LEARNING OPERATIONS FOR NEURAL PDE SOLVERS

Anonymous authors
Paper under double-blind review

ABSTRACT

In recent years neural networks have been identified as candidates for fast partial differential equation (PDE) solvers, leading to numerous approaches to designing network layers that attempt to go beyond the usual convolution-based way of working with multi-dimensional data. We take a neural architecture search (NAS) inspired approach, introducing a relaxed space of neural operations that contains numerous types of layers including multi-channel convolutions and the recent state-of-the-art Fourier Neural Operators (FNOs) of Li et al. (2021b). To do so we exploit the fact that many operations of interest are linear maps diagonalized by efficient matrices such as the discrete Fourier transform (DFT); allowing these matrices to vary as additional "architecture parameters" of the model yields our **Expressive Diagonalization** (XD) operations. Empirically, we demonstrate that XD-operations outperform FNOs using a similar network across multiple problem settings while using fewer parameters in the case of 2d and 3d domains.

1 INTRODUCTION

PDE solvers are used for a broad range of scientific and engineering problems that require a large number of solver evaluations at different initial conditions with a high discretization resolution, which can be slow when using traditional numerical methods. Deep neural network-based PDE solvers have emerged as orders-of-magnitude faster alternatives to traditional numerical solvers trained to map initial conditions to solutions using datasets of solutions generated by numerical solvers. However, baseline neural approaches such as convolutional neural networks (CNNs) perform poorly, which has led to the development of better parameterized layers specialized for this problem (Li et al., 2021b).

In this paper we approach the design of such operations from the perspective of neural architecture search (NAS). While we leave a thorough overview of the field to a survey (Elsken et al., 2019), the principle task of a NAS algorithm is to assign operations to edges in a computational graph to define a neural network. Usually this involves choosing one of a small number of operations, e.g. different types of convolution or pooling layers, for each edge. In recent years, NAS algorithms have relaxed this discrete search problem into a continuous one in which "architecture parameters" interpolating between different operations are updated using gradient-based methods (Liu et al., 2019).

We propose a different type of relaxation between operations based around the key insight that many operations consist of multiple channels that apply weights $\mathbf{w}$ to inputs $\mathbf{x}$ by computing

$$\mathbf{K}\,\mathrm{diag}(\mathbf{L}\mathbf{w})\mathbf{M}\mathbf{x} \tag{1}$$

where the fixed matrices $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$ are *efficient* (to represent and apply) and shared across channels. In particular, setting $\mathbf{L}$ and $\mathbf{M}$ to be the DFT and $\mathbf{K}$ to be its inverse yields the convolution, while doing the same except setting $\mathbf{L}$ to be the bit-reversal permutation instead results in the FNO. Thus all we need to interpolate between them is a continuous family of efficient matrices containing both permutations and DFTs, for which we use the *Kaleidoscope* or *K-matrix* family of Dao et al. (2020). Replacing all three matrices above by K-matrices yields our **Expressive Diagonalization** (XD) operations, which we show enjoy the following theoretical and empirical properties:

- XD-operations can be represented and applied in time log-linear in the input size.
- The set of XD-operations contains several operations of interest for neural PDE-solvers: convolutions, FNOs, and graph convolutions (GCNs) (Kipf & Welling, 2017). This is achieved by setting the architecture parameters $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$ appropriately, which means we can initialize or "warm-start" the optimization of our model using one of these operations. This also means XD-operations can be substituted into any standard CNN backbone with minimal tuning.

- Using the experimental setup and network topology of Li et al. (2021b), we show that XD-operations warm-started with convolutions achieve lower loss than their FNO operations on three different problems—Burgers' equation, Darcy Flow, and Navier-Stokes—each with a different input dimension. While our computational cost is higher, in most settings we use fewer parameters.
- We find that warm-starting with FNOs also outperforms their result but by a smaller margin, suggesting that XD-operations learn fundamentally distinct operations. On the other hand, XD-operations do share the FNO property of performing consistently well across resolutions.

Code implementing XD-operations and reproducing these results will be made publicly available. We relegate related work and experimental details to the supplementary material.

2 The Expressive Diagonalization Relaxation

CNNs are often the first model tried by scientists faced with multi-dimensional arrays over a regular grid (LeCun et al., 1999), including both images and the initial conditions of a PDE. However, in the latter case their performance as a PDE solver has been found to be subpar, even over rectangular meshes, leading to the design of better-performing operations (Li et al., 2021b). The main contribution of this work is the construction of a set of operations called XD-operations that generalize the convolution and can be used to obtain high-quality models for this problem. While for brevity we do not expand significantly upon the connection to architecture search, for notational convenience we will formally call an *operation* any mapping $\mathbf{Op}_\alpha : \mathcal{W} \mapsto \mathcal{F}$ from a parameter space $\mathcal{W}$ to a space $\mathcal{F} = \{\mathbf{Op}_a(\mathbf{w}) : \mathcal{X} \mapsto \mathcal{Y}\}$ of parameterized function from input space $\mathcal{X}$ to output space $\mathcal{Y}$. Here α corresponds to an *architecture parameter* specifying how to apply the weights $\mathbf{w}$ to the input; in the simplest case it could just be binary indicator saying e.g. to use convolution if $\alpha = 1$ and to use average pooling otherwise. Modern architecture search methods often treat architecture parameters as simply additional model parameters that require special treatment in terms of optimization (Li et al., 2021a), which is also the approach we will take.

The starting point of our construction of XD-operations is the convolution's efficient diagonalization by the DFT. Formally, if $\mathbf{A_w} \in \mathbb{R}^{n^2 \times n^2}$ is the matrix representing a 2d convolution with filter $\mathbf{w} \in \mathbb{R}^{\mathbf{k}}$ of kernel size $\mathbf{k} \in [n]^2$, then for any 2d input $\mathbf{x} \in \mathbb{R}^{n^2}$ we have

$$\mathbf{Conv}(\mathbf{w})(\mathbf{x}) = \mathbf{A_w x} = \mathbf{F}^{-1} \operatorname{diag}(\mathbf{F}\underline{\mathbf{w}})\, \mathbf{F x} \tag{2}$$

Here $[n] = \{1, \dots, n\}$, $\operatorname{diag}(\mathbf{z})$ denotes the diagonal matrix with nonzero entries $\mathbf{z}$, $\underline{\mathbf{w}} \in \mathbb{R}^{n^2}$ is an appropriate zero-padding of $\mathbf{w} \in \mathbb{R}^{\mathbf{k}}$, and $\mathbf{F} \in \mathbb{C}^{n^2 \times n^2}$ is the 2d DFT (a Kronecker product of two 1d DFTs). This diagonalization explicates both the convolution, as both the DFT and its inverse can be applied in time $\mathcal{O}(n \log n)$ and represented with $\mathcal{O}(n \log n)$ bits. It also suggests a natural way to generalize the convolution while preserving these efficiencies: simply replace the Fourier matrices $\mathbf{F}$ and $\mathbf{F}^{-1}$ in equation 2 by a more general family of efficient matrices. Doing so yields the single-channel version of our XD-operations:

$$\mathbf{XD}^{\mathbf{1}}_\alpha(\mathbf{w})(\mathbf{x}) = \operatorname{Real}(\mathbf{K} \operatorname{diag}(\mathbf{L}\underline{\mathbf{w}})\, \mathbf{M x}) \tag{3}$$

Here the architecture parameter $\alpha = (\mathbf{K}, \mathbf{L}, \mathbf{M})$ determines the exact matrices used to replace $\mathbf{F}$ and $\mathbf{F}^{-1}$ in Equation 2. As in convolutions, the multi-channel extension of this is straightforward.

The main remaining question is the family of efficient matrices to use, i.e. the domain of the architecture parameters $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$. For this we turn to the Kaleidoscope matrices, or *K-matrices*, of Dao et al. (2020), which generalize $\mathbf{F}$ and $\mathbf{F}^{-1}$ to include all computationally efficient linear transforms with short description length. This includes important examples, such as sparse matrices and permutations, that add significant expressivity to the DFT. To obtain this general family, K-matrices allow the DFT's butterfly factors—matrices whose products yield its efficient implementation—to take on different values. While a detailed construction of K-matrices can be found in the original paper, we need only the following useful properties: they are as (asymptotically) efficient to apply and represent as DFTs, they are differentiable and can thus be updated using gradient-based methods, and they can be composed (made "deeper") to make more expressive K-matrices.

Specifying that $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$ in Equation 3 are K-matrices largely completes our core contribution: a set of operations with K-matrix architecture parameters. We now describe two key properties of these XD-operations, as well as how we apply them in practice.

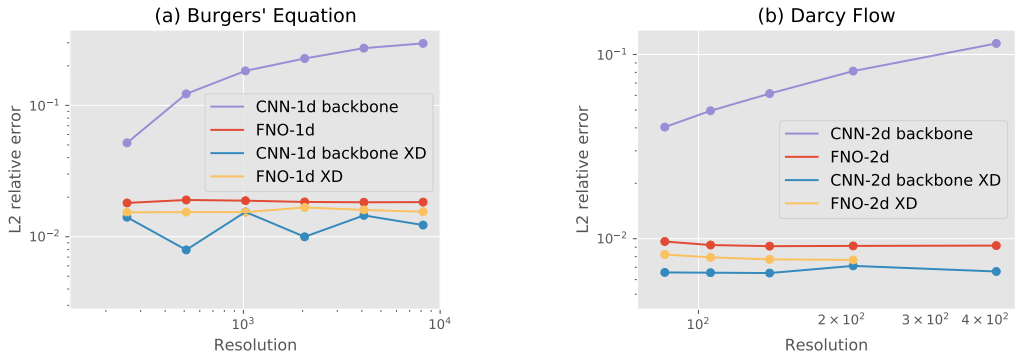

Figure 1: **(a)** and **(b)** Relative error on Burgers' equation and Darcy Flow across different resolutions, including all those considered by Li et al. (2021b).

2.1 Properties of XD-Operations

Complexity: $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ requires three K-matrices and $\mathcal{O}(1)$ filter weights to represent, which implies description length $\mathcal{O}(n \log n)$; this is more than a convolution (which has no architecture parameters) but is not quadratic like a linear layer. Applying $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ requires multiplication by three K-matrices, yielding a per-channel time complexity of $\mathcal{O}(n \log n)$. This matches the efficiency of convolutions.

Expressivity: a key aspect of XD-operations is that, for specific settings of the architecture parameters, they contain several operations of interest for solving PDEs: convolutions, FNOs (Li et al., 2021b), and GCNs (Kipf & Welling, 2017). As discussed above, convolutions can simply be obtained by setting $\mathbf{L}$ and $\mathbf{M}$ to be the DFT matrix and $\mathbf{K}$ to be its inverse. If we allow the model weights $\mathcal{W}$ to be complex, FNOs are also straightforward; simply set $\mathbf{K}$ and $\mathbf{M}$ as for convolutions and $\mathbf{L}$ to be the bit-reversal permutation,[1] which is an efficient operation contained in the set of K-matrices. Perhaps the most interesting result is for GCNs, which consist of multiplication by a sparse matrix $\mathbf{A}$ derived from the adjacency matrix followed by a pointwise convolution;[2] thus to express it as an XD-operation we can set $\mathbf{K}$ and $\mathbf{L}$ as for convolutions and $\mathbf{M}$ to be the DFT multiplied by $\mathbf{A}$.

2.2 Applying XD-Operations

Warm-starting: the major benefit of the expressivity of XD-operations is that we can initialize, or *warm-start*, our optimization procedure using these named operations; in particular, by setting the architecture parameters $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$ appropriately and using the same random initialization for model weights we can use any existing CNN backbone as a starting network. In this work we will specifically use the ResNet-like architecture of Li et al. (2021b).

Optimization: as discussed previously, we view architecture parameters α as additional model parameters that may require separate treatment during training; specifically we tune the step-size and momentum of stochastic gradient descent. Meanwhile, to train the model weights $\mathbf{w}$ we simply use the same optimizer as Li et al. (2021b) with no further tuning. These algorithms are run simultaneously as a single optimization routine. This simple procedure distinguishes our work from standard architecture search, with its more involved multi-stage search protocols (Yang et al., 2020).

3 Empirical Results

For our empirical evaluation we evaluate using the setting of Li et al. (2021b), data generated by classical PDE solvers is used to learn functions from some initial condition or setting to the corresponding PDE solution, with the goal of replacing the solver by a deep net forward pass; the latter can be orders of magnitude faster. The three PDEs they study are the Burgers' equation, Darcy Flow, and the 2d Navier-Stokes equations, which involve 1d, 2d, and 3d data, respectively. The first two are studied across multiple resolutions, while the last one is studied at different viscosities.

[1] We view the DFT as *not* including bit-reversal; if it does then we can simply set $\mathbf{L}$ to be the identity.

[2] Note that our results only holds for fixed graphs, as in the case of PDE solvers over fixed meshes.

Table 1: Relative test error on the 2d Navier-Stokes equations at two different settings of the viscosity ν and number of time steps T. Best results in each setting are **bolded**.

Method	$\nu = 10^{-4}$ $T = 30$	$\nu = 10^{-5}$ $T = 20$
CNN-3d backbone	0.325	0.278
FNO-3d (reproduced)	0.182	0.177
CNN-3d backbone XD	**0.172**	**0.168**

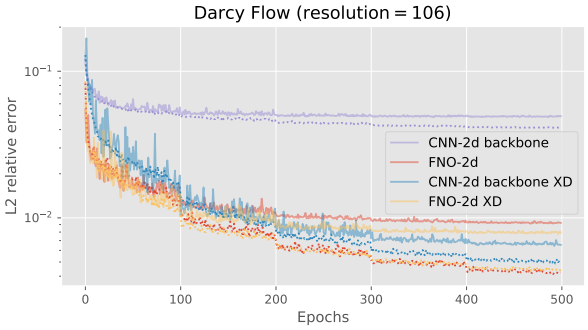

Figure 2: Training curves (dotted) and test curves (solid) on Darcy Flow.

Our first approach here is to take a simple CNN backbone—the type a scientist might use in a first attempt at a solution—and replace all convolutions by XD-operations. As a baseline, we hope to do better than this backbone; ambitiously (since we are initializing with the under-performing convolution), we hope to compete with the custom-designed FNO. The specific CNN we use is simply the FNO architecture of the appropriate dimension N from Li et al. (2021b) but with all N-dimensional FNOs replaced by N-dimensional convolutions; this performs similarly to their CNN baselines. We further compare these to the performance of XD initialized as the FNO-1d or FNO-2d architectures for the respective 1d or 2d problems. In all cases we only compare to the CNN backbone and our reproduction of the FNO results, as the latter exceeds all other neural methods; a complete results table is provided in the appendix. Our reproduction of FNO is slightly worse than their reported numbers for Burgers' equation and slightly better in the other two settings. Lastly, note that on the Navier-Stokes equations we only compare to the 3d FNO on the two settings in which we were able to reproduce their approach; moreover, we do *not* compare to their use of a 2d FNO plus a recurrent net in time, but in-principle XD-operations can also be used for the latter.

We report our results for the Burger's equation and Darcy Flow in Figure 1; for 2d Navier-Stokes the results are in Table 1. In all cases XD-operations dramatically outperform the N-dimensional CNN backbone used to warm-start them; furthermore, we also achieve better error than the FNO. As exemplified in Figure 2, despite having a higher train loss than FNO, XD-operations seem to generalize better. Notably, Figure 1 also shows that XD-operations perform consistently well across different resolutions, a major advantage of FNOs over previous methods, whose performance was tightly coupled to the discretization (Li et al., 2021b). In particular, note that CNN performance worsens with increasing resolution, whereas that of XD and FNO does not.

As a further investigation, we attempt to warm-start XD-operations using exactly the network and operations of Li et al. (2021b); this is made possible by our expressivity results. Perhaps surprisingly, Figure 1 shows that while FNO-initialized XD-operations indeed outperform FNO, CNN-initialized XD-operations outperform FNO by an even larger margin, despite the poor performance of convolutions without XD. This suggests the operations learned when warm-starting with a CNN is substantially different than FNO, which itself means that it may be fruitful to combine the two.

4 Conclusion

Our work demonstrates that our new class of XD-operations yields accurate PDE solvers in several settings and opens up several directions for future work. In particular, our results do come at a cost of slower training and inference; XD-operations are roughly an order of magnitude slower than FNOs, despite having fewer parameters in 2d and 3d. However, this still yields approximate PDE solvers that are one-to-two orders of magnitude faster than classical approaches, maintaining the usefulness of these results for the problem. We have identified several directions for ameliorating this, including a smarter treatment of padding $\mathbf{w}$ and investigating the usefulness of varying individual architecture parameters. Other future work involves investigating how XD can handle other desirable aspects such as transfer to different grids and adaptivity to irregular meshes; here combining or warm-starting with FNOs or GCNs may prove useful.

REFERENCES

Nir Ailon, Omer Leibovich, and Vineet Nair. Sparse linear networks with a fixed butterfly structure: Theory and practice. arXiv, 2020.

Keivan Alizadeh vahid, Anish Prabhu, Ali Farhadi, and Mohammad Rastegari. Butterfly transform: An efficient FFT based neural architecture design. In *Proceedings of the IEEE Conference on Conference on Computer Vision and Pattern Recognition*, 2020.

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*. 1999.

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the 9th International Conference on Learning Representations*, 2021a. To Appear.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations, 2020b.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *Proceedings of the 9th International Conference on Learning Representations*, 2021b. To Appear.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. NAS evaluation is frustratingly hard. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.04.018. URL `https://www.sciencedirect.com/science/article/pii/S0021999118302341`.

A Related Work

Many deep neural network architectures used in neural PDE solvers include convolutional layers, which outperform architectures comprising only dense layers, but perform worse as the resolution of the generated PDE solutions is increased (Zhu & Zabaras, 2018; Li et al., 2021b). Other approaches based on graph neural networks have been proposed (Li et al., 2020b;a). The recently introduced Fourier Neural Operator (FNO) of Li et al. (2021b) achieves state-of-the-art error among neural PDE solvers without suffering from drops in performance at different resolutions.

Several papers have generalized the DFT to replace layers in deep networks (Dao et al., 2019; Alizadeh vahid et al., 2020; Ailon et al., 2020), including the one whose K-matrices we apply Dao et al. (2020). These works focus on replacing *linear* layers in deep nets to speed up or add structure to models while *reducing* expressivity. In contrast, we can replace *convolutions* and other types of layers while *increasing* expressivity by extending their diagonalization using K-matrices. Note that K-matrices on their own can represent all functions returned by XD-operations, as they contain all efficient linear transforms, However, if both the input dimension N and filter size are > 1, as in most applications, then the only known way is to apply K-matrices directly to flattened inputs $\mathbf{x} \in \mathbb{R}^{n^N}$, yielding much worse description length $\mathcal{O}(n^N \log n)$. In contrast, our diagonalization approach allows the use of Kronecker products to apply DFTs to each dimension separately, yielding description length $\mathcal{O}(n \log n)$. It is thus the first (and in some sense, "right") method to use such matrices as replacements for convolutions. Furthermore, diagonalization allows us to separate model weights $\mathbf{w}$ from architecture parameters α, letting the former vary across channels while fixing the latter.

B Experimental Details

Table 2: Architecture optimizer settings on PDE tasks with CNN and FNO initializations. Note that the learning rate is updated using the same schedule as the backbone.

task	XD init	optimizer	initial lr	warmup
1d Burgers' equation	CNN	Adam	1E-3	0
1d Burgers' equation	FNO	Momentum(0.5)	1E-4	250
2d Darcy Flow	CNN	Momentum(0.5)	1E-1	0
2d Darcy Flow	FNO	Momentum(0.5)	1E-1	0
2d Navier Stokes ($\nu = 10^{-4}, T = 30$)	CNN	Momentum(0.5)	5E-3	0
2d Navier Stokes ($\nu = 10^{-5,} T = 20$)	CNN	Momentum(0.5)	1E-3	0

For our experiments, we use the code and setup from Li et al. (2021b) provided here: `https://github.com/zongyi-li/fourier_neural_operator`. We use the same training routine and settings as the backbone architecture for each task and only tune the architecture optimizer. We consider the following hyperparameters for the architecture optimizer: Adam vs. SGD (with or without momentum), initial learning rate, and number of warmup epochs. Our CNN backbone is analogous to the FNO architecture used for each problem. In particular, the CNN backbone architecture used for each task is simply the FNO architecture where FNO layers of dimension N with m modes are replaced by N-dimensional convolutional layers with filters of size $(m+1)^N$ and circular padding to match the dimensionality of FNO.

Table 3: Test relative errors on the 1d Burgers' equation. We were not able to match the FNO-1d results reported by Li et al. (2021b) using their published codebase, however, our proposed XD operations outperform our reproduction of their results at every resolution. Furthermore, we outperform their reported test relative errors on every resolution except $s = 4096$, where we roughly match their performance.

Method (source)	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
NN (Li et al., 2021b)	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN (Li et al., 2021b)	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN (Li et al., 2021b)	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN (Li et al., 2021b)	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
GNO (Li et al., 2021b)	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO (Li et al., 2021b)	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO (Li et al., 2021b)	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO-1d (Li et al., 2021b)	0.0149	0.0158	0.0160	0.0146	**0.0142**	0.0139
CNN backbone (ours)	0.0518	0.1220	0.1830	0.2280	0.2730	0.2970
FNO-1d (reproduced)	0.0181	0.0191	0.0188	0.0184	0.0183	0.0183
CNN backbone XD (ours)	**0.0141**	**0.0079**	**0.0154**	**0.0099**	0.0145	**0.0123**
FNO-1d XD (ours)	0.0153	0.0154	0.0154	0.0167	0.0160	0.0155

Table 4: Test relative errors on 2d Darcy Flow. Our reproduction of the FNO-2d results outperform those reported by Li et al. (2021b). Nonetheless, our proposed XD operations outperform both our reproduction and the reported results of Li et al. (2021b) at every resolution.

Method (source)	$s = 85$	$s = 106$	$s = 141$	$s = 211$	$s = 421$
NN (Li et al., 2021b)	0.1716	-	0.1716	0.1716	0.1716
GCN (Li et al., 2021b)	0.0253	-	0.0493	0.0727	0.1097
FCN (Li et al., 2021b)	0.0299	-	0.0298	0.0298	0.0299
PCANN (Li et al., 2021b)	0.0244	-	0.0251	0.0255	0.0259
GNO (Li et al., 2021b)	0.0346	-	0.0332	0.0342	0.0369
LNO (Li et al., 2021b)	0.0520	-	0.0461	0.0445	-
MGNO (Li et al., 2021b)	0.0416	-	0.0428	0.0428	0.0420
FNO-2d (Li et al., 2021b)	0.0108	-	0.0109	0.0109	0.0098
CNN backbone (ours)	0.0404	0.0495	0.0613	0.0813	0.1150
FNO-2d (reproduced)	0.0096	0.0092	0.0091	0.0091	0.0091
CNN backbone XD (ours)	**0.0065**	**0.0065**	**0.0065**	**0.0071**	**0.0066**
FNO-2d XD (ours)	0.0082	0.0079	0.0077	0.0076	-