MNO : A Multi-modal Neural Operator for Parametric Nonlinear BVPs

Vamshi C. Madala[1], **Nithin Govindarajan**[2], **Shivkumar Chandrasekaran**[1]

[1]University of California Santa Barbara, [2]KU Leuven

Abstract

We introduce a novel Multi-modal Neural Operator (MNO) architecture designed to learn solution operators for multi-parameter non-linear boundary value problems (BVPs). Traditional neural operators primarily map either the PDE coefficients or source terms independently to the solution, limiting their flexibility and applicability. In contrast, our proposed MNO architecture generalizes these approaches by mapping multiple parameters—including PDE coefficients, source terms, and boundary conditions—to the solution space in a unified manner. Our MNO is motivated by the hierarchical nested bases of the Fast Multipole Method (FMM) and is constructed systematically through three key components: a parameter-efficient Generalized FMM (GFMM) block, a Unimodal Neural Operator (UNO) built upon GFMM-blocks for single-parameter mappings, and most importantly, a multimodal fusion mechanism extending these components to learn the joint map. We demonstrate the multi-modal generalization capacity of our approach on both linear and nonlinear BVPs. Our experiments show that the network effectively handles simultaneous variations in PDE coefficients and source/boundary terms.

1 Introduction

We are interested in simultaneously solving a family of variable-coefficient partial differential equations (PDEs) with inhomogeneous boundary conditions on a open connected domain Ω for $u : \bar{\Omega} \to \mathbb{R}$. Specifically, our aim is to solve

$$\mathcal{D}_1(x, u; a) = c(x), \quad x \in \Omega, \tag{1a}$$
$$\mathcal{D}_2(x, u; f) = g(x), \quad x \in \partial\Omega, \tag{1b}$$

where $\mathcal{D}_1$ is a nonlinear differential operator and $\mathcal{D}_2$ another (not necessarily) differential operator of lower degree. Furthermore the coefficients a, c, f, g will be assumed to be in some compact set during training.

Related works In recent years, data-driven neural network-based approaches to solve PDEs have made considerable progress, of which, physics-informed neural networks (PINNs)(Raissi, Perdikaris, and Karniadakis 2019) and neural operators (NO) (Kovachki et al. 2023; Lu et al. 2021) are the most popular. PINNs originally took a generic DNN architecture and found the solution by setting the residual of the PDE as the loss function. They have evolved since then to be versatile and can exploit the GPU easily for solving complex nonlinear PDEs. But they can take an unpredictable amount of time to reduce the residual to a desired level, becoming very slow in some circumstances. NOs on the other hand, are pre-trained over a compact family of coefficients to produce the correct solution. So their training time can be much larger than that of a PINN, but their inference time is fixed and typically quite fast. Training NOs requires knowledge of true solutions, whereas PINNs don't have that constraint. Our setup is akin to NOs rather than PINNs.

Many variations of neural operators have been proposed in recent years to tackle different aspects of PDEs such as multiple scales, irregular grids(Li et al. 2020; Hao et al. 2023; Wang et al. 2024), operator domains(Li et al. 2021; Helwig et al. 2023; Gupta, Xiao, and Bogdan 2021; Chen et al. 2024a) and underlying neural network architectures(Raoni'c et al. 2023; Wu et al. 2024). Neural operators have been applied in a variety of domains like computational fluid dynamics(Mao et al. 2021), multi-physics(Cai et al. 2021), wave propagation(Chen et al. 2024a), weather forecasting(Pathak et al. 2022), etc. Nevertheless, most existing neural operators either map only the operator coefficient a or the source term c of the PDE to the solution u, but not both simultaneously. For example, Chen et al. (2024a) used a combined FNO and U-Net (Ronneberger, Fischer, and Brox 2015) based architecture to decouple the coefficient and source term in the Helmholtz equation to learn a joint map. Moreover, most neural operator approaches assume fixed boundary conditions (Aldirany et al. 2024; Sau and Yin 2024) or propose complex extensions to handle inhomogeneous boundary conditions such as BENO (Wang et al. 2024) where the authors use separate graph neural networks for interior and boundary source terms through connecting them with a transformer(Vaswani et al. 2017) network. MIONet (Jin, Meng, and Lu 2022) handles multiple inputs, however, its examples primarily feature zero boundary conditions (BCs) or zero right-hand sides (RHS) similar to FNO and DeepONet.

We refer to the operators that map only $a \mapsto u$ or $c \mapsto u$

github.com/vamshichowdary/MultimodalNeuralOperator

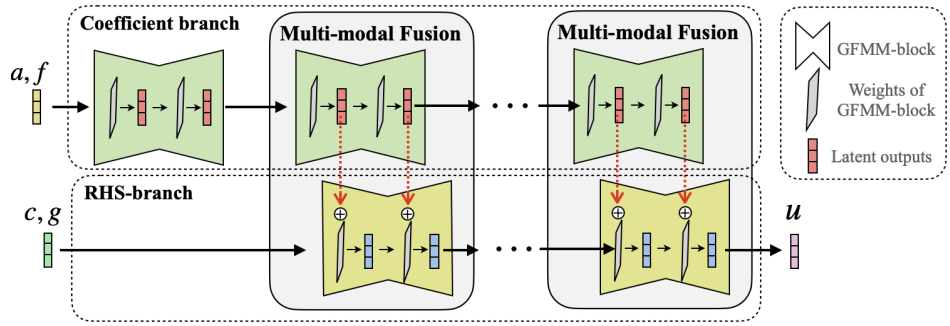

Figure 1: A schematic overview of the multi-modal neural operator (MNO).

as *uni-modal* operators, and operators that map both i.e., $a, c \mapsto u$, or more, as *multi-modal* operators. The lack of truly multi-modal neural operators in the literature partly stems from the mismatch between the types of objects being mapped: a, f parametrizes a family of operators that can act on u, whereas c, g parametrizes the range space of these operators. They typically correspond to different physical quantities with different units. The design of models that effectively handle both relations is highly useful in engineering applications. In inverse scattering, the Helmholtz equation $\Delta u + k^2(1 + a(x))u(x) = c(x)$ is routinely solved multiple times for different scattering potentials a and excitations c (Chen et al. 2024a). In structural mechanics, elasticity problems require repeated solving of PDEs to retrieve displacement fields u for varying material properties a and load configurations c. Thus, a multi-modal operator that can generalize across both material and forcing variations could reduce simulation costs and enable real-time feedback during design iterations.

Our contributions In this work, we propose a novel *multi-modal neural operator* (MNO) architecture that learns solution operators mapping multiple parameters of PDEs, boundary equations and their right hand sides (RHS) to the solution. That is, our network learns the following general solution operator for (1):

$$\mathcal{A}^\dagger : a, c, f, g \mapsto u.$$

Our proposed architecture is rigorously motivated by the hierarchical nested bases of the fast multipole method (FMM)(Greengard and Rokhlin 1987). While previous works (Fan et al. 2019; Sushnikova, Kharyuk, and Oseledets 2022; Li et al. 2020) have proposed FMM-based architectures for unimodal operators that map either a or c to u for fixed boundary conditions, ours is the first to extend this architecture to the multi-modal paradigm where all the coefficients of the PDE are inputs of the network. To achieve this functionality, our MNO is systematically built up in increasing levels of complexity as follows:

- **GFMM-block**: a parameter-efficient feed-forward network that generalizes the FMM signal flow graph for multiple parameter inputs with nonlinear activations, referred to as the Generalized FMM (GFMM) block (section 2.1).
- **UNO**: a Uni-modal Neural Operator (UNO) built on top of Generalized FMM-blocks for learning solution operators of PDEs with one parameter (e.g., a or c) as input (section 2.2).
- **Multi-modal Fusion**: a *multi-linear* extension of the UNO and GFMM-blocks called multi-modal fusion to learn the joint map $a, c \mapsto u$ (section 2.3).

We evaluate the performance of our approach on both linear and nonlinear boundary value problems (BVPs), considering both uni-modal and multi-modal scenarios. For the linear case, we use Darcy's flow, which simplifies to the Poisson equation under constant parameters. For the nonlinear case, we test our method on a first-order BVP featuring an absolute value nonlinearity. While we compare the performance of our UNO model against existing architectures such as FNO and DeepONet—showing improved generalization—we focus primarily on demonstrating the effectiveness of our MNO architecture. In particular, we show that MNO can achieve multi-modal generalization without a significant performance drop compared to its uni-modal (UNO) counterpart.

In our experiments, we put explicit effort into testing the generalization ability of our network to out-of-distribution data. Training neural operators requires ground-truth solutions for different input parameter functions. In most cases, training data is generated by constructing examples using a classical solver (Lu et al. 2021; Li et al. 2021; Long et al. 2018; Wang, Wang, and Perdikaris 2021) which might be prohibitively expensive, especially for high-dimensional and nonlinear PDEs(Gómez-Castro 2024). Given that it is much easier to sample the solution directly by using the PDE's differential operator to compute the RHS (Hasani and Ward 2024), there is much scope for training the neural operators on this synthetic data and employ them on unseen parameter distributions and PDEs (Hasani and Ward 2024; Chen et al. 2024b; Totounferoush et al. 2025; Shen, Marwah, and Talwalkar 2024). The residuals of the predicted solution when trained using this synthetic data, however, can exhibit a different distribution compared to the actual distributions observed during testing (Lerer, Ben-Yair, and Treister 2024). This discrepancy has an impact on the generalization capacity of the network.

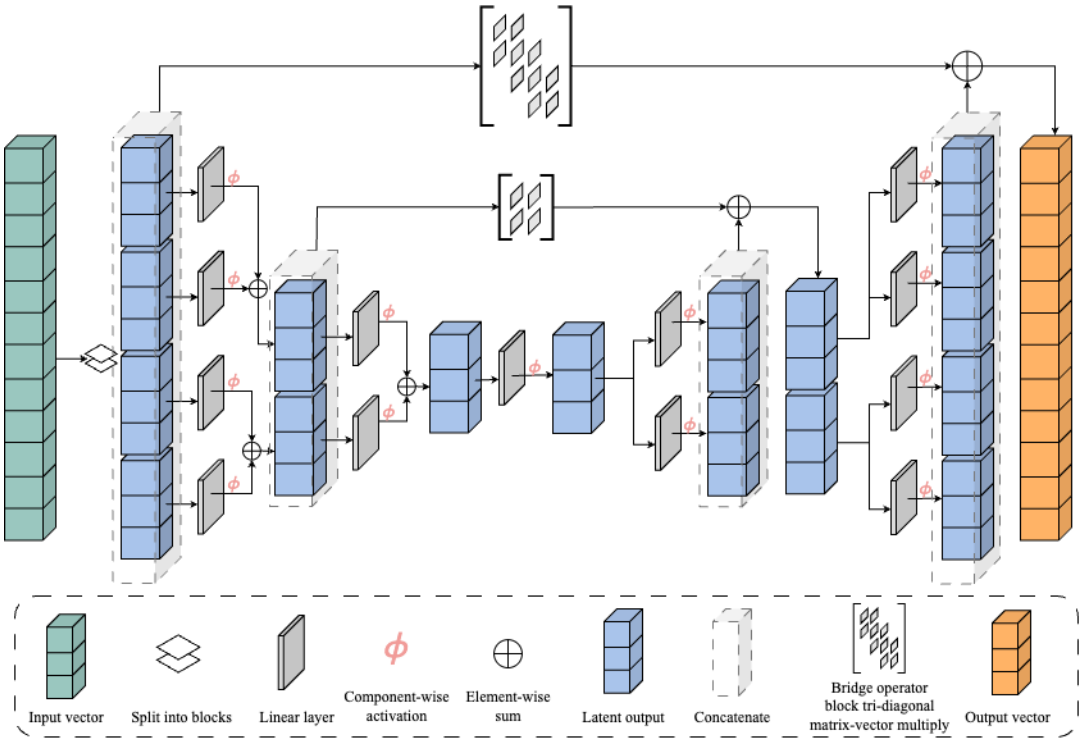

Figure 2: A schematic depiction of a one-dimensional GFMM-block. If the activation function ϕ is set to the identity map, the GFMM-block reduces to a linear block.

2 The multi-modal neural operator (MNO)

The overview of our multi-modal neural operator (MNO) is shown in Figure 1. It consists of multiple GFMM-blocks stacked along two branches. The top branch (coefficient-branch) accepts a, f as inputs and the bottom branch (RHS-branch) takes in c, g as inputs. The hierarchical nested structure of each GFMM-block produces intermediate basis representations of the respective PDE parameter inputs, shown as *latent outputs* in the figure and described in detail in the next section. In section 2.2, we develop the uni-modal neural operator (UNO) that learns the map $c \mapsto u$ for fixed coefficient PDEs. Finally, in section 2.3, we present the multi-fusion operation in MNO that updates the RHS-branch by learning the weight corrections from the latent outputs of the coefficient branch for variable coefficient PDEs.

2.1 The GFMM-block

Discretization transforms the linear PDE into a linear system $Au = c$, with A as the discretized forward operator. In the setting of PDE operators, it is well-known that A often exhibits low-rank structures. Analytic results, such as in Chandrasekaran et al. (2010), provide additional theoretical backing to these observations. Starting with foundational techniques such as FMM (Greengard and Rokhlin 1987) and hierarchical matrices(Börm, Grasedyck, and Hackbusch 2003), various algebraic representations have been proposed to capture the low-rank structures in A so that efficient linear algebra operations can be performed with them (Chandrasekaran et al. 2003; Xia et al. 2010). While most representations have fast matrix-vector product algorithms, they do not always have fast solvers which are essential as preconditioners in iterative methods. In general, finding efficient low-rank representations for inverse operators of nonlinear PDEs using classical techniques is an open problem. Deep learning techniques provide an alternative framework, and rank-structured representations have been proposed in neural networks to learn inverse operators (Kovachki et al. 2023; Fan et al. 2019; Li et al. 2020; Sushnikova, Kharyuk, and Oseledets 2022).

Linear GFMM We follow along similar lines and suggest a generalized FMM feed-forward architecture. Figure 2 illustrates a one-dimensional linear GFMM-block if ϕ is set to the identity. This block is an extension of the signal flow graph of the FMM matrix-vector product in Figure 6, but with the parameters of the FMM signal flow graph replaced with learnable weights. Furthermore, general banded matrices are included as additional "bridge" operators. As depicted in Figure 2, the GFMM block contains three types of layers: down-sampling/encoder layers, up-sampling/decoder layers, and bridge layers. Similar to the original FMM method, the number of up-sampling(down-sampling) layers L is a hyper-parameter for the network, and is typically chosen based on the physics of the underlying PDE and also memory constraints of the FMM solver. It defines the blocking size of the input vector and, consequently, the number of learnable parameters in the network. In its simplest form, all the layers are composed of $P \times P$ matrices, which apply linear transformations to their input

vectors of size P, where $P = \frac{D}{2^L}$ and D is the size of the input vector c.

Algorithm 1: GFMM-block forward pass

Input: $c \in \mathbb{R}^D$
Output: $u \in \mathbb{R}^D$
Parameters:
L: No. of up-sampling/down-sampling layers
$M = 2^L$: Number of blocks
$P = \frac{D}{M}$: Block size
$\mathbf{E}_i^l, \mathbf{D}_i^l \in \mathbb{R}^{P \times P}$ for $l = 1, \ldots, L$ and $i = 1, \ldots, \frac{M}{2^l}$: Encoder and decoder layer parameters
$\mathbf{B}^l \in \mathbb{R}^{\frac{M}{2^l}P \times \frac{M}{2^l}P}$ for $l = 0, \ldots, L$: Bridge parameters
procedure GFMM-BLOCK(c)
  $h^0 \leftarrow [c_1, c_2, \ldots, c_M]$
  for $l = 1$ to L **do**
    $h_i^l \leftarrow \text{BASISTRANSFORM}\left(\mathbf{E}_{2i-1}^l, h_{2i-1}^{l-1}\right) + \text{BASISTRANSFORM}\left(\mathbf{E}_{2i}^l, h_{2i}^{l-1}\right)$ for $i = 1$ to $\frac{M}{2^l}$
    $h^l \leftarrow [h_1^l, h_2^l, \ldots, h_{\frac{M}{2^l}}^l]$
  end for
  $z^L \leftarrow \mathbf{B}^L h^L$
  for $l = L - 1$ to 0 **do**
    $z_i^l \leftarrow \text{BASISTRANSFORM}\left(\mathbf{D}_i^l, z_{\lfloor \frac{i+1}{2} \rfloor}^{l+1}\right)$ for $i = 1$ to $\frac{M}{2^l}$
    $z^l \leftarrow [z_1^l, z_2^l, \ldots, z_{\frac{M}{2^l}}^l]$
    $z^l \leftarrow z^l + \mathbf{B}^l h^l$
  end for
  $u \leftarrow z^0$
end procedure

The full forward pass of GFMM-block is described in Algorithm 1, where `BasisTransform`$(\mathbf{F}, y)$ corresponds to the basis transformation within the feed-forward network. For a single-channel GFMM-block where y is a vector and $\mathbf{F}$ is a $P \times P$ linear layer, this operation corresponds to a matrix-vector multiply. We extend this to support multi-channel layers and inputs in Section 2.2 and Algorithm 2. A careful count reveals a total number of $(10 \times 2^L - 2L - 9)P^2$ learnable parameters in the linear one-dimensional GFMM-block. To put this into contrast, a fully dense linear layer would involve $D^2 = 2^{2L}P^2$ parameters. The one-dimensional GFMM-block can be generalized to a two-dimensional one by applying analogous modifications to the 2D FMM signal flow graph which, in essence, is a tensor product of 1D signal flow graph (see Appendix E).

Nonlinear GFMM While linear GFMM-blocks are suitable architectures to represent inverse operators of linear PDEs, they have to be extended with nonlinear activations to model nonlinear PDEs. Nonlinear activations are also required to map coefficients a, f of the PDE to the solution since this relationship is nonlinear by nature. We use the nonlinear rational function $\phi(x) = \frac{x}{1+|x|}$ as the activation function, which is motivated by the observation that the solution to a linear partial differential equation is a rational function of its coefficients. Later in our experiments (see section C), we show that this type of rational activation function seems to perform better than ReLUs.

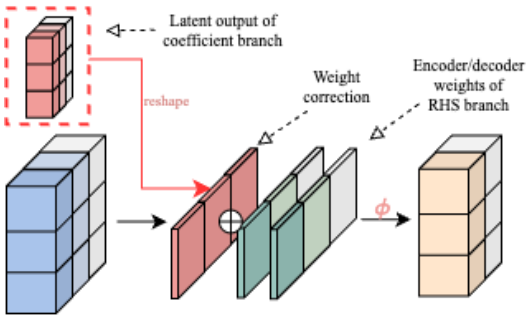

Figure 3: Multi-modal fusion: Basis transformation in one layer of the RHS-branch of MNO. Complete multi-modal fusion across the entire MNO is shown in Figure 9 in the Appendix.

2.2 Uni-modal Neural Operator (UNO) with GFMM-blocks

The primary motivation for using the GFMM-block architecture is the efficient representation of inverse operators of rank-structured linear systems. But learning the representation of a generic PDE using a single GFMM-block might be too restrictive. Experimentally, we observed that sequentially stacking multiple GFMM blocks leads to faster convergence of the training error. This is not surprising given the benefits of depth in neural networks (Telgarsky 2016) and the increase in the number of learnable parameters. Subsequently, we define the uni-modal neural operator (UNO) as a stack of GFMM-blocks with the number of blocks being a hyper-parameter that can be tuned. Moreover, to enable learning the solution operator for more than one input parameter, we extend the GFMM-blocks in the UNO architecture to support multi-channel inputs and outputs. We do this by replacing each $P \times P$ linear layer $\mathbf{F}$ in the encoder and decoder layers by a 4-dimensional tensor of size $C_{out} \times C_{in} \times P \times P$, where C_{out} is the number of output channels desired and C_{in} is the number of channels in the input. The linear transformation is applied such that each channel in the output vector z is obtained by a matrix-vector multiplication of input tensor y with the $P \times P$ weight matrix of the corresponding channel and them summed over all the input channels i.e. $z(i) = \sum_j \mathbf{F}(i, j, :, :)y(j)$.

2.3 Multi-modal Fusion

Similar to FNO and DeepONet, UNO can approximate the solution operator of PDEs that map either the coefficients or the RHS to the solution, while keeping other parameters fixed. To extend UNO to a complete MNO network involves the construction in Figure 1 with branches of stacked GFMM-blocks corresponding, respectively, to the coefficient and RHS-branch. The RHS-branch is analogous to a UNO that directly maps the RHS to the solution, but in order to learn the simultaneous map from both coefficients and RHS to the solution, we "fuse" the weights of the RHS-

Model	Number of params	Solution sampling				RHS sampling			
		$u = \alpha_k T_k$(Train)		$u = \sum_k \alpha_k T_k$(OOD)		$c = \alpha_k T_k$(OOD)		$c = \sum_k \alpha_k T_k$(OOD)	
		ϵ_{rel}	ϵ_{be}	ϵ_{rel}	ϵ_{be}	ϵ_{rel}	ϵ_{be}	ϵ_{rel}	ϵ_{be}
FNO	139,713	8.24e-03	1.72e-3	4.54e-01	3.29e-2	7.56e-01	1.29e-2	9.63e-01	2.15e-2
DeepONet	132,224	2.05e-03	1.60e-3	1.09e-03	2.40e-3	2.63e-01	1.98e-2	7.31e-01	1.45e-2
UNO	**73,216**	**5.32e-06**	**1.12e-4**	**5.72e-06**	**1.14e-4**	**8.18e-02**	**2.08e-3**	**5.76e-01**	**4.78e-3**

Table 1: Relative error and residual error of FNO, DeepONet and UNO for 1D Poisson equation. I) For "Solution sampling" columns, u is i) a random T_k (Chebyshev basis), scaled by random α_k, which is same as training data, and ii) random linear combinations of T_k which is out-of-training distribution (OOD). II) For "RHS sampling" columns, c is i) a random T_k (Chebyshev basis), scaled by random α_k and ii) random linear combinations of T_k, but in this case both constitute out-of-training distribution.

branch using the latent outputs produced by GFMM-blocks of the coefficient branch. We describe this fusing operation as *multi-modal fusion* as illustrated in Figure 3. The rationale behind this operation can be explained as follows. Consider a simple linear PDE described by

$$a(x)u'(x) + u(x) = c(x).$$

For the discrete version of this problem the linear operator (including BC) that maps c to u is a rational function of the sampled values of a. Moreover this rational dependence is also true for the FMM representation of the inverse, though that fact requires a careful perusal of the literature, but is known to experts. Since it is well-known that deep networks can generate rational functions effectively, it makes sense to use a separate DNN to generate the FMM coefficients for the inverse from the samples of a. Furthermore, by doing a perturbation analysis, one can show that localized changes to a will primarily affect the solution u in the same location. Motivated by this reasoning it also makes sense to posit that the DNN that maps a to the FMM weights, itself should have an FMM-like architecture.

3 Experiments

	Poisson's	**Darcy flow**	**Generic first order**
Equations	$Au = c$	$-\nabla.(a\nabla u) = c$	$a(x)u' + b(x)\lvert u\rvert = c(x)$
	A : 1D Laplacian	$u(0) = u_0; u(1) = u_1$	$\int f(x)u(x)dx = g$
Linearity	linear	linear	nonlinear
Coefficient	constant coeff	single variable coeff	multiple variable coeffs
BC	fixed BC	fixed BC	parametric integral BC
Map	$c \mapsto u$	$a, c \mapsto u$	$a, b, c, f, g \mapsto u$
Model	UNO	MNO	MNO

Table 2: Summary of the experiments setup.

In this section, we train and evaluate the performance of our UNO and MNO networks on both linear and nonlinear BVPs, emphasizing their ability to learn solution operators across different configurations, as shown in Table 2. We also focus on the out-of-distribution performance of the models trained using synthetic data.

We begin by showing that our linear UNO network can approximate the inverse operator of discrete 1D Poisson's equation. We compare this with FNO and DeepONet in learning the map between $c(x)$ and $u(x)$. Next we show our MNO network learning the joint map $a, c \mapsto u$ for the case of 1D Darcy flow with fixed boundary conditions. Finally, we present the experiments for the case of the full map between all the parameters of the PDE and the boundary, for multiple RHS for a generic nonlinear BVP.

3.1 Experimental setup

Training data We adopt the synthetic data approach from Hasani and Ward (2024), constructing the synthetic solution u using the Chebyshev basis: $u(x) = \sum_{k=1}^{N_\alpha} \alpha_k T_n(x)$, where T_n are the Chebyshev polynomials of the first kind ($T_n(\cos x) = \cos(nx)$), and α_k is uniformly sampled from $[-1, 1]$. The specific choice of PDE coefficients (a, f) for each problem are described in their respective sections. We use finite difference discretization of the operators $\mathcal{D}_1$ and $\mathcal{D}_2$ in (1) to generate c and g for a particular synthetic u. We resample α_k at every iteration, which serves as a regularization strategy to mitigate overfitting. We refer to this scheme as *solution sampling*.

Validation data We employ two types of validation data to evaluate the out-of-distribution generalization of our models. In the first type, we sample coefficient parameters from distributions that are different from the training distribution, while keeping u drawn from the same distribution, to test generalization across coefficient function spaces for a. In the second, we instead sample the RHS by constructing $c(x) = \sum_{k=1}^{N_\varphi} \varphi_k T_n(x)$, with φ_k sampled from $\mathcal{U}[-1, 1]$ and evaluated on the same grid. We refer to this scheme as *RHS sampling* and evaluate the models using the residual error (2). While most neural operator methods adopt RHS sampling with classical solvers to obtain ground-truth u for training and validation, such solvers are not easily available for complex nonlinear PDEs, and can prove very costly to operate when they are. Instead, we evaluate generalization on RHS-sampled data, while using only solution-sampled data for training.

Training method We use a grid of 256 points and choose the Chebyshev basis set as T_1 to T_{16}. We use single precision and the mean squared error of the predicted solution $\hat{u}$ as

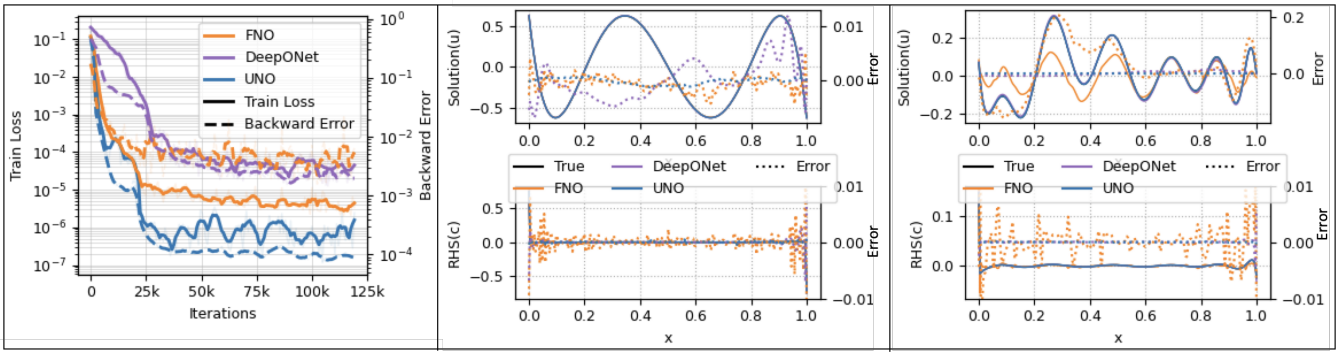

Figure 4: (Left) Training loss and backward error during training for 1D Poisson models. Predicted vs true solutions and corresponding RHS on randomly scaled (middle) and random linear combinations (right) of Chebyshev polynomials representing in-training and out-of-training distribution. The dotted lines represent the error in predictions for each model.

Model	$a \in$ (3b)(Quadratic)		$a \in$ (3c)(Log-normal)	
	ϵ_{rel}	ϵ_{res}	ϵ_{rel}	ϵ_{res}
FNO	(1.00±0.07)e+00	(4.73±0.01)e-03	(1.00±0.04)e+00	(3.32±0.00)e-03
DeepONet	(1.05±0.03)e+00	(4.67±0.02)e-03	(1.00±0.05)e+00	(2.99±0.00)e-03
MIONet	(1.07±0.01)e+02	(5.07±0.07)e-02	(1.91±0.45)e+01	(1.14±0.09)e-02
UNO-aQ	(2.71±0.04)e-02	(6.86±0.07)e-04	(4.07±0.02)e+00	(1.74±0.55)e-02
UNO-aLN	(1.22±0.002)e+01	(3.76±0.03)e-03	(1.95±0.002)e+01	(1.73±0.09)e-02
UNO-mix	(4.94±0.01)e-01	(1.96±0.04)e-03	(5.86±0.05)e-01	(5.39±0.12)e-03
UNO-multich	(5.00±0.05)e-01	(1.09±0.01)e-03	(7.01±0.04)e-01	(3.24±0.09)e-03
MNO	**(4.42±0.03)e-03**	**(3.77±0.06)e-04**	**(3.37±0.07)e-02**	**(1.53±0.01)e-03**

Table 3: Relative error and residual error of models trained on 1D Darcy flow with coefficients a sampled purely from either (3b) or (3c).

the loss function and train using the Adam (Kingma 2014) optimizer with an initial learning rate of 1e-03. We used two NVIDIA TITAN RTX GPUs for all the experiments.

Error metrics For the linear constant coefficient case, where solving the discretized PDE is equivalent to solving the linear problem $Au = c$ where A is a $D \times D$ matrix, we define the normalized backward error ϵ_{be} as in (2), where $\hat{u}$ is the model's predicted solution, and the norm is the standard 2-norm. For the variable coefficient and nonlinear case, we measure the residual error, ϵ_{res}, of the forward operator $\mathcal{D}$ applied on the model's predicted solution $\hat{u}$, and the relative solution error, ϵ_{rel}:

$$\epsilon_{\text{be}} = \frac{\|A\hat{u} - c\|}{\|A\|\|\hat{u}\| + \|c\|}; \qquad \epsilon_{\text{res}} = \|\mathcal{D}(x, \hat{u}) - c(x)\|; \qquad \epsilon_{\text{rel}} = \frac{\|\hat{u} - u\|}{\|u\|} \tag{2}$$

3.2 Linear PDEs

Uni-modal: Discrete 1D Poisson's We train UNO network to approximate the inverse operator of a linear set of equations given by $Au = c$, where A is the discrete 1D Laplacian operator. Our UNO model is composed of two linear GFMM-blocks with parameters $P = 16, L = 4$ sequentially stacked. To compare the performance of UNO, we also train an FNO with `width=32`, `modes=16` and a DeepONet with 3 branch layers and 3 trunk layers, each with `width=128`. We train all the models for 120k iterations starting with a learning rate of 1e-03 and dropping to 1e-04 after 20k iterations on *only* the Chebyshev basis polynomials, i.e., $u_i = \alpha_k T_k$.

Figure 4 (left) illustrates that while all models achieve a training loss below 1e-4, UNO achieves a backward error ϵ_{be} approximately two orders of magnitude lower than FNO and DeepONet, despite having roughly half their number of learnable parameters. As seen from Table 1, though all models accurately predict when u is a randomly scaled Chebyshev polynomial similar to the training set (see Figure 4 (middle)), UNO demonstrates better generalization compared to FNO and DeepONet when u is sampled from out-of-training distribution (see Figure 4 (right)). But when tested on data generated via RHS sampling (where c is sampled and u is derived), all models struggled to generalize, although UNO shows a slight advantage, particularly in its lower backward error (also see Figure 7).

Multi-modal: 1D Darcy Flow For the multi-modal case, we consider the steady state Darcy flow in 1D with variable coefficient $a(x)$ and fixed Dirichlet boundary conditions:

$$-\nabla(a(x)\nabla u(x)) = c(x), \quad 0 < x < 1,$$
$$u(0) = u_0, \quad u(1) = u_1.$$

Our goal is to learn the solution operator $a, c \mapsto u$. We use our multi-modal network MNO with one channel inputs for both coefficient and RHS-branches, while the boundary con-

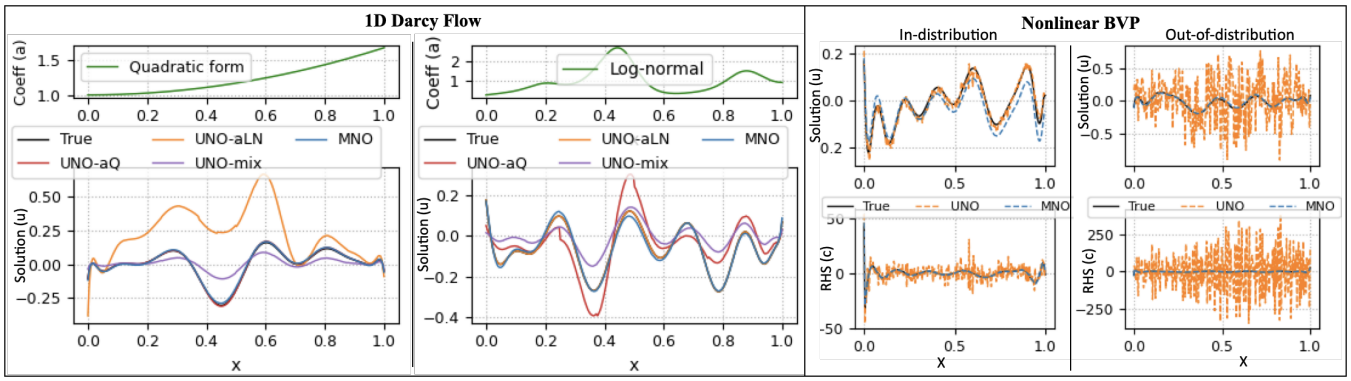

Figure 5: (Left) Test samples for 1D Darcy flow where the coefficient a is drawn from (3b) and (3c) shows that MNO models can learn the simultaneous map $a, c \mapsto u$ while the UNO models cannot. (Right) MNO learns the solution map from multiple parameters but UNO struggles for both in-distribution and out-of-distribution coefficients.

ditions are fixed. We choose $L = 4$ and $P = 16$ as the network's parameters. a is sampled from a mixture distribution of quadratic parametric form and log-normal distributions:

$$a(x) = 0.5a_1(x) + 0.5a_2(x); \tag{3a}$$
$$a_1(x) = 1 + \theta_1 x^2; \quad \text{where } \theta_1 \sim \mathcal{U}(0,1), \tag{3b}$$
$$a_2(x) = 0.1 + e^{\theta_2}; \quad \text{where } \theta_2 \sim \mathcal{N}(0, k(x,x')), \tag{3c}$$
$$k(x,x') = e^{-\frac{|x-x'|^2}{2(0.1D)^2}}.$$

In subsurface flow modeling, e.g., in groundwater flow or oil reservoir simulations governed by steady-state Darcy's law, the permeability field $a(x)$ typically exhibits high heterogeneity. Geological layers formed by sediment deposition have spatial correlations captured by log-normal gaussian processes whereas certain smooth layers are approximated by polynomial spatial variability and learning to map such heterogeneous distributions is useful in practice.

We evaluate MNO against baselines FNO, DeepONet, MIONet and four UNO networks. UNO-aQ and UNO-aLN were trained with a fixed coefficient a sampled from distinct distributions, (3b) & (3c) respectively, while UNO-mix encountered a new a (3a) in each training iteration, similar to MNO. UNO-multich corresponds to the UNO with a, c stacked as multiple channels in the inputs and so does the baseline models, whereas other UNOs only receive a single input. All models achieved a training MSE below 1e-3 over 50k iterations with a mini-batch size of 1000. Figure 5(left) demonstrates that while UNO-aQ and UNO-aLN only succeeded for their specific training a and UNO-mix struggled with novel a values despite its varied training exposure, MNO generalized effectively to unseen coefficients. Furthermore, Table 3, with results averaged over 1000 solution sampled test examples, shows that uni-modal networks failed to adapt to changes in a, whereas MNO robustly mapped both a and c simultaneously to the solution.

3.3 Nonlinear ODEs

In the section we use the following nonlinear first order BVP with a parametric form:

$$a(x)u'(x) + b(x)|u(x)| = c(x); \quad 0 < x < 1, \tag{4a}$$
$$\int f(x)u(x)dx = g; \quad 0 \le x \le 1 \tag{4b}$$

where the boundary condition (4b) is also given in a general parametric form. The absolute value of the solution term and an additional variable coefficient $b(x)$ are chosen only to test model's ability to handle nonlinearity and multiple coefficient parameters, which are sampled from following distributions:

$$\mathcal{P}_{\text{tr}} : \begin{cases} a = 1 + \theta x^2; \\ b = \sum_{i=1}^{4} \phi_i T_i(x); \\ f = \sum_{i=1}^{4} \eta_i T_i(x); \text{ where } \theta, \phi_i, \eta_i \sim \mathcal{U}[-1,1]. \end{cases}$$

Uni-modal We train a UNO network with nonlinear activation where all the parameters a, b, c, f, g are stacked as multiple channels of the input to the UNO for 20k iterations. The GFMM-blocks of UNO are initialized with $P = 16$ and $L = 4$. We evaluate the network using coefficients sampled from both in-distribution and out-of-distribution on batches of 1000 test samples. Table 4 shows the relative error, interior residual error and boundary residual error on different combinations of the sampled test data distributions. UNO achieves low relative error in the predicted solution but higher residual errors with in-distribution test data, suggesting overfitting on the solution without actually learning the joint map from the parameters. Moreover, the UNO model struggles further with out-of-distribution data, failing to attain even low relative error. Examples of UNO model's predictions in both cases are shown in Figure 5(right), where it becomes evident that UNO is unable to smoothly approximate the join map from multiple parameters to the solution. Next we discuss our multi-modal network's performance on the same setup.

Multi-modal We use the same nonlinear BVP in (4) to train our MNO that learns the solution map not only from the

		$a, b \sim \mathcal{P}_{\text{tr}}$			$a, b \nsim \mathcal{P}_{\text{tr}}$		
		ϵ_{rel}	$\epsilon_{\text{res}}^{\text{int}}$	$\epsilon_{\text{res}}^{\text{bnd}}$	ϵ_{rel}	$\epsilon_{\text{res}}^{\text{int}}$	$\epsilon_{\text{res}}^{\text{bnd}}$
$f \sim \mathcal{P}_{\text{tr}}$	FNO	(1.3±0.01)e-01	(9.9±0.05)e-01	(1.1±0.07)e+00	(1.2±0.02)e-01	(9.3±0.08)e-01	(1.2±0.09)e+00
	DeepONet	(1.0±0.03)e+00	(3.0±0.06)e+00	(5.5±0.04)e+00	(1.0±0.05)e+00	(3.3±0.01)e+00	(3.7±0.02)e+00
	MIONet	(7.1±0.09)e+01	(3.3±0.07)e+00	(2.0±0.02)e+01	(5.4±0.03)e+01	(3.6±0.05)e+00	(4.9±0.04)e+01
	UNO	(4.1±0.07)e-02	(4.8±0.04)e+00	(1.4±0.05)e-02	(1.7±0.02)e+01	(1.2±0.08)e+02	(5.6±0.03)e-02
	MNO	**(1.13±0.02)e-02**	**(3.8±0.06)e-01**	**(8.4±0.09)e-03**	**(8.0±0.05)e-02**	**(4.4±0.01)e-01**	**(6.0±0.07)e-03**
$f \nsim \mathcal{P}_{\text{tr}}$	FNO	(1.4±0.04)e-01	(1.3±0.02)e+00	(4.9±0.03)e-01	(1.4±0.05)e-01	(1.4±0.01)e+00	(7.1±0.06)e-01
	DeepONet	(1.0±0.07)e+00	(2.9±0.08)e+00	(1.0±0.06)e+00	(1.0±0.03)e+00	(3.4±0.05)e+00	(1.3±0.09)e+00
	MIONet	(1.6±0.05)e+01	(3.1±0.01)e+00	(1.1±0.04)e+00	(1.4±0.07)e+01	(3.3±0.08)e+00	(5.3±0.02)e+00
	UNO	(1.1±0.03)e+01	(1.2±0.09)e+02	(2.5±0.04)e-02	(2.8±0.06)e+01	(1.9±0.001)e+02	(7.3±0.01)e-02
	MNO	**(1.1±0.01)e-01**	**(4.2±0.37)e-01**	**(1.6±0.30)e-03**	**(1.3±0.09)e-01**	**(5.5±0.14)e-01**	**(1.2±0.42)e-03**

Table 4: Relative error (ϵ_{rel}), interior residual error ($\epsilon_{\text{res}}^{\text{int}}$), and boundary residual error ($\epsilon_{\text{res}}^{\text{bnd}}$) of UNO, MNO, FNO, DeepONet, and MIONet models for nonlinear BVP in (4). The errors are measured on 1000 test samples by sampling interior coefficients (a, b) and boundary coefficient (f) either from $\mathcal{P}_{\text{tr}}$ or from a unit normal (out-of-training) distribution.

	SS	RHS
FNO	3.50e-03	**1.48e-01**
DeepONet	1.90e-04	3.68e-01
UNO	**2.83e-05**	4.96e-01

(a) Poisson's

	SS	RHS
UNO-aQ	4.26e-04	2.64e-01
UNO-aLN	3.99e-03	1.88e-01
UNO-mix	1.98e-03	6.19e-01
MNO	**3.93e-04**	**9.22e-02**

(b) Darcy flow

	SS	RHS
UNO	6.10e+00	1.12e+02
MNO	**4.12e-01**	**2.97e+00**

(c) BVP

Table 5: Residual errors computed on test data sampled using solution sampling (SS) and RHS sampling for 1D Poisson's, 1D Darcy flow and BVP of (4) for different models.

parameters of the interior equations (a, b, c) but also boundary parametric constraints (f, g).

MNO is initialized with two nonlinear GFMM-blocks in the coefficient branch and one nonlinear and one linear GFMM-blocks in the RHS-branch with each block parameterized using $P = 16$ and $L = 4$. Figure 5 (right) shows that MNO's predicted solution is smoother and generalizes better to out-of-distribution coefficients. Lower residual errors across all distributions compared to UNO, as seen from Table 4, indicates that MNO is able to learn the solution operator mapping not only the coefficients and RHS but also the parameterized boundary conditions with better generalization.

3.4 Out-of-distribution Generalization

Table 5 shows the mean residual errors for each experiment when the models are tested on 1000 RHS sampled examples, which are out-of-training distribution. We note the large increase in error on RHS. To fix this, researchers have proposed that the output of these networks to be refined by classical iterative solvers (Zhang et al. 2024).

4 Conclusions & Future work

We presented the Multi-modal Neural Operator (MNO) for learning solution operators of parameterized PDEs under simultaneous variations in coefficients, source terms, and boundary conditions. While our current demonstrations are primarily 1D (with preliminary 2D results in Appendix E), future work will focus on extending MNO to higher dimensions, time-evolving PDEs, and diverse domain geometries. The demonstrated multi-modal capabilities of MNO also suggest its potential in applications like reducing the computational cost of neural-assisted preconditioners for classical iterative solvers. We believe our preliminary results will motivate the community to further explore and develop multi-modal architectures, significantly broadening the scope and utility of neural operators in scientific computing and engineering applications.

References

Aldirany, Z.; Cottereau, R.; Laforest, M.; and Prudhomme, S. 2024. Multi-level neural networks for accurate solutions of boundary-value problems. *Computer Methods in Applied Mechanics and Engineering*, 419: 116666.

Börm, S.; Grasedyck, L.; and Hackbusch, W. 2003. Introduction to hierarchical matrices with applications. *Engineering analysis with boundary elements*, 27(5): 405–422.

Cai, S.; Wang, Z.; Lu, L.; Zaki, T. A.; and Karniadakis, G. E. 2021. DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics*, 436: 110296.

Chandrasekaran, S.; Dewilde, P.; Gu, M.; Pals, T.; Sun, X.; van der Veen, A.; and White, D. 2003. Fast stable solvers for

sequentially semi-separable linear systems of equations and least squares problems. *SIAM Journal on Matrix Analysis and Applications*.

Chandrasekaran, S.; Dewilde, P.; Gu, M.; and Somasunderam, N. 2010. On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs. *SIAM Journal on Matrix Analysis and Applications*, 31(5): 2261–2290.

Chen, F.; Liu, Z.; Lin, G.; Chen, J.; and Shi, Z. 2024a. NSNO: Neumann series neural operator for solving Helmholtz equations in inhomogeneous medium. *Journal of Systems Science and Complexity*, 37(2): 413–440.

Chen, W.; Song, J.; Ren, P.; Subramanian, S.; Morozov, D.; and Mahoney, M. W. 2024b. Data-efficient operator learning via unsupervised pretraining and in-context learning. *Advances in Neural Information Processing Systems*, 37: 6213–6245.

Fan, Y.; Feliu-Faba, J.; Lin, L.; Ying, L.; and Zepeda-Núnez, L. 2019. A multiscale neural network based on hierarchical nested bases. *Research in the Mathematical Sciences*, 6(2): 21.

Gómez-Castro, D. 2024. Beginner's guide to aggregation-diffusion equations. *SeMA Journal*, 81(4): 531–587.

Greengard, L.; and Rokhlin, V. 1987. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2): 325–348.

Gupta, G.; Xiao, X.; and Bogdan, P. 2021. Multiwavelet-based Operator Learning for Differential Equations. In *Neural Information Processing Systems*.

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*, 12556–12569. PMLR.

Hasani, E.; and Ward, R. A. 2024. Generating synthetic data for neural operators. *arXiv preprint arXiv:2401.02398*.

Helwig, J.; Zhang, X.; Fu, C.; Kurtin, J.; Wojtowytsch, S.; and Ji, S. 2023. Group Equivariant Fourier Neural Operators for Partial Differential Equations. In *International Conference on Machine Learning*.

Jin, P.; Meng, S.; and Lu, L. 2022. MIONet: Learning multiple-input operators via tensor product. arXiv:2202.06137.

Kingma, D. P. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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.

Lerer, B.; Ben-Yair, I.; and Treister, E. 2024. Multigrid-augmented deep learning preconditioners for the Helmholtz equation using compact implicit layers. *SIAM Journal on Scientific Computing*, 46(5): S123–S144.

Li, Z.; Kovachki, N.; Azizzadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A.; and Anandkumar, A. 2021. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*.

Li, Z.-Y.; Kovachki, N. B.; Azizzadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A. M.; and Anandkumar, A. 2020. Multipole Graph Neural Operator for Parametric Partial Differential Equations. *ArXiv*, abs/2006.09535.

Long, Z.; Lu, Y.; Ma, X.; and Dong, B. 2018. Pde-net: Learning pdes from data. In *International conference on machine learning*, 3208–3216. PMLR.

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.

Mao, Z.; Lu, L.; Marxen, O.; Zaki, T. A.; and Karniadakis, G. E. 2021. DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. *Journal of computational physics*, 447: 110698.

Molina, A.; Schramowski, P.; and Kersting, K. 2019. Pad\'e activation units: End-to-end learning of flexible activation functions in deep networks. *arXiv preprint arXiv:1907.06732*.

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], 10.48550. *arXiv preprint arXiv.2202.11214*, 22.

Raissi, M.; Perdikaris, P.; and Karniadakis, G. E. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378: 686–707.

Raoni'c, B.; Molinaro, R.; Ryck, T. D.; Rohner, T.; Bartolucci, F.; Alaifari, R.; Mishra, S.; and de B'ezenac, E. 2023. Convolutional Neural Operators for robust and accurate learning of PDEs. In *Neural Information Processing Systems*.

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*, 234–241. Springer.

Sau, R. C.; and Yin, L. 2024. A Review of Neural Network Solvers for Second-order Boundary Value Problems. arXiv:2407.00442.

Shen, J.; Marwah, T.; and Talwalkar, A. 2024. UPS: Efficiently Building Foundation Models for PDE Solving via Cross-Modal Adaptation. *Transactions on Machine Learning Research*.

Sushnikova, D.; Kharyuk, P.; and Oseledets, I. 2022. FMM-Net: neural network architecture based on the Fast Multipole Method. *arXiv preprint arXiv:2212.12899*.

Telgarsky, M. 2016. Benefits of depth in neural networks. In *Conference on learning theory*, 1517–1539. PMLR.

Totounferoush, A.; Kotchourko, S.; Mahoney, M. W.; and Staab, S. 2025. Paving the way for scientific foundation models: enhancing generalization and robustness in PDEs with constraint-aware pre-training. *arXiv preprint arXiv:2503.19081*.

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

Wang, H.; Li, J.; Dwivedi, A.; Hara, K.; and Wu, T. 2024. BENO: Boundary-embedded Neural Operators for Elliptic PDEs. *ArXiv*, abs/2401.09323.

Wang, S.; Wang, H.; and Perdikaris, P. 2021. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances*, 7(40): eabi8605.

Wu, H.; Luo, H.; Wang, H.; Wang, J.; and Long, M. 2024. Transolver: A Fast Transformer Solver for PDEs on General Geometries. *ArXiv*, abs/2402.02366.

Xia, J.; Chandrasekaran, S.; Gu, M.; and Li, X. S. 2010. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications*, 17(6): 953–976.

Zhang, E.; Kahana, A.; Kopaničáková, A.; Turkel, E.; Ranade, R.; Pathak, J.; and Karniadakis, G. E. 2024. Blending neural operators and relaxation methods in PDE numerical solvers. *Nature Machine Intelligence*, 1–11.

A FMM Matrix-vector product

Figure 6 shows the computational graph of a matrix-vector product $Ax = b$ using fast multiple method (FMM)(Greengard and Rokhlin 1987). The graph shows the input block vector x being transformed through up-sweep and down-sweep recursions as:

$$g_i = V_i x_i + \sum_{j \in C(i)} W_j g_j;$$
$$f_i = R_{C^{-1}(i)} f_{C^{-1}(i)} + \sum_{j \in S(i)} B_{i,j} g_j;$$
$$b_i = D_i x_i + U_i f_i;$$

where $C(i)$ denote the set of child nodes of node i, $C^{-1}(i)$ denote the parent nodes of node i, and $S(i)$ denote the set of neighbors of node i, such that the corresponding edges are not in the tree graph. The indices are omitted from the figure for brevity. The edges B, D correspond to bridge operators and green and red paths correspond to encoder and decoder of Figure 2. The FMM recursions are sparse linear equations in x, b, f, g and so the matrix-vector product is computed efficiently.

B Full architecture of MNO

Algorithm 2: Basis Transformation inside GFMM-block

Input: $\mathbf{F} \in \mathbb{R}^{C_{out} \times C_{in} \times P \times P}$: Encoder/decoder parameters
Input: $y \in \mathbb{R}^{C_{in} \times P}$
Input: $\varepsilon \in \mathbb{R}^{P \times P}$ ▷ Latent outputs (for GFMM-blocks with multi-modal fusion)
Output: $z \in \mathbb{R}^{C_{out} \times P}$
Parameters:
C_{in}: Number of input channels
C_{out}: Number of output channels
$\phi : \mathbb{R}^d \to \mathbb{R}^d$: Element-wise nonlinear activation function
$\mathcal{E}$: True if this is a GFMM-block with multi-modal fusion.
procedure BASISTRANSFORM($\mathbf{F}, y, \varepsilon$)
    $z = \mathbf{0}$
    for $i = 1$ to C_{out} **do**
        for $j = 1$ to C_{in} **do**
            if $\mathcal{E}$ **then**
                $\mathbf{F}_{ij} \leftarrow \mathbf{F}_{ij} + \varepsilon$ ▷ Additive weight correction (multi-modal fusion)
            end if
            $z_i \leftarrow z_i + \mathbf{F}_{ij} y_j$ ▷ Matrix-vector product and sum over input channels
        end for
    end for
    $z_i \leftarrow \phi(z_i)$ for $i = 1$ to C_{out} ▷ nonlinear activation
end procedure

The full architecture of MNO for multi-channel inputs and multi-modal fusion between coefficient and RHS-branches is shown in Figure 9. The illustration shows two GFMM-blocks in the coefficient branch and one GFMM-block in the RHS-branch. Considering the example of multi-parameter BVP of (4), the coefficients of interior and boundary equations, $a(x), b(x)$ and $f(x)$ vectors are stacked and given as input to the coefficient branch. RHS terms $c(x), g(x)$ are given as input to the RHS-branch. The weights of the RHS-branch are corrected additively with the intermediate outputs from the coefficient branch and the entire network is trained end-to-end. At each encoder and decoder layer of the GFMM-blocks, the input is transformed using the `BasisTransform` described in algorithm 2 which also includes the weight correction from multi-modal fusion for the GFMM-blocks in RHS-branch.

C ReLU vs Rational Activation

We make the observation that the solution to a linear partial differential equation is a rational function of its coefficients. Moreover, to achieve the multi-modal fusion, we produce the weight corrections of the RHS-branch with the intermediate basis representations of the coefficient branch. Motivated by these, we explore the performance of rational functions as activations for our neural operator networks, in contrast to the traditional ReLU. We empirically observe that a non-linear rational function such as $\phi(x) = \frac{x}{1+|x|}$ performs better than ReLU for learning the multi-modal solution operators of both linear and non-linear BVPs. Figure 8 shows the training loss and residual error progression of MNO models on 1D Darcy flow and the nonlinear rational function outperforms ReLU by a big margin. Table 7 shows the comparison of errors of different models using different activation functions. We can see that for baseline models, *rational activation* did not result in consistent improvement. However, MNO model clearly achieves best performance with rational activation with a significant improvement, providing evidence for our *multi-modal fusion* operation in generating weight corrections of the RHS-branch with the intermediate basis representations of the coefficient branch. Previous works (Molina, Schramowski, and Kersting 2019) explored the variations of learnable rational functions as activations in neural networks. A rigorous analysis of choice of activation functions for MNOs is planned in our future work.

D 1D Poisson

D.1 Results from multiple training runs

We run the experiment outlined in section 3.2 by training the FNO, DeepONet and UNO models independently three times and aggregate the results in Table 6 which reports the mean and standard deviation of the relative and residual errors for the three models in the 1D Poisson case.

D.2 Testing on out-of-training distribution

Figure 7 shows the UNO, FNO, DeepONet models which are trained using solution sampling using randomly scaled Chebyshev polynomials i.e. $u = \alpha_k T_k(x)$ from 3.2, when they are tested on out-of-training distribution examples. For the top left plot, the example is generated by solution sampling, but groundtruth u is chosen as random linear combination of $T_1(x)$ to $T_{16}(x)$. The figure shows that UNO and

Model	Chebyshev		Mixed	
	ϵ_{rel}	ϵ_{be}	ϵ_{rel}	ϵ_{be}
FNO	(8.24±0.03)e-03	(1.72±0.05)e-3	(4.54±0.07)e-01	(3.29±0.02)e-2
DeepONet	(2.05±0.06)e-03	(1.60±0.09)e-3	(1.09±0.02)e-03	(2.40±0.03)e-3
UNO	**(5.32±0.01)e-06**	**(1.12±0.02)e-4**	**(5.72±0.03)e-06**	**(1.14±0.05)e-4**

(a) Solution Sampled

Model	Chebyshev		Mixed	
	ϵ_{rel}	ϵ_{be}	ϵ_{rel}	ϵ_{be}
FNO	(7.56±0.08)e-01	(1.29±0.04)e-2	(9.63±0.01)e-01	(2.15±0.06)e-2
DeepONet	(2.63±0.05)e-01	(1.98±0.07)e-2	(7.31±0.04)e-01	(1.45±0.08)e-2
UNO	**(8.18±0.07)e-02**	**(2.08±0.04)e-3**	**(5.76±0.09)e-01**	**(4.78±0.02)e-3**

(b) RHS Sampled

Table 6: Same as Table 1 but with mean and standard deviations reported from 3 independent training runs of each model.

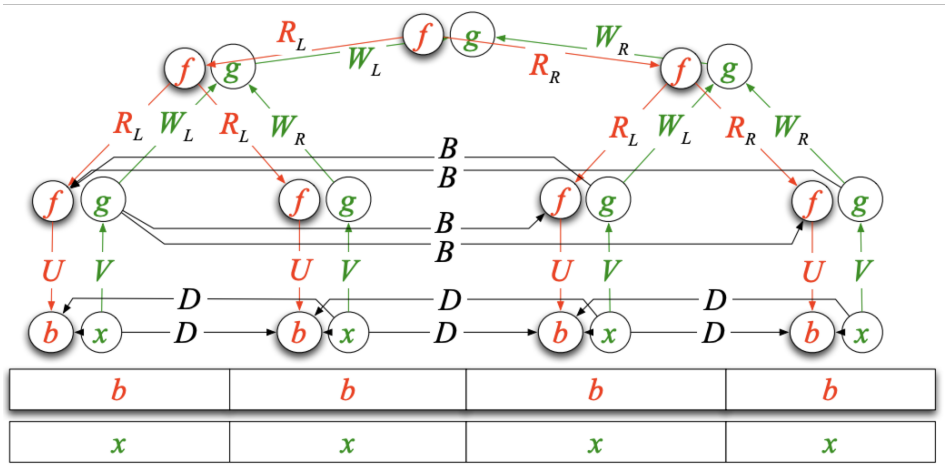

Figure 6: Computational graph of FMM matrix-vector product.

Model	**Rational**		**ReLU**		**GELU**	
	Rel Error	Residual Error	Rel Error	Residual Error	Rel Error	Residual Error
FNO	9.9469e-01	6.5013e-03	**9.9754e-01**	1.0742e-02	1.0053e+00	**2.2008e-03**
DeepONet	1.0626e+00	4.3643e-03	1.0006e+00	4.0503e-03	1.0481e+00	7.4626e-03
MIONet	1.0317e+00	8.6286e-03	2.8665e+01	6.1070e-03	1.3135e+01	5.3136e-03
MNO	**2.7362e-02**	**5.7059e-04**	9.9998e-01	**2.3622e-03**	1.8191e+02	5.2086e+00

Table 7: Comparison of different activation functions.

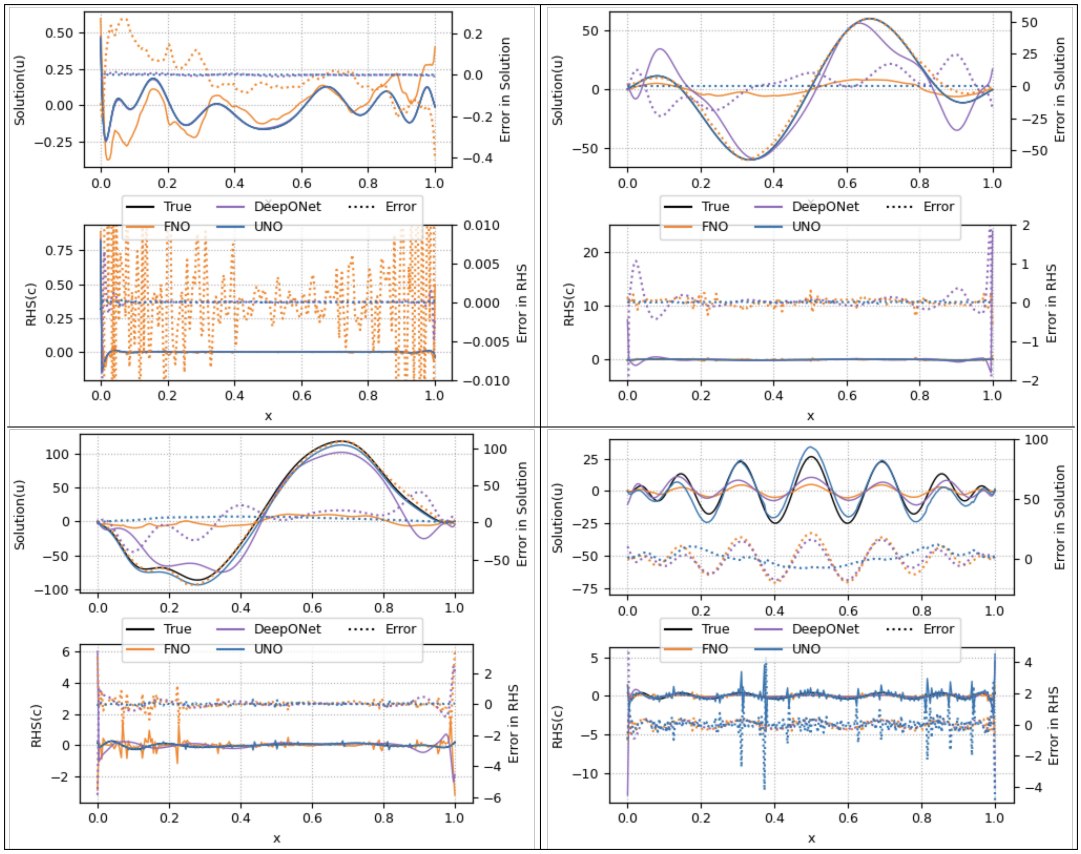

Figure 7: Examples of UNO, FNO and DeepONet evaluated on out-of-distribution 1D Poisson data. Top left sample is generated using solution sampling with random linear combinations of Chebyshev bases. Rest of the plots show samples generated using RHS sampling.

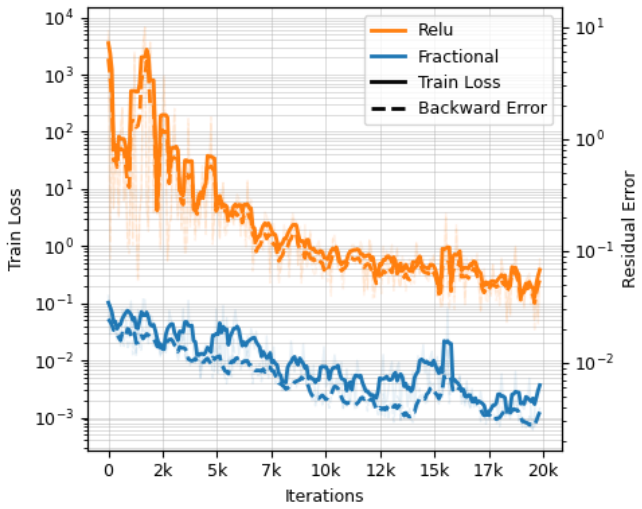

Figure 8: Training MSE loss and residual error of 1D Darcy flow MNO network with relu vs nonlinear rational activation function.

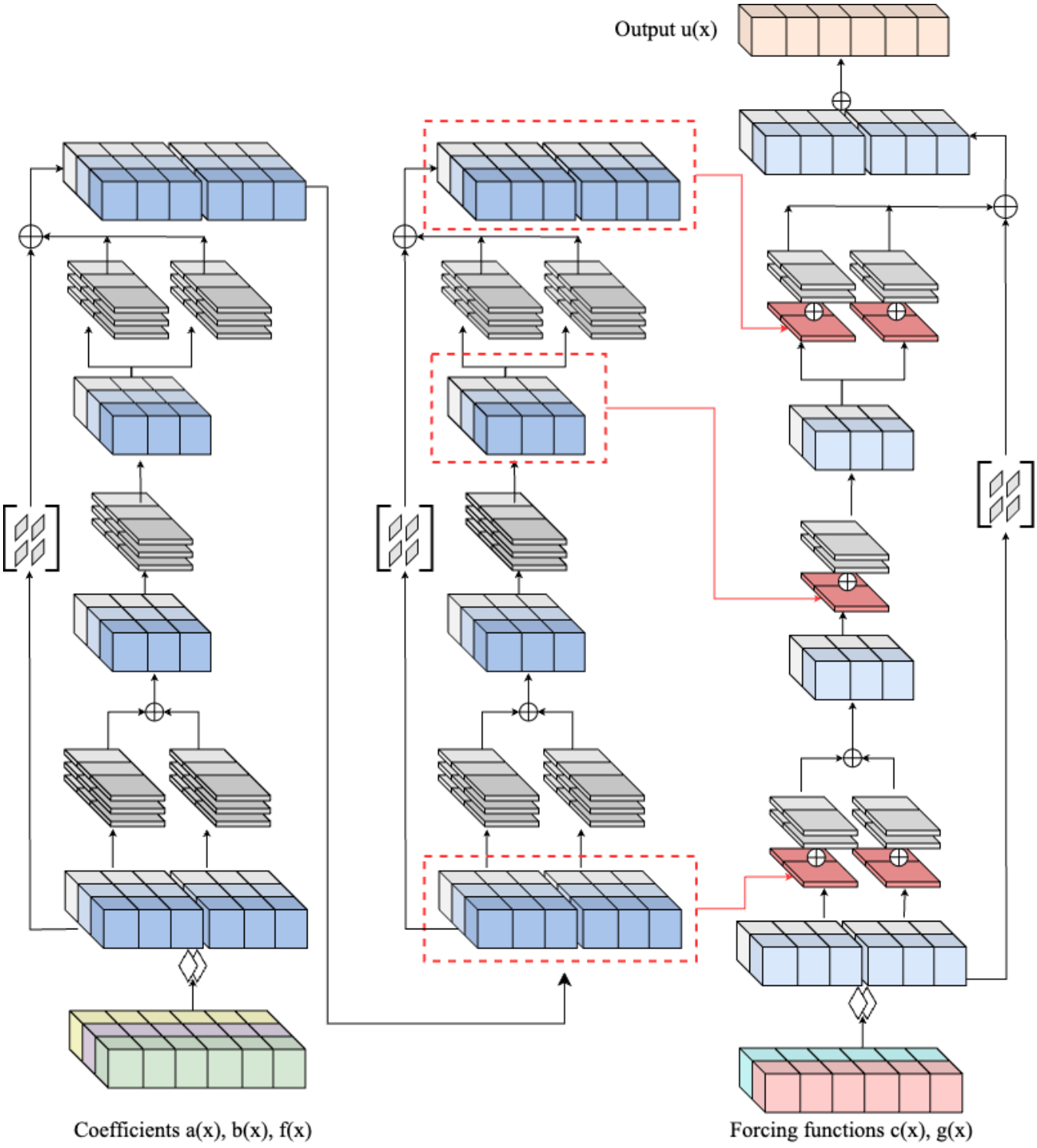

Figure 9: MNO with two GFMM-blocks in the coefficient branch and one GFMM-block in the RHS-branch showing multi-modal fusion between the two branches.

DeepONet performs better than FNO on this type of examples. In the rest of the plots, examples are generated using RHS sampling. For the top right and bottom right plots, c is chosen as randomly scaled $T_k(x)$ where $k = 4$ for top right and $k = 16$ for bottom right plot and groundtruth u is correspondingly derived. For the bottom left plot, c is chosen as a random linear combination from $T_1(x)$ to $T_{16}(x)$. For all the RHS-sampled examples, every model struggles to approximate the solution, but UNO performs better with lower residual error compared to FNO and DeepONet.

E 2D GFMM-block

E.1 Architecture

A 2D FMM computational graph is equivalent to the tensor product of a 1D FMM shown in Figure 6. Analogously, we define the 2D GFMM-block as the tensor-product of 1D GFMM-block. A schematic of the 2D GFMM-block architecture is shown in Figure 10. For the simplest case of single channel 2D input grid, the encoder and decoder graphs are quadtrees as compared to the binary trees of the 1D FMM. Each encoder node comprises of four bi-linear transformation layers that apply a *tensor contraction* operation on the input blocks and add the result into the output block. Tensor contraction is defined as AXB^T, where A and B are the linear weights and X is the input block.

The first encoder layer's input blocks are block matrices of the input 2D grid array in a fixed order. In Figure 10, we used Z-ordering (or Morton ordering) to divide the 2D input matrix into blocks. The rest of the encoder nodes follow the similar quadtree structure shown. Similarly, at every decoder layer, each input block undergoes four bi-linear transformations (tensor contraction) and the output is added to the output of bridge operator, which performs the tensor contraction on the corresponding encoder output. Finally, the output is obtained by adding the output of last decoder layer with the transformed input through the outermost bridge operator. For the bridge operators, the linear weights are fixed to be block banded matrices (block tri-diagonal in our experiments). Similar to 1D GFMM-block, we only store the blocks on the banded diagonals for memory efficiency. Additional non-linear activations can be applied after each bi-linear transformation of the encoder and decoder layers. In the next section, we show empirically that a 2D GFMM-block thus constructed can approximate the inverse operator of a discrete 2D Poisson's equation.

E.2 2D Discrete Poisson

Similar to section 3.2, we consider the linear equation $Au = c$, but here A is the 2D discrete Laplacian operator. We train the 2D GFMM-block to learn the inverse operator of A using solution sampling of random linear combinations of Chebyshev polynomials on a 128×128 uniform grid. We initialize the 2D GFMM-block with four encoder and decoder layers and a block size of 8 and trained for 200k iterations. Figure 11 shows the results on randomly sampled examples from 2D Chebyshev grid where the 2D GFMM-block network is able to approximate the inverse operator on this simple setup. It obtains a mean relative error of (4.86$\pm$0.3)e-07 and a normalized backward error of (6.48$\pm$0.005)e-05 on a random test set of 1000 examples.

Given the encouraging results, extending the 2D GFMM-block architecture with multi-modal fusion and a careful analysis of its generalization performance is the subject of our future work.

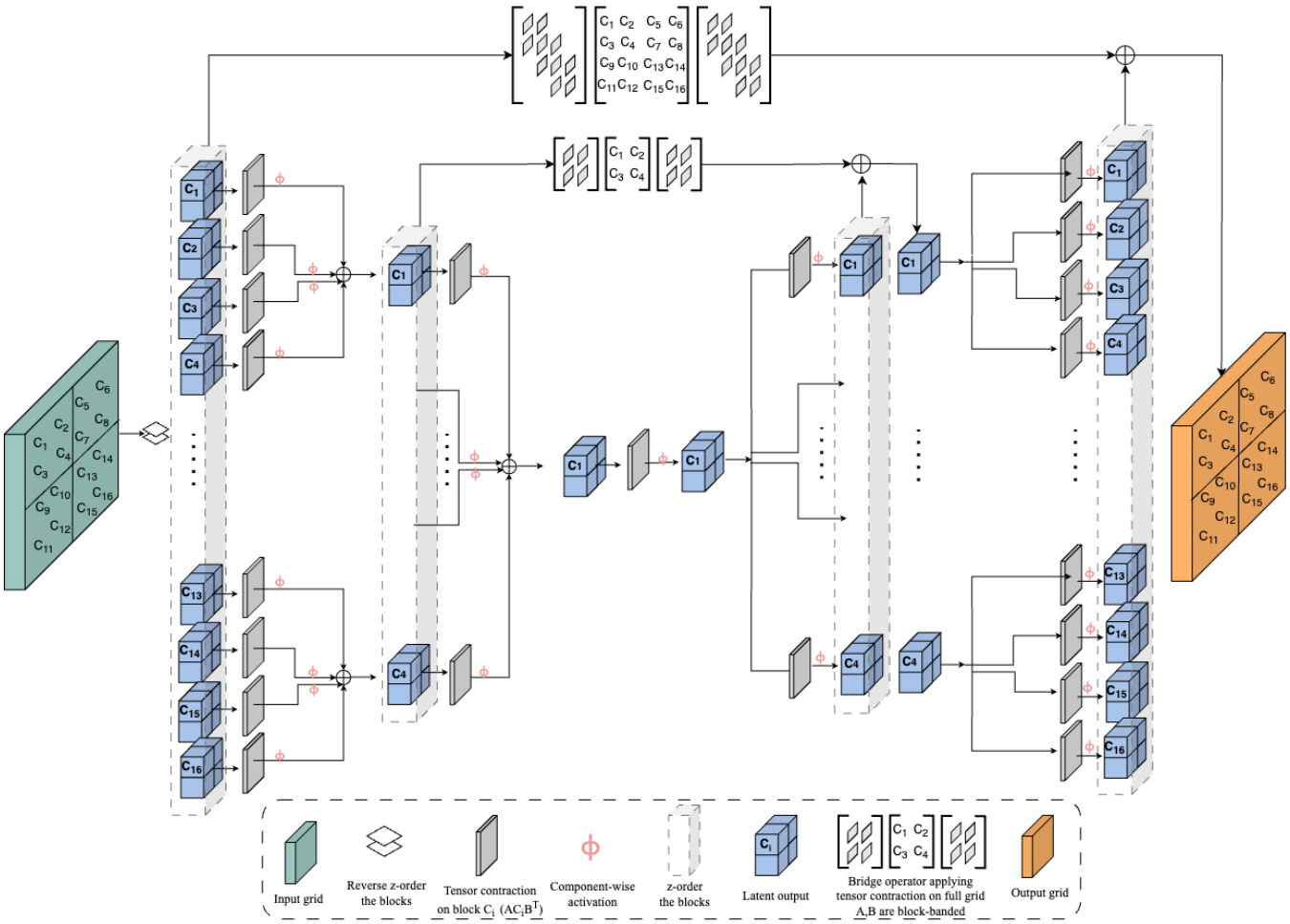

Figure 10: Schematic of a 2D GFMM block.

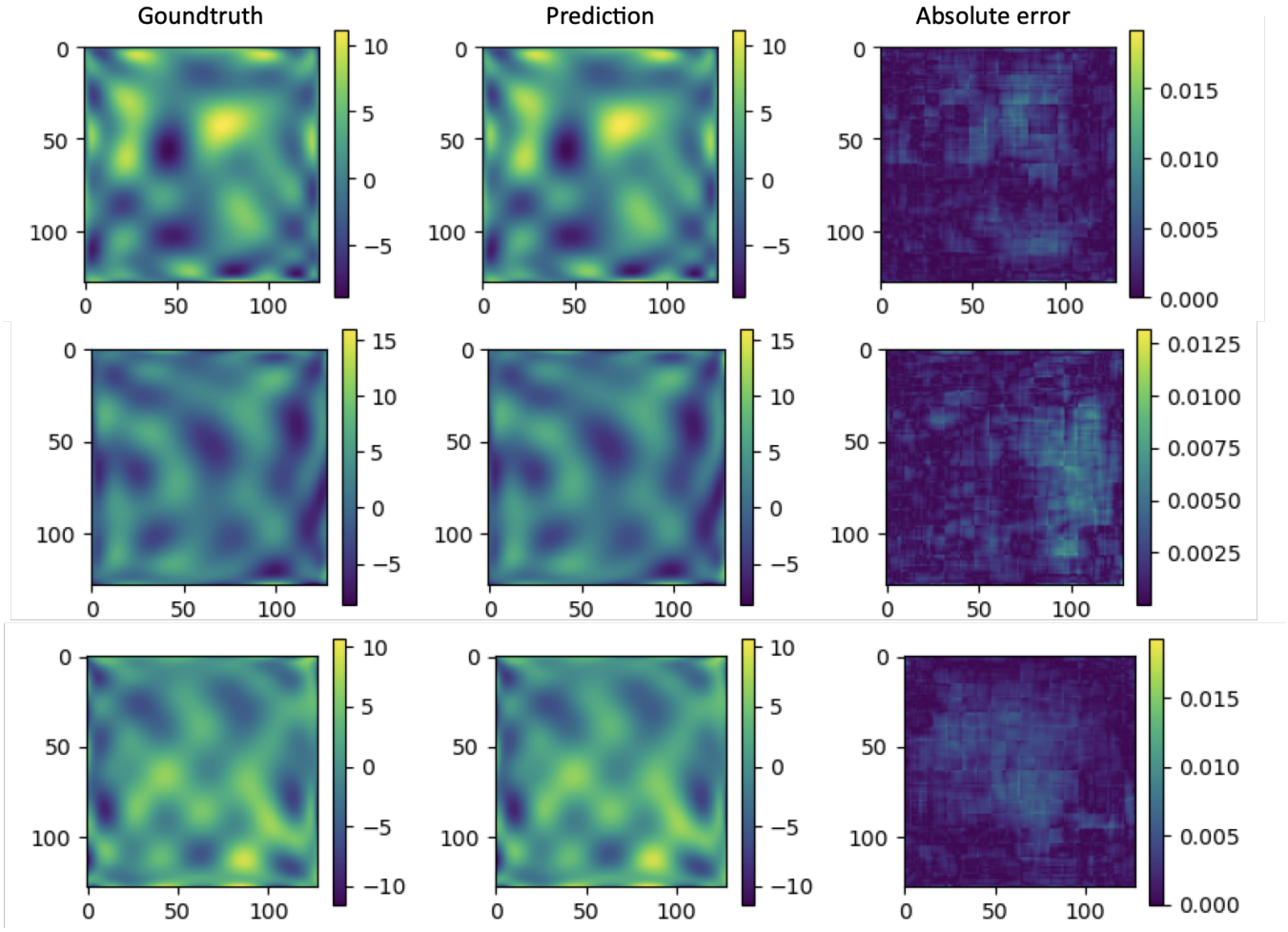

Figure 11: Groundtruth solution and the predicted output of the 2D GFMM-block trained on 2D Poisson's equation.