slowness from the measurements. The square slowness is piece-wise constant with an oval-shaped interface.
- **W-Z**: Similar to W-OVAL, except that the square slowness takes a Z-shaped interface.
- **NS**: We considered a viscous, incompressible flow governed by the 2D Navier-Stokes (NS) equation. The forward task is to predict from the initial conditions to the solution at $t = 10$, and the inverse task is to recover the initial condition from the solution at $t = 10$.
- **DR**: We employed the PDEBench dataset (Takamoto et al., 2022) for a 2D diffusion-reaction system. The forward task is to predict the activator state at time step 5 based on time step 1, while the inverse problem aims to recover the state at step 1 from step 5.
- **CFD**: We used the PDEBench dataset for computational fluid dynamics (CFD) governed by a 2D compressible Navier-Stokes (NS) equation. The forward problem involves predicting the fluid state at time step 2 from the initial state, while the inverse task aims to recover the initial state from time step 2.

We employed 800 training examples for D-LINE, D-CURV, DR and CFD, 400 for W-OVAL and W-Z, and 1,000 for NS. Each task was evaluated using 200 test examples. To access the robustness of our method against data noise and inaccuracy, we conducted additional tests on the first five benchmarks, including D-LINE, D-CURV, W-OVAL, W-Z and NS. We injected 10% and 20% white noises into the training dataset. Specifically, for each pair of $\mathbf{f}_n$ and $\mathbf{u}_n$ generated from the numerical solvers, we corrupted them via updating $\mathbf{f}_n \leftarrow \mathbf{f}_n + \eta\boldsymbol{\sigma}_f \odot \boldsymbol{\epsilon}_n$ and $\mathbf{u}_n \leftarrow \mathbf{u}_n + \eta\boldsymbol{\sigma}_u \odot \boldsymbol{\xi}_n$, where $\eta \in \{0.1, 0.2\}$ represents the noise level, $\boldsymbol{\sigma}_f$ and $\boldsymbol{\sigma}_\mathbf{u}$ are the per-element standard deviation of the sampled input and output functions, and $\boldsymbol{\epsilon}_n, \boldsymbol{\xi}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ are Gaussian white noises. The details of data generation for each benchmark are provided in Appendix Section A.1.

Methods. We compared iFNO with the invertible Deep Operator Net (iDON) (Kaltenbach et al., 2022), neural inverse operator (NIO) (Molinaro et al., 2023), the standard FNO, and a recent state-of-the-art attention-based neural operator model, GNOT (Hao et al., 2023). We evaluated all the methods in both forward and inverse tasks, except for NIO, which is exclusively designed for inverse prediction. In addition, we adapted FNO to the classical framework (Stuart, 2010) for solving inverse problems. Specifically, we trained FNO for forward prediction. Then we optimized the input to FNO to match the forward prediction and observations. To leverage the intrinsic structures within the input space, we also used β-VAE to extract a low-dimensional representation for the input functions values at the sampling locations. We optimize the embedding with ADAM. We denote this method as Input Optimization over Forward Model (IOFM). Note that we found directly optimizing the function values consistently results in large prediction errors, indicating failure. We implemented iFNO with PyTorch and used the original implementation of all the competing methods.

Experimental Settings. For each approach, we used 100 random examples as the validation set (non-overlapping with the test set) to identify the optimal hyperparameters for each task. The same validation set was used across all the methods for consistency. We employed grid search to select the best hyperparameters, with the specific ranges provided in the Appendix Section A.2. Notably, FNO and

Table 1: Relative L_2 Error on First Five Benchmarks, where "D-" and "W-" indicate Darcy flow and wave propagation, respectively, "LINE", "CURV", "OVAL", and "Z" represent linear, curved, oval, and z-shaped interfaces, respectively as explained in detail in Appendix Section A.1.1 and A.1.2; and 0%, 10%, 20% indicate the noise level in the training data. N/A indicates absence of reasonable results due to numerical instabilities.

(a) Forward Prediction

Method	IFNO	IDON	FNO	GNOT
0% noise				
D-LINE	**3.60e-2** ± 5.1e-4	4.71e-1 ± 3.7e-2	7.20e-2 ± 3.5e-3	3.90e-2 ± 1.6e-3
D-CURV	3.25e-2 ± 9.3e-4	7.68e+0 ± 3.0e-1	3.88e-2 ± 3.1e-4	**2.98e-2** ± 2.0e-4
W-OVAL	**4.41e-2** ± 3.2e-3	3.28e-1 ± 3.2e-3	5.52e-2 ± 1.3e-3	4.50e-2 ± 4.0e-3
W-Z	**3.12e-1** ± 6.9e-3	9.88e-1 ± 9.8e-4	4.15e-1 ± 3.3e-3	N/A
NS	1.94e-2 ± 5.0e-4	2.70e-1 ± 1.8e-3	**1.54e-2** ± 8.9e-5	2.54e-2 ± 5.0e-4
10% noise				
D-LINE	**5.38e-2** ±7.0e-4	5.12e-1 ±4.2e-2	8.40e-2 ±6.2e-4	6.90e-2 ±1.2e-3
D-CURV	**4.96e-2** ±1.5e-3	7.58e+0 ±5.4e-1	5.73e-2 ±1.5e-3	1.50e-1 ±4.1e-2
W-OVAL	6.15e-2 ±7.7e-3	3.38e-1 ±5.3e-3	7.88e-2 ±8.9e-4	**4.70e-2** ±4.0e-3
W-Z	**3.13e-1** ±3.1e-2	9.90e-1 ±2.4e-3	4.25e-1 ±1.9e-3	N/A
NS	**2.49e-2** ±4.0e-4	2.75e-1 ±3.9e-3	2.62e-2 ±2.6e-4	2.75e-2 ±4.0e-4
20% noise				
D-LINE	**7.29e-2** ±2.1e-3	6.74e-1 ±5.1e-2	9.96e-2 ±1.5e-3	1.06e-1 ±3.0e-3
D-CURV	**6.38e-2** ±2.8e-3	8.85e+0 ±1.4e+0	7.42e-2 ±5.3e-4	2.07e-1 ±6.8e-2
W-OVAL	7.57e-2 ±4.5e-3	3.69e-1 ±1.2e-2	1.20e-1 ±4.3e-3	**5.42e-2** ±8.0e-3
W-Z	**3.14e-1** ±7.2e-3	9.92e-1 ±3.0e-3	4.42e-1 ±1.8e-3	N/A
NS	**3.48e-2** ±9.7e-4	2.95e-1 ±1.3e-2	3.70e-2 ±3.1e-4	2.07e-1 ±6.0e-4

(b) Inverse Prediction

Method	IFNO	IDON	NIO	FNO	GNOT
0%					
D-LINE	**5.66e-2** ±1.0e-3	2.98e-1 ±6.9e-2	2.42e-1 ±5.7e-3	1.90e-1 ±2.6e-2	2.48e-1 ±1.5e-1
D-CURV	**5.54e-2** ±9.4e-4	2.20e-1 ±5.3e-3	7.83e-2 ±1.4e-3	6.84e-2 ±5.3e-4	1.19e-1 ±9.0e-3
W-OVAL	**5.74e-2** ±1.3e-3	1.42e-1 ±3.4e-3	8.51e-2 ±6.2e-4	7.64e-2 ±7.1e-4	7.50e-2 ±1.0e-2
W-Z	**2.01e-1** ±3.3e-3	2.61e-1 ±2.2e-3	2.13e-1 ±9.8e-4	2.28e-1 ±7.6e-4	4.08e-1 ±6.0e-4
NS	**5.09e-2** ±6.4e-4	1.84e-1 ±9.9e-3	7.07e-2 ±2.6e-4	5.71e-2 ±8.9e-5	1.10e-1 ±2.7e-3
10%					
D-LINE	**8.99e-2** ±1.4e-3	3.50e-1 ±4.5e-2	1.72e-1 ±1.1e-3	1.63e-1 ±3.8e-3	3.40e-1 ±1.2e-1
D-CURV	**7.94e-2** ±9.7e-4	2.25e-1 ±9.3e-3	2.14e-1 ±3.8e-3	8.80e-2 ±4.9e-4	2.09e-1 ±9.0e-3
W-OVAL	**7.19e-2** ±1.6e-3	1.47e-1 ±9.3e-4	1.08e-1 ±6.2e-4	8.49e-2 ±5.8e-4	7.90e-2 ±1.1e-2
W-Z	**1.91e-1** ±3.1e-3	2.90e-1 ±5.5e-4	2.22e-1 ±4.4e-4	2.30e-1 ±4.4e-4	4.08e-1 ±1.0e-3
NS	**6.66e-2** ±2.6e-4	1.84e-1 ±1.8e-2	1.04e-1 ±6.2e-4	8.96e-2 ±5.8e-4	1.09e-1 ±8.0e-3
20%					
D-LINE	**1.13e-1** ±9.0e-4	4.22e-1 ±7.4e-3	1.89e-1 ±2.3e-3	1.73e-1 ±5.5e-3	4.53e-1 ±3.7e-2
D-CURV	**9.58e-2** ±1.2e-3	2.51e-1 ±1.2e-2	2.39e-1 ±3.9e-3	1.07e-1 ±5.3e-4	2.23e-1 ±3.0e-2
W-OVAL	8.85e-2 ±2.6e-3	1.51e-1 ±4.0e-4	1.19e-1 ±6.7e-4	9.71e-2 ±6.7e-4	**7.92e-2** ±1.1e-2
W-Z	**2.00e-1** ±3.1e-3	2.92e-1 ±9.8e-3	2.27e-1 ±8.9e-4	2.31e-1 ±1.5e-3	4.09e-1 ±9.0e-4
NS	**7.52e-2** ±4.7e-4	1.84e-1 ±2.6e-2	1.20e-1 ±1.7e-4	1.16e-1 ±1.6e-3	1.20e-1 ±9.3e-3

Table 2: Relative L_2 Error on Benchmarks DR and CFD.

(a) Forward Prediction

Method	IFNO	IDON	FNO	GNOT
DR	**9.45e-2** ± 4.97e-4	9.08e-1 ± 5.83e-5	9.91e-2 ± 1.34e-4	6.05e-1 ± 6.70e-2
CFD	**1.43e-1** ± 2.44e-5	1.55e-1 ± 5.03e-4	1.46e-1 ± 6.26e-4	1.44e-1 ± 7.47e-5

(b) Inverse Prediction

Method	IFNO	IDON	NIO	FNO	GNOT
DR	**2.58e-1** ±9.65e-4	9.81e-1 ±8.72e-5	7.42e-1 ±1.22e-2	3.26e-1 ±8.05e-4	8.83e-1 ±2.22e-2
CFD	**8.97e-2** ±9.65e-4	1.17e-1 ±1.75e-2	1.09e-1 ±1.44e-3	9.90e-2 ±2.24e-3	9.08e-2 ±5.55e-5

GNO performed separate validation processes for forward and inverse predictions, obtaining different sets of hyperparameters optimized exclusively for each task. In contrast, iFNO and iDON performed validation only once, where the validation error is the sum of the forward and inverse prediction errors, and then they used the same set of hyperparameters to train a single model for both predictions. Following (Lu et al., 2022), for each method, we selected the optimal hyperparameters, and then conducted stochastic training for five times, reporting the average relative L_2 test error along with the standard deviation.

5.1 Predictive Performance

The relative L_2 error of each method is reported in Table 1 and 2. Due to the space limit, the error of IOFM is given in Appendix Table 5. As shown, in the vast majority of the cases, iFNO achieves the best performance, often outperforming the competing approaches by a large margin. In a few cases, iFNO is slightly second to GNOT (*e.g.,* D-CURV with 0% noise, and W-OVAL with 10% noise for forward prediction). This might be mainly caused by FNO itself, which gives 23%-120% larger error than GNOT. However, in the forward prediction tasks for W-Z, GNOT consistently encountered numerical issues, and was unable to deliver reasonable prediction errors. Hence we marked its results as N/A. iFNO greatly surpasses FNO in all the cases, except that in NS forward prediction with 0% training noise, iFNO is slightly worse than FNO. This highlights that our co-learning approach, utilizing a shared FNO architecture, can substantially enhance the performance in both forward and inverse tasks.

While iDON can also perform joint forward and inverse predictions, it employs an invertible architecture only in the branch network. For inverse prediction, iDON needs to back-solve the latent output of the branch network before predicting the input function. This additional step might introduce learning challenges and can substantially increase computational costs. We report the average running time of each method in Table 3.

Table 3: Average Running Time of Each Method, including both training and prediction, measured at a Linux workstation with NVIDIA GeForce RTX 3090. m: minutes; h: hours.

Method	D-LINE	D-CURV	W-OVAL	W-Z	NS
iFNO	17m	68m	13m	22m	50m
IDON	5.7h	15.3h	7.8h	7.9h	36.8h
NIO	5m	10m	5m	4m	13m
FNO	11m	31m	20m	15m	40m
GNOT	23m	36m	33m	29m	56m
IOFM	99m	119m	40m	35m	77m

5.2 Pointwise Error and Prediction Uncertainty

For a detailed assessment, we performed a fine-grained evaluation by visualizing the pointwise prediction error of iFNO in ten randomly selected instances within the inverse scenario. Additionally, we investigated prediction uncertainty by sampling 500 predictions from the VAE component (refer to (6)) for each instance. The standard deviation of these predictions at each location is then calculated. The pointwise error is determined based on the predictive mean of the encoder. The outcomes are depicted in Figure 2.

Overall, it is evident that the error grows as the noise level increases, mirroring a similar trend in prediction uncertainty. For problems involving Darcy flow and wave propagation (the first eight instances), the pointwise error is predominantly concentrated at the interfaces between different regions. This pattern is echoed in the prediction uncertainty of iFNO, with the standard deviation being significantly larger at the interfaces compared to other locations. These findings are not only intriguing but also intuitively reasonable. Specifically, since the ground-truth permeability and square slowness (that we aim to recover) are identical within the same region but distinct across different regions, predicting values within each region becomes relatively easier, resulting in lower uncertainty (higher confidence). Conversely, at the interfaces, where the ground-truth undergoes abrupt changes, predicting values becomes more challenging, leading to an increase in prediction uncertainty to reflect these challenges (lower confidence). In the initial condition recovery problem (the last two instances), the pointwise pre-

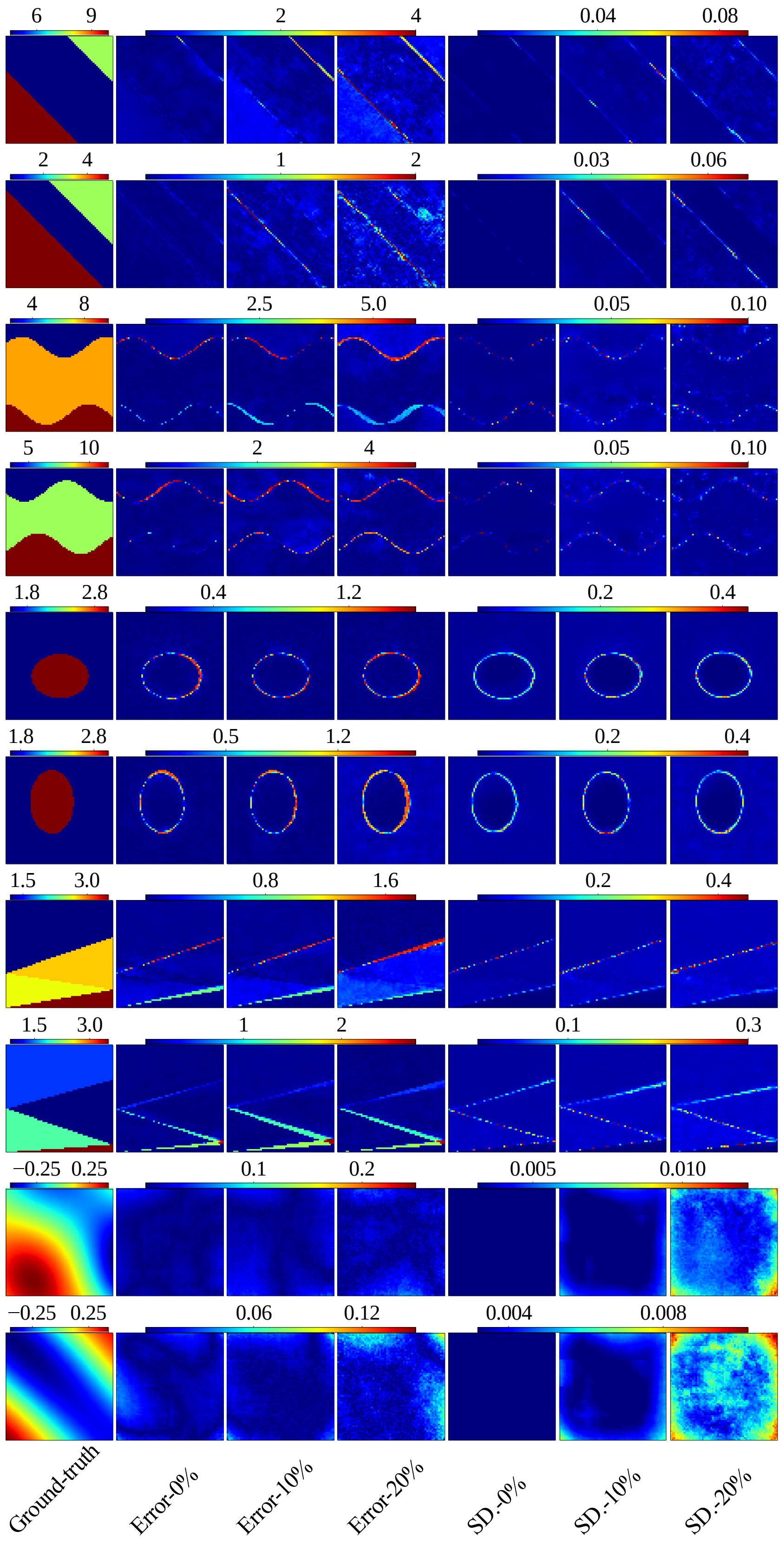


Figure 2: iFNO Pointwise Inverse Prediction Error and Predictive Standard Deviation, denoted by "Error" and "SD" respectively followed with noise levels in training data.

diction error is primarily concentrated near the boundary of the domain. Correspondingly, the predictive standard deviation of iFNO is larger near the boundary. As the noise level increases, the error at the boundary also increases, along with the calibration of uncertainty.

The comparison of pointwise errors with the competing methods is presented in Figure 4 and 5 of the Appendix, encompassing both forward and inverse predictions. The results reveal that competing methods frequently manifest significantly larger errors in various local regions, as illustrated by instances such as iDON in the second and third instance, NIO in the sixth and seventh instance, and FNO in the seventh and eighth instance in Appendix Fig. 4, and iDON and FNO in nearly all the instances of Appendix Fig. 5. We did not include the pointwise error of IOFM due to its much inferior performance as compared to all the other methods.

These results affirm that iFNO not only achieves superior global accuracy but also excels in local ground-truth recovery. Furthermore, iFNO demonstrates the ability to provide reasonable uncertainty estimates, aligning with the predictive challenges encountered across various local regions.

Ablation Study. Finally, we investigated how the performance of iFNO varies with the model size. Basically, we found that using three to four invertible Fourier blocks consistently yields optimal performance for iFNO. The detailed results and discussion are provided in Section B of the Appendix.

6 CONCLUSION

We have introduced iFNO, an invertible Fourier Neural Operator designed to jointly address forward and inverse problems. By co-learning the bi-directional tasks within a unified architecture, iFNO enhances prediction accuracy for both tasks. Additionally, iFNO demonstrates the capability to provide uncertainty calibration for inverse prediction. Through seven benchmark numerical experiments, iFNO showcases promising predictive performance. Our architecture can be easily extended to integrate with other neural operator architectures, such as attention layers. In the future, we plan to explore such extensions, and design a hybrid of exiting or innovative units in the invertible blocks, not necessarily restricted to the Fourier layers. We will continue investigating our method in a broader range of forward prediction and inverse inference tasks.

Acknowledgements

This work has been supported by MURI AFOSR grant FA9550-20-1-0358, NSF CAREER Award IIS-2046295, and NSF CSSI-2311685.

References

Batlle, P., Darcy, M., Hosseini, B., and Owhadi, H. (2023). Kernel methods are competitive for operator learning. arXiv preprint arXiv:2304.13202.

Cao, S. (2021). Choose a transformer: Fourier or galerkin. Advances in neural information processing systems, 34:24924–24940.

Chada, N. K., Iglesias, M. A., Roininen, L., and Stuart, A. M. (2018). Parameterizations for ensemble kalman inversion. Inverse Problems, 34(5):055009.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. arXiv preprint arXiv:1605.08803.

Engl, H. W. and Groetsch, C. W. (2014). Inverse and ill-posed problems, volume 4. Elsevier.

Gupta, G., Xiao, X., and Bogdan, P. (2021). Multiwavelet-based operator learning for differential equations. Advances in neural information processing systems, 34:24048–24062.

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

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2016). beta-vae: Learning basic visual concepts with a constrained variational framework. In International conference on learning representations.

Iglesias, M. A., Lu, Y., and Stuart, A. M. (2016). A Bayesian level set method for geometric inverse problems. Interfaces and free boundaries, 18(2):181–217.

Kaltenbach, S., Perdikaris, P., and Koutsourelakis, P.-S. (2022). Semi-supervised invertible deeponets for bayesian inverse problems. arXiv preprint arXiv:2209.02772.

Kaltenbach, S., Perdikaris, P., and Koutsourelakis, P.-S. (2023). Semi-supervised invertible neural operators for bayesian inverse problems. Computational Mechanics, pages 1–20.

Kashefi, A. and Mukerji, T. (2024). A novel fourier neural operator framework for classification of multi-sized images: Application to 3d digital porous media. arXiv preprint arXiv:2402.11568.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114.

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

Li, S., Yu, X., Xing, W., Kirby, R., Narayan, A., and Zhe, S. (2024). Multi-resolution active learning of fourier neural operators. In International Conference on Artificial Intelligence and Statistics, pages 2440–2448. PMLR.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., Anandkumar, A., et al. (2020). Fourier neural operator for parametric partial differential equations. In International Conference on Learning Representations.

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

Long, D., Mrvaljevic, N., Zhe, S., and Hosseini, B. (2022). A kernel approach for pde discovery and operator learning. arXiv preprint arXiv:2210.08140.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. (2021). Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature machine intelligence, 3(3):218–229.

Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. (2022). A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. Computer Methods in Applied Mechanics and Engineering, 393:114778.

Molinaro, R., Yang, Y., Engquist, B., and Mishra, S. (2023). Neural inverse operators for solving pde inverse problems. arXiv preprint arXiv:2301.11167.

Nashed, M. Z. and Scherzer, O. (2002). Inverse Problems, Image Analysis, and Medical Imaging: AMS Special Session on Interaction of Inverse Problems and Image Analysis, January 10-13, 2001, New Orleans, Louisiana, volume 313. American Mathematical Soc.

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., et al. (2022). Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. arXiv preprint arXiv:2202.11214.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. (2020). Learning mesh-based simulation with graph networks. arXiv preprint arXiv:2010.03409.

Raonic, B., Molinaro, R., De Ryck, T., Rohner, T., Bartolucci, F., Alaifari, R., Mishra, S., and de Bézenac, E. (2023). Convolutional neural operators for robust and accurate learning of pdes. Advances in Neural Information Processing Systems, 36.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18, pages 234–241. Springer.

Stuart, A. M. (2010). Inverse problems: a Bayesian perspective. Acta numerica, 19:451–559.

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. (2022). Pdebench: An extensive benchmark for scientific machine learning. Advances in Neural Information Processing Systems, 35:1596–1611.

Tanaka, M. and Dulikravich, G. S. (1998). Inverse problems in engineering mechanics. Elsevier.

Tikhonov, A. N. (1963). On the solution of ill-posed problems and the method of regularization. In Doklady akademii nauk, volume 151, pages 501–504. Russian Academy of Sciences.

Yilmaz, Ö. (2001). Seismic data analysis: Processing, inversion, and interpretation of seismic data. Society of exploration geophysicists.

Zhang, W., Joardar, A. K., et al. (2018). Acoustic based crosshole full waveform slowness inversion in the time domain. Journal of Applied Mathematics and Physics, 6(05):1086.

Zhao, Q., Lindell, D. B., and Wetzstein, G. (2022). Learning to solve pde-constrained inverse problems with graph networks. arXiv preprint arXiv:2206.00711.

Checklist

The checklist follows the references. For each question, choose your answer from the three possible options: Yes, No, Not Applicable. You are encouraged to include a justification to your answer, either by referencing the appropriate section of your paper or providing a brief inline description (1-2 sentences). Please do not modify the questions. Note that the Checklist section does not count towards the page limit. Not including the checklist in the first submission won't result in desk rejection, although in such case we will ask you to upload it during the author response period and include it in camera ready (if accepted).

In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Not Applicable]
2. For any theoretical claim, check if you include:

   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

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

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

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

Appendix

A Experimental Details

A.1 Data Preparation

A.1.1 Darcy Flow

A single-phase 2D Darcy Flow equation was employed,

$$\begin{aligned} -\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) &= g(\mathbf{x}) \quad \mathbf{x} \in (0,1)^2 \\ u(\mathbf{x}) &= 0, \quad \mathbf{x} \in \partial(0,1)^2, \end{aligned} \tag{12}$$

where $a(\mathbf{x})$ is the permeability field, $u(\mathbf{x})$ is fluid pressure, and $g(\mathbf{x})$ is an external source. We considered a practically useful case where the permeability is piece-wise constant across separate regions $\{\mathcal{R}_i\}_{i=1}^C$ where $\mathcal{R}_1 \cup \ldots \cup \mathcal{R}_C = (0,1)^2$, and

$$a(\mathbf{x}) = \sum_{i=1}^{C} q_i \mathbf{1}_{\mathcal{R}_i}(\mathbf{x}).$$

For the forward scenario, we are interested in predicting the pressure field $u(\mathbf{x})$ based on the given permeability field $a(\mathbf{x})$. Conversely, in the inverse scenario, the goal is to recover $a(\cdot)$ from the measurement of the pressure field $u(\cdot)$. We considered two benchmark problems, each featuring the permeability with a different type of geometric structures. For both problems, we fixed $g(x)$ to 1.

Linear permeability interface (D-LINE). In the first problem, the domain was divided into three regions (namely $C = 3$), and the interfaces were represented by parallel lines inclined at a 45-degree angle from the horizontal axis; see Figure 6 for an example. The permeability value in each region was sampled from a uniform distribution $U(0, 10)$. To determine the position of these sections, we sampled the endpoints of interface lines, denoted by $\{(w_1, 0), (0, w_1)\}$ for the first interface, and $\{(w_2, 0), (0, w_2)\}$ for the second interface, respectively, where $w_1 \sim \mathcal{U}(0, \frac{1}{3})$ and $w_2 \sim \mathcal{U}(\frac{1}{3}, \frac{2}{3})$.

Curved permeability interface (D-CURV). In the second problem, the domain was also partitioned into three regions, but the interfaces were curves; see Figure 7 for an illustration. We sampled the first interface as $x_2 = p_1 + 0.1\sin(2.5\pi(x_1 + r_1))$ and the second $x_2 = p_2 + 0.1\sin(2.5\pi(x_1 + r_2))$, where $p_1 \sim \mathcal{U}(0.15, 0.4), p_2 \sim \mathcal{U}(0.6, 0.85)$, and $r_1, r_2 \sim \mathcal{U}(0,1)$. The permeability value in each region was sampled from $U(0, 15)$.

To prepare the dataset, we followed (Li et al., 2020) to apply a second-order finite difference solver and collected pairs of permeability and pressure field on a 64×64 grid. To assess the robustness of our method against data noise and inaccuracy, we conducted tests by injecting 10% and 20% white noises into the training datasets, as described in the main paper. We provide the signal-to-noise ratios in Table 4.

A.1.2 Wave Propagation

We employed an acoustic seismic wave equation to simulate seismic surveys,

$$\begin{aligned} m(\mathbf{x})\frac{\mathrm{d}^2 u(\mathbf{x},t)}{\mathrm{d}t^2} - \nabla^2 u(\mathbf{x},t) + \eta \frac{\mathrm{d}u(x,t)}{\mathrm{d}t} &= q(t),\ \mathbf{x} \in \Omega, \\ u(\mathbf{x},0) = \frac{\mathrm{d}u(x,0)}{\mathrm{d}t} &= 0, \mathbf{x} \in \Omega,\ \ u(\mathbf{x},t) = 0, \mathbf{x} \in \partial\Omega, \end{aligned}$$

where $m(\mathbf{x})$ is square slowness, defined as the inverse of squared wave speed in the given physical media, $q(t)$ expresses the external wave source, and η is the damping mask. In the simulation, the physical media was placed in the domain $\Omega = (0, 1.28\,\mathrm{km})^2$, where $\forall \mathbf{x} = (x_1, x_2) \in \Omega$, x_1 and x_2 represent the depth and width, respectively. For each survey, an external source was positioned at particular location to initiate seismic waves. A row of receivers were placed at a particular depth to record the wave measurements across time; see Figure 3 for an illustration. We conducted simulation over a duration of one second. In the forward tasks, we intended to predict the wave measurements at the receivers based on the square slowness $m(\mathbf{x})$, while for the inverse tasks, the goal was to recover $m(\mathbf{x})$ from the measurements at the receivers. We designed two benchmark problems, each corresponding to a different class of structures for $m(\mathbf{x})$. In both problems, we set $q(t) = (1 - 2\pi^2 f_0^2 (t - \frac{1}{f_0})^2) e^{-\pi^2 f_0^2 (t - \frac{1}{f_0})}$, where $f_0 = 0.01$. This implies that the peak frequency of the source wave is 10 Hz.

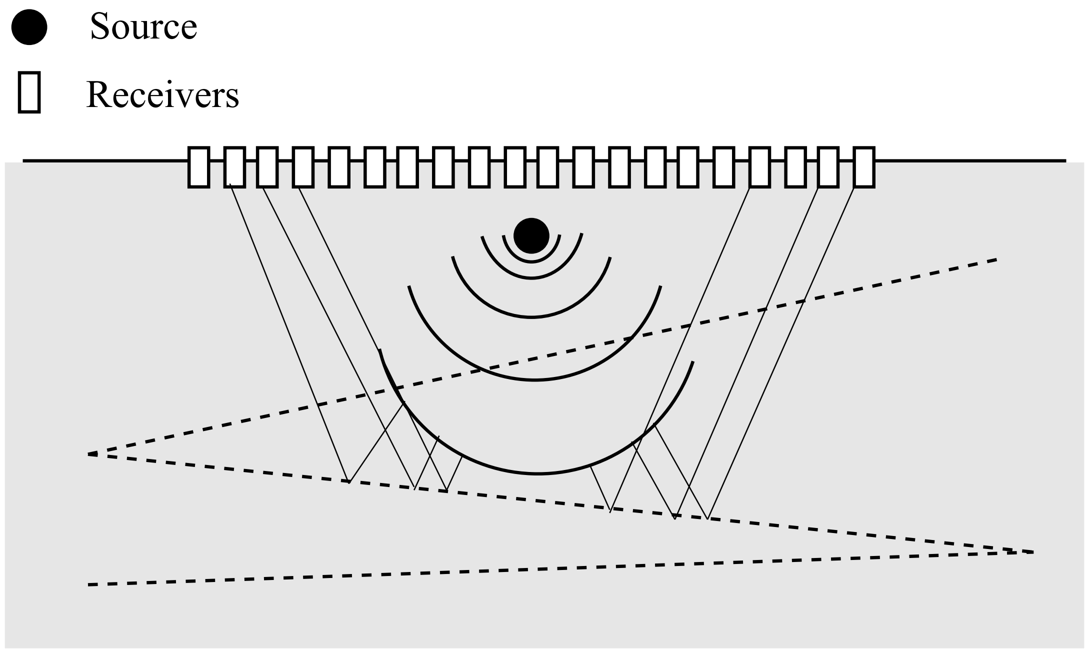


Figure 3: An Illustration of The Seismic Surveys, where the shaded region is the physical media of interest and dashed lines are the interface for different regions of square slowness.

Oval-shaped square slowness (W-OVAL). In the first problem, the source was placed at a depth of 50m and horizontally in the middle. The receivers were positioned at a depth of 20m. The domain was partitioned into two regions, with $m(\mathbf{x})$ being the same within each region. See Figure 8 for an example and the surveyed data by the receivers. The interface is an oval, defined by its center (x_{1c}, x_{2c}), and two radii w and h. We sampled $x_{1c}, x_{2c} \sim \mathcal{U}(\frac{1}{4} \times 1.28\,\mathrm{km}, \frac{3}{4} \times 1.28\,\mathrm{km})$, and $w, h \sim \mathcal{U}(\frac{1}{10} \times 1.28\,\mathrm{km}, \frac{1}{5} \times 1.28\,\mathrm{km})$. Inside the ellipse, we set the value of m to 3, while the outside the ellipse, we sampled the value from $\mathcal{U}(1, 2)$. We used the Devito library[2] to simulate the receivers' measurements. The square slowness was generated on a 128×128 grid, and the measurements were computed at 614 time steps. The data was then downsampled to a 64×64 grid and 62 time steps.

Z-shaped square slowness (W-Z). In the second problem, the source was positioned at a depth of 80m (still horizontally in the middle). The domain was divided into four regions, with the interfaces between these regions forming a z-shape. The values of $m(\mathbf{x})$ inside each region are identical; see Figure 9 for an example. We represented the end points of these interfaces by $(1.28\,\mathrm{km}, 0)$, $(z_1, 1.28\,\mathrm{km})$, $(z_2, 0)$, and $(z_3, 1.28\,\mathrm{km})$, where $z_1 \sim U(\frac{3}{4} \times 1.28\,\mathrm{km}, 1.28\,\mathrm{km})$, $z_2 \sim U(\frac{1}{2} \times 1.28\,\mathrm{km}, \frac{3}{4} \times 1.28\,\mathrm{km})$, and $z_3 \sim \mathcal{U}(\frac{1}{4} \times 1.28\,\mathrm{km}, \frac{1}{2} \times 1.28\,\mathrm{km})$. We set $m = 3.5$ in the bottom region, and sampled the value of $m(\mathbf{x})$ for each of the other three regions from $\mathcal{U}(0.5, 3.5)$. The receivers' measurements were computed at 573 time steps with spatial resolution 128, and then downsampled at 58 steps with spatial resolution 64.

A.1.3 Navier-Stoke Equation (NS)

We considered the 2D Navier-Stokes (NS) equation as used in (Li et al., 2020). The solution represents the vorticity of a viscous, incompressible fluid within the spatial domain $\mathbf{x} = (x_1, x_2) \in [0, 1]^2$. The viscosity was set to 10^{-3}. In the forward scenario, we aim to predict the vorticity at time $t = 10$ from the initial condition. Correspondingly, in the inverse scenario, the goal is to reconstruct the initial condition ω_0 from the observed vorticity at $t = 10$. We generated the initial condition by $\omega_0(x_1, x_2) = \sum_{i=1}^{2} \sum_{j=1}^{2} q_{ij} \sin(\alpha_i \pi (x_1 + c_j)) \cdot q_{ij} \cos(\alpha_i \pi (x_2 + c_j))$, where $\alpha_i \sim \mathcal{U}(0.5, 1)$, $c_j \sim \mathcal{U}(0, 1)$, and $q_{ij} \sim \mathcal{U}(-1, 1)$. An example is given by Figure 10. For each experiment, we used 1000 training examples and 200 test

[2] `https://github.com/devitocodes/devito`

examples generated at a 64×64 grid. Again, we performed additional tests by injecting 10% and 20% noises into both the training input and output function samples.

A.1.4 Diffusion Reaction (DR)

We employed a PDEBench dataset for a 2D diffusion reaction system with two non-linearly coupled variables,

$$\partial_t u = D_u \partial_{xx} u + D_u \partial_{yy} u + R_u, \tag{13}$$

$$\partial_t v = D_v \partial_{xx} v + D_v \partial_{yy} v + R_v, \tag{14}$$

where $x, y \in (-1, 1)$, $u = u(t, x, y)$ is the activator, $v = v(t, x, y)$ is the inhibitor, and the diffusion coefficients $D_u = 1 \times 10^{-3}$ and $D_v = 5 \times 10^{-3}$. The activator and inhibitor are coupled via the Fitzhugh-Nagumo equation,

$$R_u(u, v) = u - u^3 - k - v, \tag{15}$$

$$R_v(u, v) = u - v. \tag{16}$$

The initial condition is generated from a standard normal distribution. We extracted the original dataset on a 64×64 grid.

A.1.5 Computational Fluid Dynamics (CFD)

We used a CFD dataset from PDEBench. The governing equation is a 2D Compressible Navier-Stokes equation,

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{17}$$

$$\rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \eta \Delta \mathbf{v} + (\zeta + \eta/3) \nabla (\nabla \cdot \mathbf{v}), \tag{18}$$

$$\partial_t \left[\epsilon + \frac{\rho v^2}{2} \right] + \nabla \cdot \left[\left(\epsilon + p + \frac{\rho v^2}{2} \right) \mathbf{v} - \mathbf{v} \cdot \boldsymbol{\sigma}' \right] = 0, \tag{19}$$

where $\rho, \mathbf{v}, p, \epsilon$ are the mass density, velocity, gas pressure, and internal energy, respectively. We used the simulation data with Mach number $M = \frac{|v|}{c_s} = 0.1$, where c_s is the sound velocity, and with shear and bulk viscosity $\eta = \zeta = 0.1$.

A.2 Methods

All the models were trained with AdamW or the Adam optimizer with exponential decay strategy or the reducing learning rate on plateau strategy. The learning rate was chosen from $\{10^{-5}, 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 10^{-2}\}$. We varied the mini-batch size from {10, 20, 50}.

- **iFNO**. We selected the number of invertible Fourier blocks from {1, 2, 3, 4}, the channel lifting dimension from {32, 64, 128}, the number of Fourier modes (for frequency truncation) from {8, 12, 16, 32}, the number of training epochs for invertible Fourier blocks from {100, 200, 500}, the number of training epochs for β-VAE from {200, 500, 1000}, and the number of joint training epochs was set from {50, 100, 500}. The architecture of β-VAE is the same accross all the benchmarks. We employed a Gaussian encoder that includes five convolutional layers with 32, 64, 128, 256 and 512 channels respectively. The decoder first applies five transposed convolutional layers to sequntially reduce the number of channels to 512, 256, 128, 64, and 32. Then a transposed convolutional layer (with 32 output channels) and another convolutional layer (with one output channel) are applied to produce the prediction of $\mathbf{f}$. We set $\beta = 0.01$ for all the benchmarks except for Darcy flow, we set $\beta = 10^{-6}$.
- **FNO**. We used the original FNO implementation (`https://github.com/neuraloperator/neuraloperator`). To ensure convergence, we set the number of training epochs to 1000. The lifting dimension was chosen from $\{32, 64, 128\}$. The number of Fourier modes was tuned from $\{8, 12, 16, 32\}$. The number of Fourier layers was selected from $\{1, 2, 3, 4\}$.
- **iDON**. We used the Jax implementation of iDON from the authors (`https://github.com/pkmtum/Semi-supervised_Invertible_Neural_Operators/tree/main`). The number of training epochs was set to 1000. We varied the number of layers for the branch net and trunk set from {1, 3, 4, 5}. Note that to ensure invertibility, the width of the branch net must be set to the dimension of the (discretized) input function. For instance, if the input function is sampled on a 64×64 grid, the width of the branch net will be 4096. The inner-width of the trunk net was selected from {5, 10, 20} and the width of the branch net. For a fair comparison, only the forward and backward supervised loss terms were retained for training.

Table 4: The Signal-To-Noise Ratios (SNR) in DB for Datasets With Noises.

Benchmark	Forward Prediction		Inverse Prediction	
	10% noise	20% noise	10% noise	20% noise
D-LINE	22.2	16.1	28.4	22.4
D-CURV	20.1	14.1	28.5	22.5
W-OVAL	20.0	13.9	32.4	26.4
W-Z	20.0	13.9	29.7	23.6
NS	20.7	14.7	21.4	15.4

- **NIO.** We used the PyTorch implementation from the authors (`https://github.com/mroberto166/nio`). We employed 1000 training epochs. NIO used convolution layers for the branch net of the deepONet module. We tuned the number of convolution layers from {6, 8, 10}, and kernel size from {3, 5}, and padding from {1, 3}. The stride is fixed to 2. For the trunk net, we tuned the number of layers from $\{2, 3, 4\}$, and the layer width from $\{32, 64, 128, 256\}$. For the FNO component, we varied the number of Fourier layers from $\{1, 2, 3, 4\}$, lifting dimension from $\{32, 64, 128\}$, and the number of Fourier modes from $\{8, 12, 16, 32\}$.

- **IOFM.** We used the original FNO implementation from the authors (`https://github.com/neuraloperator/neuraloperator`) to train the forward model. The hyper-parameter selection and training follow the same way as we used FNO for forward and backward prediction. We used the same β-VAE as in iFNO. To conduct inverse prediction, we ran 1000 ADAM epochs optimize the input embeddings. The step-size for the optimization was selected from $\{10^{-3}, 10^{-2}, 10^{-1}, 0.2, 0.5\}$

- **GNOT.** We used the original implementation from authors (`https://github.com/HaoZhongkai/GNOT`) to train forward and inverse prediction models. For hyper-parameter selection, we varied the dimension of the embeddings from $\{64, 128, 256\}$ and number of attention layers from $\{2, 3, 4, 5\}$. We used the default data normalization(unit) as in their repository. The model was trained with the default optimizer(ADAMW), with weight decay of 5E-6, gradient clip of 0.999, with 50 warm up epochs and 500 training epochs in total.

B Ablation Study

We further investigate how the performance of iFNO varies along with model size. To this end, we ran iFNO on D-LINE and W-OVAL, with 10% noise level in the training data. For D-LINE, we fixed the number of Fourier modes to 32 and the channel lifting dimension to 64. For W-OVAL, we fixed the number of Fourier modes to 8 and the channel lifting dimension to 64. We varied the number of invertible Fourier blocks from {1, 2, 3, 4}. The test relative L_2 error for the forward and inverse prediction is shown in Fig. 11 and Fig. 12. As we can see, with more blocks, the prediction accuracy of iFNO can be further improved. The best performance is achieved at three and four blocks for D-LINE and W-OVAL, respectively, in terms of the average error for the forward and inverse prediction.

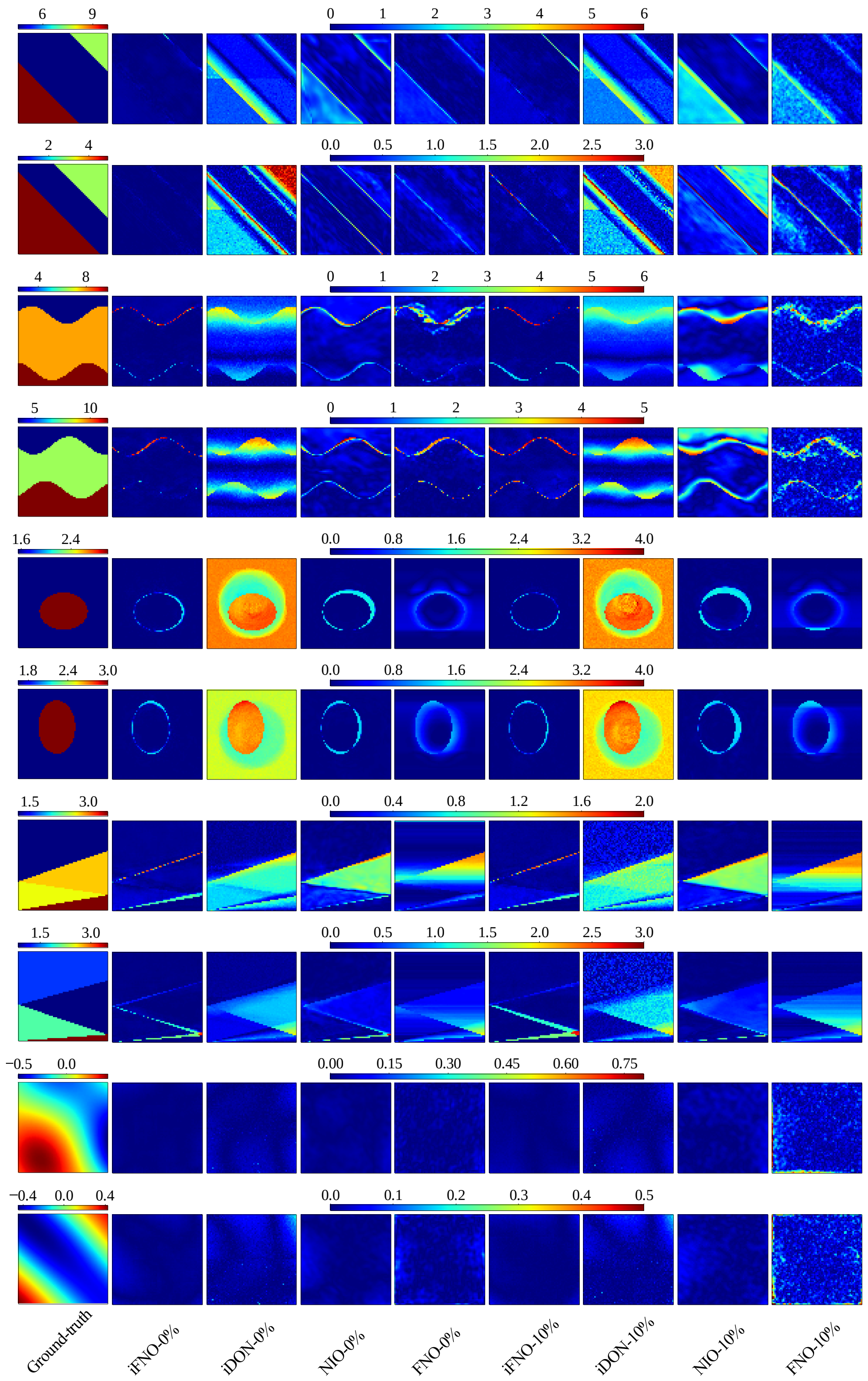


Figure 4: Pointwise Error of Inverse Prediction, where “-0%” and “-10%” indicate the noise level in the training data.

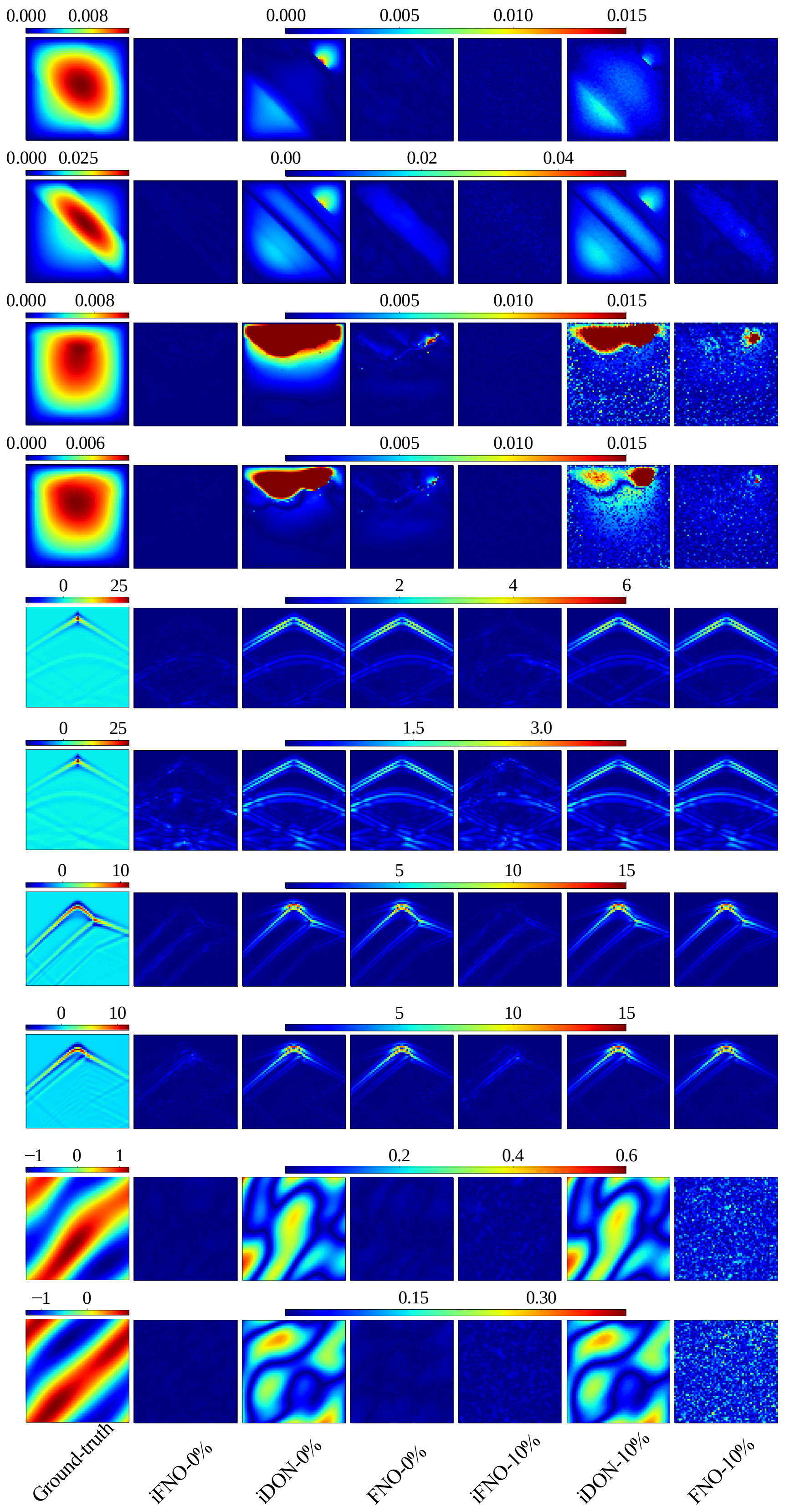
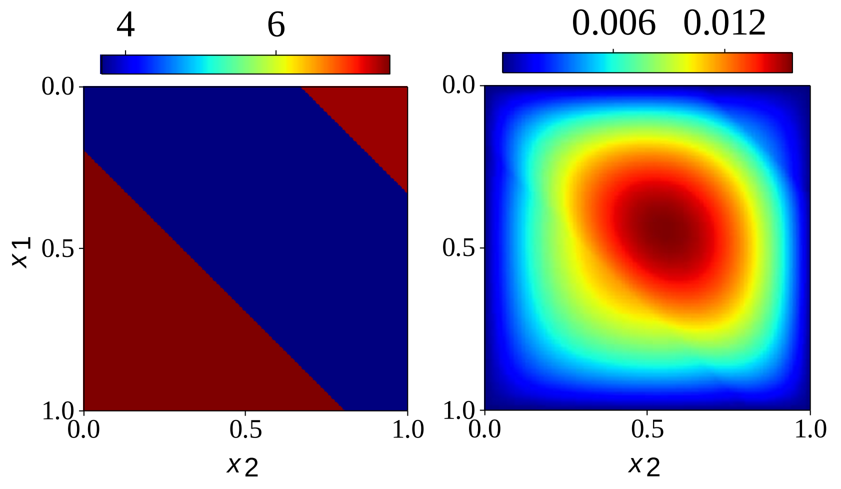
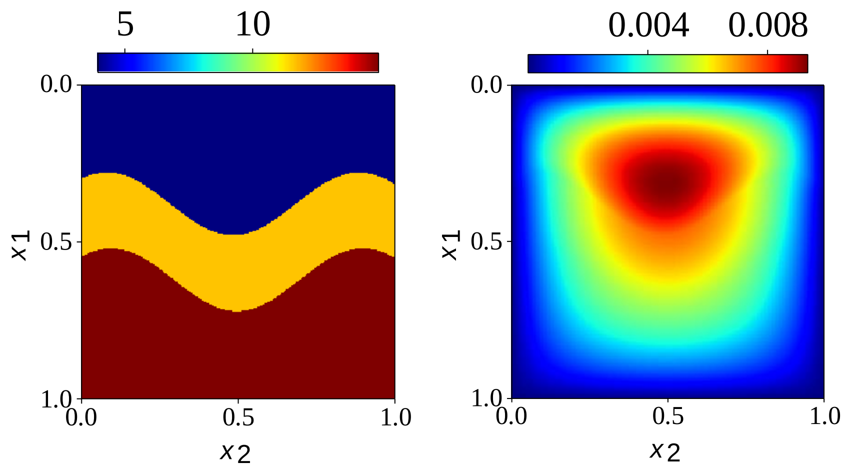


Figure 5: Pointwise Error of Forward Prediction, where “-0%” and “-10%” indicate the noise level in the training data.



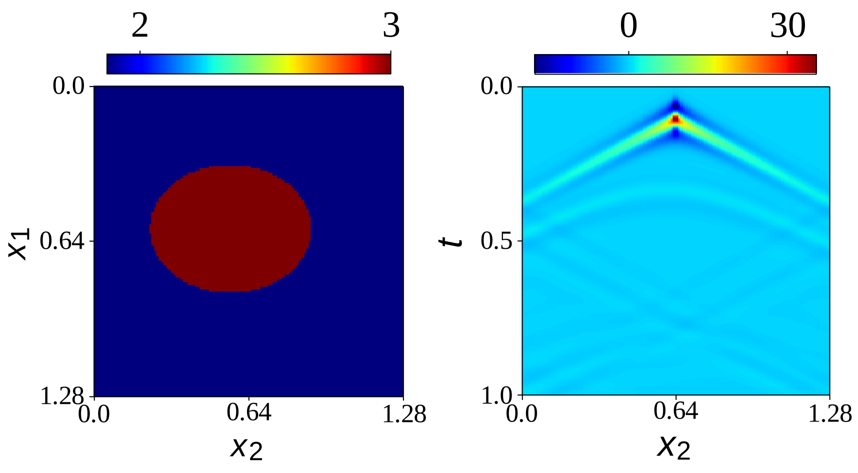
(a) *Permeability* (b) *Solution*

Figure 6: Darcy Flow with Piece-Wise Permeability and Linear Interfaces.



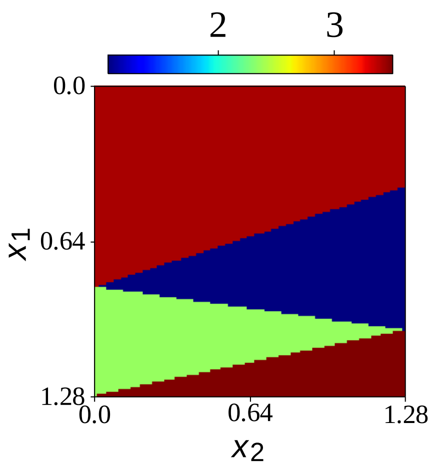
(a) *Permeability* (b) *Solution*

Figure 7: Darcy Flow with Piece-Wise Permeability and Curved Interfaces.

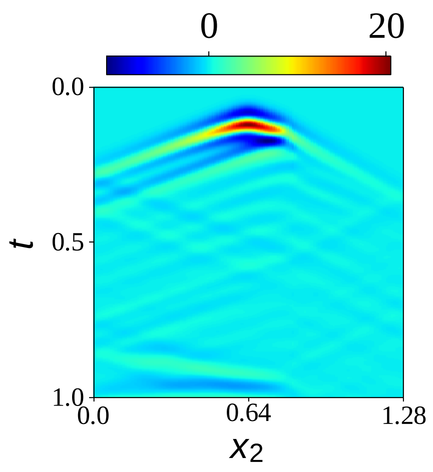


(a) *Square Slowness* (b) *Measurements at Receivers*

Figure 8: Wave Propagation via Piece-Wise Square Slowness and Oval-Shaped Interface.

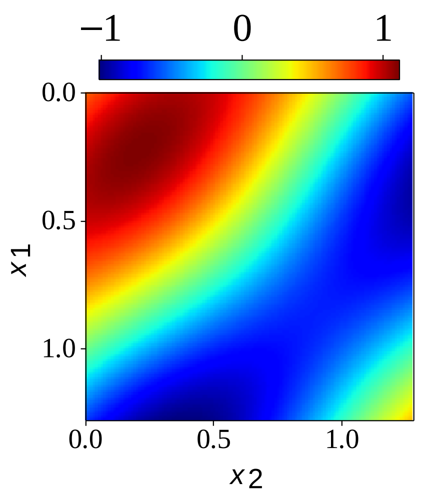


(a) *Square Slowness*

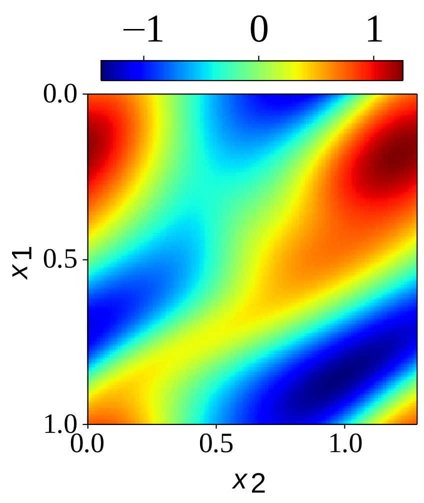


(b) *Measurements at Receivers*

Figure 9: Wave Propagation via Piece-Wise Square Slowness and Z-Shaped Interface.

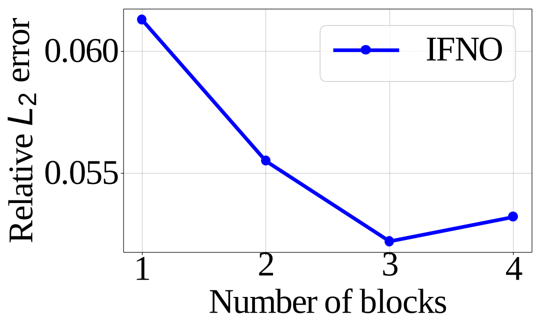


(a) *Initial Condition*

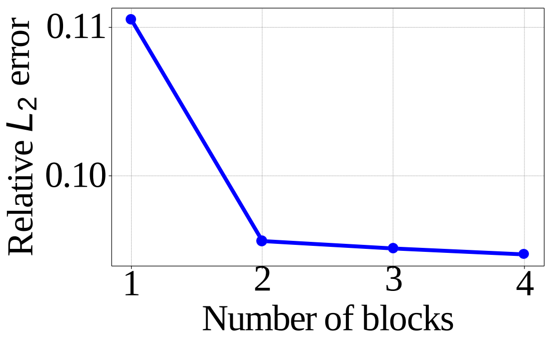


(b) *Solution at* $t = 10$

Figure 10: Solution of NS Equation.



(a) *Forward*

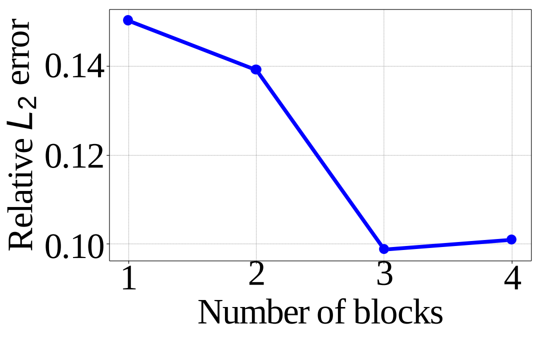


(b) *Inverse*

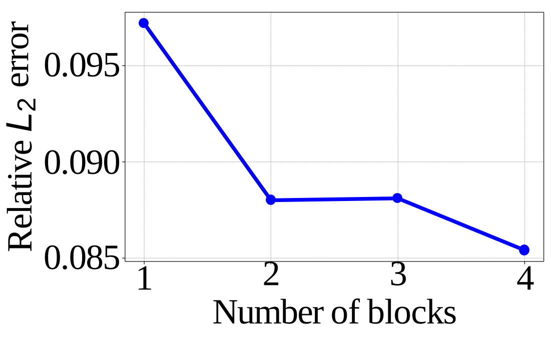
Figure 11: Relative L_2 Error of iFNO on D-LINE with 10% Noise in Training Data.

Table 5: Relative L_2 Error on Each Inverse Prediction Task with IOFM.

Method	IFNO	IOFM
0%		
D-LINE	**5.66e-2** ±1.0e-3	5.25e-1 ± 3.13e-2
D-CURV	**5.54e-2** ±9.4e-4	1.78e-1 ± 4.61e-3
W-OVAL	**5.74e-2** ±1.3e-3	2.91e-1 ± 2.33e-2
W-Z	**2.01e-1** ±3.3e-3	7.42e-1 ± 5.96e-2
NS	**5.09e-2** ±6.4e-4	5.78e-2 ± 1.79e-3
10%		
D-LINE	**8.99e-2** ±1.4e-3	4.61e-1 ± 2.05e-2
D-CURV	**7.94e-2** ±9.7e-4	2.17e-1 ± 3.40e-3
W-OVAL	**7.19e-2** ±1.6e-3	2.29e-1 ± 1.12e-2
W-Z	**1.91e-1** ±3.1e-3	7.02e-1 ± 4.38e-2
NS	**6.66e-2** ±2.6e-4	9.02e-2 ± 1.79e-3
20%		
D-LINE	**1.13e-1** ±9.0e-4	3.61e-1 ± 2.14e-2
D-CURV	**9.58e-2** ±1.2e-3	2.35e-1 ± 8.14e-3
W-OVAL	**8.85e-2** ±2.6e-3	2.01e-1 ± 7.02e-3
W-Z	**2.00e-1** ±3.1e-3	8.53e-1 ± 7.96e-2
NS	**7.52e-2** ±4.7e-4	2.36e-1 ± 6.22e-3



(a) *Forward*



(b) *Inverse*

Figure 12: Relative L_2 Error of iFNO on W-OVAL with 10% Noise in Training Data.