Transform Once Efficient Operator Learning in Frequency Domain

Michael Poli [*1] **Stefano Massaroli** [*2] **Federico Berto** [*3]
Jinkyoo Park [3] **Tri Dao** [1] **Christopher Ré** [1] **Stefano Ermon** [1]

Abstract

Spectrum analysis provides one of the most effective paradigms for information–preserving dimensionality reduction in data: often, a simple description of naturally occurring *signals* can be obtained via few terms of periodic basis functions. Neural operators designed for frequency domain learning – *frequency domain models* (FDMs) – are based on complex–valued transforms i.e. *Fourier Transforms* (FT), and layers that perform computation on the spectrum and input data separately. This design introduces considerable computational overhead: for each layer, a forward and inverse FT. Instead, this work introduces a blueprint for frequency domain learning through a single transform: *transform once* (`T1`). Our results noticeably streamline the design process of FDMs, pruning redundant transforms, and leading to speedups of 3 x to 10 x that increase with data resolution and model size. We perform extensive experiments on learning to solve partial differential equations, including incompressible Navier–Stokes, turbulent flows around airfoils and high-resolution video of smoke dynamics. `T1` models improve on the test performance of SOTA FDMs while requiring significantly less computation, with over 20% reduction in predictive error across tasks.

1. Introduction

Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry. (Feynman, 2017)

Naturally occurring *signals* are often sparse when projected on periodic basis functions (Strang, 1999). Central to recently–introduced instances of frequency–domain neural operators (Li et al., 2020; Tran et al., 2021), which we refer to as *frequency–domain models* (FDMs), is the idea of learning to modify specific frequency components of inputs to obtain a desired output in data space. With a hierarchical structure that blends learned transformations on frequency domain coefficients with regular convolutions, FDMs are able to effectively approximate global, long–range dependencies in higher resolution signals without requiring prohibitively deep architectures. Yet, state–of–the–art variants of these models suffer from several drawbacks:

1. **Poor parameter scaling:** FDMs employ complex–valued transforms, followed by parameter–inefficient complex–valued layers[1].
2. **Slow inference:** every layer of an FDM performs a forward and inverse frequency domain transform, introducing a considerable computational overhead.
3. **Suboptimal initialization:** parameter initialization schemes and layers devised to learn directly in data space can be highly suboptimal when introduced without modifications to FDMs.

Although some attempts to improve performance (Gupta et al., 2021) or alleviate the above–mentioned limitations (Tran et al., 2021), scaling FDMs to larger data resolutions and model sizes remains fundamentally challenging. In example, instantiating a medium–sized FDM (tens of layers with a non–trivial number of channels) requires a parameter count in the hundred millions.

In this work, we start by posing the question:

To reap the benefits of learning on frequency domain representations, is it necessary to construct hierarchical deep models that perform forward and inverse frequency transforms at each layer?

We provide the answer in *Transform Once* (`T1`), a model that builds representations directly in frequency domain,

[*]Equal contribution [1]Stanford University [2]Mila, Quebec [3]KAIST. Correspondence to: Michael Poli <poli@stanford.edu>.

2nd AI4Science Workshop at the 39^{th} International Conference on Machine Learning (ICML), 2022. Copyright 2022 by the author(s).

[1]A single complex–valued parameter requires separate real and imaginary parts.

after a *single* forward transform. Each aspect of `T1` addresses a specific limitation of existing FDMs:

1. **Favourable scaling:** `T1` employs a single real–valued transform (*Discrete Cosine Transform* – DCT), which reduces parameter scaling overhead w.r.t complex–valued layers.
2. **Fast:** by performing a single forward transform and optimizing directly on DCT coefficients of target data, `T1` iterations are *at least* 3x to 10x faster. When scaling to larger models and higher resolutions, the relative speedups increase as the overhead of each transform grows.
3. **Enhanced compatibility:** removing redundant transforms streamlines the design space for `T1` architectures compared to existing FDMs, allowing direct introduction of optimized layers developed for other applications e.g. UNets (Ronneberger et al., 2015).

In §2.1 we provide a (short) history on frequency domain approaches in deep learning, followed by background on FDMs in §2.3. In Section §3, we describe how to train `T1` directly in frequency domain, in §3.1 we motivate the choice of DCT, in §3.2 we derive a loss decomposition for reduced–order FDMs and in §3.3 we introduce a simple variance–preserving weight initialization scheme for all FDMs. Finally, in §4 we evaluate `T1` on a suite of benchmarks related to learning solution operators for a variety of dynamics: incompressible Navier–Stokes, flow around different airfoil geometries, and high–resolution videos of turbulent smoke plumes (Eckert et al., 2019).

Across tasks, `T1` is $3\times$ to $10\times$ faster and reduces predictive errors by 20% on average. Training `T1` models on high resolution videos (600 x 1062) of turbulent dynamics is significantly faster, requiring 5 hours instead of 32 hours (FNOs) for the same number of iterations.

2. Related Work and Background

2.1. Learning and Frequency Domain: A Short History

Links between frequency–domain signal processing and neural network architectures have been explored for decades, starting with the original CNN designs (Fukushima and Miyake, 1982). (Mathieu et al., 2013; Rippel et al., 2015) proposed replacing convolutions in pixel space with element–wise multiplications in Fourier domain. In the context of learning to solve *partial differential equations* (PDEs), *Fourier Neural Operators* (FNOs) (Li et al., 2020) popularized the state–of–the–art FDM layer structure: forward transform $\to$ learned layer $\to$ inverse transform. Similar architectures had been previously proposed for generic image classification tasks in (Pratt et al., 2017; Chi et al., 2020). Modifications to the basic FNO recipe are provided in (Tran et al., 2021; Guibas et al., 2021; Wen et al., 2022). A frequency domain representation of convolutional weights has also been used for model compression (Chen et al., 2016). Fourier features of input *domains* and periodic activation functions play important roles in deep implicit representations (Sitzmann et al., 2020; Dupont et al., 2021; Poli et al., 2022) and general–purpose PERCEIVERS (Jaegle et al., 2021).

2.2. Learning to Solve Differential Equations

A variety of deep learning approaches have been developed to solve differential equations: neural operators and physics–informed networks (Long et al., 2018; Raissi et al., 2019; Lu et al., 2019; Karniadakis et al., 2021), specialized architectures (Wang et al., 2020; Lienen and Günnemann, 2022), hybrid neural–numerical methods (Poli et al., 2020; Kochkov et al., 2021; Mathiesen et al., 2022), and FDMs (Li et al., 2020; Tran et al., 2021), the focus of this work.

2.3. Frequency–Domain Models

Let $\mathcal{D}_n$ (n-space) to be the set of real–valued discrete signals[2] of resolution N. Our objective is to develop efficient neural networks to process discrete signals $x \in \mathcal{D}_n$,

$$x_0, x_1, \dots, x_{N-1}, \quad x_n \in \mathbb{R}.$$

We define a layer of FDMs mapping x to an output signal $\hat{y} \in \mathcal{D}_n$ as the structured operator:

$$\begin{aligned} X &= \mathcal{T}(x) && \text{Forward Transform} \\ \hat{X} &= f_\theta(X) && \text{Learned Map} \\ \hat{x} &= \mathcal{T}^{-1}(\hat{X}) && \text{Inverse Transform} \\ \hat{y} &= \hat{x} + g(x) && \text{Residual} \end{aligned} \tag{1}$$

$$\begin{tikzcd} x \arrow[r] \arrow[d, no head] & \mathcal{T} \arrow[r, "X"] & f_\theta \arrow[r, "\hat{X}"] & \mathcal{T}^{-1} \arrow[r, "\hat{x}"] & \oplus \arrow[r] & \hat{y} \\ {} \arrow[rr] & & g \arrow[rr, no head] & & {} \arrow[u] & \end{tikzcd}$$

where $\mathcal{T}$ is an orthogonal (possibly *complex*) linear operator. We denote the $\mathcal{T}$–transformed n-space with $\mathcal{D}_k$ (k-space) so that $\mathcal{T} : \mathcal{D}_n \to \mathcal{D}_k$. Typically, we assume $\mathcal{T}$ to be a *Fourier–type* transform[3] (Oppenheim, 1999, Chapter 8) so that the k-space corresponds to the *frequency domain* and its elements form the *spectrum* of the input signal x.

The learned parametric map $f_\theta : \mathcal{D}_k \to \mathcal{D}_k$ is the stem of a FDM layer: it maps the k-space into itself and is typ-

[2]For clarity of exposition, models and algorithms proposed in the paper are introduced without loss of generality for one–dimensional scalar signals (i.e. $\mathcal{D}_n \equiv \mathbb{R}^n$).

[3]e.g. *discrete Fourier transform* (DFT), *discrete cosine transform* (DCT), etc.

ically chosen to be rank–deficient in the linear case, e.g. $f_\theta(X) = S_m^\top A(\theta) S_m X,\ A(\theta) \in \mathbb{C}^{m\times m}$ $(m \le N)$. The matrix $S_m \in \mathbb{R}^{n\times m}$ selects m desired elements of X, setting the rest to zero. In the case of frequency domain transforms, this allows (1) to preserve or modify only specific frequencies of the input signal x.

Residual connections or residual convolutions g (Li et al., 2020; Wen et al., 2022) are optionally added to reintroduce frequency components filtered by the S_m. An FDM mixes global transformations applied to coefficients of the chosen transform to local transformations g i.e. convolutions with finite kernel sizes. To ensure that such models can approximate generic nonlinear functions, nonlinear activations are introduced after each inverse transform.

Fourier Neural Operators Layers of the form (1) appear in recent FDMs such as *Fourier Neural Operators* (FNOs) (Li et al., 2020) and variants (Tran et al., 2021; Guibas et al., 2021; Wen et al., 2022).

In example, an FNO is recovered from (1) by letting $\mathcal{T}$ be a *Discrete Fourier Transform* (DFT)

$$\hat{x} = \mathcal{T}^{-1} \circ f_\theta \circ \mathcal{T}(x) = W^* S_m^\top A(\theta) S_m W x$$

where $W \in \mathbb{C}^{N\times N}$ is the standard N-dimensional DFT matrix and W^* its conjugate transpose. The *Discrete Fourier Transforms* (DFTs) is a natural choice of $\mathcal{T}$ as it can be computed in $O(N \log N)$ via *Fast Fourier Transform* (FFT) algorithms (Oppenheim, 1999, Chapter 9.2).

We identify two major limitations of FDMs in the form (1); each layer performs $\mathcal{T}$ and $\mathcal{T}^{-1}$, and DFTs are complex–valued, requiring parameter–inefficient complex–valued layers.

With T1, we aim to develop an FDM that does not require more than a single $\mathcal{T}$, while preserving or improving on predictive accuracy. Ideally, the transform in T1 should be (1) real–valued, to avoid introducing parameter scaling overhead, (2) universal, to allow the representation of target signals, and (3) approximately sparse or structured, to allow dimensionality reduction.

$$\begin{tikzcd}
 & \hat{y} & \\
x \arrow[ur, "g"] \arrow[d, shift left, "\mathcal{T}^{-1}"] & & \hat{x} \arrow[ul] \arrow[d, shift left, "\mathcal{T}^{-1}"] \\
X \arrow[u, shift left, "\mathcal{T}"] \arrow[rr, "f_\theta"'] & & \hat{X} \arrow[u, shift left, "\mathcal{T}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
x \arrow[d, shift right, "W"'] & \hat{x} \arrow[d, shift right, "W"'] \\
X \arrow[u, shift right, "W^*"'] \arrow[r, "S_m^\top A(\theta) S_m"'] & \hat{X} \arrow[u, shift right, "W^*"']
\end{tikzcd}$$

Commutative diagrams for FDM layers (1) and linear FNOs (frequency domain part).

3. Transform Once: The T1 Recipe

With T1, we introduce major modifications to the way FDMs are designed and optimized. In particular, T1 is defined, inferred and trained directly in the frequency domain with only a **single** direct transform required to process data. Hence follows the name: *transform once* (T1).

Direct learning in the frequency domain Consider two signals $x \in \mathcal{D}_n, y \in \mathcal{D}_n$ and suppose there exists a function $\varphi : \mathcal{D}_n \to \mathcal{D}_n$ mapping x to y, i.e.

$$y = \varphi(x).$$

Then, there must also exist another function $\psi : \mathcal{D}_k \to \mathcal{D}_k$ that relates the spectra of the two signals, i.e. $Y = \psi(X)$ being $X = \mathcal{T}(x)$ and $Y = \mathcal{T}(y)$. In particular,

$$\varphi(x) = \mathcal{T}^{-1} \circ \psi \circ \mathcal{T}(x) \quad \Leftrightarrow \quad \mathcal{T} \circ \varphi(x) = \psi \circ \mathcal{T}(x)$$

It follows that, from a learning perspective, we can aim to approximate ψ directly in the k-space rather than φ in the n-space. To do so, we define a learnable parametric function $f_\theta : \mathcal{D}_k \to \mathcal{D}_k$ and train it to minimize the approximation error J_θ of the output signal spectrum Y in the k–space. Given a distribution $p(x)$ of input signals, T1 is characterized by the following nonlinear program

$$\begin{aligned}
\min_\theta \quad & \mathbb{E}_x \left[\|\mathcal{T}(y) - \hat{Y}\| \right] \\
\text{subject to} \quad & \hat{Y} = f_\theta \circ \mathcal{T}(x) \\
& x \sim p(x) \\
& y = \varphi(x)
\end{aligned} \tag{2}$$

$$\begin{tikzcd}
x \arrow[r] & \mathcal{T} \arrow[r, dash] & X \arrow[r] & f_\theta \arrow[r, "\hat{Y}"] & J_\theta \\
y \arrow[r] & \mathcal{T} \arrow[r, dash] & Y \arrow[urr] & &
\end{tikzcd}$$

If $\mathcal{T}$ is a DFT, the above turns out to be a close approximation (or equivalent, depending on the function class of f_θ) to the minimization of $\|y - \hat{y}\|$ in n-space by the *Parseval-Plancherel identity*.

Theorem 3.1 (Parseval-Plancherel Identity (Stein and Shakarchi, 2011, pp. 223))**.** *Let $\mathcal{T}$ be the normalized DFT. Given a signal $v \in \mathcal{D}_n$ and its transform $V = \mathcal{T}(v)$, it holds $\|v\| = \|V\|$.*

This result also applies to any other norm–preserving transform $\mathcal{T}$, e.g. a normalized type–II DCT (Oppenheim, 1999, pp. 679). For the linear transforms considered in this work, $\mathcal{T}(x) = Wx$, $W \in \mathbb{C}^{N\times N}$, condition for Th. 3.1 to hold is W to be orthonormal, i.e. $W^* W = \mathbb{I}$.

Note that T1 retains, in principle, the same universal approximation properties of FNOs (Kovachki et al., 2021) as f_θ is allowed to operate on the entirety of the input spectrum. Given enough capacity, f_θ can arbitrarily approximate ψ, implicitly reconstructing φ via $\mathcal{T}^{-1} \circ f_\theta \circ \mathcal{T}$.

Speedup measurements We provide a concrete example of the effect of pruning redundant transforms on computational costs. We measure wall–clock inference time speedups of depth d `T1`

$$\texttt{T1}(x) := f_d \circ \cdots \circ f_2 \circ f_1 \circ \mathcal{T}(x)$$

over an equivalent depth d FNO with layers (1). The only difference concerns the application of transforms between layers.

Figure 3.1 provides the speedups on two–dimensional signals: on the left, we fix model depth $d = 6$ and investigate the scaling in signal width (i.e. number of channels) and signal resolution. On the right, we fix signal width to be 32 and visualize the interaction of model depth and signal resolution. For common experimental settings e.g. resolutions of 64 or 128, 6 layers and width 32, `T1` is at least 10 x faster than other FDMs. It will later be shown (§4) that `T1` also preserves or improves on predictive accuracy of other FDMs across tasks.

When `T1` is not preceded by online preprocessing steps for inputs x, such as other neural networks or randomized data augmentations, the transform on $\mathcal{T}(x)$ can be done once on the dataset, amortizing the cost over training epochs, and increasing the speed of `T1` further.

3.1. Choosing the right transform

The transform $\mathcal{T}$ in `T1` is chosen to be in the class of *Discrete Cosine Transforms* (DCTs) (Ahmed et al., 1974; Strang, 1999), in particular the normalized DCT–II,

$$X_k = 2\sum_{n=0}^{N-1} \beta_k x_n \cos\frac{(2n+1)k\pi}{2N}$$

$$\beta_k = \begin{cases} \frac{1}{2\sqrt{N}}, & k = 0 \\ \frac{1}{\sqrt{2N}}, & k = 1, \dots, N-1 \end{cases}.$$

DCT–II transforms can be computed in $O(N \log N)$ via an extended FFT (Makhoul, 1980), are real–valued, and can be shown to provide effective representations of smooth continuous signals leveraging classical results on function approximation with Chebyshev polynomials (Trefethen, 2019). Moreover, DCT–II transforms are the backbone of modern compression codecs for digital images i.e. JPEG, providing further evidence of approximate sparsity induced in $\mathcal{D}_k$ by this transform.

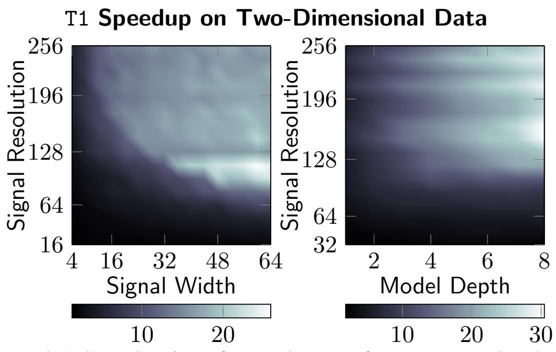


Figure 3.1. Speedup in a forward pass of `T1` over FNOs sharing the same transform $\mathcal{T}$ (DFT) on two–dimensional signals of increasing resolution. The speedup for a given configuration (point on the plane) is shown as background color gradient. The improvement grows with signal width, resolution and model depth.

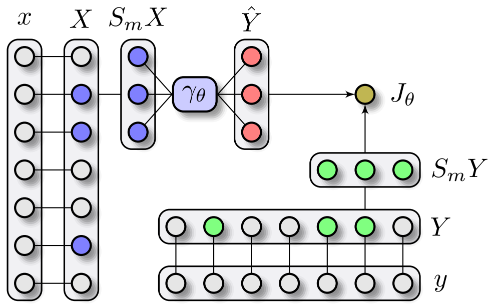
A real–valued transform allows `T1` to at least halve the required number of parameters by f_θ, avoiding the overhead of complex–valued parameters. Although our initial design of `T1` included training deep f_θ in FFT k-spaces, we observe poor performance and optimization instabilities, also noted by (Li et al., 2020). Along with parameter inefficiency, these issues are resolved by letting $\mathcal{T}$ be a DCT–II.

3.2. Reduced–Order `T1` Model and Irreducible Loss Bound

We seek to leverage structure induced in $\mathcal{D}_k$ by $\mathcal{T}$. To this end we allow `T1`, similarly to (1), to modify specific elements of X and consequently affect only certain frequency components of x.

The reduced–order `T1` model is designed to operate only on $m < N$ elements (selected by $S_m \in \mathbb{R}^{N\times m}$) of the input k-space, i.e. on a *reduced* k-space $\mathcal{D}_m \equiv \mathbb{R}^m$ of lower dimension. Thus, we can employ a smaller neural network $\gamma_\theta : \mathcal{D}_m \to \mathcal{D}_m$ for mapping $S_m X$ to the corresponding m elements $S_m Y$ of the output k-space. Thus, training involves a truncated objective that compares predictions with elements in the output signal spectrum also selected by S_m:

$$\begin{aligned} \min_\theta \quad & \mathbb{E}_x \left[\|S_m \circ \mathcal{T}(y) - \hat{Y}\|\right] \\ \text{subject to} \quad & \hat{Y} = \gamma_\theta \circ S_m \circ \mathcal{T}(x) \\ & x \sim p(x) \\ & y = \varphi(x) \end{aligned} \tag{3}$$



Irreducible losses Without any loss of generality, let us assume to select with S_m the first m elements of the k-

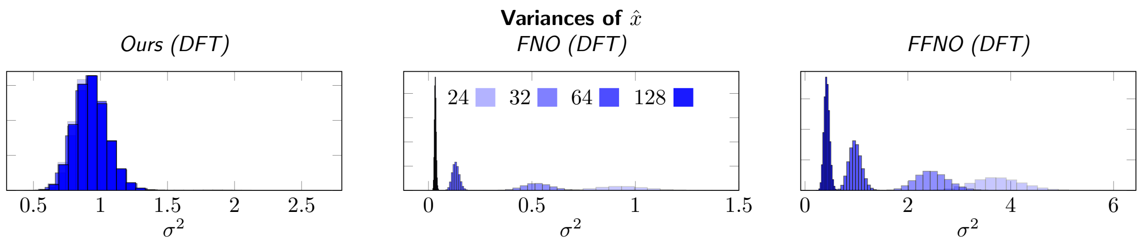


Figure 3.2. Output variance histogram in layer outputs $\hat{x} = W_m^* S_m^\top A(\theta) S_m W_N$, for a finite sample of inputs x and a single sample of θ. Color indicates signal resolution.

space, i.e.

$$S_m = m\left\{\left[\underbrace{\begin{matrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{matrix}}_{m} \quad \underbrace{\begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix}}_{N-m}\right]\right. .$$

The truncated training objective $J_\theta(X, Y)$ reads as

$$J_\theta(X, Y) = \|S_m Y - \hat{Y}\| = \sum_{k=0}^{m-1} |Y_k - \gamma_{\theta,k} \circ S_m(X)|,$$

However, J_θ does not represent a good metric to measure the real performance of the reduced–order `T1` model. For that we need to evaluate the complete loss L_θ of the approximation task, including the $N - m$ elements of the output k-space discarded by our model, i.e.

$$\begin{aligned} L_\theta(X, Y) &= \|Y - S_m^\top \hat{Y}\| \\ &= \underbrace{\sum_{k=0}^{m-1} |Y_k - \gamma_{\theta,k} \circ S_m(X)|}_{J_\theta(X,Y)} + \underbrace{\sum_{k=m}^{N-1} |Y_k - 0|}_{R_o(Y)} . \end{aligned}$$

It follows that the overall loss L_θ is higher than `T1`'s training objective J_θ, i.e. $L_\theta = J_\theta + R_o > J_\theta$, whilst R_o represents the *irreducible* residual loss due to truncation of the predictions $\hat{Y}_k$.

The choice of elements and thus magnitude of irreducible loss is a strong inductive bias for reduced–order `T1`. Nonetheless, given a dataset of input–output signals it is possible to perform an *a priori* analysis on R_o to inform the choice of m and S_m. Often, we empirically observe the irreducible error R_o for reduced–order `T1` to be smaller than for non–reduced–order FDMs i.e $R_o < \sum_{k=m}^{K-1} \|Y_k - \mathcal{T}_k(\hat{y})\|$ with layers of type (1)[4].

We note further that the reachable component J_θ of the objective cannot always be minimized to zero regardless of the approximation power of γ_θ. For each $k < m$, S_m discards $N - m$ frequency components of the input signal which, if different than zero, likely contain the necessary information to approximate $\psi_k(X)$ exactly. Specifically, the irreducible lower bound on J_θ should depend on "how much" the output's m frequency components depend on the discarded $N - m$ input's elements. A rough quantification of such bound can be obtained by inspecting the mismatch between the gradients of $\psi_k - \gamma_{\theta,k} \circ S_m$ with respect to X. In particular, it holds

$$\begin{aligned} &\sum_{j=0}^{N-1} \left| \frac{\partial \psi_k(X)}{\partial X_j} - \frac{\partial \gamma_{\theta,k}(S_m X)}{\partial X_j} \right| \\ &= \sum_{j=0}^{m-1} \left| \frac{\partial \psi_k(X)}{\partial X_j} - \frac{\partial \gamma_{\theta,k}(S_m X)}{\partial X_j} \right| + \sum_{j=m}^{N-1} \left| \frac{\partial \psi_k(X)}{\partial X_j} \right| \end{aligned}$$

Unless $\partial_{X_j} \psi_k(X) = 0$ holds for all $j = m, \dots, N-1$ and all k i.e. no dependency of the ground truth map in k-space on the truncated elements, there will be an irreducible overall gradient mismatch and thus a nonzero J_θ.

3.3. Weight Initialization for Reduced–Order FDMs

FDMs (Li et al., 2020; Tran et al., 2021; Wen et al., 2022) opt for a standard Xavier–like (Glorot and Bengio, 2010) initialization distribution that takes into account the "fan-in" input dimensions c to a layer i.e. $\mathcal{N}(0, \frac{1}{\sqrt{c}})$. However, well–known variance response properties of Xavier schemes do not hold for FDM layers truncating $N - m$ elements of the k-space. Notably, the standard deviation of the weight initialization distribution is not scaled based on the number of elements m kept after truncation of the spectrum performed by f_θ, leading to the *collapse* of the outputs to zero.

To avoid this issue in `T1` and other FDMs, we develop a simple *variance–preserving* (vp) that introduces a variance scaling factor based on m and the class of transform.

[Variance Preserving (`vp`) Initialization] Let $\hat{x} = W^* S_m^\top A S_m W x$ be a k-space reduced–order layer and W is a normalized DCT–II transform. If $x \in \mathbb{R}^N$ is a random vector with

$$\mathbb{E}[x] = \mathbb{0}, \quad \mathbb{V}[x] = \sigma^2 \mathbb{I}.$$

Then,

$$A_{ij} \sim \mathcal{N}\left(0, \frac{N}{m^2}\right) \Rightarrow \mathbb{V}[\hat{x}] = \mathbb{V}[x].$$

Corollary 3.1 (`vp` initialization for DFTs)**.** *Under the as-*

[4]See Fig. 4.1 and Appendix B3 for experimental evidence in support of this phenomenon.

sumptions of Theorem 3.3, if W is a normalized DFT matrix we have $\mathrm{Re}(A_{ij}), \mathrm{Im}(A_{ij}) \sim \mathcal{N}(0, \frac{N}{2m^2}) \Rightarrow \mathbb{V}[\hat{x}] = \mathbb{V}[x]$.

We report informal proofs of the above results in Appendix A. The collapse phenomenon is empirically shown in Figure 3.2 for $m = 24$, comparing a single layer of FNO and FFNO (with Xavier initialization) with FNO equipped with the proposed vp scheme. Under the assumptions of Corollary 3.3, we sample A and compute empirical variances of $\hat{x} = W^* S_m^\top A(\theta) S_m W x$ for several finite batches of input signals x. We repeat the experiment for signals of different lengths N. The vp scheme preserves unitary variances whereas the other layers concentrate output variances towards zero at a rate that grows with $N - m$. When the learned frequency–domain transformation f_θ is obtained, instead of the single low–rank linear layer $f_\theta = A(\theta) S_m X$, as the composition of several layers, preserving variances can be achieved by applying the vp scheme only to the first layer. For some variants of FDMs e.g. FNO that truncate the spectrum at each layer, vp initialization should instead be applied to all.

4. Experiments

We validate `T1` on learning to approximate solution operators of dynamical systems from images.

- In Section 4.1, we apply `T1` on the standard task of learning solution operators for incompressible Navier–Stokes, comparing against other FDMs. In 4.1 we perform a series of ablation experiments on each ingredient of the `T1` recipe, including weight initialization and architecture. In 4.1 we provide scaling laws.
- Section 4.2 we deal with fluid–solid interaction dynamics in the form of higher resolution images (128). We consider turbulent flows around varying airfoil geometries, benchmarking against current SOTA (Thuerey et al., 2020).
- In Section 4.3 we show how the computational efficiency of `T1` allows learning on unwieldy data without downsampling or building low–resolution meshes. We consider learning on high–resolution video (600 × 1062) capturing the turbulent dynamics of smoke (Eckert et al., 2019).

Configuration and model details are reported in the supplementary material.

4.1. Incompressible Navier–Stokes

We show that `T1` matches or outperforms SOTA FDMs with less computation on the standard incompressible Navier–Stokes benchmark. Losses are reported in n-space (signal space) for comparison.

Setup We consider two–dimensional Navier–Stokes equations for incompressible fluid in vorticity form as described in (Li et al., 2020). Given a dataset of initial conditions, we train all models to approximate the solution operator at time 50 seconds for high viscosity ($\nu = 1e^{-3}$) and at time 15 for lower viscosity ($\nu = 1e^{-4}$). As a metric, we report *normalized mean squared error* (N–MSE). Both initial condition as well as solution are provided as images of resolution 64.

We include as baseline established FDMs, such as Fourier Neural Operators (FNOs) (Li et al., 2020) and *Factorized Fourier Neural Operators* (FFNOs) (Tran et al., 2021). We indicate with the suffix *vp* models that employ the proposed variance preserving initialization scheme. All models truncate to $m = 24$, except FFNOs with $m = 32$.

Results We perform 20 training runs for each model and report mean and standard deviation in Table 4.1. `T1` reduces solution error w.r.t FNOs by over 20% and FFNOs by over 40%. A single forward pass of `T1` models is on average 2x faster than FNO and 10x than FFNOs. We note that FFNOs are designed to share parameters between layers, and thus require deeper architectures – and slower, due to more transforms. In particular, training time (500 epochs) for `T1` is cut to 20 minutes down from 40 of FNOs, matching the model speedup. Finally, we report an improvement in performance for FNOs with parameters initialized following our proposed scheme (FNOvp). Figure 4.1 provides sample predictions in n-space (left) to contextualize the task, in addition to prediction errors in frequency domain (right). Despite being a reduced order model with $m = 24$, `T1`+vp produces smaller errors on truncated k-space elements ($k > m$) compared to FNOvp and FFNO.

Ablations on weight scheme and architecture We repeat the previous experiment and report prediction errors for four variants of `T1`: same architecture and weight initialization scheme as FNOs (`T1`), `T1` with our proposed vp scheme (`T1`vp), a reduced–order variant with k-space model f_θ defined as a UNet architecture (`T1`+), and `T1`+ with variance preserving scheme (`T1`+vp). The results in Table 4.2 provide empirical evidence in support of the vp scheme and its synergistic effect with the proposed architecture. In particular, combining vp scheme and UNet structure in frequency domain reduces error by half compared to the naive `T1` approach.

Scaling laws We verify whether the reduction in predictive error of `T1` over neural operator baselines is preserved as the size of training dataset grows. We perform 10 train-

Method	Param. (M)	Size (MB)	Step (ms)	high ν	low ν
FFNO (Tran et al., 2021)	8.9	35	294	1.002±0.011	1.016±0.010
FNO (Li et al., 2020)	14.2	56	31	0.379±0.006	0.327±0.004
FNOvp	14.2	56	32	0.351±0.007	0.314±0.005
`T1+vp`	10.2	40	19	**0.260**±0.011	**0.238**±0.003

Table 4.1. Benchmarks on incompressible Navier–Stokes. Direct long–range prediction errors (N–MSE) in n-space (signal space) of different models.

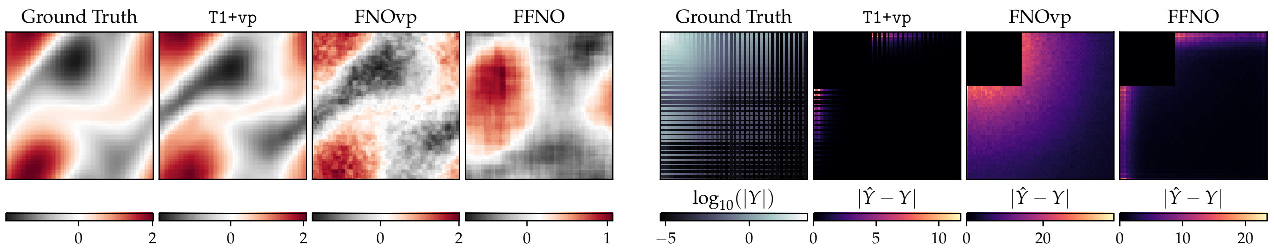


Figure 4.1. **[Left]** Direct predictions at $T = 50$s on high viscosity Navier–Stokes. **[Right]** Ground–truth spectrum and absolute errors in k-space (DCT–II). Despite predicting only the first $m = 24$ elements, reduced–order `T1` models produce smaller errors even in other regions of the k-space.

Method	high ν	low ν
`T1`	0.491	0.449
`T1vp`	0.304	0.280
`T1+`	0.295	0.260
`T1+vp`	**0.260**	**0.238**

Table 4.2. Ablation on the effect of the proposed weight initialization scheme and `T1` architecture.

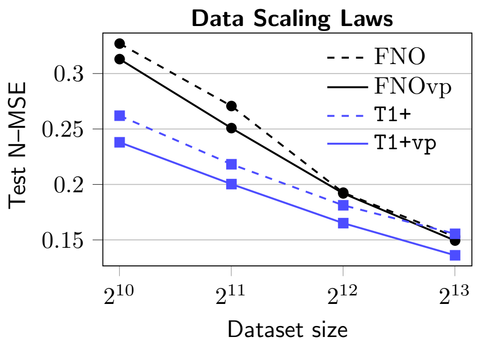


Figure 4.2. Scaling laws for N–MSE.

ing runs on the Navier–Stokes $\nu = 1e^{-4}$ experiment, each time with a larger dataset size, and report the scaling laws in Figure 4.2. With additional data, the gaps in test errors narrow slightly, with noticeable improvements obtained by applying the `vp` scheme to both FNO and `T1+`.

4.2. Flow Around Airfoils

We investigate the performance of `T1` in predicting steady–state solutions of flow around airfoils.

Setup We use data introduced in (Thuerey et al., 2020) in the form of 10000 training pairs of initial conditions, specifying freestream velocities and the airfoil mask, with the target steady–state velocity and pressure fields. This task introduces additional complexity in the form of higher resolution input images (128) and a full k-space due to the discontinuity in the field produced by the mask.

We compare a SOTA UNet architecture (DFPNet) introduced by (Thuerey et al., 2020) to FNOs and `T1` with `vp` initialization schemes. We perform a search on the most representative hyperparameters (detailed in the Appendix). Averages for 5 runs are reported in Table 4.3.

Method	N–MSE	Time (hrs)
DFPNet	0.026	2.1
FNO	0.019	6.8
`T1+`	0.020	2.2

Table 4.3. Test N–MSE and total training time on the flow around airfoil task.

Results All models are able to accurately predict steady–state solutions for different airfoils. FDMs and `T1` offer a slight improvement over DFPNets (Thuerey et al., 2020), with FNOs introducing a significant computational overhead. Training of `T1` is as fast as DFPNets and as accurate as FNOs. We note standard deviation of the results $\sigma = 0.001$; differences between FNO are explainable by natural variation `T1` ($\leq 1\sigma$) whereas improvements over DFPNet are $\geq 6\sigma$.

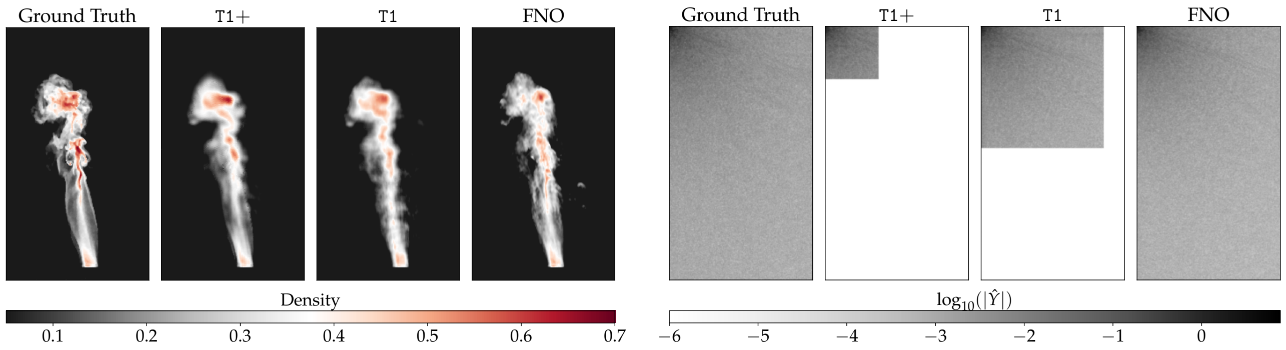


Figure 4.3. **[Left]** 10-step rollout predictions on ScalarFlow. FNOs produce non–physical artifacts and accumulate error more rapidly in time compared to `T1` models **[Right]** Log–absolute values of predictions in k-space (DCT–II). Although `T1` is limited to $m = 512$ and `T1+` to $m = 224$ k-space elements, the predictions are overall more physically accurate in n-space.

4.3. Turbulent Smoke

We investigate the performance of `T1` in predicting iterative rollouts from high–resolution video of real rising smoke plumes.

Setup We use the ScalarFlow dataset introduced in (Eckert et al., 2019) consisting of 104 sequences of 150 frames each collected from video recordings of rising hot smoke plumes. The dataset consists of raw video data at high–resolution (600 × 1062) collected at 60 fps. This task scales up complexity by involving real–world high–definition data, capturing highly–turbulent dynamics. We perform rollouts iteratively based on previous predictions: all models are trained on 3–step rollouts and evaluated over 10–steps extrapolation to test their generalization in time. We compare FNOs against `T1` and `T1+` of similar model sizes, after performing a search on most representative hyperparameters (Appendix B).

Method	N–MSE	Time (hrs)
FNO	0.220	32.4
`T1`	0.214	8.1
`T1+`	0.203	4.7

Table 4.4. Test 10–steps rollout n-space prediction errors (N–MSE) and total training time on the ScalarFlow dataset.

Results Figure 4.3 provides a sample rollout of different model predictions in k-space (DCT–II). `T1` accumulates smaller errors over the rollout and is less prone to generation of non–physical artifacts by performing prediction only on a subset of the k-space (Table 4.4). Notably, `T1` and `T1+` are $4\times$ to $7\times$ faster, providing a reduction in training time from 32.4 hours to 4.7. In Appendix B, we include additional visualization, including prediction errors on k-space.

5. Conclusion

We present a streamlined class of *frequency domain models* (FDM): *Transform Once* (`T1`). `T1` models are optimized directly in frequency domain, after a single transform, and achieve similar or improved predictive performance at a fraction of the computational cost (3x to 10x speedups across tasks). Further, a simple truncation–aware weight initialization scheme is introduced and shown to improve performance of `T1` and existing FDMs.

References

N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93, 1974.

W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing convolutional neural networks in the frequency domain. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1475–1484, 2016.

L. Chi, B. Jiang, and Y. Mu. Fast fourier convolution. *Advances in Neural Information Processing Systems*, 33: 4479–4488, 2020.

E. Dupont, A. Goliński, M. Alizadeh, Y. W. Teh, and A. Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021.

M.-L. Eckert, K. Um, and N. Thuerey. Scalarflow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

W. Falcon et al. Pytorch lightning. *GitHub. Note: https://github. com/PyTorchLightning/pytorch-lightning*, 3:6, 2019.

R. Feynman. *The Character of Physical Law, with new foreword*. MIT press, 2017.

K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

J. Guibas, M. Mardani, Z. Li, A. Tao, A. Anandkumar, and B. Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

G. Gupta, X. Xiao, and P. Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in Neural Information Processing Systems*, 34, 2021.

C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

H. Jasak, A. Jemcov, Z. Tukovic, et al. Openfoam: A c++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20. IUC Dubrovnik Croatia, 2007.

G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

N. Kovachki, S. Lanthaler, and S. Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22:Art–No, 2021.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

M. Lienen and S. Günnemann. Learning the dynamics of physical systems from sparse observations with finite element networks. *arXiv preprint arXiv:2203.08852*, 2022.

Z. Long, Y. Lu, X. Ma, and B. Dong. PDE-net: Learning PDEs from data. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3208–3216. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/long18a.html.

L. Lu, P. Jin, and G. E. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

J. Makhoul. A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):27–34, 1980.

F. Mathiesen, B. Yang, and J. Hu. Hyperverlet: A symplectic hypersolver for hamiltonian systems. In *AAAI*, 2022.

M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

A. V. Oppenheim. *Discrete-time signal processing*. Pearson Education India, 1999.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

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

M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park. Hypersolvers: Toward fast continuous-depth models. *Advances in Neural Information Processing Systems*, 33:21105–21117, 2020.

M. Poli, W. Xu, S. Massaroli, C. Meng, K. Kim, and S. Ermon. Self-similarity priors: Neural collages as differentiable fractal representations. *arXiv preprint arXiv:2204.07673*, 2022.

H. Pratt, B. Williams, F. Coenen, and Y. Zheng. Fcnn: Fourier convolutional neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 786–798. Springer, 2017.

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

O. Rippel, J. Snoek, and R. P. Adams. Spectral representations for convolutional neural networks. *Advances in neural information processing systems*, 28, 2015.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

E. M. Stein and R. Shakarchi. *Fourier analysis: an introduction*, volume 1. Princeton University Press, 2011.

G. Strang. The discrete cosine transform. *SIAM review*, 41 (1):135–147, 1999.

S. H. Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020.

A. Tran, A. Mathews, L. Xie, and C. S. Ong. Factorized fourier neural operators. *arXiv preprint arXiv:2111.13802*, 2021.

L. N. Trefethen. *Approximation Theory and Approximation Practice, Extended Edition*. SIAM, 2019.

L. Verlet. Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical review*, 159(1):98, 1967.

R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020.

G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson. U-fno–an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, page 104180, 2022.

Transform Once
Supplementary Material

A. Weight Initialization

Let us assume the input sequence x to be real and distributed according to an unknown distribution p, i.e. $x \sim p(x)$. We seek a principled initialization scheme for A that allows the first central moments of p_X be preserved by a reduced order FDM layer.

Moments under a DFT:
Let x be a real–valued input distributed according to

$$p_{\text{Re}(x)} = \mathcal{N}(0, \sigma^2 \mathbb{I}) \quad p_{\text{Im}(x)} = \delta(\mathbb{0}).$$

Consider a single element of X

$$X_k = \sum_{n=0}^{N-1} v_n$$

with

$$v_n = \frac{1}{\sqrt{N}} e^{\frac{2\pi jnk}{N}} x_n = \frac{1}{\sqrt{N}} \cos \frac{2\pi nk}{N} x_n + j \frac{1}{\sqrt{N}} \sin \frac{2\pi nk}{N} x_n.$$

For clarity, we will treat the real part $\text{Re}(X_k)$ first.

$$\text{Re}(v_n) = \frac{1}{\sqrt{N}} \cos \frac{2\pi nk}{N} \text{Re}(x_n)$$

and

$$\mathbb{E}[v_n] = \frac{1}{N} \cos^2 \frac{2\pi nk}{N} \mathbb{E}[x_n] = 0$$

$$\mathbb{V}[v_n] = \frac{1}{N} \cos^2 \frac{2\pi nk}{N} \mathbb{V}[x_n] = \frac{\sigma^2}{N} \cos^2 \frac{2\pi nk}{N}$$

where we have used the fact that

$$\text{Im}(w^{nk}) \, \text{Im}(x_n) = 0.$$

Thus,

$$\mathbb{E}[\text{Re}(X_k)] = 0$$

$$\mathbb{V}[\text{Re}(X_k)] = \sum_{n=0}^{N-1} \frac{\sigma^2}{N} \cos^2 \frac{2\pi nk}{N}$$

We observe that (a) the first central moment is preserved and (b) while the variance term can be simplified as

$$\begin{aligned}
\mathbb{V}[\text{Re}(X_k)] &= \sum_{n=0}^{N-1} \frac{\sigma^2}{N} \cos^2 \frac{2\pi nk}{N} \\
&= \frac{\sigma^2}{N} \sum_{n=0}^{N-1} \cos^2 \frac{2\pi nk}{N} \\
&= \frac{\sigma^2}{N} \frac{N}{2} \\
&= \frac{\sigma^2}{2}
\end{aligned}$$

We follow a similar procedure for $\mathrm{Im}(X_k)$, arriving at

$$\mathbb{E}[\mathrm{Im}(X_k)] = 0$$
$$\mathbb{V}[\mathrm{Im}(X_k)] = \sum_{n=0}^{N-1} \frac{\sigma^2}{N} \sin^2 \frac{2\pi nk}{N}$$

where the variance again simplifies to

$$\sum_{n=0}^{N-1} \frac{\sigma^2}{N} \sin^2 \frac{2\pi nk}{N} = \frac{\sigma^2}{2}$$

Since $X_k = \mathrm{Re}(X_k) + j\,\mathrm{Im}(X_k)$, its statistics are given by

$$\mathbb{E}[X_k] = \mathbb{E}[\mathrm{Re}(X_k)] + j\mathbb{E}[\mathrm{Im}(X_k)] = 0 + j0$$
$$\mathbb{V}[X_k] = \mathbb{V}[\mathrm{Re}(X_k)] + \mathbb{V}[\mathrm{Im}(X_k)] = \sigma^2$$

A similar argument can be developed using basic properties of circular–symmetry of complex Normals.

It is critical that the normalization factor $\frac{1}{\sqrt{N}}$ be included in W in order to preserve the variance of $\mathbb{V}[X]$.

Indeed, normalization factors used in different conventions lead to different results

$$\text{forward factor } \frac{1}{N} \implies \mathbb{V}[X_k] = \frac{\sigma^2}{N}$$
$$\text{backward factor } 1 \implies \mathbb{V}[X_k] = N\sigma^2$$

As N can easily be in the order of hundreds or thousands for generic signals, explosion of variance can be an issue if the orthogonalization factor $\frac{1}{N}$ is not applied to W.

A.1. Variance response of transformations in spectral layers

Having determined the preservation of probability under Fourier Transforms, we now analyze of the first two central moments of the distribution of a spectral layer output $\hat{x}$ are affected by a learned transformation on X.

Gaussian Initialization with Dense Linear Layer Case:
Once again, we consider a single element $\hat{X}_k$ of $\hat{X}$

$$\begin{aligned}\hat{X}_k &= \sum_{i=0}^{N-1} A_{ki} S_{m,ki} X_i \\ &= \sum_{i=0}^{m} A_{ki} X_i\end{aligned} \tag{4}$$

In this case, $p_{\hat{X}_k}$ is a sum of product distributions involving independent random variables A_{ki} and X_t.
The first central moment is readily obtained

$$\mathbb{E}[\hat{X}_k] = \sum_{i=0}^{m-1} \mathbb{E}[A_{ki}]\mathbb{E}[X_i] = 0$$

if at least one of the conditions

$$\forall n:\ \mathbb{E}[x_n] = 0 \implies \forall k:\ \mathbb{E}[X_k] = 0$$

or $\forall\, k, i:\ \mathbb{E}[A_{ki}] = 0$ holds.
The variance

$$\begin{aligned}
\mathbb{V}[\hat{X}_k] &= \sum_{i=0}^{m-1} \Big(\mathbb{V}[A_{ki}] + \mathbb{E}[A_{ki}]^2)(\mathbb{V}[X_i] + \mathbb{E}[X_i]^2) - \mathbb{E}[A_{ki}]^2\mathbb{E}[X_i]^2\Big) \\
&= \sum_{i=0}^{m-1} \mathbb{V}[A_{ki}]\mathbb{V}[X_i] \\
&= \sum_{i=0}^{m-1} \sigma^2\mathbb{V}[A_{ki}] \\
&= \sum_{i=0}^{m-1} \sigma^2(\mathbb{V}[\mathrm{Re}(A_{ki})] + \mathbb{V}[\mathrm{Im}(A_{ki})])
\end{aligned}$$

$$\mathbb{V}[\hat{X}_k] = 2m\sigma^2\sigma_A^2$$

Since

$$\begin{aligned}
\mathbb{V}[\hat{x}_n] &= \frac{1}{N}\sum_{k=0}^{m-1} \mathbb{V}[\hat{X}_k] \ \forall \ k, n \\
&= \frac{2m^2}{N}\sigma^2\sigma_A^2 \quad \square
\end{aligned}$$

Solving for σ_A, we arrive at a sufficient condition for variance preservation i.e.

$$\sigma_A^2 = \frac{N}{2m^2}\sigma^2 \implies \forall n : \mathbb{V}[\hat{x}_n] = \mathbb{V}[x_n]$$

The result for DCT can be derived by following the same steps. Since the k–space for DCTs is real, the variance sums to

$$\begin{aligned}
\mathbb{V}[\hat{X}_k] &= \sum_{i=0}^{m} \Big(\mathbb{V}[A_{k,i}] + \mathbb{E}[A_{k,i}]^2)(\mathbb{V}[X_i] + \mathbb{E}[X_i]^2) - \mathbb{E}[A_{k,i}]^2\mathbb{E}[X_i]^2\Big) \\
&= \sum_{i=0}^{m-1} \mathbb{V}[A_{k,i}]\mathbb{V}[X_i] \\
&= \sum_{i=0}^{m-1} \sigma^2\mathbb{V}[A_{k,i}] \\
&= m\sigma^2\sigma_A^2
\end{aligned}.$$

B. Additional Details

Broader impact FDMs are widely used in the context of learning to predict the evolution of dynamical systems. The model class presented in this work, `T1`, provides an accessible way to train and evaluate large–scale FDMs, reducing memory overhead and overall training times. When predicting the solution of e.g. a *partial differential equation* (PDE), care should be taken especially when the prediction is used to inform downstream decision making, as many systems are optimally predictable only for a certain time scale (Strogatz, 2018, pp. 366). We anticipate a potential positive environmental impact from the adoption of `T1` as a replacement for the largest FDMs currently in use.

Experimental setup Experiments have been performed on an NVIDIA© DGX workstation equipped with a 128 threads AMD© EPYC 7742 CPU, 512GB of RAM and four NVIDIA© A100 GPUs. The main software implementation has been done within the `PyTorch` (Paszke et al., 2017) ecosystem building upon the `pytorch-lightning` (Falcon et al., 2019) framework.

Common experimental settings

B.1. Incompressible Navier–Stokes

Dataset We use data generated in (Li et al., 2020) in the form of pairs of initial conditions and solutions of the incompressible Navier–Stokes equations in vorticity form solved with a pseudospectral method. The dataset [5] is comprised rollouts of solutions as images of resolution 64.

Models and training The training configuration is shared by all models:

```yaml
datamodule:
    ntrain: 1000
    ntest: 200
    batch_size: 64
    history_size: 1
train:
    optimizer:
        type: AdamW
        learning_rate: 1e-3
        weight_decay: 1e-4
    scheduler:
        type: Step
        step_size: 100
        gamma: 0.5
        scheduler_interval: epoch
loss_fn: RelativeL2Loss
```

For the high viscosity ($1e^{-3}$) setting, the models are trained to predict the solution at time $T = 50$ seconds directly, without producing rollouts and supervising the model with solutions at times between 0 and 50. Crucially, this ensures that the task is much more challenging than that of (Li et al., 2020), where for a single training sample the entire rollout is used as supervision. For the low viscosity setting ($1e^{-4}$), target times are $T = 15$ seconds.

Model configurations are given below:

FNO

```yaml
modes: 24
nlayers: 6
width: 32
```

T1

```yaml
modes: 24
nlayers: 6
width: 48
```

FFNO

```yaml
modes: 32
nlayers: 10
width: 82
```

where each layer in a model shares the same structure. In FNOs and FFNOs, we employ a regular FDM layer following (Li et al., 2020; Tran et al., 2021) with k–space convolutions and residual connections given by n–space layers (pointwise convolutions for FNOs, dense for FFNOs). `T1` uses a similar layer without n–space residual paths. The differences in number of layers and width have been introduced to keep parameter counts comparable. At a given channel width, FNOs require the largest number of parameters due to k–space convolutions on complex numbers given by the DFT coefficients. Although FFNOs (Tran et al., 2021) are most parameter efficient due to parameter sharing, we found them unable to tackle the task and produce high–quality predictions.

`T1+` employs a UNet on the patch constructed by the elements of the k–space kept, and shares its structure with `T1` otherwise. The `vp` parameter initialization scheme in `T1` is applied only to the first layer performing the truncation in k–space, not to the following layers which use standard Kaiming initialization (He et al., 2015). In FNOvp the scheme is applied to all layers.

Hyperparameter tuning We start with the basic model structure of FNOs as detailed (Li et al., 2020) and perform a basic hyperparameter search on a small slice of the training set, with the goal of ensuring proper convergence of a model. We did not find the number of layers to have a significant impact on convergence. Width plays an important role and is best kept above 24.

[5]Data can be downloaded here: Google Drive link. High viscosity: `NavierStokes_V1e-3_N5000_T50`, Low viscosity: `NavierStokes_V1e-4_N10000_T30`.

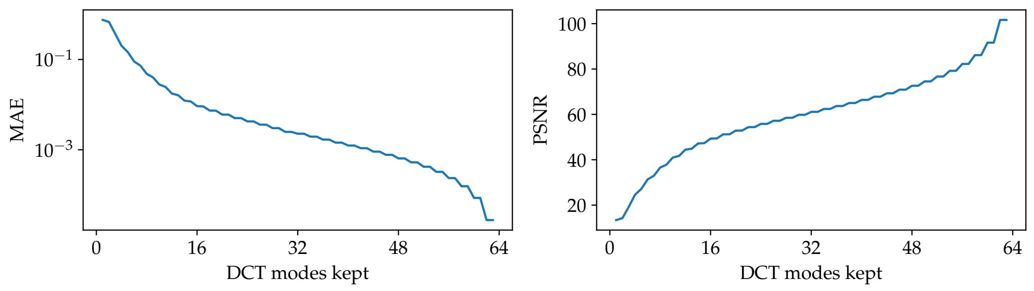


Figure B.1. Incompressible Navier–Stokes: metrics vs number of DCT modes (i.e. m elements) kept (i.e. not pruned).

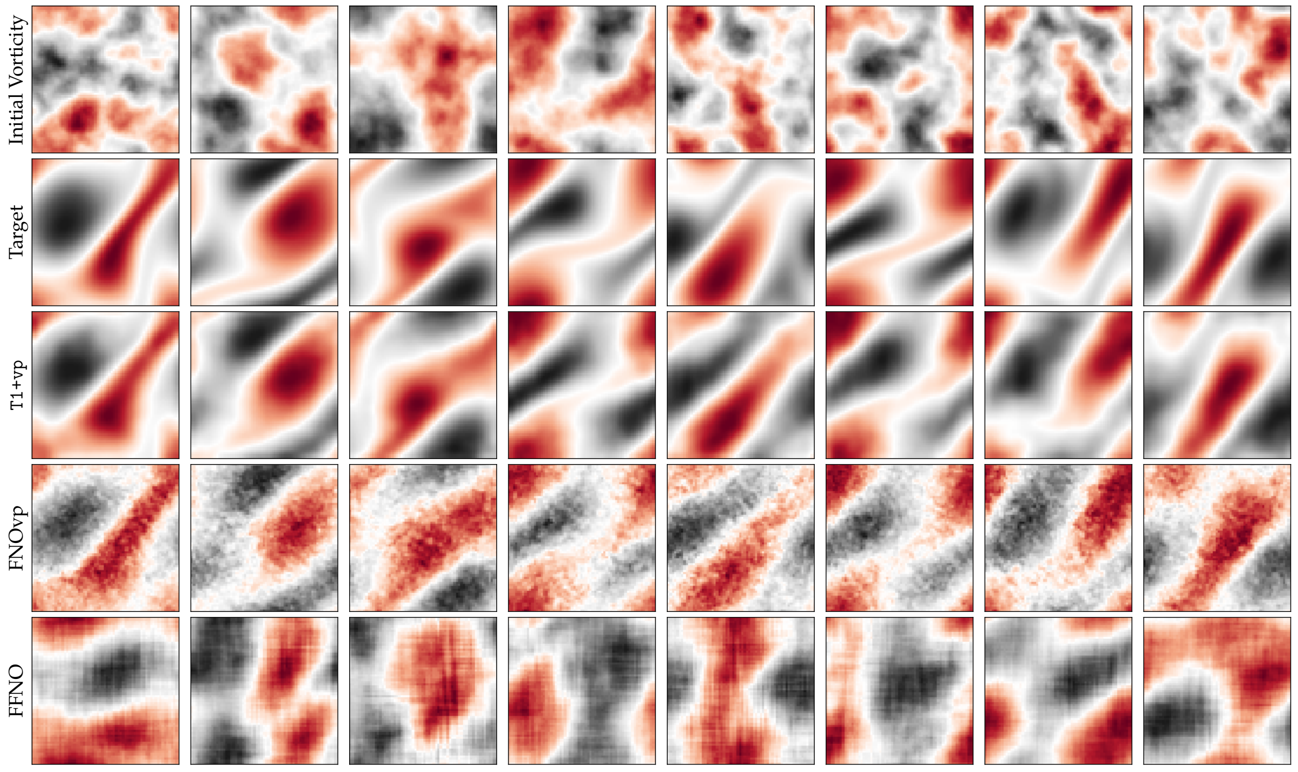


Figure B.2. Initial conditions, ground truth solutions at time $T = 50$ seconds, and models predictions for incompressible Navier-Stokes in vorticity form (high viscosity of $1e^{-3}$). `T1` reduces solution error w.r.t FNOs by over 20% and FFNOs by over 40%. A single forward pass of `T1` models is on average $2\times$ faster than FNO and $10\times$ than FFNOs.

Scaling laws We use the same settings as the main experiment, repeating separate training runs for the low viscosity setting. In particular, we increase the dataset size for each set of runs by a factor of 2: $1024, 2048, 4096, 8192$. The total number of epochs is kept fixed, so that more iterations are performed for larger datasets. The same test set of size 200 is used in all cases.

Further comments Additional predictions are provided in Figure B.2. Figure B.1 shows the approximation error on the Navier-Stokes solutions due to truncation at different number of k–space elements m.

B.2. Flow Around Airfoils

Dataset We use a slice of the dataset introduced by (Thuerey et al., 2020) in the form of 11000 training pairs of initial conditions and solutions. The solutions are obtained via `OpenFOAM` (Jasak et al., 2007) SIMPLE, a steady–state solver for incompressible and turbulent flows. In particular, the initial conditions are specified as freestream velocities over the domain (two–directional components), in addition to a specification of the airfoil in point cloud format. Delaunay triangulation is used for mesh generation.

After simulation, data is provided as initial condition and steady–state solution pairs. The initial condition is a three channel 128×128 image: two channels for freestream velocities and one for the airfoil mask. The solution is a three channel 128×128 image: a velocity field and a scalar pressure field. All data is normalized using training set statistics.

Models and training Training configuration is given as

```
datamodule:
    ntrain: 8000
    nval: 2000
    ntest: 1000
    batch_size: 64
train:
    optimizer:
        type: AdamW
        learning_rate: 1e-3
        weight_decay: 1e-4
    scheduler:
        type: Step
        step_size: 100
        gamma: 0.6
        scheduler_interval: epoch
loss_fn: RelativeL2Loss
```

The baseline UNet matches the architecture of (Thuerey et al., 2020) (DFPNet). The FNO architecture is comprised of a standard stack of FDM layer as discussed in B.1. The k–space UNet in `T1+` has the same structure as a DFPNet.

FNO
```
modes: 24
nlayers: 6
width: 48
```

DFPNET
```
channel_exponent: 6
```

T1+
```
modes: 100
channel_exponent: 5
```

Hyperparameter tuning This is an example of a dataset where the k–space is full due to discontinuity in the solution given by the airfoil mask.

We use the training and validation sets to inspect the k–space and set m to 100 for the irreducible loss term to be sufficiently small as shown in Figure B.5. We swept over m for FNOs and found larger than 24 to perform worse, likely due to k–space convolution being sufficient to capture higher frequency components. We observe DFPNets with larger channel exponents perform worse due to overfitting.

Further comments A sample of predictions is given in Fig. B.3. Figure B.4 shows the n–space and corresponding DCT k–space of a data point. As can be observed, the k–space is structured but full due to the discontinuity caused by the airfoil mask. Figure B.5 shows the approximation error on solution fields due to truncation in k–space at different m. In this task, the DCT is more efficient, given a budget of modes to keep, as it yields lower errors. This error provides a theoretical lower bound for the predictive error achievable by a `T1` model with a given budget, reachable only if the `T1` predicts the first m modes perfectly.

The vertical line indicates the budget used for the main text `T1` experiments ($m = 100$), and the horizontal one the test N–MSE achieved. Various segments of the vertical line indicate reducible and irreducible components of the loss as discussed in Section 3.2. The theoretical limit at $m = 100$ is well below what has been empirically achieved by `T1` and other models. Indeed, the irreducible loss is an order of magnitude smaller than what the best model (including non–reduced–order variants) achieves on the task.

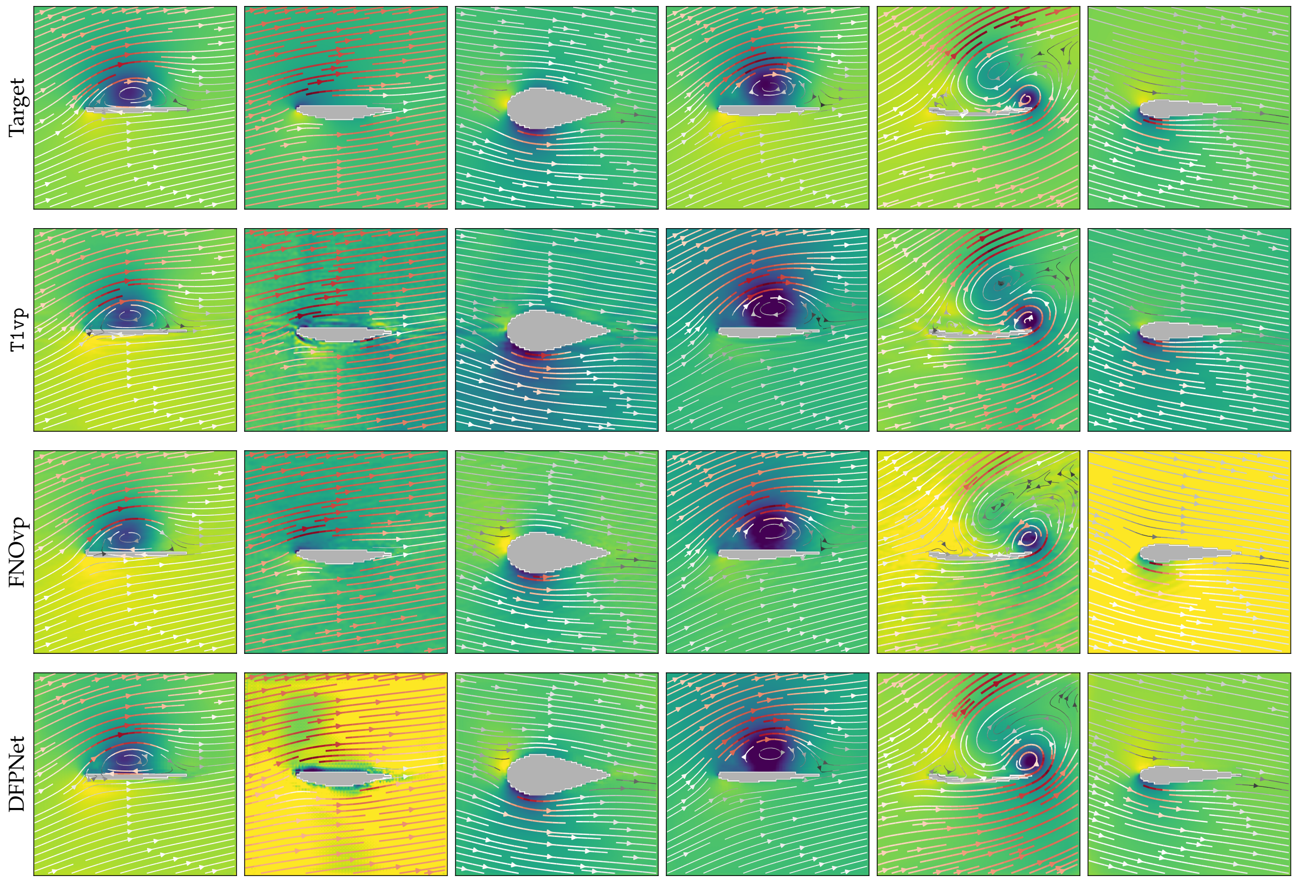


Figure B.3. Ground truth solutions and predictions with different airfoil designs and angles of attack of the flow. The background color is the scalar pressure value while the vector field represents the velocity field: arrow colors indicate its "strength" i.e. 2–norm.

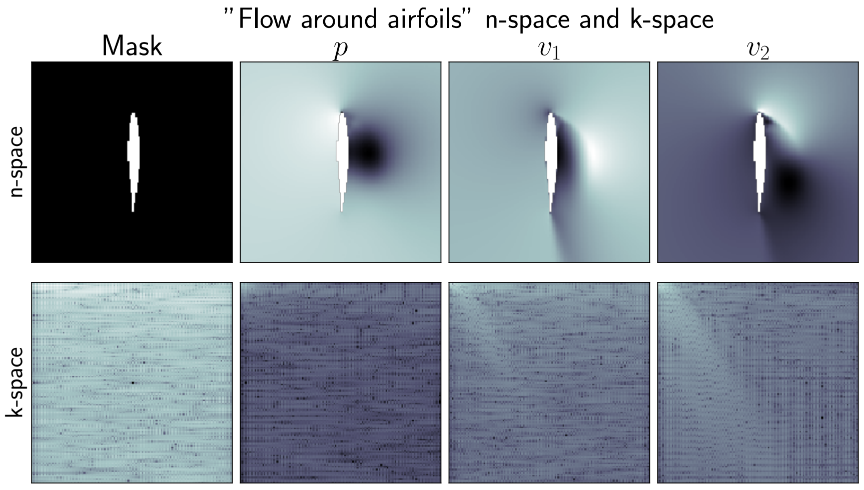


Figure B.4. Flow around airfoils: example of n-space: input mask, output pressure p and velocity field v_1, v_2. Below, the corresponding DCT k–space in abs–log i.e. $\log(|\mathcal{T}(x)|)$ to highlight its structure.

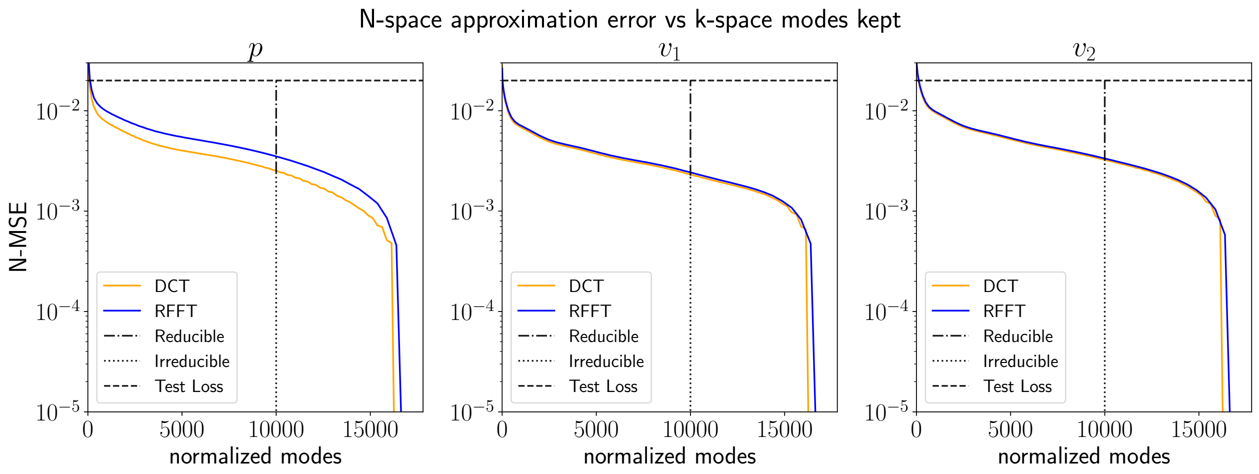


Figure B.5. Average approximation error (N–MSE) due to truncation in k–space at different number of elements m for the *flow around airfoils* dataset. In blue, the real FFT k–space, in orange the regular DCT k–space. On the x–axis, the normalized cost for a number of modes m: for DCTs, since the k–space is real, truncation at m modes requires m^2 floats, for real FFTs with complex k–space and conjugacy the cost in floats is $4m^2$. The vertical line indicates the budget used for `T1` used in this task ($m = 100$), while the horizontal line is the test N–MSE achieved.

B.3. Turbulent Smoke

Dataset We employ for this experiment the ScalarFlow dataset introduced in (Eckert et al., 2019) which is available online under the Creative Commons license CC-BY-NC-SA 4.0[6]. (Eckert et al., 2019) created an environment for controlling the release of smoke plumes: a fog machine generated fog inside of a container; the fog was then heated up by a heating cable and a valve controlled its release. Data was captured via multiple calibrated cameras in high resolution at 60 fps (frames per second) for 150 frames.

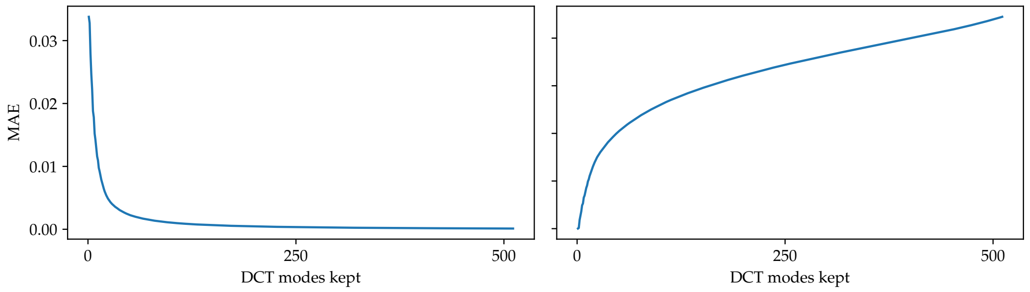


Figure B.6. ScalarFlow dataset: reconstruction error versus number of kept DCT modes.

The dataset contains 3D reconstructions of the smoke plumes and 2D input and rendered images: input images are used by (Eckert et al., 2019) to solve an optimization problem in which the goal is to generate a 3D reconstruction that minimizes the difference between input and rendered images. 2D input images are obtained directly from raw data on which only post–processing is applied by (Eckert et al., 2019) in the form of gray scaling and denoising: these are saved in compressed `numpy` (Harris et al., 2020) arrays named `imgsTarget_000xxx.npz`. Each resulting frame comprises 5 different camera views 600×1062 in size. Since we want to use `T1` on high–resolution experimental data, we directly utilize the central camera view of these input images in our learning task without any further downsampling or data processing. Similarly to (Lienen and Günnemann, 2022), we divide the 104 recordings into the first 64 for training and use the remaining 20 for validation and 20 for testing.

Data is normalized to the $[0, 1]$ range based on training dataset statistics.

[6]ScalarFlow dataset download: https://ge.in.tum.de/publications/2019-scalarflow-eckert/

Hyperparameter selection and tuning We performed a search on the most representative hyperparameters. One of the most important hyperparameters to choose from is the number of DCT modes to keep, i.e. first m elements in k–space. We note that for simplicity as well as for compatibility with the UNet inside of `T1+`, we consider a *square* mode pruning, i.e. we keep the same number of frequencies on both height and width of the image and refer to the modes kept in both dimensions as m. Figure B.7 and Figure B.6 show trends of DCT modes in terms of errors and visual quality: while the first modes m contribute the most to the quality of the representation in n–space, the last elements contribute only to high–frequency details whose effect is minor on the overall reconstruction. Thus, we set `T1+` to $m = 224$ and consequently `T1` to $m = 512$ to have comparable model sizes. We set $m = 48$ for FNO due to memory and model size limitations, noting that its residual connections effectively enlarge the training spectrum to all possible frequencies as shown in Figure 4.3. Similarly to other experiments (B2), we observe raising m in FNO to not significantly improve predictive error, even when the additional k–space elements would include a larger portion of the dataset.

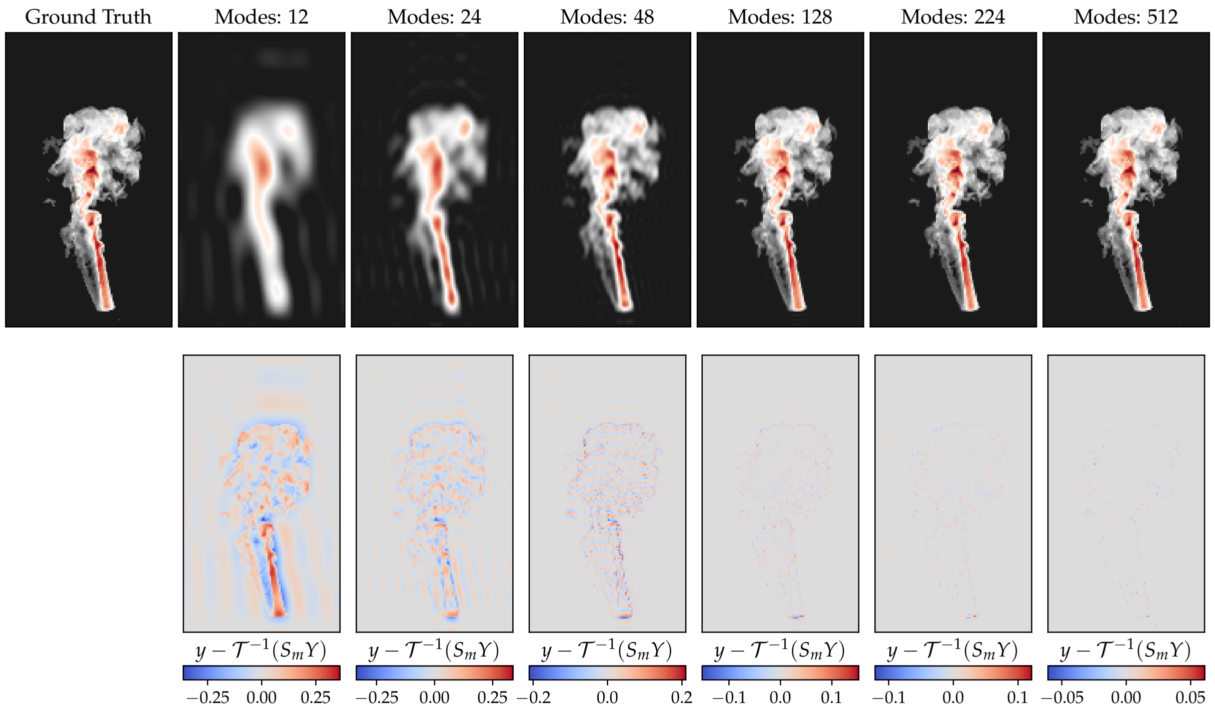


Figure B.7. **[Top]** Visual comparison of ScalarFlow frames with changing number of DCT modes kept (i.e. first m elements) . **[Bottom]** Error between the ground truth frame y and its inverse transformation after mode pruning from k–space back to n–space. As expected, the first few k–space elements are crucial to minimizing reconstruction errors, with higher frequency components contributing minimally.

We also experiment with different iterative rollout update strategies as in (Pfaff et al., 2020). We consider the time step Δt to be unitary, i.e. $\Delta t = 1$, given that the training frames are sampled consistently at 60 fps. We call 0–order integration an update of the type: $x_{t+1} = h_\theta(x_t; x_{t-1}, \dots, x_{t-H})$ in which h_θ denotes a learned model which takes as inputs the current state x_t and optionally a history of size H of past states $x_{t-1}, \dots, x_{t-H}$ and directly predicts the next state x_{t+1}. A 1–order integrator performs the following update: $x_{t+1} = x_t + h_\theta(x_t; \cdot)$, in which the model predicts the state update, i.e. the *velocity*, similarly to an Euler step. A 2–order integrator, also known as basic Störmer—Verlet (Verlet, 1967) can be written as following: $x_{t+1} = 2x_t - x_{t-1} + h_\theta(x_t; \cdot)$; the model h_θ predicts the *acceleration* of the system. We empirically found the zero–order integration to be more prone to generating artifacts with slower convergence, which may be because the model has to directly predict the next step with no "help" from the current step information. We found models trained with first–order integrators to have lower predictive errors than those trained with second–order ones, and we thus use it in all the experiments. As for the history size, we selected $H = 1$ since it provided noticeable benefits compared to $H = 0$, in which the model has no way of knowing previous states and thus inferring velocities. Larger history sizes did not seem to provide any improvements and only made the models larger as also noted in (Pfaff et al., 2020).

Models and training All models share the following configuration for training:

```yaml
datamodule:
    ntrain: 64
    nval: 20
    ntest: 20
    batch_size: 1
    history_size: 1
    target_steps_train: 3
    target_steps_val_test: 10
train:
    optimizer:
        type: AdamW
        learning_rate: 1e-3
        weight_decay: 1e-4
    scheduler:
        type: CosineAnnealingWarmRestarts
        T_0: 32
        step_size: 1
        scheduler_interval: step
loss_fn: RelativeL2Loss
```

Where we used the implementation in `PyTorch` of the cosine annealing schedule with warm restarts[7]. The FNO architecture comprises a standard stack of FDM layers as discussed in B.1. The k–space UNet in T1+ has the same structure as a DFPNet.

FNO

```yaml
modes: 48
nlayers: 4
width: 48
```

T1

```yaml
modes: 512
nlayers: 4
width: 8
```

T1+

```yaml
modes: 224
nlayers: 1
width: 4
channel_exponent: 7
```

where we note that all models employ `GeLU` (Hendrycks and Gimpel, 2016) activation functions between inner layers.

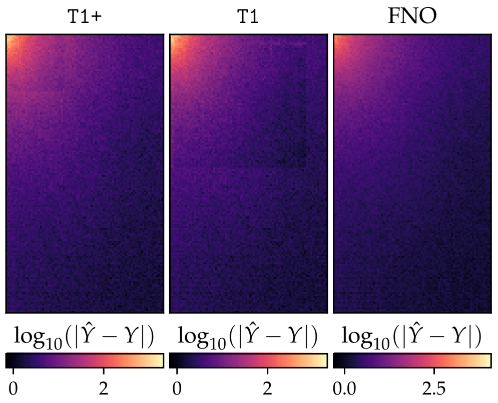


Figure B.8. Mean log–absolute values of predictions in k-space (DCT–II) of a 20–elements batch in the test dataset. Although `T1` is limited to $m = 512$ and `T1+` to $m = 224$ k-space elements (visible as square "shadows" in the error plots), its predictions are overall more physically accurate in n-space.

Analysis of results Table B.1 provides a larger version of the table in the main text, including 1–step mean absolute errors (MAE). We note that while FNO produces smaller errors in one–step predictions, it quickly accumulates larger

[7]We used the scheduler `torch.optim.lr_scheduler.CosineAnnealingWarmRestarts` with the number of iterations for the first restart $T\_0 = 32$/. All other hyperparameters are the same as in the reference implementation.

errors in extrapolation. Figure B.8 shows mean errors in k–space of FNO vs `T1` and `T1+`. `T1` demonstrate smaller overall errors and lower maxima compared to the FNO.

Table B.1. Full benchmark on the ScalarFlow dataset. N-MSE refers to 10-step test rollouts. `T1` models generate more stable rollouts.

Method	Param (M)	Size (MB)	Time (hrs)	N-MSE	MAE 1 step ($\times 10^{-3}$)	MAE 10 steps ($\times 10^{-2}$)
FNO	84.9	339	32.4	0.220	2.15	1.20
`T1`	83.9	335	8.1	0.214	2.89	1.12
`T1+`	67.8	271	4.7	0.203	2.29	1.11