Multi-Grid Tensorized Fourier Neural Operator for High-Resolution PDEs

Anonymous authors
Paper under double-blind review

Abstract

Memory complexity and data scarcity have so far prohibited learning solution operators of partial differential equations (*PDE*) at high resolutions. We address these limitations by introducing a new data efficient and highly parallelizable operator learning approach with reduced memory requirement and better generalization, called multi-grid tensorized neural operator (*MG-TFNO*). *MG-TFNO* scales to large resolutions by leveraging local and global structures of full-scale, real-world phenomena, through a decomposition of both the input domain and the operator's parameter space. Our contributions are threefold: i) we enable parallelization over input samples with a novel multi-grid-based domain decomposition, ii) we represent the parameters of the model in a high-order latent subspace of the Fourier domain, through a global tensor factorization, resulting in an extreme reduction in the number of parameters and improved generalization, and iii) we propose architectural improvements to the backbone *FNO*. Our approach can be used in any operator learning setting. We demonstrate superior performance on the turbulent Navier-Stokes equations where we achieve less than half the error with over 150× compression. The tensorization combined with the domain decomposition, yields over 150× reduction in the number of parameters and 7× reduction in the domain size without losses in accuracy, while slightly enabling parallelism.

1 Introduction

Real-world scientific computing problems often time require repeatedly solving large-scale and high-resolution partial differential equations (*PDE*s). For instance, in weather forecasts, large systems of differential equations are solved to forecast the future state of the weather. Due to internal inherent and aleatoric uncertainties, multiple repeated runs are carried out by meteorologists every day to quantify prediction uncertainties. Conventional *PDE* solvers constitute the mainstream approach used to tackle such computational problems. However, these methods are known to be slow and memory-intensive. They require an immense amount of computing power, are unable to learn and adapt based on observed data, and oftentimes require sophisticated tuning (Slingo & Palmer, 2011; Leutbecher & Palmer, 2008; Blanusa et al., 2022).

Neural operators are a new class of models that aim at tackling these challenging problems (Li et al., 2020a). They are maps between function spaces whose trained models emulate the solution operators of *PDE*s (Kovachki et al., 2021b). In the context of *PDE*s, these deep learning models are orders of magnitude faster than conventional solvers, can easily learn from data, can incorporate physically relevant information, and recently enabled solving problems deemed to be unsolvable with the current state of available *PDE* methodologies (Liu et al., 2022; Li et al., 2021c). Among neural operator models, Fourier neural operators (*FNO*s), in particular, have seen successful application in scientific computing for the task of learning the solution operator to *PDE*s as well as in computer vision for classification, in-painting, and segmentation (Li et al., 2021b; Kovachki et al., 2021a; Guibas et al., 2021). By leveraging spectral theory, *FNO*s have successfully advanced frontiers in weather forecasts, carbon storage, and seismology (Pathak et al., 2022; Wen et al., 2022; Yang et al., 2021).

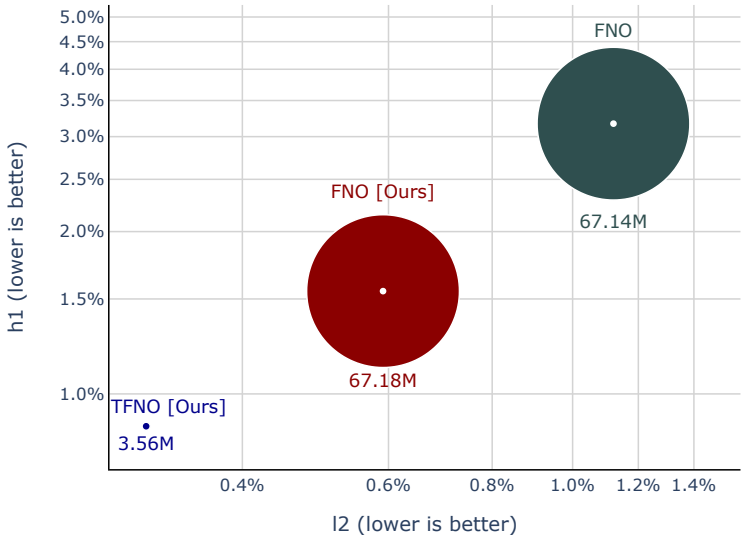


Figure 1: **Comparison of the performance on the relative L^2 and H^1 test errors (lower is better) on a log-scale** of our approach, compared with both our improved backbone (*FNO*) and the original FNO, on Navier-Stokes. Our approach enables large compression for both input and parameter, while outperforming regular *FNO*.

While *FNO*s have shown tremendous speed-up over classical numerical methods, their efficacy can be limited due to the rapid growth in memory needed to represent complex operators. In the worst case, large memory complexity is required and, in fact, is unavoidable due to the need for resolving fine-scale features globally. However, many real-world problems, possess a local structure not currently exploited by neural operator methods. For instance, consider a weather forecast where predictions for the next hour are heavily dependent on the weather conditions in local regions and minimally on global weather conditions. Incorporating and learning this local structure of the underlying *PDE*s is the key to overcoming the curse of memory complexity.

In this work, we propose a new, scalable neural operator that addresses these issues by leveraging the structure in both the domain space and the parameter space, Figure 2. Specifically, we introduce the multi-grid tensor operator (*MG-TFNO*), a model that exploits locality in physical space by a novel multi-grid domain decomposition approach to compress the input domain size by up to 7× while leveraging the global interactions of the model parameters to compress them by over 100× without any loss of accuracy.

In the input space, to predict the solution in any region of the domain, *MG-TFNO* decomposes the input domain into small local regions to which hierarchical levels of global information are added in a multi-grid fashion. Since a local prediction depends most strongly on its immediate spatial surroundings, the farther field information is downsampled to lower resolutions, progressively, based on its distance from the region of interest. Thus, *MG-TFNO* allows parallelization over the input domain as it relies on high-resolution data only locally and coarse-resolution data globally. Due to its state-of-the-art performance on *PDE* problems and efficient FFT-based implementation, we use the *FNO* as the backbone architecture for our method. It is worth noting that the multi-grid approach is readily amendable to neural network settings and, moreover, any other neural operator architecture can be used in place of *FNO* as a backbone.

In the parameter space, we exploit the spatiotemporal structure of the underlying *PDE* solution operator by parameterizing the convolutional weights within the Fourier domain with a low-rank tensor factorization.

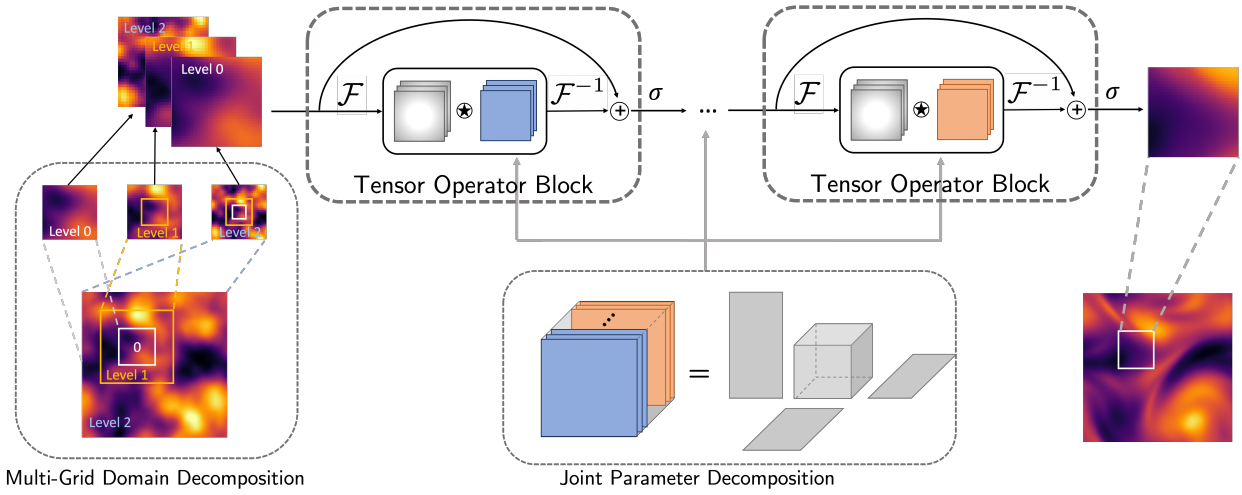


Figure 2: **Overview of our approach**. First (left), a multi-grid approach is used to create coarse to fine inputs that capture high-resolution details in a local region while still encoding global context. The resulting regions are fed to a tensorized Fourier operator (middle), the parameters of which are jointly represented in a single latent space via a low-rank tensor factorization (here, a Tucker form). Here $\mathcal{F}$ denotes Fourier transform. Finally, the outputs (right) are stitched back together to form the full result. Smoothness in the output is ensured via the choice of the loss function.

Specifically, we impose a coupling between all the weights in the Fourier space by jointly parameterizing them with a single tensor, learned in a factorized form such as Tucker or Canonical-Polyadic (Kolda & Bader, 2009). This coupling allows us to limit the number of parameters in the model without limiting its expressivity. On the contrary, this low-rank regularization on the model mitigates over-fitting and improves generalization. Intuitively, our method can be thought of as a fully-learned implicit scheme capable of converging in a small, fixed number of iterations. Due to the global nature of the integral kernel transform, the *FNO* avoids the Courant–Friedrichs–Lewy (CFL) condition plaguing explicit schemes, allowing convergence in only a few steps (Courant et al., 1928). Our weight coupling ensures maximum communication between the steps, mitigating possible redundancies in the learned kernels and imposing a low-rank regularizing of the network Panagakis et al. (2021).

In summary, we make the following contributions:

- **We propose architectural improvements to the backbone** which we validated through thorough ablations.
- **We propose** *MG-TFNO*, a novel neural operator parameterized in the spectral domain by a single low-rank factorized tensor, allowing its size to grow linearly with the size of the problem.
- **Our tensor operator achieves better performance with a fraction of the parameters**: we outperform *FNO* on solving the turbulent Navier Stokes equations with more than 400× weight compression ratio, Figure 7.
- **Our method overfits less and does better in the low-data regime**. In particular, it outperforms *FNO* with less than half the training samples, Figure 9.
- **We introduce a novel multi-grid domain decomposition approach**, a technique which allows the operator to predict the output only on local portions of the domain, thus reducing the memory usage by an order of magnitude with no performance degradation.
- **Combining tensorization with multi-grid domain decomposition leads to** *MG-TFNO*, which is more efficient in terms of task performance, computation, and memory. *MG-TFNO* achieves 2.5× lower error with 10× model weight compression, and 1.8× domain compression.

- **A unified codebase** to run all configurations and variations of *FNO* and *MG-TFNO* will be released, along with the Navier-Stokes data used in this paper, along with the paper.

2 Background

Here, we review related works and introduce the background necessary to explain our approach.

Many physical phenomena are governed by *PDE*s and a wide range of scientific and engineering computation problems are based on solving these equations. In recent years, a new perspective to *PDE*s dictates to formulate these problems as machine learning problems where solutions to *PDE*s are learned. Prior works mainly focused on using neural networks to train for the solution map of *PDE*s (Guo et al., 2016; Zhu & Zabaras, 2018; Adler & Oktem, 2017; Bhatnagar et al., 2019; Gupta et al., 2021). The use of neural networks in the prior works limits them to a fixed grid and narrows their applicability to *PDE*s where maps between function spaces are desirable. Multiple attempts have been made to address this limitation. For example mesh free methods are proposed that locally output mesh-free solution (Lu et al., 2019; Esmaeilzadeh et al., 2020), but they are still limited to fixed input gird.

A new deep learning paradigm, neural operators, are proposed as maps between function spaces (Li et al., 2020a; Kovachki et al., 2021b). They are discretization invariants maps. The input functions to neural operators can be presented in any discretization, mesh, resolution, or basis. The output functions can be evaluated at any point in the domain, either on a regular grid (e.g. FFT based architectures) or for any arbitrary geometry (e.g. graph based architectures). Variants of neural operators deploy a variety of Nyström approximation to develop new neural operator architecture. Among these, multi-pole neural operators (Li et al., 2020b) utilize the multi-pole approach to develop computationally efficient neural operator architecture. Inspired by the spectral method, Fourier-based neural operators show significant applicability in practical applications (Li et al., 2021b; Yang et al., 2021; Wen et al., 2022; Rahman et al., 2022a), and the architectures have been used in neural networks for vision and text tasks (Guibas et al., 2021; Dao et al., 2022). Principle component analysis and u-shaped methods are also considered (Bhattacharya et al., 2020; Liu et al., 2022; Rahman et al., 2022b; Yang et al., 2022). It is also shown that neural operators can be additionally trained using *PDE*s, resulting in physics-informed neural operators, opening new venues for hybrid data and equation methods (Li et al., 2021c) to tackle problems in scientific computing.

Decomposing the domain in smaller subdomains is at the core of many methods in computational sciences(Dryja & Widlund, 1990; Chan & Mathew, 1994) and extensively developed in deep learning (Dosovitskiy et al., 2020). Prior deep learning methods on neural networks propose to decompose the input finite dimension vector to multiple patches, accomplish local operations, and aggregate the result of such process in the global sense (Dosovitskiy et al., 2020; Guibas et al., 2021). Such methods do not decompose the output domain and directly predict the entire output vector. In contrast, *MG-TFNO* works on function spaces, and not only decomposes the input domain, but also decomposes the domain of the output functions, and separately predicts the output at each subdomain.

As we move beyond learning from simple structures to solving increasingly complex problems, the data we manipulate becomes more structured. To efficiently manipulate these structures, we need to go beyond matrix algebra and leverage the spatiotemporal structure. For all purposes of this paper, tensors are multi-dimensional arrays and generalize the concept of matrices to more than 2 modes (dimensions). For instance, RGB images are encoded as third-order (three-dimensional) tensors, videos are 4^{th} order tensors and so on and so forth. Tensor methods generalize linear algebraic methods to these higher-order structures. They have been very successful in various applications in computer vision, signal processing, data mining and machine learning (Panagakis et al., 2021; Janzamin et al., 2019; Sidiropoulos et al., 2017; Papalexakis et al., 2016).

Using tensor decomposition Kolda & Bader (2009), previous works have been able to compress and improve deep networks for vision tasks. Either a weight matrix is tensorized and factorized Novikov et al. (2015), or tensor decomposition is directly to the convolutional kernels before fine-tuning to recover-for lost accuracy, which also allows for an efficient reparametrization of the network (Lebedev et al., 2015; Kim et al., 2016; Gusak et al., 2019). There is a tight link between efficient convolutional blocks and tensor factorization and

Variable	Meaning	Dimensionality
$\mathbf{T}$	Tensor of weights in the Fourier domain	$\text{Cov}^{\alpha\times\cdots\times\alpha\times m\times n}$
$\mathbf{W}$	Weight tensor parameterizing the entire operator	$\text{Cov}^{\alpha\times\cdots\times\alpha\times n\times n\times 2^{d-1}L}$
$\mathcal{A}$	Input function space	Infinite
$\mathcal{U}$	output function space	Infinite
a	Input function	Infinite
u	Output function	Infinite
$D_{\mathcal{A}}$	Domain of function a	d
$D_{\mathcal{U}}$	Domain of function u	d
$d_{\mathcal{A}}$	Dimension of the co-domain of the input functions	1
$d_{\mathcal{U}}$	Dimension of the co-domain of the output functions	1
$\mathcal{F}$	Fourier transform	Infinite
$\mathcal{F}^{-1}$	Fourier transform	Infinite
L	Number of integral operation layers	In $\mathbb{N}$
l	Layer index	Between 1 and L
σ	Point-wise activation operation	Infinite
b	Bias vector	
v	Function at each layer	Infinite
α	Number of kept frequencies in Fourier space	Between 1 and $\frac{1}{2}\min\{s_1,\cdots,s_d\}$

Table 1: **Table of notation**

factorized higher-order structures (Kossaifi et al., 2020). Similar strategies have been applied to multi-task learning (Bulat et al., 2020a) and NLP (Papadopoulos et al., 2022; Cordonnier et al., 2020). Of all these prior works, none has been applied to neural operator. In this work, we propose the first application of tensor compression to learning operators and propose a Tensor OPerator (*TFNO*).

3 Methodology

Here, we briefly review operator learning as well as the Fourier Neural Operator, on which we build to introduce our proposed Tensor OPerator (*TFNO*) as well as the Multi-Grid Domain Decomposition, which together form our proposed *MG-TFNO*.

3.1 Operator Learning

Let $\mathcal{A} := \{a : D_{\mathcal{A}} \to \mathbb{R}^{d_{\mathcal{A}}}\}$ and $\mathcal{U} := \{u : D_{\mathcal{U}} \to \mathbb{R}^{d_{\mathcal{U}}}\}$ denote two input and output function spaces respectively. Each function a, in the input function space $\mathcal{A}$, is a map from a bounded, open set $D_{\mathcal{A}} \subset \mathbb{R}^d$ to the $d_{\mathcal{A}}$-dimensional Euclidean space. Any function in the output function space $\mathcal{U}$ is a map from a bounded open set $D_{\mathcal{U}} \subset \mathbb{R}^d$ to the $d_{\mathcal{U}}$-dimensional Euclidean space. In this work we consider the case $D = D_{\mathcal{A}} = D_{\mathcal{U}} \subset \mathbb{R}^d$.

We aim to learn an operator $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ which is a mapping between the two function spaces. In particular, given a dataset of N samples (e.g. PDE solves) $\{(a_j, u_j)\}_{j=1}^N$, where the pair (a_j, u_j) are functions satisfying $\mathcal{G}(a_j) = u_j$, we build an approximation of the operator $\mathcal{G}$. As a backbone operator learning model, we use neural operators as they are consistent and universal learners in function spaces. We refer the reader to Kovachki et al. (2021b) for an overview of theory and implementation. We specifically use the *FNO* and give details in the forthcoming section (Li et al., 2021b).

3.2 Notation

We summarize the notation used throughout the paper in Table 1.

3.3 Fourier Neural Operators

For simplicity, we will work on the d-dimensional unit torus $\mathbb{T}^d$ and first describe a single, pre-activation *FNO* layer mapping $\mathbb{R}^m$-valued functions to $\mathbb{R}^n$-valued functions. Such a layer constitutes the mapping $\mathcal{G} : L^2(\mathbb{T}^d; \mathbb{R}^m) \to L^2(\mathbb{T}^d; \mathbb{R}^n)$ defined as

$$\mathcal{G}(v) = \mathcal{F}^{-1}\big(\mathcal{F}(\kappa) \cdot \mathcal{F}(v)\big), \qquad \forall\, v \in L^2(\mathbb{T}^d; \mathbb{R}^m) \tag{1}$$

where $\kappa \in L^2(\mathbb{T}^d; \mathbb{R}^{n\times m})$ is a function constituting the layer parameters and $\mathcal{F}, \mathcal{F}^{-1}$ are the Fourier transform and its inverse respectively. The Fourier transform of the function κ is parameterized directly by some fixed number of Fourier nodes denoted $\alpha \in \mathbb{N}$.

To implement equation 1, $\mathcal{F}, \mathcal{F}^{-1}$ are replaced by the discrete fast Fourier transforms $\hat{\mathcal{F}}, \hat{\mathcal{F}}^{-1}$. Let $\hat{v} \in \mathbb{R}^{s_1\times\cdots\times s_d\times m}$ denote the evaluation of the function v on a uniform grid discretizing $\mathbb{T}^d$ with $s_j \in \mathbb{N}$ points in each direction. We replace $\mathcal{F}(\kappa)$ with a weight tensor $\mathbf{T} \in \mathrm{Cov}^{s_1\times\cdots\times s_d\times n\times m}$ consisting of the Fourier modes of κ which are parameters to be learned. To ensure that κ is parameterized as a $\mathbb{R}^{n\times m}$-valued function with a fixed, maximum amount of wavenumbers $\alpha < \frac{1}{G2}\min\{s_1, \cdots, s_d\}$ that is independent of the discretization of $\mathbb{T}^d$, we leave as learnable parameters only the first α entries of $\mathbf{T}$ in each direction and enforce that $\mathbf{T}$ have conjugate symmetry.

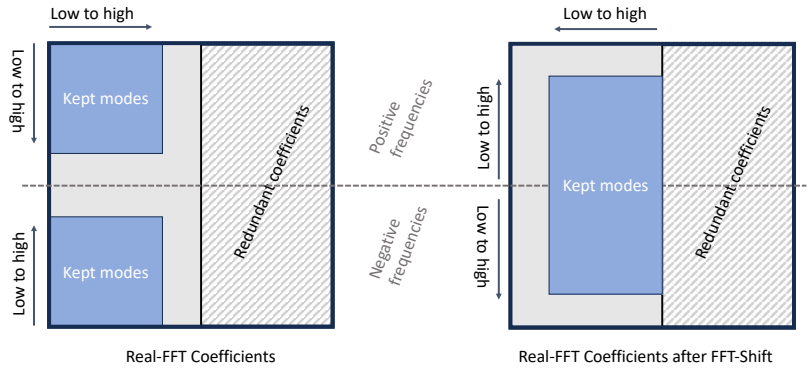


Figure 3: Visualization of the hyper-corners containing the low-frequency in the real-FFT. On the left, we show the low-frequency modes used by our algorithm. Equivalently, we can first apply an FFT-shift (right) and keep the central coefficients.

In particular, we parameterize half the corners of the d-dimensional hyperrectangle with 2^{d-1} hypercubes with length size α. That is, $\mathbf{T}$ is made up of the free-parameter tensors $\tilde{\mathbf{T}}_1, \cdots, \tilde{\mathbf{T}}_{2^{d-1}} \in \mathrm{Cov}^{\alpha\times\cdots\times\alpha\times n\times m}$ situated in half of the corners of $\mathbf{T}$. Each corner diagonally opposite of a tensor $\tilde{\mathbf{T}}_j$ is assigned the conjugate transpose values of $\tilde{\mathbf{T}}_\mathbf{j}$. All other values of $\mathbf{T}$ are set to zero. This is illustrated in the middle-top part of Figure 3.3 for the case $d = 2$ with $\tilde{\mathbf{T}}_1$ and $\tilde{\mathbf{T}}_2$. Note that, equivalently, we can apply an FFT-shift to the output of the real FFT so the low-frequency coefficients are located in one block in the middle. In that case, we then apply another FFT-shift to the output of the contraction before applying the inverse real FFT. We will use the notation $\mathbf{T}(k, \cdots) = \tilde{\mathbf{T}}_k$ for any $k \in [2^{d-1}]$. The discrete version of equation 1 then becomes the mapping $\hat{\mathcal{G}} : \mathbb{R}^{s_1\times\cdots\times s_d\times m} \to \mathbb{R}^{s_1\times\cdots\times s_d\times n}$ defined as

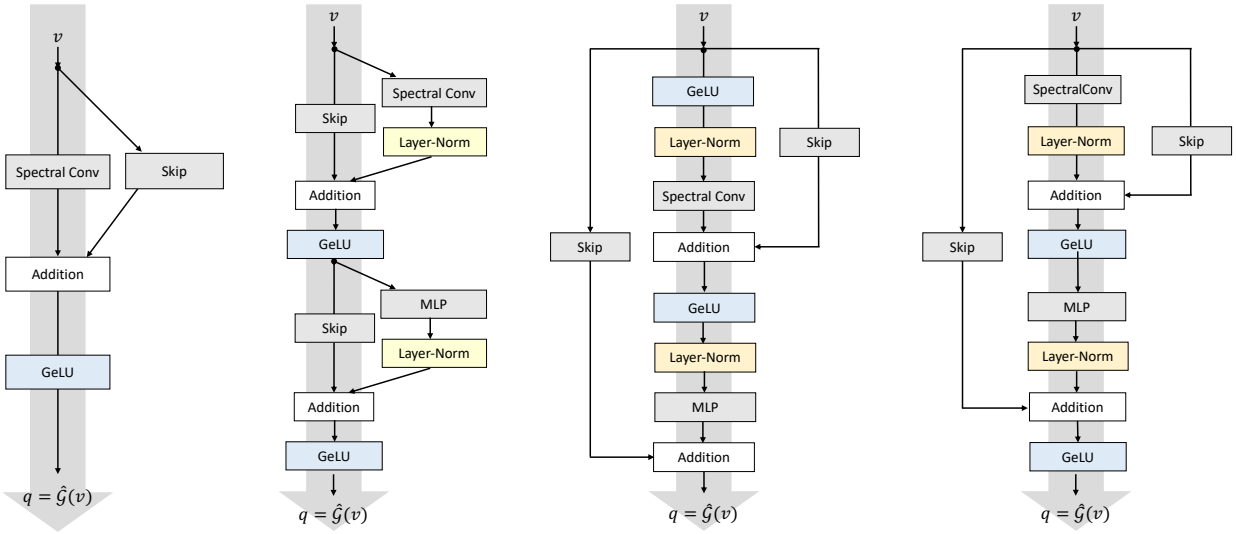
$$\hat{\mathcal{G}}(\hat{v}) = \hat{\mathcal{F}}^{-1}\big(\mathbf{T} \cdot \hat{\mathcal{F}}(\hat{v})\big), \qquad \forall\, \hat{v} \in \mathbb{R}^{s_1\times\cdots\times s_d\times m} \tag{2}$$

where the $\cdot$ operation is simply a tensor contraction along the last dimension. Specifically, we have

$$\big(\mathbf{T} \cdot \hat{\mathcal{F}}(\hat{v})\big)(l_1, \ldots, l_d, j) = \sum_{i=1}^{m} \mathbf{T}(l_1, \ldots, l_d, j, i)\big(\hat{\mathcal{F}}(\hat{v})\big)(l_1, \ldots, l_d, i). \tag{3}$$

From equation 2, a full FNO layer is build by adding a point-wise linear action to $\hat{v}$, a bias term, and applying a non-linear activation. In particular, from an input $\hat{v} \in \mathbb{R}^{s_1\times\cdots\times s_d\times m}$, the output $\hat{q} \in \mathbb{R}^{s_1\times\cdots\times s_d\times n}$ is given as

$$\hat{q}(l_1, \cdots, l_d, :) = \sigma\big(\mathbf{Q}\hat{v}(l_1, \cdots, l_d, :) + \hat{\mathcal{G}}(\hat{v}) + b\big)$$



(a) **Original FNO.** (b) **Double Skip.** (c) **Improved backbone - pre-activation.** (d) **Improved backbone.**

Figure 4: **Original FNO and Improved Backbone Architecture.** The original FNO architecture (Li et al., 2021b) is composed of simply a Spectral Convolution, with a (linear, soft-gating, or identity) skip connection to recover high-frequency information and handle non-periodic inputs (4(a)). We improve the architecture as detailed in section 3.4. In particular, we have a version with a double (sequential) skip connection (4(b)), while our best architecture uses nested skip connections, and can be made both with and without preactivation (subfigures 4(c) and 4(d), respectively). The latter, subfigure 4(d), is our best architecture.

with $\sigma : \mathbb{R} \to \mathbb{R}$ a fixed, non-linear activation, and $b \in \mathbb{R}^n$, $\mathbf{Q} \in \mathbb{R}^{n \times m}$, $\tilde{\mathbf{T}}_1, \cdots, \tilde{\mathbf{T}}_{2^{d-1}} \in \mathrm{Cov}^{\alpha \times \cdots \times \alpha \times n \times m}$ are the learnable parameters of the layer. The full FNO model consists of $L \in \mathbb{N}$ such layers each with weight tensors $\mathbf{T}_1, \cdots, \mathbf{T}_L$ that have learnable parameters $\tilde{\mathbf{T}}_k^{(l)} = \mathbf{T}_l(k, \cdots)$ for any $l \in [L]$ and $k \in [2^{d-1}]$. In the case $n = m$ for all layers, we introduce the joint parameter tensor $\mathbf{W} \in \mathrm{Cov}^{\alpha \times \cdots \times \alpha \times n \times m \times 2^{d-1} L}$ so that

$$\mathbf{W}\left(\ldots, 2^{d-1}(l-1) + k + 1\right) = \tilde{\mathbf{T}}_k^{(l)}. \tag{4}$$

A perusal of the above discussion reveals that there are $(2^d \alpha^d + 1)mn + n$ total parameters in each FNO layer. Note that, since m and n constitute the respective input and output channels of the layer, the number of parameters can quickly explode due to the exponential scaling factor $2^d \alpha^d$ if many wavenumbers are kept. Preserving a large number of modes could be crucial for applications where the spectral decay of the input or output functions is slow such as in image processing or the modeling of multi-scale physics. In the following section, we describe a tensorization method that is able to mitigate this growth without sacrificing approximation power.

3.4 Architectural improvements

Our proposed approach uses *FNO* as a backbone. To improve its performance, we first study various aspects of the Fourier Neural Architecture and perform thorough ablation to validate each aspect. In particular, we propose improvements to the base architecture that improve performance.

Normalization in neural operators While normalization techniques, such as Batch-Normalization Ioffe & Szegedy (2015), have proven very successful in training neural networks, additional consideration must be given when applying those to neural operators in order to preserve its properties, notably discretization invariance. Specifically, it cannot depend on the spatial variables and therefore has to be either a global or

a function-wise normalization. We investigate several configurations using instance normalization Ulyanov et al. (2016) and layer-normalization Ba et al. (2016), in conjunction with the use-of preactivation He et al. (2016).

Channel mixing *FNO* relies on a global convolution realized in the spectral domain. Inspired by previous works, e.g. Guibas et al. (2021), we propose adding an MLP in the *original* space, after each Spectral convolution. In practice, we found that two-layer bottleneck MLP works well, e.g. we decrease the co-dimension by half in the first linear layer before restoring it in the second one.

Boundary conditions: domain padding and skip connections Fourier neural operators circumvent the limitation of traditional Fourier methods to inputs with periodic boundaries only. This is achieved through a local linear transformation added to the spectral convolution. This can be seen as a linear skip connection. We investigate replacing these with an identity skip-connection and a soft-gated skip-connection Bulat et al. (2020b).

We represent in Figure. 4 the original FNO architecture (Li et al., 2021b), subfigure 4(a), the improved version with double (sequential) skip connections (subfigure 4(b)) and our best architecture, both with and without preactivation (subfigures 4(c) and 4(d), respectively).

We also investigate the impact of domain-padding, found by Li et al. (2021b) to improve results, especially for non-periodic inputs, and padding for the multi-grid decomposition. In fact, the multi-grid domain decomposition inherently breaks any periodic boundary conditions (if the function itself is periodic, its patches typically likely are not). The effect of this can be mitigated by padding the inputs, which increases computational cost, resulting in a tradeoff between padding and computational efficiency.

3.5 Tensor Fourier Neural Operators

In the previous section, we introduced a unified formulation of FNO where the whole operator is parametrized by a single parameter tensor $\mathbf{W}$. This enables us to introduce the tensor operator, which parameterizes efficiently $\mathbf{W}$ with a low-rank, tensor factorization. We introduce the method for the case of a Tucker decomposition, for its flexibility. Other decompositions, such as Canonical Polyadic, can be readily integrated. This joint parametrization has several advantages: i) it applies a low-rank constraint on the entire tensor $\mathbf{W}$, thus regularizing the model. These advantages translate into i) a huge reduction in the number of parameters, ii) better generalization and an operator less prone to overfitting. We show superior performance for low-compression ratios (up to 200×) and very little performance degradation when largely compressing ($> 450\times$) the model, iii) better performance in a low-data regime.

In practice, we express $\mathbf{W}$ in a low-rank factorized form, e.g. Tucker or CP. In the case of a Tucker factorization with rank $(R_1, \cdots, R_d, R_L, R_I, R_O)$, where R_L controls the rank across layers, $R_I = R_O$ control the rank across the input and output co-dimension, respectively, and $R_1, \cdots, R_d$ control the rank across the dimensions of the operator:

$$\mathbf{W} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_d=1}^{R_d} \sum_{r_i=1}^{R_I} \sum_{r_o=1}^{R_O} \sum_{r_l=1}^{R_L} \mathbf{G}(r_1, \cdots, r_d, r_i, r_o, r_l) \cdot \mathbf{U^{(1)}}(:, r_1) \cdot \ \cdots \ \cdot \mathbf{U^{(d)}}(:, r_d) \cdot \mathbf{U^{(I)}}(:, r_i) \cdot \cdot \mathbf{U^{(O)}}(:, r_o) \cdot \mathbf{U^{(L)}}(:, r_l). \tag{5}$$

Here, $\mathbf{G}$ is the core of size $R_L \times R_I \times R_O \times R_1 \times \cdots \times R_d$ and $\mathbf{U^{(L)}}, \mathbf{U^{(I)}}, \mathbf{U^{(O)}}, \mathbf{U^{(1)}}, \cdots, \mathbf{U^{(d)}}$ are factor matrices of size $(R_L \times L), (R_I \times I), (R_O \times O), (R_1 \times \alpha), \cdots, (R_d \times \alpha)$, respectively.

Note that the mode (dimension) corresponding to the co-dimension can be left uncompressed by setting $R_L = L$ and $\mathbf{U^{(L)}} = \mathrm{Id}$. This leads to layerwise compression. Also note that having a rank of 1 along any of the modes would mean that the slices along that mode differ only by a (multiplicative) scaling parameter. Also note that during the forward pass, we can pass $\mathbf{T}$ directly in factorized form to each layer by selecting the corresponding rows in $\mathbf{U^{(L)}}$. While the contraction in equation 3 can be done using the reconstructed

tensor, it can also be done directly by contracting $\hat{\mathcal{F}}(\hat{v})$ with the factors of the decomposition. For small, adequately chosen ranks, this can result in computational speedups.

A visualization of the Tucker decomposition of a third-order tensor can be seen in Figure 5). Note that we can rewrite the entire weight parameter for this Tucker case, equivalently, using the more compact n-mode product as:

$$\mathbf{W} = \mathbf{G} \times_1 \mathbf{U^{(1)}} \cdots \times_d \mathbf{U^{(d)}} \times_{d+1} \mathbf{U^{(I)}} \times_{d+2} \mathbf{U^{(O)}} \times_{d+3} \mathbf{U^{(L)}}$$

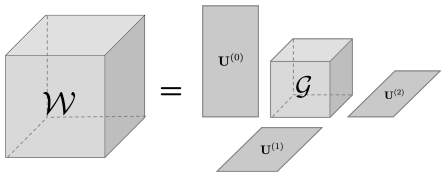


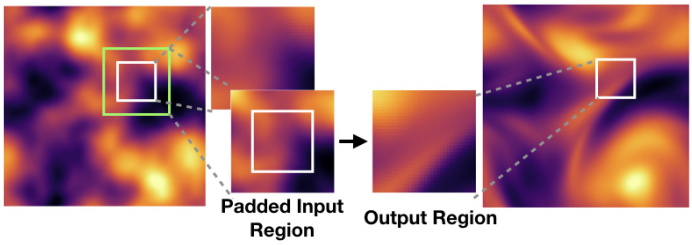
Figure 5: **Illustration of a Tucker decomposition.** For clarity , we show $\mathbf{W}$ as a 3^{rd}-order tensor weight.

We can efficiently perform an iFFT after contraction with the tensorized kernel. For any layer l, the (j_1, j_2) coordinate of the matrix-valued convolution function $\kappa(x)$ is as follows,

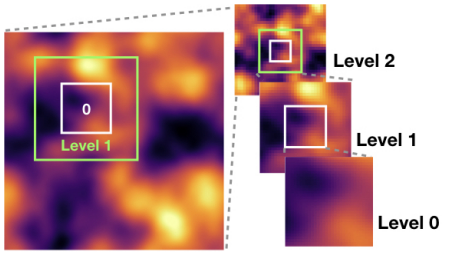
$$[\kappa_l(x)]j1, j_2 = \sum_{i_1=1}^{m_1} \cdots \sum_{i_d=1}^{m_d} \sum_{r_l=1}^{R_L} \sum_{r_i=1}^{R_I} \sum_{r_o=1}^{R_O} \sum_{r_1=1}^{R_1} \cdots \sum_{r_d=1}^{R_d} \mathbf{G}(r_1, \cdots, r_d, r_i, r_o, r_l) \cdot$$

$$\mathbf{U^{(1)}}(i_1, r_1) \cdots \mathbf{U^{(d)}}(i_d, r_d) \cdot \mathbf{U^{(I)}}(j_1, r_i) \cdot \mathbf{U^{(O)}}(j_2, r_o) \cdot \mathbf{U^{(L)}}(l, r_l) \cdot \exp(2\pi \sum_{k=1}^{d} i x_k i_k)$$

This joint factorization along the entire operator allows us to leverage redundancies both locally and across the entire operator. This leads to a large reduction in the memory footprint, with only a fraction of the parameter. It also acts as a low-rank regularizer on the operator, facilitating training. Finally, through global parametrization, we introduce skip connections that allow gradients to flow through the latent parametrization to all the layers jointly, leading to better optimization.



(a) **Predicting with padded regions.** Local region in the input is padded and used to predict the corresponding region in the output.



(b) **MG-Domain Decomposition.** Progressively larger spatial regions are added to a local region by subsampling.

Figure 6: **Domain decomposition in space (6(a)) and our Multi-Grid based approach. (6(b))**. White squares represent the region of interest while yellow squares the larger embeddings.

Importantly, this formulation is general and works with any tensor factorization. For instance, we also explore a Canonical-Polyadic decomposition (CP) which can be seen as a special case of Tucker with a super-diagonal core. In that case, we set a single rank R and express the weights as a weighted sum of R rank-1 tensors. Concretely:

$$\mathbf{W} = \sum_{r=1}^{R} \lambda_r \mathbf{U^{(1)}}(:, r) \cdot \ \cdots \ \cdot \mathbf{U^{(d)}}(:, r) \cdot \tag{6}$$

$$\mathbf{U^{(I)}}(:, r) \cdot \mathbf{U^{(O)}}(:, r) \cdot \mathbf{U^{(L)}}(:, r).$$

where $\mathbf{U^{(L)}}, \mathbf{U^{(I)}}, \mathbf{U^{(O)}}, \mathbf{U^{(1)}}, \cdots, \mathbf{U^{(d)}}$ are factor matrices of size $(R{\times}L), (R{\times}I), (R{\times}O), (R{\times}\alpha), \cdots, (R{\times}\alpha)$, respectively and $\lambda \in \mathbb{R}^{\mathbf{R}}$. Note that the CP, contrarily to the Tucker, has a single rank parameter, shared between all the dimensions. This means that to maintain the number of parameters the same, R needs to

be very high, which leads to memory issues. This makes CP more suitable for large compression ratios, and indeed, we found it leads to better performance at high-compression / very low-rank. In this paper, we also explore the tensor-train decomposition Oseledets (2011). A rank-$(1, R_1, \cdots, R_N, R_I, R_O, R_L, 1)$ TT factorization expresses $\mathbf{W}$ as:

$$\mathbf{W}(i_1, \cdots, i_d, i_c, i_o, i_l) = \mathbf{G}_1(i_1) \cdot \times \mathbf{G}_N(i_d)\mathbf{G}_I(i_c) \times \cdots \mathbf{G}_O(i_o) \times \cdots \mathbf{G}_L(i_l).$$

Where each of the factors of the decompositions $\mathbf{G}_k$ are third order tensors of size $R_k \times I_k \times R_{k+1}$.

In the experimental section 4.3, we show results of *TFNO* trained with a Tucker, TT and CP factorization.

Separable Fourier Convolution The proposed tensorization approach introduces a factorization of the weights in the spectral domain. When a CP Kolda & Bader (2009) is used, this induces separability over the learned kernel. In the case where the number of input channels is equal to the number of output channels ($n = m$ in eq. 1), we propose to make this separability explicit by not performing any channel mixing in the spectral domain and relying on the MLP introduced above to do so. Specifically, we can now rewrite the parameters in Eq. 4 as $\mathbf{W} \in \text{Cov}^{\alpha \times \cdots \times \alpha \times n \times 2^{d-1}L}$ (notice this has one less dimension that for the non-separable case). The operation in the spectral domain simplifies from a tensor contraction to just an element-wise matrix multiplication:

$$\big(\mathbf{T} \cdot \hat{\mathcal{F}}(\hat{v})\big)(l_1, \ldots, l_d, i) = \mathbf{T}(l_1, \ldots, l_d, i) * \big(\hat{\mathcal{F}}(\hat{v})\big)(l_1, \ldots, l_d, i).$$

The separable Spectral convolution can be considered a depthwise convolution performed in the Fourier domain, e.g. without any channel mixing, and with separability along each dimension. The mixing between channels is instead done in the spatial domain. This results in a significant reduction in the number of parameters while having minimal impact on performance (we found it necessary to increase the depth of the network, however, to ensure the network retained enough capacity).

3.6 Multi-Grid Domain Decomposition

Having introduced our decomposition in the operator's parameter space, we now introduce our novel multi-grid approach to decompose the problem domain.

Domain decomposition is a method commonly used to parallelize classical solvers for time-dependent PDEs that is based on the principle that the solution for a fixed local region in space depends mostly on the input at the same local region (Chan & Mathew, 1994). In particular, since the time-step $h > 0$ of the numerical integrator is small, the solution $u(x, t + h)$, for any point $x \in D$ and $t \in \mathbb{R}_+$, depends most strongly on the points $u(y, t)$ for all $y \in B\big(x, r(h)\big)$ where $B\big(x, r(h)\big)$ denotes the ball centered at x with radius $r(h)$. This phenomenon is easily seen for the case of the heat equation where, in one dimension, the solution satisfies

$$\begin{aligned} u(x, t + h) &\propto \int_{-\infty}^{\infty} \exp\left(\frac{-(x - y)^2}{4h}\right) u(y, t)\, \mathsf{d}y \\ &\approx \int_{x-4h}^{x+4h} \exp\left(\frac{-(x - y)^2}{4h}\right) u(y, t)\, \mathsf{d}y \end{aligned}$$

with the approximation holding since 99.9937% of the kernel's mass is contained within $B(x, 4h)$. While some results exist, there is no general convergence theory for this approach, however, its empirical success has made it popular for various numerical methods (Albin & Bruno, 2011). Domain deomposition methods are also widely used for boundary value, for example, in solving elliptic PDEs (Dryja & Widlund, 1990). Differences in methods vary based on how boundary data is handled between adjecent subdomains. In our proposed appraoch we supplemant global data to each subdomain by using overlapping patches and stacking channels with an increasing receptive between at coarser scales.

To exploit this localization , the domain D is split in $q \in \mathbb{N}$ pairwise-disjoint regions $D_1, \cdots, D_q$ so that $D = \cup_{j=1}^{q} D_j$. Each region D_j is then embedded into a larger one $Z_j \supset D_j$ so that points away from the

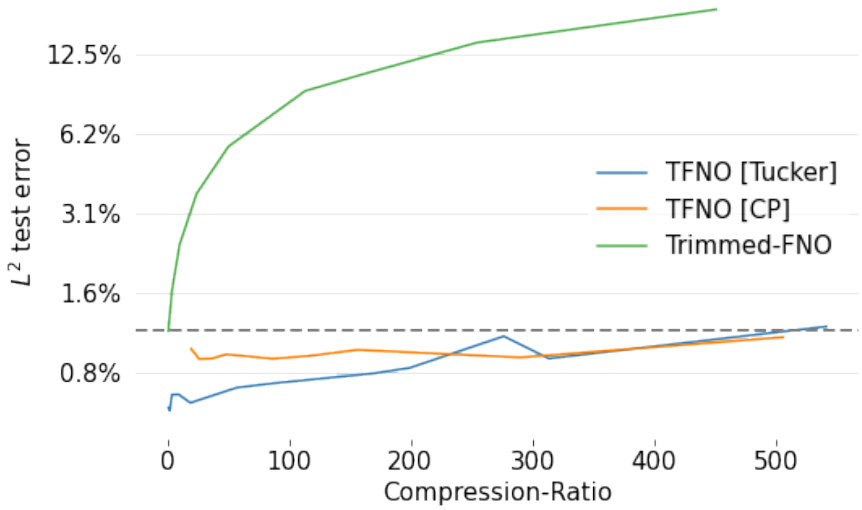


Figure 7: **Tensorization: error in logscale as a function of the compression ratio.** We compare the tensor neural operator with an FNO with the same number of parameters (*trimmed*). **We achieve over 100x compression ratio with better performance that the original FNO**

center of D_j have enough information to be well approximated. A model can then be trained so that the approximation $\mathcal{G}(a|_{Z_j})|_{D_j} \approx u|_{D_j}$ holds for all $j \in [q]$. This idea is illustrated in Figure 6(a) where $D = [0,1]^2$ and all D_j, Z_j are differently sized squares. This allows the model to be ran fully in parallel hence its time and memory complexities are reduced linearly in q.

Multi-Grid. Domain decomposition works well in classical solvers when the time step $h > 0$ is small because the mapping $u(\cdot, t) \mapsto u(\cdot, t+h)$ is close to the identity. However, the major advancement made by machine learning-based operator methods for PDEs is that a model can approximate the solution, in one shot, for very large times i.e. $h > 1$. But, for larger h, the size of Z_j relative to D_j must increase to obtain the same approximation accuracy, independently of model capacity. This causes any computational savings made by the decomposition approach to be lost.

To mitigate this, we propose a multi-grid based domain decomposition approach where global information is added hierarchically at different resolutions. While our approach is inspired by the classical multi-grid method, it is not based on the V-cycle algorithm (McCormick, 1985). For ease of presentation, we describe this concept when a domain $D = \mathbb{T}^2$ is uniformly discretized by $2^s \times 2^s$ points, for some $s \in \mathbb{N}$, but note that generalizations can readily be made. Given a final level $L \in \mathbb{N}$, we first sub-divide the domain into 2^{2L} total regions each of size $2^{s-L} \times 2^{s-L}$ and denote them $D_1^{(0)}, \cdots, D_{2^{2L}}^{(0)}$. We call this the zeroth level. Then, around each $D_j^{(0)}$, for any $j \in [2^{2L}]$, we consider the square $D_j^{(1)}$ of size $2^{s-L+1} \times 2^{s-L+1}$ that is equidistant, in every direction, from each boundary of $D_j^{(0)}$. We then subsample the points in $D_j^{(1)}$ uniformly by a factor of $\frac{1}{2}$ in each direction, making $D_j^{(1)}$ have $2^{s-L} \times 2^{s-L}$ points. We call this the first level. We continue this process by considering the squares $D_j^{(2)}$ of size $2^{s-L+2} \times 2^{s-L+2}$ around each $D_j^{(1)}$ and subsample them uniformly by a factor of $\frac{1}{4}$ in each direction to again yield squares with $2^{s-L} \times 2^{s-L}$ points. The process is repeated until the Lth level is reached wherein $D_j^{(L)}$ is the entire domain subsampled by a factor of 2^{-L} in each direction. The input to the model is then the concatenation each of these levels, in particular,

$$\mathbf{a}_j = [D_j^{(0)}, \ldots, D_j^{(L)}] \in \mathbb{R}^{L+1 \times 2^{s-L} \times 2^{s-L}}$$

The process is illustrated for the case $L = 2$ in Figure 6(b). Since we work with the torus, the region of the previous level is always at the center of the current level.

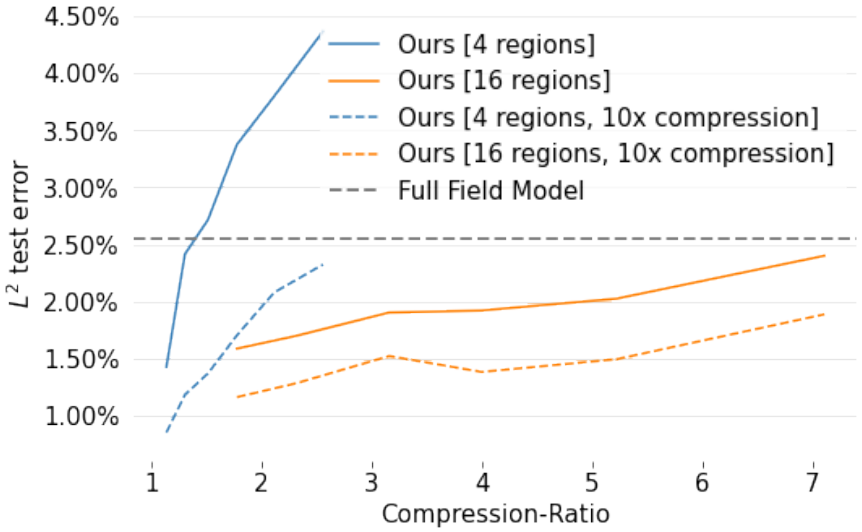


Figure 8: **MG-Domain Decomposition: error as a function of the domain compression ratio.** We compare *MG-TFNO* with different numbers of multigrid regions both with and without weight tensor compression to a full field FNO model. **We achieve over 7x input space compression, 10x parameter space compression ratios and better performance than the original FNO.**

The intuition behind this method is that since the dependence of points inside a local region diminishes the further we are from that region, it is enough to have coarser information, as we go farther. We combine this multi-grid method with the standard domain decomposition approach by building appropriately padded squares $Z_j^{(l)}$ of size $2^{s-L}+2p \times 2^{s-L}+2p$ around each $D_j^{(l)}$ where $p \in \mathbb{N}$ is the amount of padding to be added in each direction. We then take the evaluations of the input function a at each level and concatenate them as channels. In particular, we train a model so that $\hat{\mathcal{G}}\big((a|_{Z_j^{(0)}}, \cdots, a|_{Z_j^{(L)}})\big)|_{D_j^{(0)}} \approx u|_{D_j^{(0)}}$. Since the model only operates on each padded region separately, we reduce the total number of grid points used from 2^{2s} to $(2^{s-L}+2p)^2$ and define the domain compression ratio as the quotient of these numbers. Furthermore, note that, assuming a is $\mathbb{R}^{d_{\mathcal{A}}}$-valued, a model that does not employ our multi-grid domain decomposition uses inputs with $d_{\mathcal{A}}$ channels while our approach builds inputs with $(L+1)d_{\mathcal{A}}$ channels. In particular, the number of input channels scales only logarithmically in the number of regions hence global information is added at very little additional cost. Indeed, FNO models are usually trained with internal widths much larger than $d_{\mathcal{A}}$ hence the extra input channels cause almost no additional memory overhead.

4 Experiments

In this section, we first introduce the data, experimental setting and implementation details before empirically validating our approach through thorough experiments and ablations.

4.1 Data.

We experiment on a dataset of 10K training samples and 2K test samples of the two-dimensional Navier-Stokes equation with Reynolds number 500. We also experiment with the one-dimensional viscous Burgers' equation.

Navier-Stokes. We consider the vorticity form of the two-dimensional Navier-Stokes equation,

$$\begin{aligned} \partial_t \omega + \nabla^\perp \phi \cdot \omega &= \frac{1}{\mathrm{Re}} \Delta \omega + f, \qquad x \in \mathbb{T}^2,\ t \in (0, T] \\ -\Delta \phi = \omega, \quad \int_{\mathbb{T}^2} \phi &= 0, \qquad x \in \mathbb{T}^2,\ t \in (0, T] \end{aligned} \tag{7}$$

with initial condition $\omega(0, \cdot) = 0$ where $\mathbb{T}^2 \cong [0, 2\pi)^2$ is the torus, $f \in \dot{L}^2(\mathbb{T}^2; \mathbb{R})$ is a forcing function, and $\mathrm{Re} > 0$ is the Reynolds number. Then $\omega(t, \cdot) \in \dot{H}^s(\mathbb{T}^2; \mathbb{R})$ for any $t \in (0, T]$ and $s > 0$, is the unique weak solution to equation 7 (Temam, 1988). We consider the non-linear operator mapping $f \mapsto \omega(T, \cdot)$ with $T = 5$ and fix the Reynolds number $\mathrm{Re} = 500$. We define the Gaussian measure $\mu = \mathcal{N}(0, C)$ on the forcing functions where we take the covariance $C = 27(-\Delta + 9I)^{-4}$, following the setting in (De Hoop et al., 2022). Input data is obtained by generating i.i.d. samples from μ by a KL-expansion onto the eigenfunctions of C (Powell et al., 2014). Solutions to equation 7 are then obtained by a pseudo-spectral scheme (Chandler & Kerswell, 2013). Generating a single instance of the data at the 128×128 resilution on a single GPU takes approximately one minute. All machine learning methods compared in this work have an execution time of less than one second, providing, at least, $100\times$ speed-up. The cost of generating the high resolution data scales quadrtically due to the CFL condition and the speed-up due to the machine learning methods becomes much larger.

Burgers' Equation. We consider the one-dimensional Burgers' equation on the torus,

$$\begin{aligned} \partial_t u + u u_x &= \nu u_{xx}, \qquad x \in \mathbb{T},\ t \in (0, T] \\ u|_{t=0} &= u_0, \qquad x \in \mathbb{T} \end{aligned} \tag{8}$$

for initial condition $u_0 \in L^2(\mathbb{T}; \mathbb{R})$ and viscosity $\nu > 0$. Then $u(t, \cdot) \in H^s(\mathbb{T}; \mathbb{R})$, for any $t \in \mathbb{R}_+$ and $s > 0$, is the unique weak solution to 8 (Evans, 2010). We consider the non-linear operator $u_0 \mapsto u(T, \cdot)$ with $T = 0.5$ or 1 and fix $\nu = 0.01$. We define the Gaussian measure $\mu = \mathcal{N}(0, C)$ where we take the covariance $C = 3^{5/2}(-\frac{d^2}{dx^2} + 9I)^{-3}$. Input data is obtained by generating i.i.d. samples from μ by a KL-expansion onto the eigenfunctions of C. Solutions to equation 8 are then obtained by a pseudo-spectral solver using Heun's method. We use 8K samples for training and 2K for testing.

4.2 Implementation details

Implementation We use PyTorch Paszke et al. (2017) for implementing all the models. The tensor operations are implemented using TensorLy Kossaifi et al. (2019) and TensorLy-Torch Kossaifi (2021). Our code will be released under the permissive MIT license, as a Python package that is well-tested and comes with extensive documentation, to encourage and facilitate downstream scientific applications. It will be made available along withe the final version of the manuscript

Hyper-parameters We train all models via gradient backpropagation using a mini-batch size of 16, the Adam optimizer, with a learning rate of $1e^{-3}$, weight decay of $1e^{-4}$, for 500 epochs, decreasing the learning rate every 100 epochs by a factors of $\frac{1}{2}$. The model width is set in all cases to 64 except when specified otherwise (for the Trimmed *FNO*), meaning that the input was first lifted (with a linear layer) from the number of input channels to that width. The projection layer projects from the width to 256 and a prediction linear layer outputs the predictions. 10000 samples were used for training, as well as a separate set of 2000 samples for testing. All experiments are done on a NVIDIA Tesla V100 GPU. The resolution is 128×128 unless specified otherwise.

To disentangle the effect of each of our components, the comparisons between the original FNO, the MG-FNO, *TFNO*, and the *MG-TFNO* were conducted in the same setting, with a mini-batch size of 32, modes of 42 and 21 for the height and width, respectively, and an operator width of 64.

For the comparison between our best models, we use all the modes (64 and 32) and a mini-batch size of 16, which leads to improved performance for all models but longer training times. For each comparison, the same setting and hyper-parameters were used for all models.

Table 2: **Comparing the performance of *MG-TFNO* with previous works, on the Navier-Stokes equations**. Our method achieves superior performance with a fraction of the parameters while largely compressing the weights (*TFNO*) and the input-domain (*MG-TFNO*).

Method	**L^2 test error (%)**	**# Params**	**Model CR**	**Input CR**
FNO Li et al. (2021b)	1.34 %	67M	-	-
FFNO Tran et al. (2023)	1.15 %	1M	67×	-
UNO Rahman et al. (2022b)	0.5 %	96M	-	-
TUNO (CP)	0.49%	1M	72×	-
TFNO (CP)	0.29%	890K	75×	-
TFNO (CP)	0.47%	447K	150×	-
MG-TFNO (CP)	0.49 %	447K	40×	1.9×
MG-TFNO (Tucker)	0.42 %	447K	19×	1.9×

Training the operator. Since *MG-TFNO* predicts local regions which are then stitched together to form a global function without any communication, aliasing effects can occur where one output prediction does not flow smoothly into the next. To prevent this, we train our model using the H^1 Sobolev norm (Czarnecki et al., 2017; Li et al., 2021a). By matching derivatives, training with this loss prevents any discontinuities from occurring and the output prediction is smooth. We note that this approach is only applicable if the PDE solution is globally H^1 which may not always be the case, for example, in problems which contains shocks. A new approach for smoothly stiching the patch-wise solution will be needed in such cases and we leave this to future work. We note that all problems considered in our experimental section satisfy the H^1 assumption.

4.3 Experimental results

In this section, we empirically verify the performance of our method, by comparing it with previous works, and through thorough ablation studies.

Comparison with previous works In this section, we compare our approach with both the regular FNO Li et al. (2021b), the Factorized-FNO (FFNO) Tran et al. (2023) and the U-Shaped Neural Operator Rahman et al. (2022b).

FFNO separately applied FFT along each mode before combining the results. We used the same settings as described in the original paper Tran et al. (2023). U-NO uses a U-shaped architecture and reduces the spacial resolution while increasing the co-domain size.

In all cases, our approach achieves superior performance with a fraction of the parameters, as can be seen in Table 4.4.4.

Tensorizing: better compression. In Figure 7, we show the performance of our approach (TNO) compared to the original *FNO*, for varying compression ratios. In the Trimmed-*FNO*, we adjust the width in order to match the number of parameters in our TNO. We focus on the width of the network as it was shown to be the most important parameter (De Hoop et al., 2022). Our method massively outperforms the Trimmed-*FNO* at every single fixed parameter amount. Furthermore, even for very large compression ratios, our FNO outperforms the full-parameter FNO model. This is likely due to the regularizing effect of the tensor factorization on the weight, showing that many of the ones in the original model are redundant.

Tensorizing: better generalization. Figure 9 (left) shows that our TNO generalizes better with less training samples. Indeed, at every fixed amount of training samples, the TNO massively outperforms the full-parameter FNO model. Even when only using half the samples, our TNO outperforms the FNO trained on the full dataset. Furthermore, Figure 9 (right) shows that our TNO overfits significantly less than FNO, demonstrating the regularizing effect of the tensor decomposition. This result is invaluable in the *PDE* setting

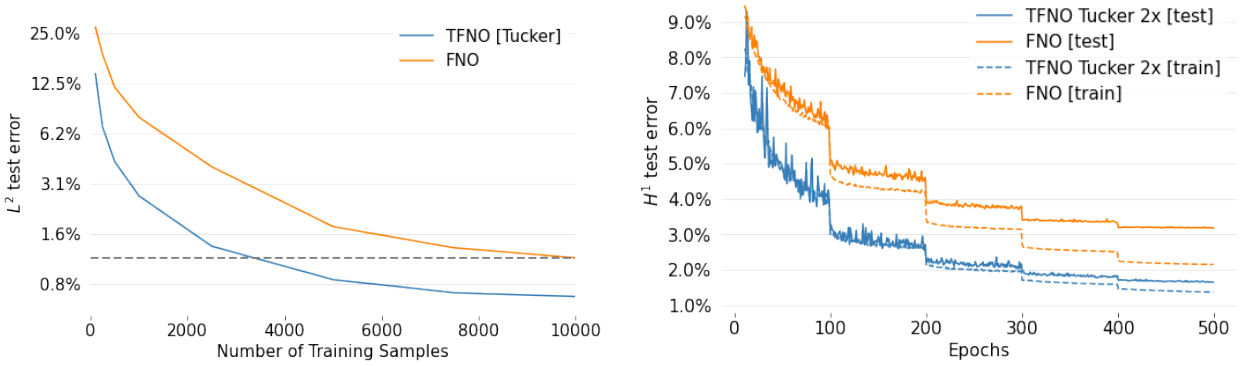


Figure 9: **Error as a function of the number of training samples (left) and training VS testing loss.** We compare *TFNO* with a regular *FNO*. Note that on the left we show the testing L^2 error while, for training, the H^1 loss is used and that is compared with the H^1 test error on the right. Our approach generalizes better while requiring fewer training samples.

where very few training samples are typically available due to the high computational cost of traditional *PDE* solvers.

Multi-Grid Domain Decomposition. In Table 4.4.4, we compare our *MG-TFNO* with the baseline FNO and the *TFNO*, respectively. *MG-TFNO* enables compressing both the weight tensor but also the input domain. On the other hand, preserving resolution invariance requires padding the patches, which decreases performance, resulting in a tradeoff between input domain compression and prediction accuracy.

We also show the impact of multi-grid domain decomposition on performance in Figure 8. We find that lower compression ratios (corresponding to a larger amount of padding in the decomposed regions) perform better which is unsurprising since more information is incorporated into the model. More surprisingly, we find that using a larger number of regions (16) performs consistently better than using a smaller number (4) and both can outperform the full-field FNO. This can be due to the fact that: i) the domain decomposition acts as a form of data augmentation, exploiting the translational invariance of the *PDE* and more regions yield larger amounts of data, and ii) the output space of the model is simplified since a function can have high frequencies globally but may only have low frequencies locally. Consistently, we find that the tensor compression in the weights acts as a regularizer and improves performance across the board.

Architectural improvements to the backbone In addition to the ablation performed on our *MG-TFNO*, we also investigate architectural improvements to the *FNO* backbone, see Sec 3.4 for details. In particular, we find that, while instance normalization decreases performance, layer normalization helps, especially when used in conjunction with a pre-activation. Adding an MLP similarly improves performance, we found that a bottleneck (expansion factor of 0.5) works well in practice, resulting in an absolute improvement of 0.87% in relative L^2 error. We found the ordering of normalization, activation, and weights (including preactivation), did not have a significant impact on performance. Finally, when not using multi-grid domain decomposition, the inputs are periodic and padding is not necessary. In that case, not padding the input improves performance. We use all these improvements for the backbone of the best version of our *MG-TFNO*, Fig 1 where we show that our improved backbone significantly outperforms the original FNO, while our approach significantly outperforms both, with a small fraction of the parameters, opening the door to the application of *MG-TFNO* to high-resolution problems.

We experiment with various number of layers and compare the original FNO with our improved backbone in Table. 4.3.

Table 3: **Impact of our architectural improvements**.

Method	**Layers**	L^2 **test error**	H^1 **test error**	**# Params**	**Model CR**
FNO Li et al. (2021b)	4	1.34%	3.78%	67,142,657	-
FNO Li et al. (2021b)	6	0.90%	2.59%	100,705,409	0.7×
FNO Li et al. (2021b)	8	0.73%	2.09%	134,268,161	0.5×
TFNO (CP)	4	0.47%	1.20%	447,105	150×
TFNO (CP)	6	0.27%	0.74%	662,081	101×
TFNO (CP)	8	0.22%	0.59%	877,057	77×

Table 4: **Resolution invariance of *TFNO*.** Since the model is an operator, it is resolution invariant. In particular, here, we trained our model in resolution 128×128 and test it on unseen samples in various resolutions and show it generalizes, with virtually no loss of performance to higher resolutions unseen during training.

Method	128×128		256×256		512×512		1024×1024	
	L^2 **error**	H^1 **error**	L^2 **error**	H^1 **error**	L^2 **error**	H^1 **error**	L^2 **error**	H^1 **error**
CP *TFNO*	0.3%	0.87%	0.3%	0.93%	0.3%	0.93%	0.3%	0.93%
CP *MG-TFNO*	0.49%	1.2%	0.49%	1.3%	0.49%	1.5%	0.49%	1.6%

4.4 Ablation studies

In this section, we further study the properties of our model through ablation studies. We first look at how *TFNO* suffers less from overfitting thanks to the low-rank constraints before comparing its performance with various tensor decompositions. Finally, we perform ablation studies for our multi-grid domain decomposition on Burger's equation.

4.4.1 Resolution invariance

TFNO is resolution invariant, meaning that it can be trained on one resolution and tested on a different one. To illustrate this, we show zero-shot super-resolution results: we trained our best model (Table 4.4.4) on images of resolution 128×128 and tested it on unseen samples at higher resolutions (256×256 and 512×512), Table 4. As can be seen, our method does as well on unseen, higher-resolution unseen testing samples as it does on the training resolution, confirming the resolution invariance property of our neural operator.

4.4.2 Training on higher-resolution with Multi-grid

One important advantage of our multi-grid domain decomposition is that it enables training much larger models on large inputs by distributing over patches. We demonstrate this, by training on larger resolution (512x512 discretization) and using the largest FNO and TFNO that fits in memory, on a V100 GPU. For the original FNO, this corresponds to a width of 12, first row in table 5. We then compare its performance with the multigrid approach with a neural operator as large as fits into the same V100 GPUs i.e. each width in the table has been optimized to be as large as memory allows. As we can see, our approach allows to fit a larger model and reaches a much lower relative L^2 error.

4.4.3 Overfitting and Low-Rank Constraint

Here, we show that lower ranks (higher compressions) lead to reduced overfitting. In Figure 5, we show the training and testing H^1 errors for our TOP with Tucker decomposition at varying compression ratios (2x, 49x and 172x). We can see how, while the test error does not vary much, the gap between training and test errors reduces as we decrease the rank. As we can see, while being the most flexible, Tucker does not

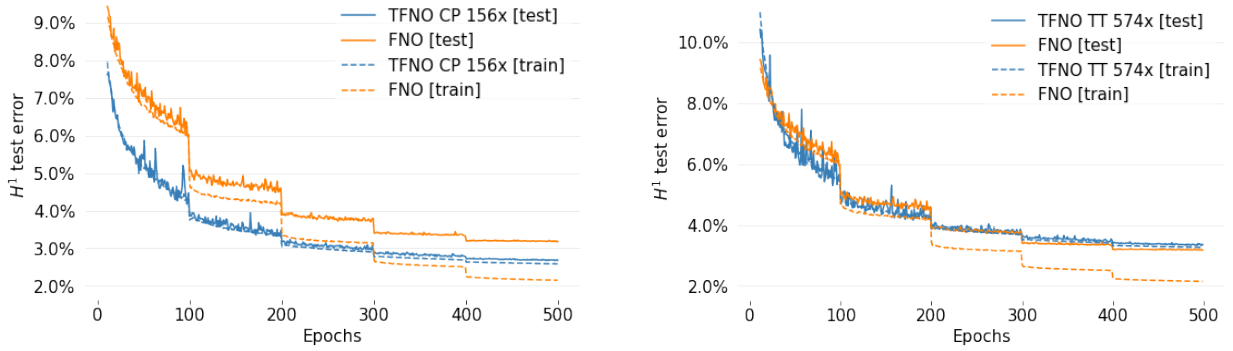
Table 5: **Training on 512x512**. Multi-grid domain decomposition allows us to fit larger models into memory by distributing patches in the domain space, thus reaching a lower relative error.

Model	Width	Patches	Padding	L^2 error
FNO	12	0	0	6.1
MG-FNO	42	4	70	2.9
MG-FNO	66	4	53	2.4
MG-FNO	88	16	40	1.8
Tucker MG-TFNO	80	16	46	1.3

Table 6: **Relative L^2 test error of our *MG-TFNO* approach for different tensor decompositions**. We empirically found that Tucker works best for small compression ratio, CP excels at large compression ratio ($\approx 100\times$) but becomes computationally heavy for smaller ones. TT tends to be unstable at low-compression ratios but preserves a good performance for extreme compression ratio ($> 500\times$).

Method	**L^2 test error**	**# Params**	**Model CR**
FNO Li et al. (2021b)	1.12%	67 M	0×
TFNO [Tucker]	0.37%	28 M	2.3×
TFNO [CP]	0.46%	808 K	83×
TFNO [TT]	1.18%	117 K	574×

perform as well at higher compression ratios. In those extreme cases, CP and Tensor-Train lead to lower errors.



(a) **Train VS Test error over time for a TOP with a CP factorization).**

(b) **Train VS Test error over time for a TOP with a TT factorization.**

Figure 10: **Train/test curve for a TOP-CP (10(a)) and TOP-TT (10(b))**

4.4.4 Tensor-Train and TOP

Our approach is independent of the choice of tensor decomposition. We already showed how Tucker is most flexible and works well across all ranks. We also showed that while memory demanding for high rank, a CP decomposition leads to better performance and low rank. Our method can also be used in conjunction with other decompositions, such as tensor-train. To illustrate this, we show the convergence behavior of TNO with a Tensor-Train decomposition for a compression ratio of 178, figure 10(b).

We also compare in Table 4.4.4 our *TFNO* with different tensor decompositions.

Table 7: **Ablation comparing the performance on the relative L^2 test error of our *MG-TFNO* approach, compared with its parts *TFNO* and MG-FNO and the regular FNO, on Navier-Stokes.** CR stands for compression ratio. Tensorization and multi-grid domain decomposition both individually improve performance while enabling space savings. The two techniques combined lead to further improvements, enabling large compression for both input and parameter, while outperforming regular *FNO*.

Method	L^2 test error	# Params	Model CR	Domain CR
FNO (Li et al., 2021b)	2.54%	58 M	0×	0×
TFNO [Tucker]	1.39%	41 M	1.5×	0×
TFNO [CP]	2.24%	130 K	**482**×	0×
MG-FNO	1.43%	58 M	0×	1.4×
MG-TFNO [Tucker]	**0.85**%	5.5 M	10×	1.78×
MG-TFNO [Tucker]	1.89%	5.5 M	10×	**7**×

4.4.5 Decomposing domain and weights: *MG-TFNO*.

Tensorization and multi-grid domain decomposition not only improve performance individually but their advantages compound and lead to a strictly better algorithm that scales well to higher-resolution data by decreasing the number of parameters in the model as well as the size of the inputs thereby improving performance as well as memory and computational footprint. Table 7 compares FNO with Tensorization alone, multi-grid domain decomposition alone, and our joint approach combining the two, *MG-TFNO*. In all cases, for α, we keep 40 Fourier coefficients for height and 24 for the width and use an operator width of 64. Our results imply that, under full parallelization, the memory footprint of the model's inference can be reduced by 7× and the size of its weights by 10× while also improving performance.

Consistently with our other experiments, we find that the tensor compression in the weights acts as a regularizer and improves performance across the board. Our results imply that, under full parallelization, the memory footprint of the model's inference can be reduced by 7× and its weight size by 10× while also improving performance.

4.4.6 Burgers' Equation

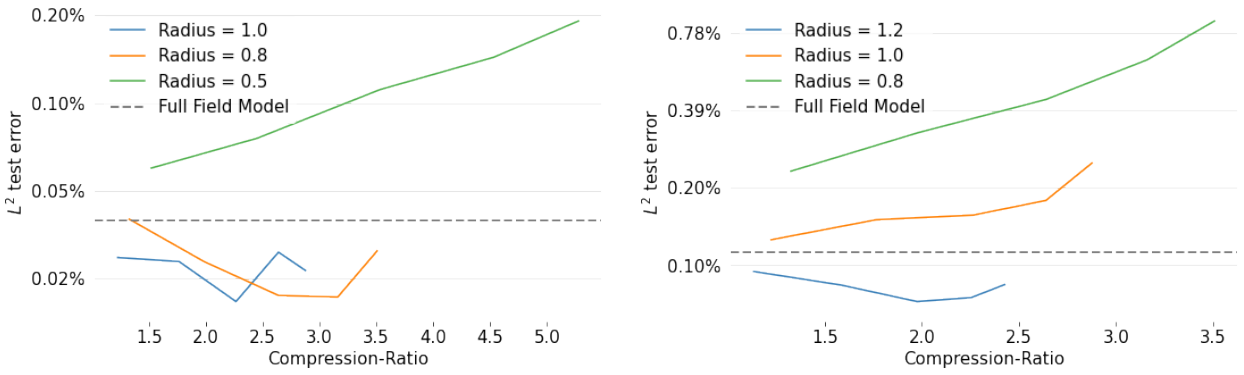


Figure 11: **Error on Burgers' equation with $T = 0.5$ (left) and $T = 1$ (right) as a function of domain compression ratio using standard domain decomposition without our multi-grid approach.** We evaluate the performance of the standard domain decomposition approach. The radius indicates the size, in physical space, of the padding added to each region.

We test the efficacy of the standard domain decomposition approach by training on two separate Burgers problems: one with a final time $T = 0.5$ and one with $T = 1$. As described in Section 3.6, we expect that

Model	# Params	GPU RAM (nvml)	Max GPU RAM (PyTorch)	Runtime
FNO	69 M	4.465 Gb	3.400 Gb	0.023 s
FFNO	1 M	5.465 Gb	4.493 Gb	0.043 s
UNO	79 M	5.048 Gb	3.341 Gb	0.029 s
TUNO [CP-fac]	344 K	4.952 Gb	3.347 Gb	0.026 s
TUNO [TUCKER-fac]	4 M	4.957 Gb	3.376 Gb	0.022 s
TUNO [TUCKER-fac]	836 K	4.860 Gb	3.234 Gb	0.023 s
TFNO [CP-fac]	167 K	4.476 Gb	3.416 Gb	0.019 s
TFNO [CP-fac]	282 K	4.776 Gb	3.678 Gb	0.025 s
TFNO [CP-fac]	514 K	5.289 Gb	4.209 Gb	0.031 s
TFNO [CP-rec]	514 K	6.840 Gb	5.792 Gb	0.025 s
TFNO [TT-fac]	1 M	4.604 Gb	3.473 Gb	0.022 s
TFNO [TT-rec]	1 M	5.254 Gb	4.169 Gb	0.031 s
TFNO [TUCKER-fac]	4 M	4.600 Gb	3.472 Gb	0.022 s
TFNO [TUCKER-fac]	724 K	4.355 Gb	3.297 Gb	0.020 s

Table 8: **Training**, Network width of 64

for $T = 1$, each region requires more global information thus significantly more padding need to be used in order to reach the same error. The results of Figure 11 indeed confirm this. The domain compression ratios needed for the approach to reach the performance of the full-field model are higher, indicating the need for incorporating global information. These results motivate our multi-grid domain decomposition approach.

4.5 Computational efficiency

In this section, we perform ablation studies to explore the impact of our proposed tensorization on memory consumption and training and inference times. The tensorized formulation enables an efficient implementation obtained by directly contracting the activations with the factors of the decomposition, allowing us to further scale the size of the models we can fit in the memory. However, this speedup depends heavily on the hardware and parameters. Going forward, we plan to further improve computational efficiency and memory savings.

To experimentally measure the gains, we created an FNO keeping all the modes, and we benchmarked two network widths, 64 (Tables 4.5 and 4.5) and 128 (Tables 4.5 and 4.5), for a batch size of 16.

In both cases, we measured the memory usage of both a regular FNO and our proposed Tucker TFNO, on a single RTX 3080 GPU, for a single forward pass (inference) and forward-backward pass (training), using [1] pynvml (`pynvml.nvmlDeviceGetMemoryInfo`) and [2] PyTorch (`torch.cuda.max_memory_allocated`).

4.6 Super-evaluation

In this section, we show some examples of input forcing functions, the corresponding ground-truth target function and the function predicted by our proposed *MG-TFNO*.

In particular, we visualize the predictions from our proposed method for two different settings: in both cases we train the model on a fixed size of 128×128 input functions but we either test with i) the same resolution or ii) with a new, much higher resolution, unseen during training (*zero-shot super-evaluation*).

Training and evaluating on the same resolution: In this setting, we train our proposed method on the full training set of functions discretized at resolution 128×128 and test on new unseen samples at the *same* resolution, Figure. 4.6.

Our model correctly learns the target function.

Model	# Params	GPU RAM (nvml)	Max GPU RAM (PyTorch)	Runtime
FNO	69 M	1.967 Gb	0.832 Gb	0.019 s
FFNO	1 M	1.522 Gb	0.653 Gb	0.041 s
UNO	79 M	2.133 Gb	1.188 Gb	0.021 s
TUNO [CP-fac]	344 K	1.840 Gb	0.896 Gb	0.021 s
TUNO [TUCKER-fac]	4 M	1.840 Gb	0.910 Gb	0.020 s
TUNO [TUCKER-fac]	836 K	1.784 Gb	0.898 Gb	0.020 s
TFNO [CP-fac]	167 K	1.709 Gb	0.575 Gb	0.017 s
TFNO [CP-fac]	282 K	1.709 Gb	0.575 Gb	0.018 s
TFNO [CP-fac]	514 K	1.709 Gb	0.576 Gb	0.021 s
TFNO [CP-rec]	514 K	1.725 Gb	0.790 Gb	0.027 s
TFNO [TT-fac]	1 M	1.713 Gb	0.579 Gb	0.017 s
TFNO [TT-rec]	1 M	1.528 Gb	0.580 Gb	0.021 s
TFNO [TUCKER-fac]	4 M	1.709 Gb	0.588 Gb	0.017 s
TFNO [TUCKER-fac]	724 K	1.711 Gb	0.577 Gb	0.016 s

Table 9: **Inference**, Network width of 64

Model	# Params	GPU RAM (nvml)	Max GPU RAM (PyTorch)	Runtime
FNO	277 M	2.752 Gb	1.926 Gb	0.039 s
FFNO	5 M	2.358 Gb	1.294 Gb	0.080 s
UNO	96 M	2.198 Gb	1.252 Gb	0.023 s
TUNO [CP-fac]	381 K	1.717 Gb	0.896 Gb	0.022 s
TUNO [TUCKER-fac]	5 M	1.860 Gb	0.913 Gb	0.021 s
TUNO [TUCKER-fac]	1 M	1.719 Gb	0.898 Gb	0.022 s
TFNO [CP-fac]	314 K	1.967 Gb	0.896 Gb	0.028 s
TFNO [CP-fac]	496 K	1.844 Gb	0.897 Gb	0.029 s
TFNO [CP-fac]	858 K	1.911 Gb	0.898 Gb	0.033 s
TFNO [CP-rec]	858 K	2.885 Gb	1.566 Gb	0.062 s
TFNO [TT-fac]	2 M	1.914 Gb	0.902 Gb	0.030 s
TFNO [TT-rec]	2 M	2.366 Gb	1.102 Gb	0.046 s
TFNO [TUCKER-fac]	14 M	2.022 Gb	0.950 Gb	0.028 s
TFNO [TUCKER-fac]	3 M	1.967 Gb	0.905 Gb	0.026 s

Table 10: **Inference**, Network width of 128

Model	# Params	GPU RAM (nvml)	Max GPU RAM (PyTorch)	Runtime
FNO	277 M	6.362 Gb	5.435 Gb	0.038 s
FFNO	5 M	9.805 Gb	8.698 Gb	0.082 s
UNO	96 M	5.054 Gb	3.562 Gb	0.026 s
TUNO [CP-fac]	381 K	4.966 Gb	3.504 Gb	0.023 s
TUNO [TUCKER-fac]	5 M	5.095 Gb	3.600 Gb	0.022 s
TUNO [TUCKER-fac]	1 M	4.950 Gb	3.480 Gb	0.022 s
TFNO [CP-fac]	314 K	6.033 Gb	4.678 Gb	0.029 s
TFNO [CP-fac]	496 K	6.485 Gb	4.939 Gb	0.032 s
TFNO [CP-fac]	858 K	6.755 Gb	5.477 Gb	0.038 s
TFNO [CP-rec]	858 K	11.852 Gb	10.721 Gb	0.059 s
TFNO [TT-fac]	2 M	6.513 Gb	4.994 Gb	0.031 s
TFNO [TT-rec]	2 M	9.476 Gb	8.503 Gb	0.044 s
TFNO [TUCKER-fac]	14 M	6.889 Gb	5.480 Gb	0.031 s
TFNO [TUCKER-fac]	3 M	6.157 Gb	4.858 Gb	0.030 s

Table 11: **Training**, Network width of 128

Zero-shot super-evaluation: In this setting, we take the same model as above, trained on the full training set of functions discretized at resolution 128×128.

However, this time, we give our model as input new unseen samples at *new, unseen* resolution of 1024×1024, much higher than the training one, Figure. 4.6. The model is able to use high-frequency information in the new resolution and produce results as good as in the training resolution.

A natural question is whether there is an advantage compared to just downscaling the input, feeding it at the training resolution to the model and reupsampling it. In other words, is the model able to leverage high-frequency information *it has not seen* during training.

To answer this question, we generated 1000 samples of the Navier-Stokes equation, in the same setting as previously introduced, but with resolution 1024×1024. We trained our proposed *TFNO* on the full training set of samples at resolution 128×128. We then evaluated the model on the high resolution, unseen, test samples of resolution 1024×1024 in two settings:

1. [**upscaling**] In that case, we downscale the input to the training resolution (128×128), predict the solution which we then re-upsample to the target high-resolution (1024×1024) for evaluation. In that case, we get a relative l2 error of 4.5%. As expected, the model is not able to leverage high-frequency information since that is lost when downsampling.

2. [**zero-shot super-evaluation**] In that case, we leverage the fact that our proposed method is a neural operator Li et al. (2021b) and therefore can take as input functions at any regular discritization. We directly feed it the samples at high-resolution (1024×1024) and use the prediction (funtions on the same discretization and resolution) for evaluation. In that case, we get a relative l2 error over 10 times better of only 0.4%.

This enables us to empirically verify the ability of our neural operator to leverage high-frequency information unseen during training.

5 Conclusion

In this work, we introduced i) a novel tensor operator (*TFNO*) as well as a multi-grid domain decomposition approach which together form *MG-TFNO*, ii) an operator model that outperforms the *FNO* with a fraction of the parameters and memory complexity requirements, and iii) architectural improvements to the *FNO*. Our method scales better, generalizes better, and requires fewer training samples to reach the same performance;

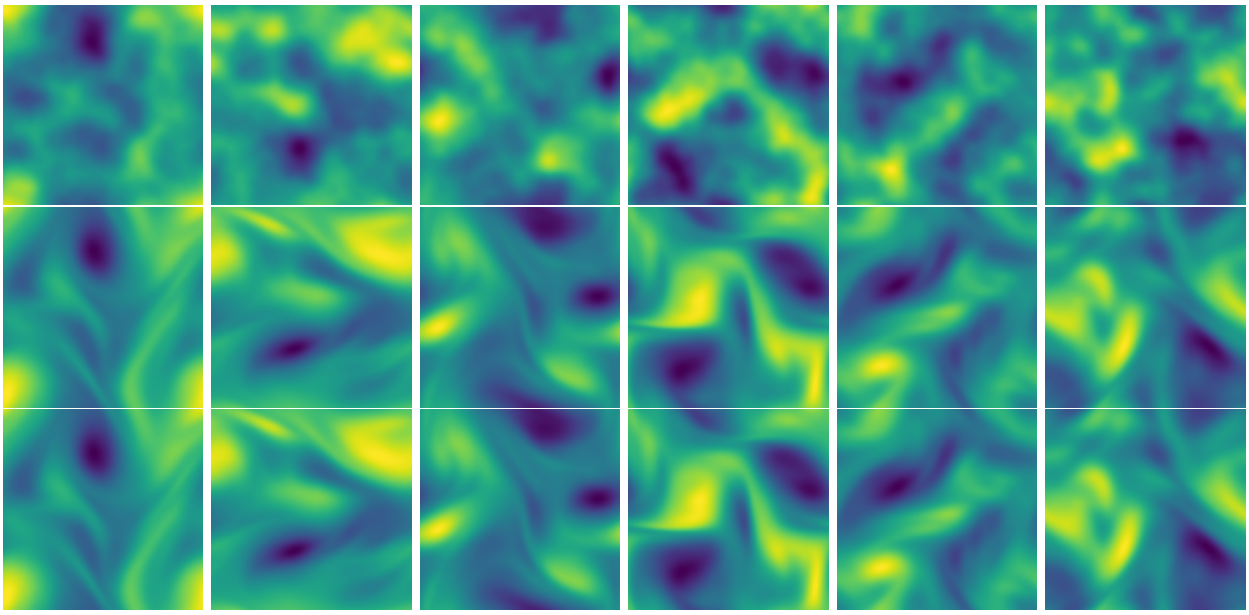


Figure 12: **Visualization of evaluation of our model with samples from the Navier-Stokes equation at resolution** 128×128**.** We show the input initial conditions (top-row), the corresponding target ground-truth function (middle-row), and the prediction function from our proposed model (bottom-row). Our model has learned to correctly predict the target function and can solve the Navier-Stokes equation with very high accuracy for *any initial conditions*.

while the multi-grid domain decomposition enables parallelism over huge inputs. This paves the way to applications on very high-resolution data and in our future work, we plan to deploy *MG-TFNO* to large-scale weather forecasts for which existing deep learning models are prohibitive.

References

Jonas Adler and Ozan Oktem. Solving ill-posed inverse problems using iterative deep neural networks. *Inverse Problems*, nov 2017.

Nathan Albin and Oscar P. Bruno. A spectral fc solver for the compressible navier–stokes equations in general domains i: Explicit time-stepping. *Journal of Computational Physics*, 230(16):6248–6270, 2011.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, pp. 1–21, 2019.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *arXiv preprint arXiv:2005.03180*, 2020.

Mackenzie L Blanusa, Carla J López-Zurita, and Stephan Rasp. The role of internal variability in global climate projections of extreme events. *arXiv preprint arXiv:2208.08275*, 2022.

Adrian Bulat, Jean Kossaifi, Georgios Tzimiropoulos, and Maja Pantic. Incremental multi-domain learning with network latent tensor factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 10470–10477, 2020a.

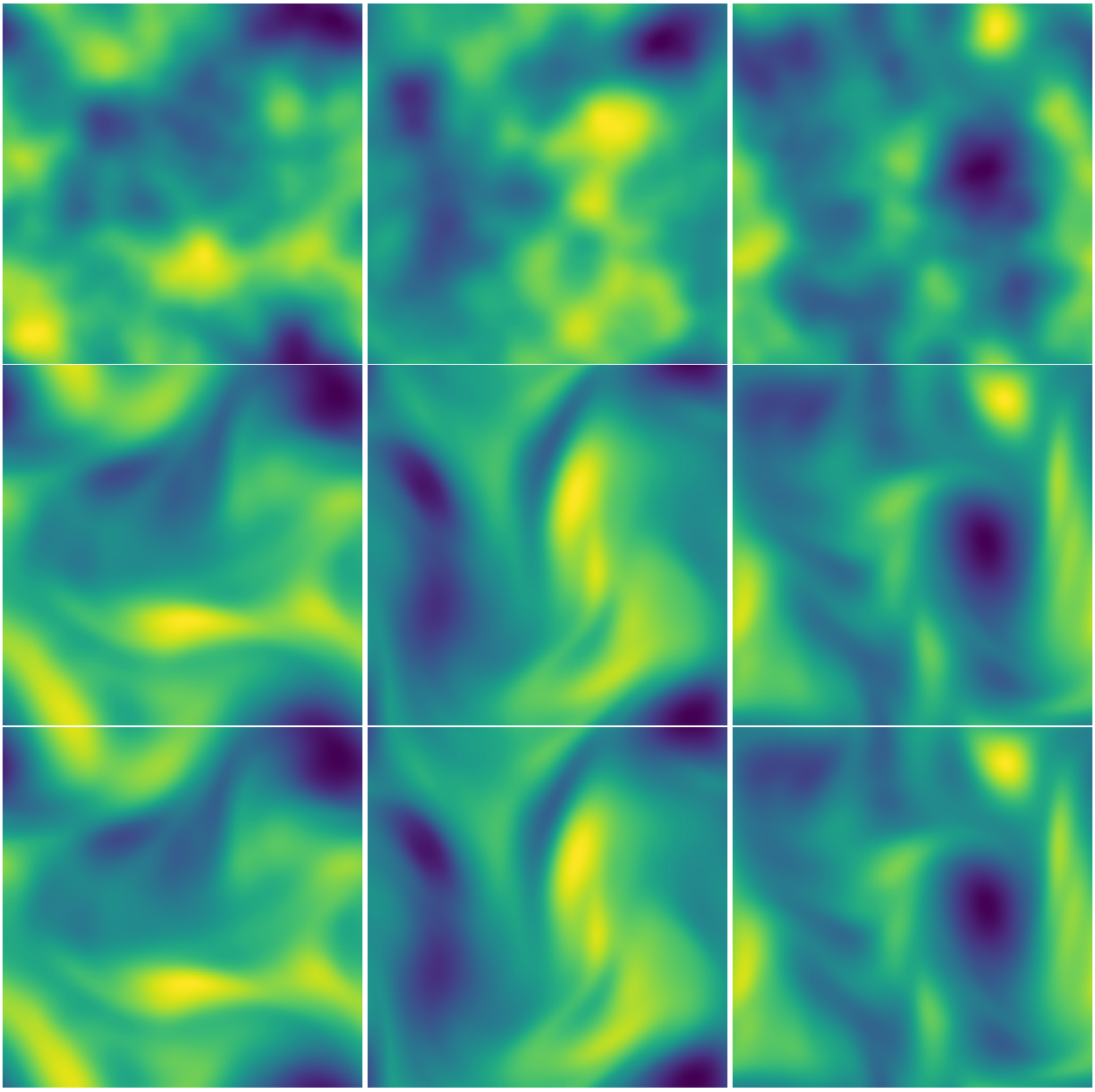


Figure 13: **Visualization of super-evaluation with our proposed models for samples from the Navier-Stokes equation at resolution** 1024×1024**.** Here, we trained our proposed method with rank-128 CP factorization, on data of resolution 128×128. However, we evaluate on data at a much higher resolution of 1024×1024. This is possible because our approach is a neural operator that can be evaluated and queried on any arbitrary resolution (here on a regular grid as it leverages the Fast Fourier Transform). Even though our model has never seen the higher-frequency information from the Navier-Stokes equation obtained directly from a solver at high-resolution, it is able to generalize. We show the input initial conditions (top-row), the corresponding target ground-truth function (middle-row), and the prediction function from our proposed model (bottom-row). Our model has learned to correctly predict the target function and can solve the Navier-Stokes equation with very high accuracy for *any initial conditions* and *new unseen resolution*.

Adrian Bulat, Jean Kossaifi, Georgios Tzimiropoulos, and Maja Pantic. Toward fast and accurate human pose estimation via soft-gated skip connections. In *2020 15th IEEE International Conference on Automatic Face & Gesture Recognition*, 2020b.

Tony F. Chan and Tarek P. Mathew. Domain decomposition algorithms. *Acta Numerica*, 3:61–143, 1994.

Gary J. Chandler and Rich R. Kerswell. Invariant recurrent solutions embedded in a turbulent two-dimensional kolmogorov flow. *Journal of Fluid Mechanics*, 722:554–595, 2013.

Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. Multi-head attention: Collaborate instead of concatenate. *arXiv preprint arXiv:2006.16362*, 2020.

R. Courant, K. Friedrichs, and H. Lewy. Uber die partiellen differenzengleichungen der mathematischen physik. *Mathematische annalen*, 100(1):32–74, 1928.

Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pp. 4690–4721. PMLR, 2022.

Maarten De Hoop, Daniel Zhengyu Huang, Elizabeth Qian, and Andrew M Stuart. The cost-accuracy trade-off in operator learning with neural networks. *arXiv preprint arXiv:2203.13181*, 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Maksymilian Dryja and Olof B. Widlund. Chapter 16 - some domain decomposition algorithms for elliptic problems. In David R. Kincaid and Linda J. Hayes (eds.), *Iterative Methods for Large Linear Systems*, pp. 273–291. Academic Press, 1990.

Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A Tchelepi, Philip Marcus, Mr Prabhat, Anima Anandkumar, et al. Meshfreeflownet: A physics-constrained deep continuous space-time super-resolution framework. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2020.

Lawrence C. Evans. *Partial differential equations*. American Mathematical Society, 2010.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in Neural Information Processing Systems*, 34:24048–24062, 2021.

Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Philip Blagoveschensky, Andrzej Cichocki, and Ivan Oseledets. Automated multi-stage compression of neural networks. Oct 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pp. 630–645. Springer, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Majid Janzamin, Rong Ge, Jean Kossaifi, Anima Anandkumar, et al. Spectral learning on matrices and tensors. *Found. and Trends® in Mach. Learn.*, 12(5-6):393–536, 2019.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. 2016.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, 2009.

Jean Kossaifi. Tensorly-torch. `https://github.com/tensorly/torch`, 2021.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research (JMLR)*, 20(26), 2019.

Jean Kossaifi, Antoine Toisoul, Adrian Bulat, Yannis Panagakis, Timothy M Hospedales, and Maja Pantic. Factorized higher-order CNNs with an application to spatio-temporal emotion estimation. pp. 6060–6069, 2020.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22(290):1–76, 2021a.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021b.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. 2015.

Martin Leutbecher and Tim N Palmer. Ensemble forecasting. *Journal of computational physics*, 227(7): 3515–3539, 2008.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020a.

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

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Markov neural operators for learning chaotic systems. *arXiv preprint arXiv:2106.06898*, 2021a.

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 *International Conference on Learning Representations*, 2021b.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021c.

Burigede Liu, Nikola Kovachki, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, Andrew M Stuart, and Kaushik Bhattacharya. A learning-based multiscale method and its application to inelastic impact problems. *Journal of the Mechanics and Physics of Solids*, 158:104668, 2022.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

S. F. McCormick. Multigrid methods for variational problems: General theory for the v- cycle. *SIAM Journal on Numerical Analysis*, 22(4):634–643, 1985.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. pp. 442–450, 2015.

I. V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, September 2011.

Yannis Panagakis, Jean Kossaifi, Grigorios G. Chrysos, James Oldfield, Mihalis A. Nicolaou, Anima Anandkumar, and Stefanos Zafeiriou. Tensor methods in computer vision and deep learning. *Proceedings of the IEEE*, 109(5):863–890, 2021. doi: 10.1109/JPROC.2021.3074329.

Christos Papadopoulos, Yannis Panagakis, Manolis Koubarakis, and Mihalis Nicolaou. Efficient learning of multiple nlp tasks via collective weight factorization on bert. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 882–890, 2022.

Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Trans. Intell. Syst. and Technol. (TIST)*, 8 (2):1–44, 2016.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

Catherine E. Powell, Gabriel Lord, and Tony Shardlow. *An Introduction to Computational Stochastic PDEs*. Texts in Applied Mathematics. Cambridge University Press, United Kingdom, 1 edition, August 2014. ISBN 9780521728522.

Md Ashiqur Rahman, Manuel A Florez, Anima Anandkumar, Zachary E Ross, and Kamyar Azizzadenesheli. Generative adversarial neural operators. *arXiv preprint arXiv:2205.03017*, 2022a.

Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022b.

Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *Transactions Signal Processing*, 65(13):3551–3582, 2017.

Julia Slingo and Tim Palmer. Uncertainty in weather and climate prediction. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1956):4751–4767, 2011.

Roger Temam. *Infinite-dimensional dynamical systems in mechanics and physics*. Applied mathematical sciences. Springer-Verlag, New York, 1988.

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=tmIiMPl4IPa`.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

Yan Yang, Angela F Gao, Jorge C Castellanos, Zachary E Ross, Kamyar Azizzadenesheli, and Robert W Clayton. Seismic wave propagation and inversion with neural operators. *The Seismic Record*, 1(3):126–134, 2021.

Yan Yang, Angela F Gao, Jorge C Castellanos, Zachary E Ross, Kamyar Azizzadenesheli, and Robert W Clayton. Accelerated full seismic waveform modeling and inversion with u-shaped neural operators. *arXiv preprint arXiv:2209.11955*, 2022.

Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 2018. ISSN 0021-9991.