Merging Memory and Space: A State Space Neural Operator

Anonymous authors
Paper under double-blind review

Abstract

We propose the *State Space Neural Operator* (SS-NO), a compact architecture for learning solution operators of time-dependent partial differential equations (PDEs). Our formulation extends structured state space models (SSMs) to joint spatiotemporal modeling, introducing two key mechanisms: *adaptive damping*, which stabilizes learning by localizing receptive fields, and *learnable frequency modulation*, which enables data-driven spectral selection. These components provide a unified framework for capturing long-range dependencies with parameter efficiency. Theoretically, we establish connections between SSMs and neural operators, proving a universality theorem for convolutional architectures with full field-of-view. Empirically, SS-NO achieves strong performance across diverse PDE benchmarks—including 1D Burgers' and Kuramoto–Sivashinsky equations, and 2D Navier–Stokes and compressible Euler flows—while using significantly fewer parameters than competing approaches. Our results demonstrate that state space modeling provides an effective foundation for efficient and accurate neural operator learning.

1 Introduction

Many problems in scientific computing require approximating nonlinear operators that map input functions to output functions, often governed by partial differential equations (PDEs). Neural operators (NOs) (Kovachki et al., 2023) provide a mesh-independent framework for this task, operating directly on functions and generalizing across discretizations.

The Fourier Neural Operator (FNO) (Li et al., 2021) is a widely used NO, as its global convolution kernels effectively capture long-range spatial correlations. However, this fully global design comes with substantial computational and memory costs, scaling poorly in higher dimensions. Factorized variants (Tran et al., 2023; Lehmann et al., 2023; 2024) address this by decomposing operator learning into lower-dimensional subspaces, achieving linear scaling in dimension—a crucial advantage for high-dimensional spatial applications. Yet, these approaches offer limited flexibility in adjusting receptive fields or frequency modes to different spatial regions or temporal dynamics.

In a complementary line of work, state space models (SSMs) (Gu et al., 2022b), such as the diagonalized S4D architecture (Gu et al., 2022a), have achieved remarkable success in modeling long-range *temporal* dependencies. Like factorized FNOs, SSMs scale linearly in time and memory, but they provide an additional advantage: data-dependent parameterization of convolutional filters that can learn to emphasize different frequency components and temporal scales. Despite these parallels, the connection between FNOs (primarily spatial) and SSMs (primarily temporal) has remained underexplored.

This paper makes that connection explicit. We introduce the **State Space Neural Operator (SS-NO)**, a unified operator learning framework that applies SSMs directly over joint spatiotemporal domains. SS-NO can be interpreted in two complementary ways: as extending SSMs from purely temporal modeling to spatiotemporal operator learning, or as generalizing FNOs into more expressive architectures with adaptive receptive fields, frequency content, and temporal causality.

Theoretically, we prove that any continuous operator can be approximated arbitrarily well by convolutional NOs with full receptive fields, a condition satisfied by both FNO and SS-NO. Importantly, the spatial-only variant of SS-NO subsumes factorized FNO as a special case while adding two key capabilities: learned

damping coefficients for stability control and data-driven frequency modulation. Empirically, we validate SS-NO on challenging 1D and 2D benchmarks, including chaotic Kuramoto–Sivashinsky dynamics, variants of Kolmogorov flow, Richtmyer–Meshkov instability, and gravitational Rayleigh–Taylor turbulence. Across tasks, SS-NO surpasses existing methods using fewer parameters. These results highlight SS-NO as a principled, scalable, and practical architecture for data-driven modeling in engineering and the physical sciences.

2 Related Work

2.1 Neural Operators

Data-driven neural operators have emerged as a powerful framework for learning PDE solution operators (Chen & Chen, 1995; Bhattacharya et al., 2021; Lu et al., 2021; Kovachki et al., 2023). Theoretical work established that several neural operators can universally approximate nonlinear operators (Chen & Chen, 1995; Kovachki et al., 2021; 2024; Lanthaler et al., 2024), including DeepONet (Lu et al., 2021; Lanthaler et al., 2022) and the Fourier Neural Operator (FNO), which implements global convolution via the Fourier transform (Li et al., 2021). Extensions such as the Factorized Fourier Neural Operator (FFNO) (Tran et al., 2023) enhance efficiency and expressivity in practice, but their theoretical expressivity remains unknown. Alternative inductive biases have also been explored, including U-Net–based convolutional operators (Raonic et al., 2023; Gupta & Brandstetter, 2023; Rahman et al., 2023; Takamoto et al., 2022) and attention-based operators like the Galerkin Transformer (GKT) (Cao, 2021) and the FactFormer (Li et al., 2023).

While some studies suggest that purely Markovian models can be performant for learning time-evolution PDEs (Tran et al., 2023; Lippe et al., 2023), recent work has shown that incorporating past states improves accuracy, particularly under low-resolution or noisy conditions. This has led to the integration of memory mechanisms into neural operators. Buitrago et al. (2025) introduce the Memory Neural Operator (MemNO) framework, which embeds temporal S4-based recurrence within general neural operator architectures, motivated by the Mori–Zwanzig formalism (Mori, 1965; Zwanzig, 1961). Our work extends this direction by examining scenarios with missing contextual information and introducing a spatiotemporal factorization that distributes memory across *both time and space*.

2.2 Structured State Space Models (SSMs)

Structured state space models (SSMs) have recently achieved strong results on long-range sequence tasks in natural language processing and vision. The Structured State Space sequence model (S4) (Gu et al., 2022b;a) uses a continuous-time linear SSM layer to capture very long-range dependencies efficiently, and the Mamba model (Gu & Dao, 2024) extends this idea by making the SSM parameters input-dependent, further improving performance on large-scale language modeling tasks. These SSM-based architectures have been shown to match or exceed Transformer performance on a variety of benchmark tasks.

SSMs have also begun to be used in operator learning. For example, Hu et al. (2024) incorporate Mamba-style SSM layers to predict dynamical systems efficiently. In the context of neural PDE solvers, recent works like Cheng et al. (2024) and Zheng et al. (2024) embed Mamba SSM modules to capture spatial correlations in solution fields. These methods primarily focus on spatial modeling, applying SSM layers across spatial dimensions or treating time steps sequentially. By contrast, our work focuses on applying SSMs jointly in both space and time, employing a unified spatiotemporal SSM architecture for neural operator learning.

3 Methodology

3.1 Problem Formulation

Let $\Omega \subset \mathbb{R}^d$ be a bounded spatial domain, and consider the solution $u \in C([0,T]; L^2(\Omega; \mathbb{R}^V))$ of a time-dependent partial differential equation (PDE), where C denotes the space of continuous functions and L^2 denotes the space of square-integrable functions. We assume access to a dataset of N solution trajectories $u^{(i)}(t,x)_{i=1}^N$ generated from varying initial or parametric conditions.

In practice, we discretize the spatial domain Ω using an equispaced grid $\mathcal{S}$ of resolution f, and the temporal domain $[0, T]$ using an equispaced grid $\mathcal{T}$ with $N_t + 1$ points. Let $T_{\text{in}} < T$ denote the fixed input horizon.

Given the trajectory segment $u(t, x)|_{t\in[0,T_{\text{in}}]}$ on $\mathcal{S}$, our goal is to predict the future evolution $u(t, x)|_{t\in[T_{\text{in}},T]}$ on the same grid. Rather than forecasting a single step, we aim to learn the full system dynamics conditioned on the initial segment. Formally, the NO model defines a parametric map $\Psi_\theta : C(\mathcal{T}_{\text{in}}; L^2(\Omega; \mathbb{R}^V)) \to C(\mathcal{T}_{\text{out}}; L^2(\Omega; \mathbb{R}^V))$, where $\mathcal{T}_{\text{in}} = [0, T_{\text{in}}]$ and $\mathcal{T}_{\text{out}} = [T_{\text{in}}, T]$, mapping the input segment to the predicted solution.

3.2 Structured State Space Models (S4 and S4D)

We begin with the continuous-time linear state space model (SSM) defined as:

$$\begin{aligned} \frac{d}{dt}v(t) &= Av(t) + Bu(t), \\ y(t) &= Cv(t) + Du(t), \end{aligned} \tag{1}$$

where $v(t) \in \mathbb{R}^H$ is the hidden state, H is the number of states, $u(t) \in \mathbb{R}$ is the input, and $y(t) \in \mathbb{R}$ is the output. The matrices $A \in \mathbb{R}^{H\times H}$, $B \in \mathbb{R}^{H\times 1}$, $C \in \mathbb{R}^{1\times H}$, and $D \in \mathbb{R}$ define the system dynamics.

The Structured State Space (S4) model (Gu et al., 2022b) introduces a parameterization where A is structured to allow efficient computation of the convolution kernel $\kappa(t)$ corresponding to the system's impulse response. Specifically, S4 utilizes a diagonal plus low-rank (DPLR) structure, $A = \Lambda + PQ^\top$, where $\Lambda \in \mathbb{C}^{H\times H}$ is diagonal, and $P, Q \in \mathbb{C}^{H\times r}$ with $r \ll H$. This structure enables computation of $\kappa(t)$ via the Cauchy kernel and allows for efficient implementation with the Fast Fourier Transform (FFT).

To further simplify the model, S4D (Gu et al., 2022a) restricts A to be diagonal, i.e., $A = \text{diag}(\lambda_1, \ldots, \lambda_N)$, and initializes the eigenvalues $\{\lambda_i\}$ to approximate the behavior of the original S4 model. This diagonalization reduces the computational complexity and memory footprint while retaining the ability to model long-range dependencies.

In both S4 and S4D, after computing the convolution with the kernel $\kappa(t)$, a pointwise nonlinearity $\sigma(\cdot)$ (e.g., GELU) is applied, followed by a residual connection.

3.3 Markovian Spatial State Space Model

To model spatial dependencies in PDEs, we extend the S4D framework to spatial dimensions. A key requirement for universality of factorized convolutional neural operators is a full field of view, as formalized in Theorem 4.1. A unidirectional SSM does not satisfy this condition because each spatial location only aggregates information from one side, limiting expressivity. Empirically, we confirm in Appendix C.1 that unidirectional scans underperform, motivating the use of bidirectional processing.

For a 1D spatial domain discretized into X points, we define two SSM models: $\mathcal{M}_{fwd}$ and $\mathcal{M}_{bwd}$, where the $_{fwd}$ and $_{bwd}$ labels serve as identifiers rather than indicating scan direction. The processing is computed as

$$y_{fwd} = \mathcal{M}_{fwd}(u), \qquad y_{bwd} = \text{flip}\left(\mathcal{M}_{bwd}(\text{flip}(u))\right), \tag{2}$$

with the final output $y = y_{fwd} + y_{bwd}$. For multi-dimensional grids, such as $X \times Y$, bidirectional SSMs are applied sequentially along each axis (first x, then y), ensuring every spatial location has a full field of view and satisfying Theorem 4.1.

Comparison with Existing Methods. Unlike Vision Mamba (Zhu et al., 2024), which processes 2D data in a zigzag manner using bidirectional SSMs, our approach applies SSMs separately along each spatial dimension. While Factorized Fourier Neural Operators (F-FNO) (Tran et al., 2023) process each spatial dimension in parallel and sum the results, our method applies SSMs sequentially, allowing more expressive modeling of spatial interactions. Due to memory constraints, we do not employ Mamba-based SSMs directly but explore alternative 2D and factorized spatial SSM configurations using S4D in Appendix D.

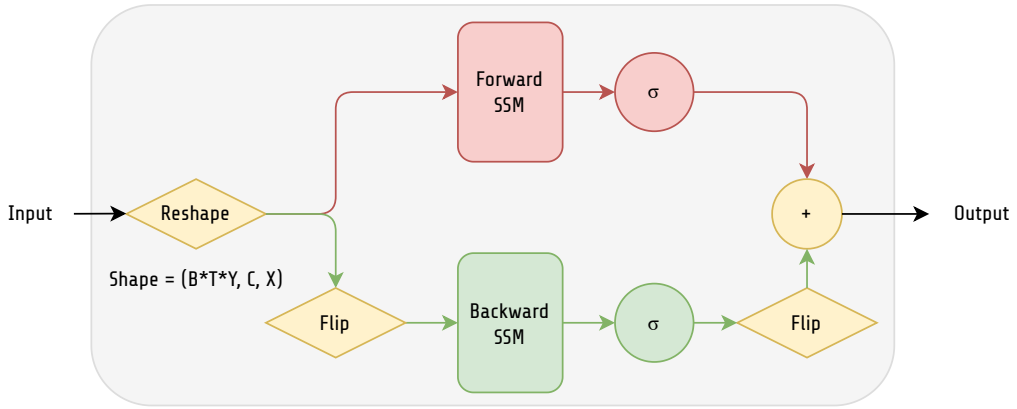

Figure 1: Detailed illustration of the spatial bidirectional SSM module. B: batch size, T: temporal length, X: spatial dimensions, C: input channels, σ: pointwise nonlinearity, and $+$: element-wise addition. The input is processed through both a forward spatial SSM and a flipped backward spatial SSM. Each path includes a residual connection and nonlinear activation, and their outputs are aggregated to form the final output.

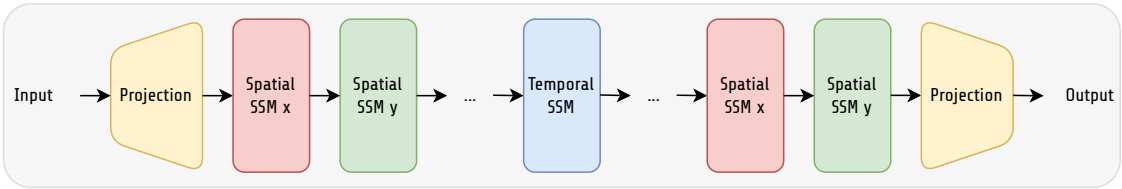

Figure 2: Architecture combining Markovian 1D spatial SSM modules with a single temporal SSM following the MemNO framework. The spatial SSMs are applied sequentially across spatial dimensions, while the temporal SSM's position within the stack is a tunable hyperparameter.

3.4 Full SS-NO Model with Temporal Memory

The SS-NO architecture combines Markovian spatial SSM layers (Section 3.3) with a single non-Markovian temporal memory module following the MemNO framework to capture spatiotemporal dependencies in PDEs. Spatial layers process local interactions sequentially across dimensions, while the temporal module aggregates information across previous timesteps. During inference, the temporal module maintains and updates a hidden state at each step, which is combined with spatial processing for next-step predictions.

The number of spatial layers and the placement of the temporal layer within the network stack are tunable hyperparameters optimized for specific PDE characteristics. Figures 1 and 2 illustrate the architecture: spatial SSM modules perform bidirectional processing with residual connections, while the full model integrates these with the temporal module. Complete implementation details are provided in Appendix G.

4 Theory

We view the proposed SS-NO architecture as an instance of a popular neural operator paradigm Kovachki et al. (2023), combining two types of layers: (a) pointwise composition with nonlinear activations, and (b) application of nonlocal convolution operators, and with prototypical hidden layers of the form,

$$\mathcal{L}: \; v(x) \mapsto \underbrace{\sigma(Wv(x) + b}_{\text{(a) nonlinear}} + \underbrace{\textstyle\int_D \kappa(x-y)v(y)\,dy}_{\text{(b) nonlocal}}), \tag{3}$$

where $v : D \to \mathbb{R}^H$ is a hidden state, σ is a standard activation function (e.g., GELU), $W \in \mathbb{R}^{H\times H}$ is a weight matrix, $b \in \mathbb{R}^H$ a bias vector and $\kappa : D \to \mathbb{R}^{H\times H}$ is a learnable integral-kernel.

To give a unified discussion, we will say that Ψ is a **convolutional NO**, if it is of the form $\Psi(u) = \mathcal{Q}\circ\mathcal{L}_L\circ\cdots\circ\mathcal{L}_1\circ\mathcal{R}(u)$, with hidden layers $\mathcal{L}_\ell$ as in equation 3, and choice of parametrized kernel $\kappa_\ell(x) = \kappa_\ell(x;\theta)$. In addition, we have a lifting layer $\mathcal{R}(u)(x) := R(u(x), x)$ and projection layer $\mathcal{Q}(v(x)) = Q(v(x))$ by composition with shallow neural networks R, Q. We next discuss three architectures exemplifying this approach, highlighting their commonality and giving a unified description. This results in a sharp *criterion for universality* for any such convolutional NO architecture.

Fourier neural operator (FNO). The FNO parametrizes the convolution kernel by a truncated Fourier series, $\kappa(x) = \sum_{|k|_\infty\leq K} \hat{\kappa}_k e^{ikx}$, with cut-off parameter K, and $|k|_\infty = \max_{j=1,\dots,d} |k_j|$. The Fourier coefficients $\hat{\kappa}_k \in \mathbb{C}^{H\times H}$ are complex-valued matrices. The parametrization of the integral kernel $\kappa(x)$ of FNO in d dimensions entails a considerable memory footprint, requiring $O(LH^2K^d)$ parameters. This can render FNO prohibitive in high-dimensional applications.

Factorized FNO (F-FNO). To lessen the computational demands of FNO, so-called "factorized" architectures have been introduced. Here, convolutions are taken along one dimension at a time. Mathematically, this corresponds to choosing kernels of the form $\kappa(x) = \kappa_s(x_j)\prod_{k\neq j}\delta(x_k)$, where $\kappa_s(x_j)$ is a 1d FNO kernel and $\delta(x_k)$ the Dirac delta function. Thus, integration is effectively only performed with respect to x_j. The resulting F-FNO only requires $O(LH^2Kd)$ parameters Tran et al. (2023).

Spatial SSM. We next consider the spatial convolution layers of the proposed SS-NO architecture. In this case, solution of the ODE system equation 1 leads to an explicit formula for the corresponding kernel. For a 1d spatial domain, this results in a kernel $\kappa(x) = \kappa_+(x) + \kappa_-(x)$, where $\kappa_\pm(x)$ correspond to the backward and forward scans, respectively, and $\kappa_\pm(x)$ is of the form

$$\kappa_\pm(x) = 1_{\mathbb{R}_\pm}(x)\ \sum_{k=1}^{K} e^{r_k|x|}e^{i\omega_k x}\, C_k B_k^T,\ r_k = \mathrm{Real}(\lambda_k),\ \omega_k = \mathrm{Imag}(\lambda_k),\ B_k, C_k \in \mathbb{R}^H. \tag{4}$$

Here $1_{\mathbb{R}_\pm}(x)$ is the indicator function of $\mathbb{R}_\pm = \{\pm x \geq 0\}$. The main difference with FNO lies in the parametrization of the convolutional kernel κ, which now has tunable frequency parameters $\omega_k \in \mathbb{R}$, and tunable damping parameters $r_k < 0$. The parameter count of the resulting factorized spatial SSM architecture in d-dimensions requires at most $O(LH(H+K)d)$ parameters.

A comparison of the model parameter count entailed by the above choices is summarized below:

Model	**FNO**	**F-FNO**	**spatial-SSM**
# Parameter (scaling)	LH^2K^d	LH^2Kd	$L(H^2+HK)d$

4.1 A Sharp Criterion for Universality of Convolutional NOs.

As highlighted above, FNO, factorized FNO and SSMs all share a common structure, distinguished by the chosen kernel parametrization. *What can be said about the expressivity of such architectures?* Our first goal is to derive a sharp, general condition for the universality of such convolutional NO architectures.

A minimal requirement for the universality of any neural network architecture is that the value of each output pixel must be informed by the values of *all* input pixels, i.e. the model needs to have a "full field of view". In the context of convolutional NO architectures, this leads to the following definition:

Definition. *A convolutional NO architecture with layer kernels $\kappa_1, \dots, \kappa_L$ has a **full field of view**, if the iterated kernel $\bar{\kappa} := \kappa_L * \kappa_{L-1} * \cdots * \kappa_1$ is non-vanishing, i.e. $\bar{\kappa}(x-y) \neq 0$ for all $x, y \in D$.*

The next result shows that this necessary condition is actually already sufficient for the universality of a convolutional architecture, requiring no additional assumptions on the convolution operators:

Theorem 4.1. *A (factorized) convolutional NO architecture is universal if it has a full field of view.*

We refer to Appendix B for the precise statement and proof. To the best of our knowledge, the criterion identified above is both the simplest and most widely applicable result for convolutional NO architectures;

it implies universality of (vanilla) FNO, factorized FNOs and combinations of (F-)FNO and SSMs, SS-NO and even gives a sharp criterion for the universality of *localized* convolutional NOs (similar to the CNO Raonic et al. (2023)), for which κ_ℓ has localized support. A theoretical basis for both factorized and localized architectures had been missing from the literature. The above result closes this gap. Appendix C.1 contrasts empirical results with a unidirectional SSM that violates the criterion.

4.2 Adaptivity and Enhanced Model Expressivity of SS-NO

From the formula equation 4 it is evident that (factorized) spatial SSM is able to exactly recover FNO (in 1d), or F-FNO. In fact, it can represent any 1d convolutional kernel of the form

$$\kappa(x) = \sum_{k=1}^{K} c_k e^{-\rho_k |x|} e^{i\omega_k x}, \quad \rho_k \geq 0, \quad \omega_k \in \mathbb{R}. \tag{5}$$

When restricting this choice to $\rho_k = 0$ and $\omega_k = k$, we recover the FNO kernel parametrization. However, since the damping and frequency are *tunable parameters* within SS-NO, this adds further adaptivity: (1) choice of $\rho_k > 0$ allows effective *kernel localization*; the model can optimize the support of its convolutional kernel, interpolating between the global kernels of FNO ($\rho_k \approx 0$) and very localized CNN-like kernels ($\rho_k \gg 1$). (2) additional adaptivity comes from *adaptive mode-filtering*; the model can optimize the frequencies $\omega_1, \ldots, \omega_K$, most relevant for nonlocal processing. Thus, in theory, this added adaptivity implies enhanced model expressivity of SS-NO over F-FNO.

5 Experiment Setup

5.1 Setup and Dataset Description

We evaluate our models on a suite of one-dimensional (1D) and two-dimensional (2D) partial differential equation (PDE) benchmarks commonly used in operator learning. These datasets span a variety of dynamical regimes—from chaotic behavior to turbulent flows—and are designed to assess the models' ability to capture long-range temporal dependencies. For 1D problems, we use datasets based on the Burgers' equation (nonlinear convection–diffusion) and the Kuramoto–Sivashinsky equation (chaotic dynamics), following the same data sources as Buitrago et al. (2025) and generating low-resolution versions through uniform spatial downsampling.

For 2D problems, we evaluate several widely used benchmarks, including the `TorusLi`, `TorusVis`, and `TorusVisForce` datasets (Li et al., 2021; Tran et al., 2023), as well as compressible Euler benchmarks such as the Richtmeyer–Meshkov instability (CE-RM) and the Rayleigh–Taylor instability with gravitational forcing (GCE-RT) (Herde et al., 2024). The `TorusVis` and `TorusVisForce` datasets incorporate variable, time-dependent forcing under randomly sampled viscosities in the range $\nu \in [10^{-5}, 10^{-4})$. Together, these benchmarks test the models' ability to capture turbulence, long-range interactions, shocks, chaotic dynamics, and interface instabilities.

For evaluation, we compare SS-NO against a suite of popular baseline models including U-Net (Gupta & Brandstetter, 2023), GKT (Cao, 2021), Factformer (Li et al., 2023), and FFNO (Tran et al., 2023; Buitrago et al., 2025) for 1D benchmarks, and additionally FNO with a two-dimensional kernel (2D FNO) (Li et al., 2021) for 2D problems. Following the MemNO framework (Buitrago et al., 2025), we augment all baselines and SS-NO with a temporal S4 module using a memory window of $K = 4$ for fair comparison. The only exception is GKT, for which we use a multi-input variant with $K = 4$ where the temporal dimension is mixed with features, as we found the model struggled to converge with an additional temporal S4 module.

Data Preprocessing and Training Setup. We follow standard practices from prior work (Tran et al., 2023), normalizing all data to the range $[0, 1]$ and adding Gaussian noise with variance 10^{-3} during training for stability. Models are trained to minimize the step-wise normalized relative ℓ_2 loss.

Unless otherwise specified, all models use four blocks with consistent hidden dimensions. Full details on baseline models, data generation, preprocessing, and training are provided in Appendices E and F.

5.2 Training Objective.

We train the model by minimizing the empirical risk over the dataset of trajectories. Given a loss function $\ell : L^2(\Omega; \mathbb{R}^V) \times L^2(\Omega; \mathbb{R}^V) \to \mathbb{R}$, we solve:

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \frac{1}{N_{\text{out}}} \sum_{t \in \mathcal{T}_{\text{out}}} \ell \left(u^{(i)}(t,x),\ \Psi_{\theta,t} \left(u^{(i)}(t',x)|_{t' \in \mathcal{T}_{\text{in}}} \right) \right), \tag{6}$$

where $\Psi_{\theta,t}(\cdot)$ denotes the prediction at time $t \in \mathcal{T}_{\text{out}}$. As our loss ℓ, we employ relative $\mathcal{L}^2$-error, discussed below. The formulation equation 6 enables the model to learn the entire future evolution of the system dynamics from a finite observed window, rather than stepwise or autoregressive forecasting.

5.3 Evaluation Metric.

Performance is reported using the relative $\mathcal{L}^2$ error:

$$\textit{Relative } \mathcal{L}^2(u(t,x), \hat{u}(t,x)) = \frac{\|u(t,x) - \hat{u}(t,x)\|_2}{\|u(t,x)\|_2}, \tag{7}$$

where $\| \cdot \|_2$ denotes the squared norm over all spatial locations and time steps in the output horizon.

All results reported in the main text represent the mean performance, measured by the **best validation loss**, across **five independent runs** with different random seeds. To contextualize performance relative to model complexity, we also report the **number of parameters** for each model, defined as the total count of learnable parameters. We note that for all 1D experiments, the standard deviation is less than $\pm\mathbf{5}\%$ of the reported mean at resolutions 128 and 64, and less than $\pm\mathbf{7}\%$ at resolution 32, while for all 2D experiments, the standard deviation is below $\pm\mathbf{0.3}\%$ in relative $\mathcal{L}^2$ error. The exceptions to this are GKT and Factformer (2D), which exhibit slightly higher variability.

Table 1: Relative $\mathcal{L}^2$ error on 1D benchmarks at varying resolutions.

Resolution	**Architecture**	**# Parameters**	**Relative $\mathcal{L}^2$ Error**			
			KS			Burgers'
			$\nu = 0.075$	$\nu = 0.1$	$\nu = 0.125$	$\nu = 0.001$
128	U-Net	2,728,641	0.0831	0.0703	0.0377	0.0682
	GKT	291,521	0.0163	0.0054	0.0033	0.0294
	Factformer (1D)	677,345	0.0154	0.0094	0.0063	0.0089
	FFNO	2,192,833	0.0110	0.0045	0.0031	0.0085
	SS-NO (ours)	203,713	**0.0086**	**0.0026**	**0.0013**	**0.0070**
64	U-Net	2,728,641	0.0782	0.0387	0.0318	0.0613
	GKT	160,449	0.0478	0.0122	0.0035	0.0180
	Factformer (1D)	677,345	0.0559	0.0096	0.0053	0.0147
	FFNO	1,144,257	0.0199	0.0066	0.0039	0.0146
	SS-NO (ours)	203,713	**0.0143**	**0.0048**	**0.0029**	**0.0121**
32	U-Net	2,728,641	0.1158	0.0688	0.0638	0.0580
	GKT	94,913	0.1135	0.0979	0.0636	0.0301
	Factformer (1D)	677,345	0.0641	0.0277	0.0136	0.0224
	FFNO	619,969	0.0601	0.0231	0.0129	0.0217
	SS-NO (ours)	203,713	**0.0472**	**0.0162**	**0.0088**	**0.0200**

6 Results

6.1 1D Burgers' Equation

We evaluate model performance on the canonical 1D Burgers' equation with $\nu = 0.001$ across temporal resolutions of 128, 64, and 32. As shown in Table 1, SS-NO achieves competitive accuracy at all resolutions while demonstrating exceptional parameter efficiency.

Notably, SS-NO delivers superior performance with just over $200k$ parameters—nearly an order of magnitude fewer than many baselines. Unlike Fourier-based models whose parameter counts scale with input resolution and spectral modes, SS-NO maintains a constant parameter count independent of resolution. This architectural advantage is particularly valuable for applications requiring deployment across multiple spatial or temporal scales.

The Burgers' equation exhibits smooth dynamics dominated by diffusion and mild nonlinearity, and SS-NO's consistent performance across resolutions highlights its robustness in handling such well-behaved systems. The model's ability to maintain accuracy while drastically reducing parameter count represents a significant advancement in efficient operator learning.

6.2 1D Kuramoto–Sivashinsky Equation

We next evaluate models on the 1D Kuramoto–Sivashinsky (KS) equation across three viscosities ($\nu = 0.075, 0.1, 0.125$) and multiple temporal resolutions. The KS equation presents a more challenging benchmark with increasingly chaotic dynamics at lower viscosities.

As shown in Table 1, SS-NO demonstrates consistent performance improvements, achieving the best results among evaluated methods across all viscosity regimes. At resolution 128, SS-NO outperforms the second-best model (FFNO) by 22% at $\nu = 0.075$, 42% at $\nu = 0.1$, and 58% at $\nu = 0.125$. These improvements are particularly pronounced in the most challenging low-viscosity regime where chaotic behavior dominates.

The performance advantage amplifies at coarser temporal resolutions, suggesting that SS-NO's damping mechanisms provide crucial stabilization under complex dynamics. At resolution 32, SS-NO maintains a 21% improvement over FFNO at $\nu = 0.075$ and 30% at $\nu = 0.1$. This effect diminishes at higher viscosities where smoother system behavior allows all models to perform more comparably, though SS-NO still maintains superiority.

Notably, SS-NO maintains consistent performance gains even as resolution decreases, outperforming other temporal S4-augmented models across all tested conditions. The architecture's ability to capture long-range temporal dependencies proves particularly valuable under limited-resolution conditions, where traditional methods struggle. These results collectively demonstrate that SS-NO achieves an optimal balance of accuracy, efficiency, and robustness—excelling in both smooth and chaotic regimes while maintaining parameter efficiency across varying resolutions.

6.3 Ablation Study

We conduct a comprehensive ablation study to understand the individual contributions of model capacity, learnable frequencies, and explicit damping mechanisms. Four distinct configurations are evaluated: **All**, where both frequencies and damping are learnable; **Damping only**, where frequencies remain fixed while damping is learnable; **Frequency only**, where frequencies are learnable but damping is explicitly set to zero (absent); and **Fixed**, where neither frequencies nor damping are learnable.

Experiments are conducted on three variants of the KS equation with viscosities $\nu \in \{0.075, 0.1, 0.125\}$, all at resolution 128. Performance is quantified using relative ℓ_2 error. Results are presented in Table 2 to isolate the effects of architectural components across different capacity regimes.

6.3.1 Observations

Explicit damping enables remarkable parameter efficiency. The most striking finding is that proper damping mechanisms allow models to maintain strong performance even under extreme capacity constraints. The 16-state damping-only model achieves performance competitive with much larger models, demonstrating that damping serves as a powerful regularization and stabilization mechanism that dramatically improves parameter efficiency. This efficiency is particularly evident in challenging regimes ($\nu = 0.075$), where the 16-state damping-only model approaches the performance of fixed-configuration models with 2-4$\times$ more states, despite its significantly reduced capacity.

Model capacity dramatically affects damping requirements. Higher-capacity models can implicitly compensate for architectural limitations through additional spectral modes, while lower-capacity models rely critically on explicit damping for stability. The performance degradation in 16-state models without damping versus 64-state models confirms that damping becomes increasingly essential as model capacity decreases. This progressive dependency relationship underscores damping's role as a crucial stabilization mechanism in capacity-constrained settings.

Damping significance escalates with problem difficulty. The performance gap between models with and without damping grows substantially with increasing chaos (decreasing viscosity). For $\nu = 0.075$, proper damping provides significant absolute improvement, while for $\nu = 0.125$, the benefit reduces substantially. This differential effect confirms that chaotic dynamics benefit substantially from explicit damping mechanisms, while smoother regimes can tolerate their absence.

Learnable frequencies provide consistent spectral adaptation benefits. Across all configurations, the frequency-only setting consistently outperforms the fixed configuration, demonstrating the value of data-driven spectral basis adaptation. This benefit is particularly pronounced in constrained settings and complex regimes like KS $\nu = 0.075$, where learnable frequencies allow models to dynamically allocate their limited spectral resources to the most relevant temporal modes. Rather than being constrained to a fixed Fourier basis, models can adapt their frequency representations to focus on the specific oscillatory patterns most critical for the target dynamics, leading to more efficient temporal representation learning. While these gains are generally more modest than those provided by explicit damping—particularly in challenging regimes—they confirm that learnable frequencies provide meaningful improvements in spectral efficiency.

Results are not driven by overparameterization. The consistent performance pattern across capacity levels confirms that our findings are not artifacts of excessive model size. Even the smallest models (16 states) achieve competitive performance when equipped with appropriate inductive biases, demonstrating that architectural design rather than parameter count drives performance improvements.

Table 2: Relative $\mathcal{L}^2$ error on KS benchmarks at varying state sizes and training settings.

State Size	**Setting**	**# Parameters**	**Relative $\mathcal{L}^2$ Error**		
			KS		
			$\nu = 0.075$	$\nu = 0.1$	$\nu = 0.125$
64	All	203,713	0.0086	0.0026	0.0013
	Damping only	187,329	0.0086 (+0.0000)	0.0027 (+0.0001)	0.0013 (+0.0000)
	Frequency only	187,329	0.0110 (+0.0024)	0.0030 (+0.0003)	0.0014 (+0.0001)
	Fixed	170,945	0.0116 (+0.0030)	0.0033 (+0.0006)	0.0016 (+0.0003)
32	All	154,561	0.0090	0.0027	0.0012
	Damping only	146,369	0.0094 (+0.0004)	0.0028 (+0.0001)	0.0014 (+0.0002)
	Frequency only	146,369	0.0137 (+0.0047)	0.0038 (+0.0011)	0.0018 (+0.0006)
	Fixed	138,177	0.0158 (+0.0068)	0.0042 (+0.0015)	0.0020 (+0.0008)
16	All	129,985	0.0115	0.0038	0.0017
	Damping only	125,889	0.0131 (+0.0016)	0.0044 (+0.0006)	0.0019 (+0.0002)
	Frequency only	125,889	0.0198 (+0.0083)	0.0069 (+0.0031)	0.0031 (+0.0014)
	Fixed	121,793	0.0227 (+0.0112)	0.0068 (+0.0030)	0.0031 (+0.0014)

6.3.2 Damping Analysis: Connecting Ablation Results to Learned Behavior

The ablation results naturally raise the question: how do these performance differences manifest in the actual learned damping behavior? To answer this, we conduct a detailed analysis of the learned damping distributions across different configurations, aggregating results from five independently trained models with different random seeds for each case to ensure statistical robustness.

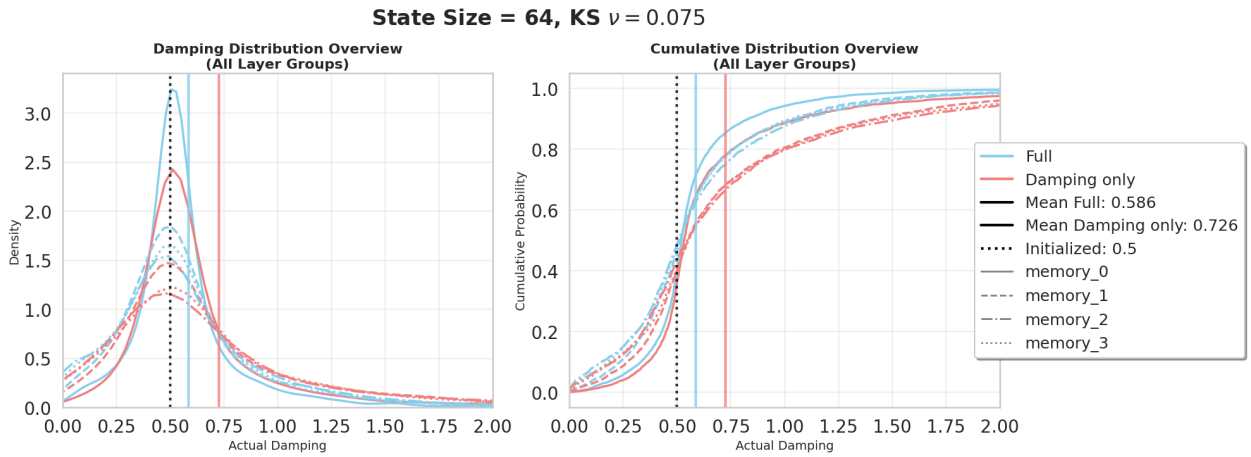

Figure 3: Damping coefficient distributions for full vs. damping-only models at $\nu = 0.075$ with 64 states.

Architectural constraints drive stronger damping. Quantitative analysis reveals that damping-only models learn 24% **stronger mean damping** than full models (0.726 vs. 0.586, $p = 5.2 \times 10^{-118}$), representing a 163% **larger relative increase** from initialization (45.3% vs. 17.2% increase from 0.5), as shown in Figure 3. This compensatory strengthening explains the ablation results: when frequency adaptation is disabled, models intensify their damping mechanisms to maintain stability, directly corresponding to the observed patterns in Table 2.

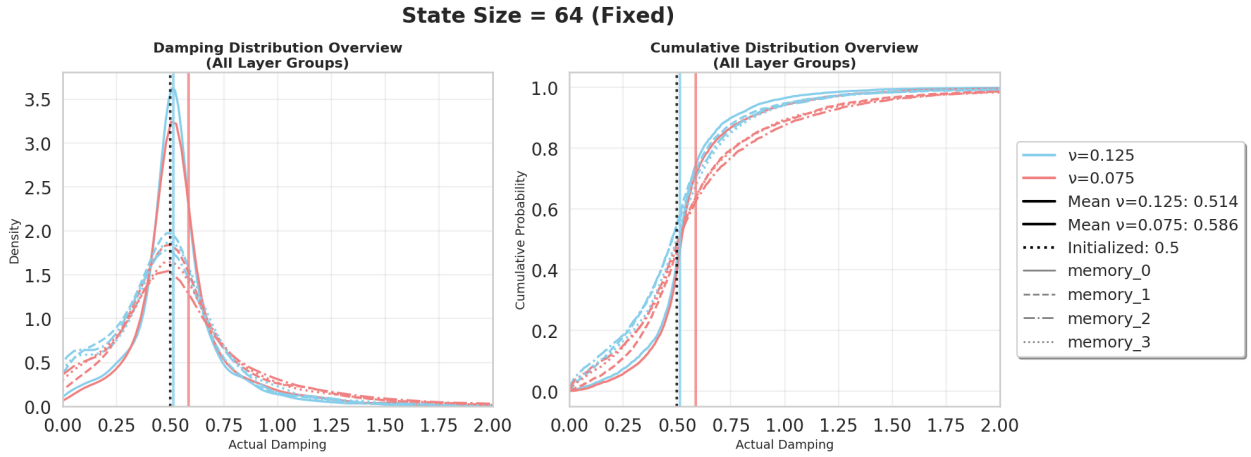

Figure 4: Damping coefficient distributions for $\nu = 0.125$ vs. $\nu = 0.075$ models with 64 states.

Problem difficulty modulates damping strength. Models trained on more chaotic dynamics ($\nu = 0.075$) learn 14% **stronger damping** than those on smoother regimes ($\nu = 0.125$, 0.586 vs. 0.514, $p = 5.8 \times 10^{-71}$), with a 537% **larger relative increase** from initialization (17.2% vs. 2.7% increase from 0.5), as visualized in Figure 4. This dramatic differential learning directly mirrors the ablation findings where damping provides exponentially greater benefits in challenging regimes.

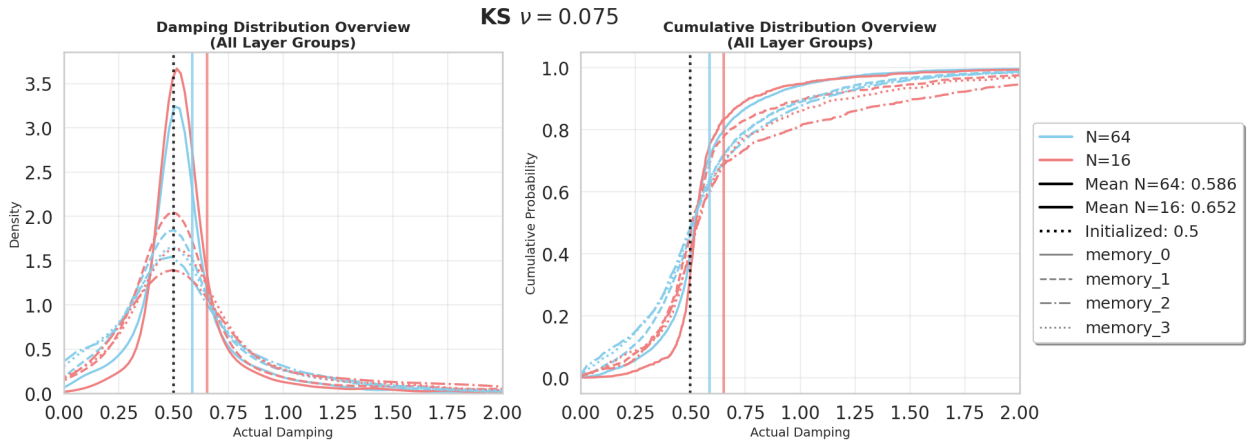

Figure 5: Damping coefficient distributions at $\nu = 0.075$ for different state sizes.

Capacity constraints amplify damping requirements. Lower-capacity models (16 states) learn 11% **stronger damping** than higher-capacity counterparts (64 states, 0.652 vs. 0.586, $p = 2.2 \times 10^{-15}$), with a 77% **larger relative increase** from initialization (30.4% vs. 17.2% increase from 0.5), as demonstrated in Figure 5. This intensified damping in capacity-constrained models explains their ability to maintain stability despite reduced parameter counts, directly connecting to the efficiency results in our ablation study.

The consistent pattern across all three analyses—supported by overwhelming statistical significance ($p < 10^{-15}$ in all cases)—demonstrates that **damping serves as a crucial stabilization mechanism that scales intelligently with architectural and problem constraints**. Models automatically learn to strengthen damping in response to constraints, providing a mechanistic explanation for the performance patterns observed in our ablation study. This adaptive damping behavior represents a key innovation that enables both parameter efficiency and robustness across diverse dynamical regimes.

Table 3: Relative $\mathcal{L}^2$ error on 2D Navier–Stokes datasets at 64×64 resolution.

Architecture	**# Parameters**	**Relative $\mathcal{L}^2$ Error**				
		`TorusLi`	`TorusVis`	`TorusVisForce`	`CE-RM`	`GCE-RT`
U-Net	$7,783,777$	0.0611	0.0474	0.0558	0.0644	0.0300
Factformer (2D)	$1,006,433$	0.0628	0.0337	0.0395	0.0642	0.0315
GKT	$8,418,049$	0.0619	0.0404	0.0742	0.0691	0.0172
FNO2D	$67,197,700$	0.0760	0.0466	0.0444	0.0717	0.0155
FFNO	$2,192,897$	0.0409	0.0231	0.0326	0.0688	0.0196
SS-NO (ours)	$369,665$	**0.0345**	**0.0218**	**0.0263**	**0.0583**	**0.0138**

6.4 2D Navier–Stokes Equations

Navier–Stokes with Fixed Viscosity. We evaluate our model on the `TorusLi` dataset, which contains vorticity fields generated by solving the 2D incompressible Navier–Stokes equation on the unit torus with a fixed forcing term and low viscosity $\nu = 10^{-5}$. This regime is particularly challenging due to reduced diffusion, leading to highly nonlinear and chaotic dynamics.

Table 3 reports the relative $\mathcal{L}^2$ errors of various baselines. Among them, FNO2D achieves an error of 0.0760 with over 67 million parameters. In contrast, SS-NO reduces the error to 0.0345—a 54.5% improvement—while using just 369k parameters, a 182× reduction in model size.

This performance gap is notable given the differences in scaling. FNO2D leverages a full 2D Fourier transform to model frequency interactions jointly across both spatial dimensions, with cost scaling as $\mathcal{O}(N^D)$, where D

is the number of dimensions and N is the resolution per dimension. In contrast, SS-NO applies separate 1D operators along each axis in a factorized manner, achieving more favorable $\mathcal{O}(N \cdot D)$ complexity while retaining or improving accuracy. These results demonstrate the advantage of temporal-state modeling even in time-invariant contexts, highlighting the role of state-space modeling for capturing fluid dynamics in low-viscosity regimes. Among several factorized connection variants we tested, our configuration delivered the best performance (see Appendix D).

2D Navier-Stokes: Problems with Varying Viscosities and Forces. We also evaluate our model on the `TorusVis` and `TorusVisForce` datasets from the FFNO benchmark Tran et al. (2023), where both viscosity and external forcing vary across trajectories. In `TorusVis`, viscosity is sampled uniformly between 10^{-5} and 10^{-4} and the forcing function is time-independent. `TorusVisForce` introduces a time-varying forcing function with phase shift $\delta = 0.2$.

Table 3 summarizes the results. SS-NO achieves the lowest errors across both datasets, consistently outperforming all baselines. This demonstrates that SS-NO effectively captures spatiotemporal dynamics across mixed-viscosity regimes, where larger viscosity values in the data are significantly easier to predict, while still handling the challenging low-viscosity cases.

Compressible Euler Benchmarks (CE-RM and GCE-RT). We also evaluate SS-NO on compressible Euler benchmarks: the Richtmeyer–Meshkov instability (CE-RM) and the Rayleigh–Taylor instability with gravitational forcing (GCE-RT). These problems involve complex interface dynamics and shocks, making them particularly challenging for predictive modeling.

Table 3 shows that CE-RM has relatively high errors across all models, reflecting its intrinsic difficulty. Interestingly, on GCE-RT, GKT and FNO2D perform better compared to other factorized baselines, suggesting that explicitly modeling full 2D spatial interactions can be beneficial for this problem.

Despite being factorized, SS-NO achieves the lowest relative $\mathcal{L}^2$ errors on both CE-RM (0.0583) and GCE-RT (0.0138), demonstrating that it can efficiently capture the dominant spatiotemporal dynamics even in problems with shocks and complex interface interactions. These results highlight that SS-NO combines the efficiency of factorized operations with strong modeling capacity, making it competitive with full 2D approaches in capturing challenging fluid behavior.

7 Conclusion and Future Work

We have presented the State Space Neural Operator (SS-NO), a compact neural operator designed to efficiently learn solution operators for time-dependent partial differential equations. Grounded in a theoretical universality result for convolutional operator learning with full field-of-view, SS-NO generalizes and improves upon prior factorizations such as FNO by integrating adaptive mechanisms—including damping for receptive field localization and learnable frequency modulation for data-driven mode selection—while maintaining temporal causality. Our spatial-only variant exactly recovers F-FNO, yet SS-NO dramatically reduces parameter count by exploiting a linear $\mathcal{O}(N \cdot D)$ scaling compared to the $\mathcal{O}(N^D)$ complexity of high-dimensional convolutions.

Empirically, SS-NO achieves competitive performance across a suite of 1D and 2D benchmarks, including the Burgers', Kuramoto–Sivashinsky, Navier–Stokes, and Euler equations, demonstrating strong accuracy with significantly fewer parameters. In addition, we proposed a dimensionally factorized variant of SS-NO, which scales favorably in higher dimensions and achieves competitive results on challenging 2D fluid dynamics datasets. These findings highlight that state space modeling is not only compatible with operator learning but also offers unique advantages in long-term memory, adaptivity, and efficiency.

While our formulation provides clear benefits for structured spatiotemporal domains, one limitation is the lack of a systematic connection between damping-based receptive field localization and irregular geometries. Developing principled extensions of SS-NO to unstructured meshes and complex domains remains an important direction for future work. Beyond this, we aim to apply SS-NO to higher-dimensional and multi-physics systems, including astrophysical, cosmological, and space plasma dynamics, where efficient and generalizable PDE operators are critical.

References

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

Johannes Brandstetter, Max Welling, and Daniel E Worrall. Lie point symmetry data augmentation for neural pde solvers. In *The International Conference on Machine Learning (ICML)*, 2022.

Ricardo Buitrago, Tanya Marwah, Albert Gu, and Andrej Risteski. On the benefits of memory for modeling time-dependent pdes. In *The International Conference on Learning Representations (ICLR)*, 2025.

Shuhao Cao. Choose a transformer: Fourier or galerkin. In *Advances in neural information processing systems (NeurIPS)*, 2021.

Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE transactions on neural networks*, 6(4):911–917, 1995.

Chun-Wun Cheng, Jiahao Huang, Yi Zhang, Guang Yang, Carola-Bibiane Schönlieb, and Angelica I Aviles-Rivero. Mamba neural operator: Who wins? transformers vs. state-space models for pdes. *arXiv preprint arXiv:2410.02113*, 2024. URL https://arxiv.org/abs/2410.02113.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *The Conference on Language Modeling (COLM)*, 2024.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022a.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*, 2022b.

Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *Transactions on Machine Learning Research*, 2023.

Maximilian Herde, Bogdan Raonic, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Zheyuan Hu, Nazanin Ahmadi Daryakenari, Qianli Shen, Kenji Kawaguchi, and George Em Karniadakis. State-space models are accurate and efficient neural operators for dynamical systems. *arXiv preprint arXiv:2409.03231*, 2024. URL https://arxiv.org/abs/2409.03231.

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, 2021. URL http://jmlr.org/papers/v22/21-0806.html.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Nikola B. Kovachki, Samuel Lanthaler, and Andrew M. Stuart. Operator learning: Algorithms and analysis. In *Handbook of Numerical Analysis*. Elsevier, 2024. doi: 10.1016/bs.hna.2024.05.009. URL https://www.sciencedirect.com/science/article/pii/S1570865924000097.

Samuel Lanthaler, Siddhartha Mishra, and George E Karniadakis. Error estimates for DeepONets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001, 2022. URL https://academic.oup.com/imatrm/article-pdf/6/1/tnac001/42785544/tnac001.pdf.

Samuel Lanthaler, Zongyi Li, and Andrew M. Stuart. Nonlocality and nonlinearity implies universality in operator learning, 2024. URL https://arxiv.org/abs/2304.13221.

Fanny Lehmann, Filippo Gatti, Michaël Bertin, and Didier Clouteau. Seismic hazard analysis with a factorized fourier neural operator (f-FNO) surrogate model enhanced by transfer learning. In *NeurIPS 2023 AI for Science Workshop*, 2023. URL https://openreview.net/forum?id=xiNRyrBAjt.

Fanny Lehmann, Filippo Gatti, Michaël Bertin, and Didier Clouteau. 3d elastic wave propagation with a factorized fourier neural operator (f-fno). *Computer Methods in Applied Mechanics and Engineering*, 420:116718, 2024. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2023.116718. URL https://www.sciencedirect.com/science/article/pii/S0045782523008411.

Zijie Li, Dule Shu, and Amir Barati Farimani. Scalable transformer for pde surrogate modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, et al. Fourier neural operator for parametric partial differential equations. In *The International Conference on Learning Representations (ICLR)*, 2021.

Phillip Lippe, Bas Veeling, Paris Perdikaris, Richard Turner, and Johannes Brandstetter. Pde-refiner: Achieving accurate long rollouts with neural pde solvers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

Hazime Mori. Transport, collective motion, and brownian motion. *Progress of theoretical physics*, 33(3): 423–455, 1965.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *The International Conference on Machine Learning (ICML)*, 2010.

Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators. *Transactions on Machine Learning Research*, 2023.

Bogdan Raonic, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel de Bézenac. Convolutional neural operators for robust and accurate learning of pdes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. In *The International Conference on Learning Representations (ICLR)*, 2023.

Jianwei Zheng, Wei Li, Ni Xu, Junwei Zhu, and Xiaoqin Zhang. Alias-free mamba neural operator. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. In *The International Conference on Machine Learning (ICML)*, 2024.

Robert Zwanzig. Memory effects in irreversible thermodynamics. *Physical Review*, 124(4):983, 1961.

A Appendix

B Universality of Convolutional NOs

We here provide a rigorous definition of convolutional NOs having a "full field of view", and provide the precise statement of our sharp universality condition.

B.1 Full Field of View

We describe the required full field of view property in greater detail, which we informally stated as a definition before Theorem 4.1 in the main text.

The following assumptions were implicit in that definition: The convolutional NO architecture is of the form

$$\Psi(u) = \mathcal{Q} \circ \mathcal{L}_L \circ \cdots \circ \mathcal{L}_1 \circ \mathcal{R}(u),$$

where the lifting layer $\mathcal{R}(u)(x) := R(u(x), x)$ and projection layer $\mathcal{Q}(v(x)) = Q(v(x))$ are given by composition with shallow neural networks R, Q of width H (both applied pointwise to the respective input functions). The hidden layers $\mathcal{L}_\ell$ are given by

$$\mathcal{L}_\ell(v)(x) = \sigma\left(W_\ell v(x) + b_\ell + \int_D \kappa_\ell(x-y;\theta_\ell) v(y)\, dy\right),$$

with channel width H. We will assume that H can be chosen as large as required. For the integral kernel κ_ℓ, we assume that, for any choice of H and matrix $A_\ell \in \mathbb{R}^{H\times H}$ acting on the channel dimension, there exists a setting of the parameters θ_ℓ^*, such that $\kappa_\ell(x;\theta_\ell^*) \in \mathbb{R}^{H\times H}$ is a scalar multiple of A, i.e. the kernel $\kappa_\ell(\,\cdot\,;\theta_\ell^*)$ is the form

$$\kappa_\ell(x;\theta_\ell^*) \in \mathbb{R}^{H\times H} = g_\ell(x)\, A_\ell, \quad g_\ell : D \to \mathbb{R} \text{ some scalar function.} \tag{8}$$

To our knowledge, all proposed architectures, including FNO, factorized FNO, SSM, as well as (UNet-style) convolutional NOs with localized kernels have this property.

The precise definition of the full field of view property is then the following:

Definition (rigorous)**.** *An architecture is said to have a full field of view, if for any channel width H and $A_\ell \in \mathbb{R}^{H\times H}$ with product $A_L \cdots A_1 \neq 0$, we can find parameters such that equation 8 holds, and the iterated kernel*

$$\bar{\kappa}(x) = (\kappa_L * \cdots * \kappa_1)(x) \equiv (g_L * \cdots * g_1)(x)\, A_L \cdots A_1,$$

is nowhere-vanishing, i.e. for any $x, y \in D$, we have $\bar{\kappa}(x-y) \neq 0$.

B.2 A Sharp Universality Condition

We can now state and prove our universality result for general convolutional NOs.

Theorem B.1. *Let $\mathcal{G} : C(D;\mathbb{R}^{d_i}) \to C(D;\mathbb{R}^{d_o})$ be a continuous operator, with either $D \subset \mathbb{R}^d$ compact or $D = \mathbb{T}^d$ the periodic torus. Let $\mathcal{K} \subset C(D;\mathbb{R}^{d_i})$ be a compact set of input functions. Let $\Psi(u)$ be a convolutional NO architecture with a full field of view. Then for any $\epsilon > 0$, there exists a channel width H, and a setting of the weights of Ψ, such that*

$$\sup_{u\in\mathcal{K}} \sup_{x\in D} |\mathcal{G}(u)(x) - \Psi(u)(x)| \leq \epsilon.$$

Proof. For simplicity, we will assume that $d_i = d_o = 1$. The proof easily extends to the more general case. The space of continuous operators $\mathcal{G} : \mathcal{K} \to C(D)$, $u(x) \mapsto \mathcal{G}(u)(x)$, is isometrically isomorphic to the space $C(\mathcal{K} \times D)$, by identifying $\mathcal{G}(u, x) := \mathcal{G}(u)(x)$. By assumption, both $\mathcal{K}$ and D are compact sets.

Given the choice of convolutional NO architecture, we now fix $\kappa_1, \ldots, \kappa_L$ such that $\bar{\kappa}$ is nowhere vanishing. Let us now introduce the set $\mathbb{A}$ consisting of all operators that can be represented by a choice of channel width H and a choice of tunable parameters:

$$\mathbb{A} := \left\{ \begin{array}{c} \Psi = \mathcal{Q} \circ \mathcal{L}_L \circ \cdots \circ \mathcal{L}_1 \circ \mathcal{R},\ \Psi \text{ is a convolutional NO} \\ \text{with kernels } \kappa_1, \ldots, \kappa_L \text{ and channel width } H \end{array} \right\}.$$

Our goal is to show that $\mathbb{A} \subset C(\mathcal{K} \times D)$ is dense. We denote by $\overline{\mathbb{A}}$ the closure of $\mathbb{A}$ in $C(\mathcal{K} \times D)$ (set of limit points). To prove that $\mathbb{A} \subset C(\mathcal{K} \times D)$ is dense, we will use the following result, which follows from the Stone-Weierstrass theorem:

Lemma B.2. *A subset $\mathbb{A} \subset C(\mathcal{K} \times D)$ is dense, if*

- *The constant function $1 \in \mathbb{A}$,*
- *$\mathbb{A}$ separates points: For any $(u_1, x_1), (u_2, x_2) \in \mathcal{K} \times D$, there exists $\Psi \in \mathbb{A}$ such that $\Psi(u_1, x_1) \neq \Psi(u_2, x_2)$.*
- *$\mathbb{A}$ is an approximate vector subalgebra:*
  - *$\mathbb{A}$ is closed under addition and scalar multiplication (i.e. vector subspace),*
  - *$\mathbb{A}$ is approximately closed under multiplication: For any Ψ_1, Ψ_2, the product $\Psi_1 \cdot \Psi_2 \in \overline{\mathbb{A}}$, where*
  $$(\Psi_1 \cdot \Psi_2)(u, x) = \Psi_1(u, x)\Psi_2(u, x),$$
  is the pointwise multiplication.

Our goal is to show these properties for $\mathbb{A}$ to conclude that $\mathbb{A} \subset C(\mathcal{K} \times D)$ is dense.

$\mathbb{A}$ contains constants. It is very easy to show that the constant function $1 \in \mathbb{A}$, by defining a convolutional NO that disregards the input and has constant output $= 1$.

$\mathbb{A}$ is an approximate subalgebra. To show the other properties, we first note that $\mathbb{A}$ is closed under scalar multiplication and under addition, i.e.

$$\Psi_1, \Psi_2 \in \mathbb{A} \quad \Rightarrow \quad \lambda_1 \Psi_1 + \lambda_2 \Psi_2 \in \mathbb{A}, \quad \forall \lambda_1, \lambda_2 \in \mathbb{R}.$$

If H_1 and H_2 are the channel widths of Ψ_1 and Ψ_2, respectively, this conclusion follows by a simple parallelization of Ψ_1 and Ψ_2, and employing the last projection $\mathcal{Q}$-layer to sum (and scale) the parallelized results. Thus, $\mathbb{A}$ is a vector subspace of $C(\mathcal{K} \times D)$.

To show that $\mathbb{A}$ is approximately closed under multiplication, i.e. $\Psi_1, \Psi_2 \in \mathbb{A}$ implies that $\Psi_1 \cdot \Psi_2 \in \overline{\mathbb{A}}$, we fix arbitrary $\Psi_1, \Psi_2 \in \mathbb{A}$. We will denote by $\hat{\Psi}_j$ the NO Ψ_j for $j = 1, 2$, but where the last $\mathcal{Q}$ projection-layer has been removed:

$$\Psi_1 \equiv \mathcal{Q}_1 \circ \hat{\Psi}_1, \qquad \Psi_2 \equiv \mathcal{Q}_2 \circ \hat{\Psi}_2.$$

Assuming wlog that the channel width $H_1 = H_2 = H$ (otherwise, we can pad the channel width by zeros), we note that these incomplete NOs $\hat{\Psi}_1$ and $\hat{\Psi}_2$ define continuous mappings $\mathcal{K} \times D \to \mathbb{R}^H$. Since $\mathcal{K} \times D$ is compact, if follows that also the images $\hat{\Psi}_j(\mathcal{K} \times D) \subset \mathbb{R}^H$ are compact (since compact sets are mapped to compact sets under continuous maps). In particular, there exists $B > 0$, such that

$$\hat{\Psi}_1(u, x), \hat{\Psi}_2(u, x) \in [-B, B]^H, \quad \forall\, u \in \mathcal{K},\, x \in D.$$

Consider now $\mathcal{Q}_1$ and $\mathcal{Q}_2$, i.e. the last layers of Ψ_1 and Ψ_2, respectively. The following product mapping

$$[-B, B]^H \times [-B, B]^H \to \mathbb{R}, \quad (v_1, v_2) \to \mathcal{Q}_1(v_1) \cdot \mathcal{Q}_2(v_2),$$

is continuous. By the universality of shallow neural networks, for any $\epsilon > 0$, there thus exists a shallow net $\mathcal{Q} : \mathbb{R}^{2H} \to \mathbb{R}$, such that

$$\sup_{|v_1|_\infty, |v_2|_\infty \leq B} |\mathcal{Q}(v_1, v_2) - \mathcal{Q}_1(v_1) \cdot \mathcal{Q}_2(v_2)| \leq \epsilon.$$

Define now a new NO as $\Psi(u, x) := \mathcal{Q}([\hat{\Psi}_1(u, x), \hat{\Psi}_2(u, x)])$, i.e. $\mathcal{Q}$ applied to the parallelization of $\hat{\Psi}_1$ and $\hat{\Psi}_2$. Then

$$\begin{aligned}
&\sup_{\mathcal{K} \times D} |\Psi(u, x) - \Psi_1(u, x) \cdot \Psi_2(u, x)| \\
&\qquad = \sup_{\mathcal{K} \times D} |\mathcal{Q}([\hat{\Psi}_1(u, x), \hat{\Psi}_2(u, x)]) - \mathcal{Q}_1(\hat{\Psi}_1(u, x)) \cdot \mathcal{Q}_2(\hat{\Psi}_2(u, x))| \\
&\qquad \leq \sup_{|v_1|_\infty, |v_2|_\infty \leq B} |\mathcal{Q}([v_1, v_2]) - \mathcal{Q}_1(v_1) \cdot \mathcal{Q}_2(v_2)| \\
&\qquad \leq \epsilon.
\end{aligned}$$

Since $\epsilon > 0$ was arbitrary, this shows that $\Psi_1 \cdot \Psi_2$ is a limit point of $\Psi \in \mathbb{A}$, thus $\Psi_1 \cdot \Psi_2 \in \overline{\mathbb{A}}$. We conclude from the above that $\mathbb{A}$ is an approximate vector subalgebra.

$\mathbb{A}$ separates points. To conclude our proof, it only remains to show that $\mathbb{A}$ separates points. Let $(u_1, x_1) \neq (u_2, x_2)$ be two distinct elements in $\mathcal{K} \times D$. There are two cases: Either $x_1 \neq x_2$, or $x_1 = x_2$ and $u_1 \neq u_2$.

Case 1: $x_1 \neq x_2$. We want to construct $\Psi \in \mathbb{A}$, such that $\Psi(u_1, x_1) \neq \Psi(u_2, x_2)$. This is easy, since we can always use the lifting layer $\mathcal{R}$ to eliminate the dependency of Ψ on the u-variable. Once the weights acting on the u-variable have been set to zero, we then have that $\Psi(u, x) = \psi(x)$ is an ordinary multilayer perceptron. Since $x_1 \neq x_2$, we can easily choose ψ such that e.g. $\psi(x_1) = 0$ and $\psi(x_2) = 1$. Thus $\Psi(u_1, x_1) = \psi(x_1) \neq \psi(x_2) = \Psi(u_2, x_2)$.

Case 2: $x_1 = x_2 = x$ and $u_1 \neq u_2$. This is the more difficult case. In the following argument, we will appeal to the universality of shallow neural networks, and their compositions, multiple times. We will forego the tedious ϵ-δ estimates, and instead sketch out the general idea; filling in the details is a straight-forward (and boring...), exercise that would not provide any additional insight.

We construct Ψ as follows: First, given matrices A_ℓ and functions g_ℓ as in the (rigorous) definition of "full field of view", we choose the weights of the hidden layers $\mathcal{L}_\ell$, such that

$$\mathcal{L}_\ell(v)(x) = \sigma\left(A_\ell \int_D g_\ell(x-y)v(y)\,dy + b_\ell\right).$$

This is possible by assumption on the convolutional NO architecture, cp. equation 8. By universality of shallow neural networks, we can choose H sufficiently large, and choose matrices C_*, A_* and biases α_*, β_*, such that for all relevant input vectors $\xi = (\xi_1, 0, \dots, 0)$ (effectively one-dimensional), we have

$$C_*\sigma\left(A_*\xi + \beta_*\right) + \alpha_* \approx \xi, \tag{9}$$

i.e. the resulting shallow neural network is an approximation of the identity on one-dimensional ξ, to any desired accuracy.

We now momentarily focus on the case $L = 2$. In this case, we choose $A_1 = A_*$, $b_1 = b_*$, then choose $A_2 = A_*C_*$, and bias $b_2 = \alpha_* \int_D g(x-y)\,dy + \beta_*$, so that

$$\mathcal{L}_2 \circ \mathcal{L}_1(v)(x) = \sigma\left(A_2 \int_D g_2(x-y)\mathcal{L}_1(v)(y)\,dy + b_2\right),$$

where

$$A_2 \int_D g_2(x-y)\mathcal{L}_1(v)(y)\,dy + b_2 = A_* \int_D g_2(x-y)\left\{C_*\mathcal{L}_1(v)(y) + \alpha_*\right\}\,dy + b_*.$$

We now note that, by construction, we have for any hidden state function of the form $v(y) = [v_1(y), 0, \dots, 0]^T$:

$$C_*\mathcal{L}_1(v)(y) + \alpha_* = C_*\sigma\left(A_*(g_1 * v)(y) + \beta_*\right) + \alpha_* \approx \int_D g_1(y-z)v(z)\,dz,$$

where we have used equation 9 and note that this approximation is to any desired accuracy. It follows that also

$$\begin{aligned}\mathcal{L}_2 \circ \mathcal{L}_1(v)(x) &\approx \sigma\left(A_* \int_D g_2(x-y) \int_D g_1(y-z)v(z)\,dz\,dy + b_*\right)\\ &= \sigma\left(A_* \int_D (g_2 * g_1)(x-y)v(y)\,dy + b_*\right),\end{aligned}$$

to any desired accuracy. This shows that for two hidden layers $\mathcal{L}_1, \mathcal{L}_2$ there exists a choice of the parameters, such that for a hidden state $v(x) = (v_1(x), 0, \dots, 0)$, we can obtain an arbitrarily good approximation

$$\mathcal{L}_2 \circ \mathcal{L}_1(v)(x) \approx \sigma\left(A_* \int_D (g_2 * g_1)(x-y)v(y)\,dy + b_*\right).$$

Iterating this argument, for general $L \geq 2$, we start from $\mathcal{L}_L \circ \mathcal{L}_{L-1} \circ \ldots \mathcal{L}_1$, and we first choose the weights of $\mathcal{L}_L$ and $\mathcal{L}_{L-1}$ such that

$$\mathcal{L}_L \circ \mathcal{L}_{L-1}(v)(x) \approx \sigma\left(A_* \int_D (g_L * g_{L-1})(x-y)v(y)\,dy + b_*\right).$$

This can be done by the argument employed for the case $L = 2$. Next, we can apply the same argument again to choose the weights of $\mathcal{L}_{L-2}$, such that

$$\begin{aligned}\mathcal{L}_L \circ \mathcal{L}_{L-1} \circ \mathcal{L}_{L-2}(v)(x) &= (\mathcal{L}_L \circ \mathcal{L}_{L-1}) \circ \mathcal{L}_{L-2}(v)(x) \\ &\approx \sigma\left(A_* \int_D (g_L * g_{L-1} * g_{L-2})(x-y)v(y)\,dy + b_*\right),\end{aligned}$$

and continue similarly, until

$$\mathcal{L}_L \circ \cdots \circ \mathcal{L}_1(v)(x) \approx \sigma\left(A_* \int_D (g_L * \cdots * g_1)(x-y)v(y)\,dy + b_*\right),$$

to any desired accuracy. This argument is based on the assumption that $v(x) = (v_1(x), 0, \ldots, 0)$.

We next add a projection layer $\mathcal{Q}$ to this composition, of the form $\mathcal{Q}(v)(x) = Q(C_* v(x) + \alpha_*)$, where Q is a shallow neural network, to obtain

$$\mathcal{Q} \circ \mathcal{L}_L \circ \cdots \circ \mathcal{L}_1(v)(x) \approx Q\left(\int_D \bar{g}(x-y)v(y)\,dy\right), \quad \bar{g}(x) := (g_L * \cdots * g_1)(x).$$

Recall that $\bar{g}(x-y) \neq 0$ for all $x, y \in D$, by assumption (full field of view).

Pre-composing with a lifting layer $\mathcal{R}(u)(x) = R(u(x), x)$, which we choose to have only a non-vanishing first component on the output-side, i.e. $R(u(x), x) = (R_1(u(x), x), 0, \ldots, 0)$, it follows that we can construct $\Psi(u)(x) := \mathcal{Q} \circ \mathcal{L}_L \circ \cdots \circ \mathcal{L}_1 \circ \mathcal{R}$, such that

$$\Psi(u)(x) \approx Q\left(\int_D \bar{g}(x-y)R_1(u(y), y)\,dy\right),$$

where the approximation error can be made arbitrarily small. Here, we are still free to choose Q and R_1. Our goal is to choose them in such a way that for our given functions $u_1 \neq u_2$ and the given point $x \in D$, we have $\Psi(u_1)(x) \neq \Psi(u_2)(x)$. We will choose Q as an approximation of the identity on the first component, so that

$$\Psi(u)(x) \approx \int_D \bar{g}(x-y)R_1(u(y), y)\,dy.$$

By assumption, $\bar{g}(x-y) \neq 0$ is non-vanishing for all $y \in D$. Since $u_1 \neq u_2$, there exists $y_0 \in D$ such that $u_1(y_0) \neq u_2(y_0)$. We may wlog assume that $u_1(y_0) < \tau_1 < \tau_2 < u_2(y_0)$ for some $\tau_1, \tau_2 \in \mathbb{R}$. By continuity of u_1, u_2, there exists $\delta > 0$, such that $\max_{B_\delta(y_0)} u_1(y) < \tau_1 < \tau_2 < \min_{B_\delta(y_0)} u_2(y_0)$. By the universality of the shallow network R_1, we can choose R_1, such that we approximately have

$$R_1(\eta, y) \approx \rho_\delta(y - y_0)h_{\tau_1,\tau_2}(\eta),$$

where $\rho_\delta(\,\cdot\, - y_0)$ is a non-negative function supported inside $B_\delta(y_0)$, and $h_{\tau_1,\tau_2}(\,\cdot\,)$ is a non-negative function, such that

$$h_{\tau_1,\tau_2}(\eta) = \begin{cases} 1, & (\eta > \tau_2), \\ 0, & (\eta < \tau_1). \end{cases}$$

Since $\bar{g}(x-y)$ is non-vanishing for all $x, y \in D$, we can further refine our choice of ρ_δ, to ensure that

$$\int_D \bar{g}(x-y)\rho_\delta(y-y_0)\,dy \neq 0.$$

It then follows that

$$R_1(u_1(y), y) \approx \rho_\delta(y - y_0) h_{\tau_1,\tau_2}(u_1(y)) \equiv 0,$$

and

$$R_1(u_2(y), y) \approx \rho_\delta(y - y_0) h_{\tau_1,\tau_2}(u_2(y)) \equiv \rho_\delta(y - y_0).$$

In particular, we conclude that – to any desired accuracy – we can construct $\Psi(u)$, such that

$$\Psi(u_1)(x) \approx \int_D \bar{g}(x - y) R_1(u_1(y), y)\, dy \approx 0$$

and

$$\Psi(u_2)(x) \approx \int_D \bar{g}(x - y) R_1(u_2(y), y)\, dy \approx \int_D \bar{g}(x - y) \rho_\delta(y - y_0)\, dy \neq 0.$$

In particular, upon making the approximation errors sufficiently small, it follows that there exists $\Psi \in \mathbb{A}$ such that $\Psi(u_1)(x) \neq \Psi(u_2)(x)$. This finally shows that $\mathbb{A}$ separates points.

Our proof of the universality of $\mathbb{A} \subset C(\mathcal{K} \times D)$ now concludes by application of the Stone-Weierstrass theorem (cp. Lemma B.2).

□

C Further Ablation Studies

C.1 Unidirectional Spatial SSM

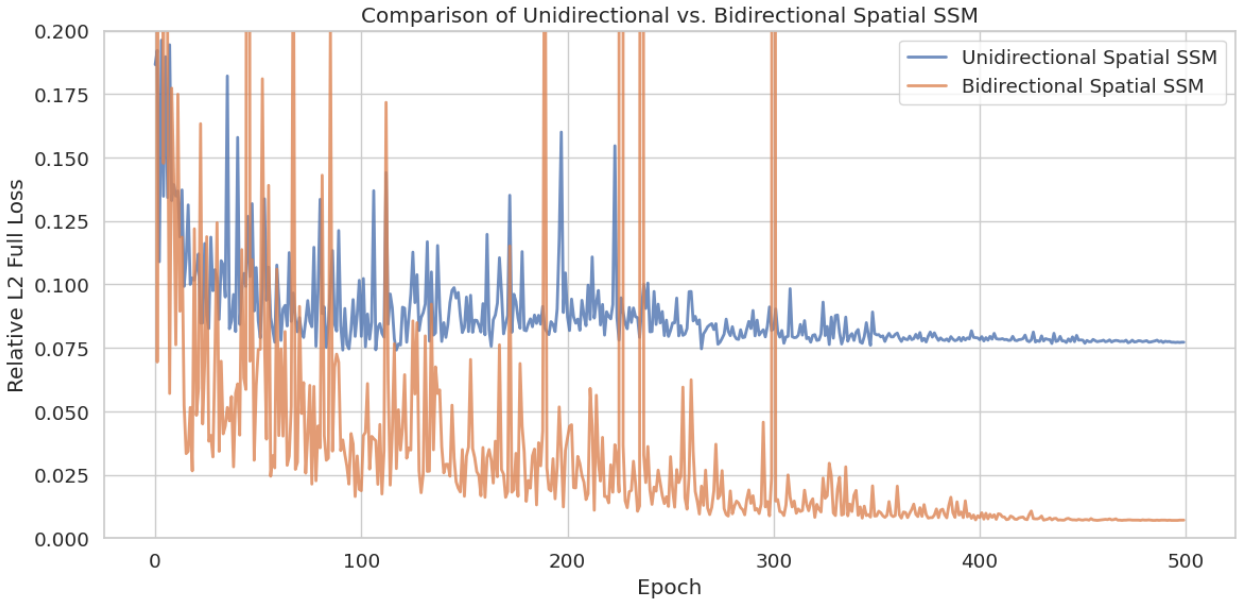

Figure 6: Training curves comparing the unidirectional and bidirectional spatial SSMs on the `TorusLi` dataset. The unidirectional variant plateaus early, failing to capture global spatial dependencies.

We conduct an ablation study to evaluate the importance of the bidirectional spatial module in our SS-NO architecture on 1D Burgers' Equation. To ensure a fair comparison that isolates the effect of directionality, we construct a unidirectional model with the same parameter count as the full SS-NO. This is done by stacking 8 layers of forward-only spatial SSM blocks and inserting a single temporal SSM layer in the middle.

As shown in Figure 6, the unidirectional variant fails to capture the dynamics effectively. Its training curve plateaus early, and the relative $\mathcal{L}^2$ error stagnates around 0.075. In contrast, the full bidirectional model continues to improve and ultimately reaches a much lower error of approximately 0.007—more than ten

times better. These results highlight that unidirectional spatial SSMs are fundamentally limited in their representational capacity for 2D PDEs and act as non-universal approximators in this setting – indeed, this unidirectional spatial SSM *is lacking* a full field of view, violating the criterion for universality in Theorem 4.1. Bidirectional context is essential to capture the long-range spatial dependencies needed for accurate forecasting.

C.2 Effect of Missing Contextual Information and the Role of Temporal Modeling

To better understand the role of temporal modeling in our architecture, we evaluate the full SS-NO against an ablated variant that removes the temporal state-space module (S4), as well as two strong baselines (FFNO and FNO2D). Figure 7 shows the relative $\mathcal{L}^2$ error across three levels of contextual information: *all context*, *only forcing*, and *no context*, on both the `TorusVis` and `TorusVisForce` datasets.

Three main observations emerge from these results. First, the temporal S4 component is necessary in the most challenging *No Context* setting, where both viscosity and forcing inputs are missing. Without temporal modeling, SS-NO suffers from very high errors (e.g., 0.1156 vs. 0.0411 on `TorusVis`), whereas the full model remains comparatively robust by leveraging temporal dependencies to compensate for missing information. These results extend the observations made by Buitrago et al. (2025), who showed that temporal memory helps mitigate the effects of resolution loss and noisy data. In our case, we show that memory-based modeling also helps compensate for missing physical parameters—highlighting an additional benefit of incorporating temporal memory into spatiotemporal PDE models.

Second, in the *Only Force* setting, the presence of forcing information appears to be sufficient for accurate predictions in our architecture: both the full and ablated SS-NO achieve consistently low errors, while FFNO exhibits a slight degradation. This suggests that temporal modeling is not strictly necessary when forcing is available, as even the ablated variant performs competitively.

Finally, in the extreme case where all context is missing, FFNO surprisingly outperforms SS-NO (full), despite performing worse in other settings. We hypothesize this could be the result of the difference between how FFNO and FNO/SS-NO make residual connections, since it seems like the degree of performance loss is similar between SS-NO (full) and FNO2D, but we do not do further investigation as it is out of the scope of this work.

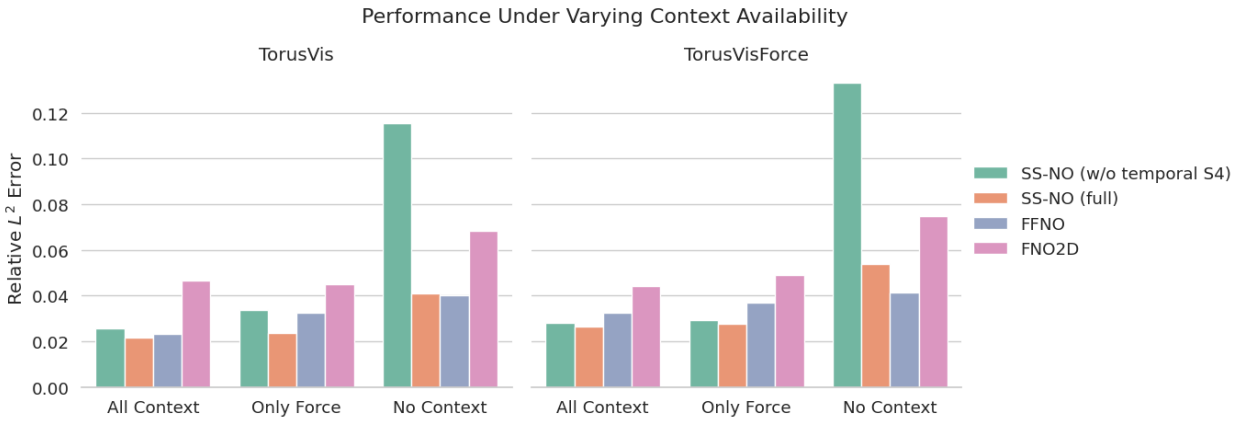

Figure 7: Relative $\mathcal{L}^2$ error across three levels of contextual information: full context, only forcing, and no context. Lower is better.

C.3 Effect of Varying the Memory Window Size

We investigate how the choice of memory window size K affects the performance of our SS-NO model on the KS dataset with $\nu = 0.075$. This hyperparameter controls how many past spatial feature maps are accessible

to the model during temporal state updates. All results reported in this section are obtained from models trained *without teacher forcing* to better reflect deployment conditions.

As shown in Figure 8, increasing K consistently improves performance up to around $K = 8$. Beyond this point, the performance gains become increasingly marginal, and the validation loss begins to plateau. Notably, setting $K = 0$, which corresponds to no temporal memory, results in significantly degraded performance. These results highlight the critical importance of temporal memory in modeling complex spatiotemporal dynamics, while also suggesting diminishing returns beyond a moderate window size.

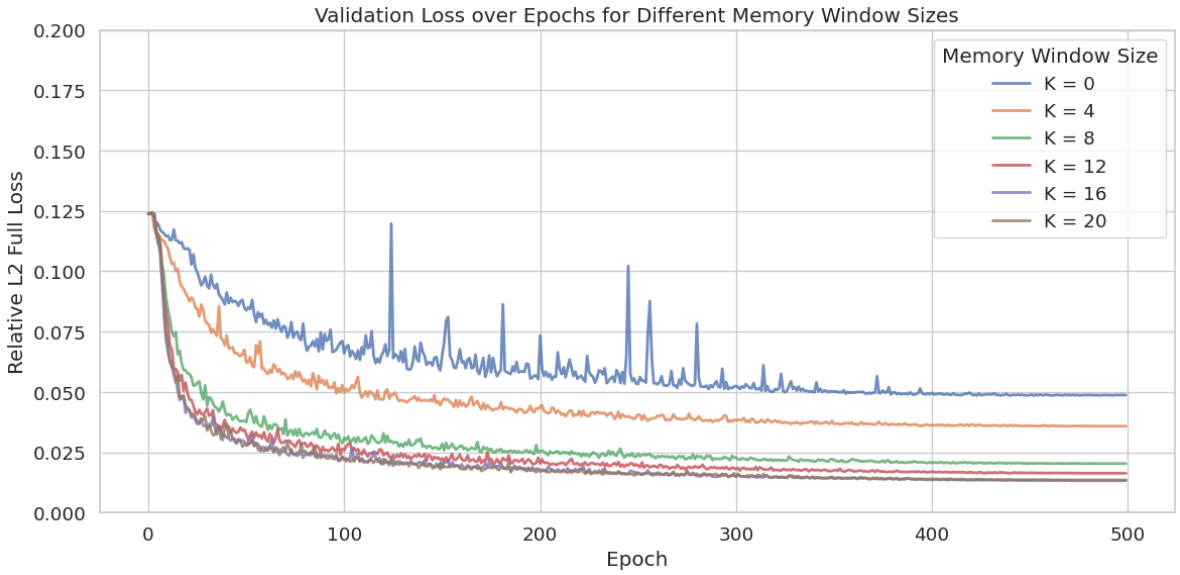

Figure 8: Validation $\mathcal{L}^2$ loss over training epochs for different memory window sizes K on the KS dataset ($\nu = 0.075$). Increasing the window improves performance until around $K = 8$, after which gains diminish.

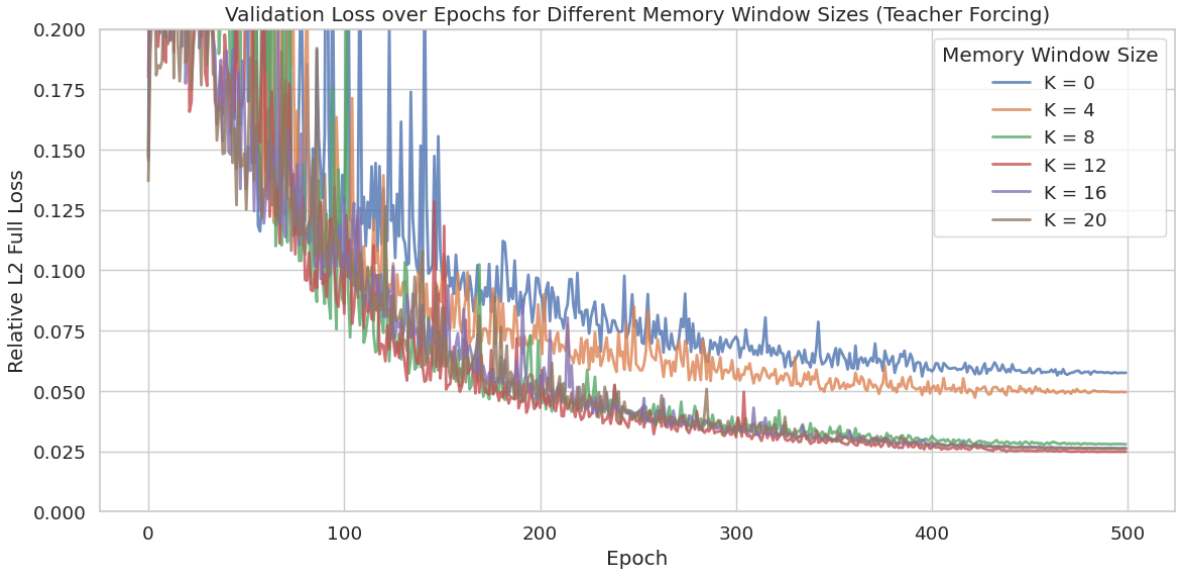

Figure 9: Validation $\mathcal{L}^2$ loss over training epochs on the KS dataset ($\nu = 0.075$) using teacher forcing for various memory window sizes K. Compared to Figure 8, training is less stable and results in higher loss.

C.4 Effect of Teacher Forcing on SS-NO Performance

Case Study: 1D KS with $\nu = 0.075$ at Low Resolution. While prior work such as FFNO (Tran et al., 2023) suggests that teacher forcing can improve model performance by stabilizing training, we observe a different trend for our SS-NO architecture, particularly in low-resolution regimes. To investigate this, we revisit the 1D Kuramoto–Sivashinsky (KS) dataset with $\nu = 0.075$, where reduced spatial resolution poses a significant challenge.

We train SS-NO with teacher forcing across various memory window sizes ($K = 0$ to 20), and present the results in Figure 9. Compared to the non-teacher-forced variant (Figure 8), training with teacher forcing leads to noticeably less stable optimization, slower convergence, and consistently higher final relative $\mathcal{L}^2$ errors. Moreover, while performance still improves with longer memory, the gains plateau earlier, and the benefits beyond $K = 8$ diminish more rapidly than in the non-teacher-forced case.

These findings suggest that in this setting, SS-NO benefits more from fully autoregressive training, likely due to its strong inductive bias for modeling long-term temporal dependencies without relying on ground-truth guidance.

General Trends Across 1D and 2D Benchmarks. To better understand the broader impact of teacher forcing, we compare SS-NO models trained with and without teacher forcing across all 1D and 2D benchmarks (Tables 4 and 5).

In 1D settings, results are mixed: at higher resolutions (e.g., 128), teacher forcing slightly improves performance in some cases (e.g., KS with $\nu = 0.1$ and $\nu = 0.125$), but at lower resolutions, non-teacher-forced models generally perform better—especially in the challenging $\nu = 0.075$ KS setup. These results suggest that autoregressive training may confer more robustness in low-resolution or harder regimes, where model predictions diverge more easily from ground truth.

In contrast, across all 2D Navier–Stokes benchmarks, teacher forcing consistently yields better performance. This suggests that for higher-dimensional systems with complex spatiotemporal interactions, teacher forcing can help stabilize training and guide the model toward more accurate trajectories.

Overall, while the effect of teacher forcing appears to be dataset- and resolution-dependent, we find that non-teacher-forced training can be highly effective in 1D regimes, particularly under low resolution. In contrast, for complex 2D flows, teacher forcing remains a valuable tool for improving predictive accuracy.

Table 4: Relative $\mathcal{L}^2$ error on 1D benchmarks at varying resolutions.

Resolution	**Architecture**	**Relative $\mathcal{L}^2$ Error**			
		KS			Burgers'
		$\nu = 0.075$	$\nu = 0.1$	$\nu = 0.125$	$\nu = 0.001$
128	SS-NO (Teacher Forcing)	0.0086	**0.0027**	**0.0013**	**0.0070**
	SS-NO (No Teacher Forcing)	**0.0076**	0.0034	0.0019	**0.0070**
64	SS-NO (Teacher Forcing)	0.0143	0.0048	0.0029	0.0121
	SS-NO (No Teacher Forcing)	**0.0135**	**0.0042**	**0.0023**	**0.0114**
32	SS-NO (Teacher Forcing)	0.0472	0.0162	0.0088	0.0200
	SS-NO (No Teacher Forcing)	**0.0358**	**0.0135**	**0.0071**	**0.0171**

D Comparison of Factorized Architectural Variants

In this section, we explore how different architectural factorization strategies affect performance on the `TorusLi` dataset. Specifically, we examine two popular alternatives to our default SS-NO spatial block.

Table 5: Relative $\mathcal{L}^2$ error on 2D Navier–Stokes datasets. All models are evaluated at a spatial resolution of 64×64.

Architecture	**Relative $\mathcal{L}^2$ Error**		
	TorusLi	TorusVis	TorusVisForce
SS-NO (Teacher Forcing)	**0.0345**	**0.0218**	**0.0263**
SS-NO (No Teacher Forcing)	0.0546	0.0385	0.0371

```
# SS-NO-VM (Vision Mamba Style)
x_flat = flatten_spatial(x)

# First zigzag sweep
x_fwd = SSM_flat_forward(x_flat)
x_bwd = SSM_flat_backward(reverse(x_flat))
z1 = permute_channels(x_fwd + x_bwd)
x = linearT(z1) + x_flat

# Second zigzag sweep
x_fwd2 = SSM_flat_forward(x)
x_bwd2 = SSM_flat_backward(reverse(x))
z2 = permute_channels(x_fwd2 + x_bwd2)
x = linearT2(z2) + x

x_out = reshape_spatial(x)
```

Listing 1: SS-NO-VM (Vision Mamba style)

```
# SS-NO-FF (FFNO-style)
residual = x

# X-axis sweep
x1 = reshape_for_x_sweep(x)
x1_fwd = SSM_x_forward(x1)
x1_bwd = reverse(SSM_x_backward(reverse(x1)))
x1_combined = reshape_back(x1_fwd + x1_bwd)

# Y-axis sweep
y1 = reshape_for_y_sweep(x)
y1_fwd = SSM_y_forward(y1)
y1_bwd = reverse(SSM_y_backward(reverse(y1)))
y1_combined = reshape_back(y1_fwd + y1_bwd)

# Combine and project
z = x1_combined + y1_combined
z = backcast_ff(z)
output = z + residual
```

Listing 2: SS-NO-FF (FFNO-style connection)

Figure 10: Pseudocode for two spatial architectural variants of SS-NO: Vision Mamba-style (left) and FFNO-style (right).

SS-NO-VM (Vision Mamba style). This variant adapts the Vision Mamba architecture by replacing the Mamba core with our spatial SSM block. The key idea is to preserve Vision Mamba's flattened zigzag scanning structure in 2D, allowing bidirectional processing over a 1D sequence obtained from flattening the spatial grid. After bidirectional processing over the first spatial axis, the result is passed through a mixing linear projection before a second pass along the alternate axis, again using bidirectional spatial SSMs and linear mixing. This model maintains Vision Mamba's emphasis on alternating directional fusion while leveraging the structure of our SSM.

SS-NO-FF (FFNO style). In this variant, we discard the original spatial block of SS-NO and instead adopt the FFNO-style connection, where spatial context is aggregated through two 1D sweeps: one across the x-axis and one across the y-axis. Each sweep involves a forward and backward spatial SSM applied independently along that axis, and the outputs from both axes are summed together. This approach highlights the role of directional decoupling in factorized Fourier models and contrasts with the fused zigzag sweep in SS-NO-VM.

Pseudocode for both SS-NO-FF and SS-NO-VM variants is provided in Figure 10.

FNO2D-R (Reduced Fourier Kernel). To verify that the gains of SS-NO are not trivially due to kernel simplification, we construct a reduced FNO2D model, denoted as FNO2D-R. In this variant, we remove the output channel mixing in the Fourier kernel by modifying the kernel from:

```
self.weights = nn.Parameter(self.scale * torch.rand(
    in_channels, out_channels, modes1, modes2, dtype=torch.cfloat))
```

```
...
return torch.einsum("bixy,ioxy->boxy", input, weights)
```

to:

```
self.weights = nn.Parameter(self.scale * torch.rand(
    in_channels, modes1, modes2, dtype=torch.cfloat))
...
return input * weights
```

This isolates the contribution of full channel-wise Fourier mixing and allows for a more controlled comparison against SS-NO.

As shown in Table 6, our original SS-NO achieves the best performance among all tested variants, demonstrating that both our architectural choices and structured state-space modeling contribute meaningfully to the observed improvements.

Table 6: Relative $\mathcal{L}^2$ error on the `TorusLi` dataset for different factorized architectural variants. All models are evaluated at a spatial resolution of 64×64.

Architecture	**# Parameters**	**Relative $\mathcal{L}^2$ Error**
		`TorusLi`
SS-NO-VM	402,945	0.0495
SS-NO-FF	503,297	0.0403
FNO2D-R	$1,115,841$	0.0718
SS-NO (ours)	369,665	**0.0345**

E Data Preprocessing and Training Details

E.1 Data Preprocessing and Training Setup

We normalize all input data to the range $[0, 1]$ and add fixed-variance Gaussian noise during training. Through empirical validation, we found $\sigma = 0.005$ optimal for Burgers' equation to stabilize training while preserving signal integrity, while $\sigma = 0.001$ works best for all other datasets. All models are trained for 500 epochs using the AdamW optimizer with an initial learning rate of 10^{-3}, weight decay of 10^{-4}, and a cosine annealing learning rate schedule. We employ teacher forcing during training unless otherwise noted.

To simulate low-resolution scenarios, we generate downsampled versions of the original datasets by uniformly subsampling in space. All models are trained to minimize the normalized step-wise relative ℓ_2 loss, and evaluation is reported using the full-sequence relative ℓ_2 error.

E.2 Model Architecture Specifications

All models use four blocks with a consistent hidden dimension of 64 across both 1D and 2D problems. Intermediate and output linear layers maintain a hidden size of 128. With the exception of FFNO and our SS-NO on 1D problems, we adopt the hyperparameter settings reported by Buitrago et al. (2025). Unlike their setup which uses hidden dimension 128 for 1D and 64 for 2D problems, we found no statistical difference in performance with a unified hidden dimension of 64, simplifying architecture design while maintaining competitive results.

All models except GKT follow the MemNO framework and incorporate a temporal S4 module with window size $K = 4$ positioned in the middle of the network stack. For GKT, we use a multi-input variant where the temporal dimension is mixed with features.

We adopt a simple spatial positional encoding scheme shared across all models. In 1D, for a grid with f equispaced points over the interval $[0, L]$, the positional encoding $E \in \mathbb{R}^f$ is defined as $E_i = \frac{i}{L}$ for $0 \leq i < f$.

In 2D, for a $f \times f$ grid over $[0, L_x] \times [0, L_y]$, the encoding is $E_{ij} = (\frac{i}{L_x}, \frac{j}{L_y})$. The input lifting operator r_{in} is a shared linear layer that maps the concatenation of input features and positional encoding to the hidden space, applied pointwise across the spatial grid.

E.3 Baseline Model Architectures

Unless otherwise specified, all models use four blocks with a channel width of 64. Intermediate and output linear layers have a hidden size of 128 (i.e., twice the channel width). **With the exception of FFNO, multi-input FNO, and our SS-NO on 1D problems, we adopt the "optimal" hyperparameter settings (i.e., number of layers, hidden dimension, and expansion size of linear layers) reported by Buitrago et al. (2025).** The U-Net we use is even larger, containing more parameters than this standard baseline. For memory-augmented architectures, the memory module is inserted after the first two blocks and before the last two. For all non-Markovian models, we fix the memory window size to $K = 4$ across experiments; the impact of varying K on SS-NO is explored in Appendix C.3.

We adopt a simple spatial positional encoding scheme shared across all models. In 1D, for a grid with f equispaced points over the interval $[0, L]$, the positional encoding $E \in \mathbb{R}^f$ is defined as $E_i = \frac{i}{L}$ for $0 \leq i < f$. In 2D, for a $f \times f$ grid over $[0, L_x] \times [0, L_y]$, the encoding is $E_{ij} = (\frac{i}{L_x}, \frac{j}{L_y})$. The input lifting operator r_{in} is a shared linear layer that maps the concatenation of input features and positional encoding from $\mathbb{R}^{2+k}$, where k is the number of features, to the hidden space $\mathbb{R}^h$, applied pointwise across the spatial grid. Similarly, a shared decoder r_{out} maps from $\mathbb{R}^h$ back to $\mathbb{R}$. For all models containing an FNO or FFNO module, we follow the setup in Buitrago et al. (2025) and retain all available Fourier modes by setting $k_{\max} = \lfloor \frac{f}{2} \rfloor$, where f denotes the spatial resolution.

Factorized Fourier Neural Operator (FFNO). The Factorized Fourier Neural Operator (FFNO), introduced by Tran et al. (2023), extends the original Fourier Neural Operator (FNO) (Li et al., 2021) by modifying how the spectral integral kernel is applied. We use the standard FFNO implementation from `https://github.com/alasdairtran/fourierflow/`. Each FFNO layer operates on a spatial grid of size $|S|$ and a hidden dimension h, and is defined as a residual block:

$$\ell(v) = v + \text{Linear}_{h \to h'} \circ \sigma \circ \text{Linear}_{h' \to h} \circ \mathcal{K}(v),$$

where σ denotes the ReLU activation function (Nair & Hinton, 2010), and h' is an intermediate dimensionality used within the nonlinear mapping. The integral operator $\mathcal{K}$ transforms v in the Fourier domain and is computed by aggregating across spatial dimensions:

$$\mathcal{K}(v) = \sum_{\alpha=1}^{d} \text{IFFT}_\alpha \left[R_\alpha \cdot \text{FFT}_\alpha(v) \right],$$

where FFT_α and IFFT_α are the (inverse) discrete Fourier transforms applied along the α-th spatial axis, and $R_\alpha \in \mathbb{C}^{h^2 \times k_{\max}}$ are learned complex-valued projection matrices for each dimension.

Fourier Neural Operator (FNO). For 2D problems, we use a modified version of the original Fourier Neural Operator (FNO) (Li et al., 2021) where the output of the integral kernel is passed through a nonlinearity before the residual skip connection, following the implementation at `https://github.com/lilux618/fourier_neural_operator/blob/master/fourier_2d_time.py`, which we found to perform better.

Galerkin Transformer (GKT). We use the Galerkin Transformer (GKT) implementation from `https://github.com/scaomath/galerkin-transformer`. We employ a multi-input variant where the model is made non-Markovian by providing the last $K = 4$ steps as input and processing the temporal dimension as normal channels concatenated with other features, rather than as an independent dimension. We use a hidden size of 32 with dropout rates of 0.05 in attention layers and 0.025 in feedforward layers.

Factformer 2D. As part of our evaluation, we include the original Factformer model from Li et al. (2023), which is designed for spatiotemporal modeling over 2D spatial domains. While Buitrago et al.

(2025) only evaluated the 1D variant, we adopt the same architectural configuration for the 2D case to ensure fair comparison: four attention layers, a hidden dimension of 64, and 4 attention heads with a total projection dimension of 512. Factformer 2D applies linear attention sequentially along each spatial axis. Given a hidden tensor $w \in \mathbb{R}^{S_x \times S_y \times H}$, two separate MLPs ($\text{MLP}_x$ and MLP_y) are applied to compute keys and queries across S_x and S_y, respectively. After averaging over the complementary axis, we obtain $q_x, k_x \in \mathbb{R}^{S_x \times H}$ and $q_y, k_y \in \mathbb{R}^{S_y \times H}$. Values $v \in \mathbb{R}^{S_x \times S_y \times H}$ are computed through a shared linear projection, and attention is applied first along x and then y. Our implementation follows the original GitHub repository `https://github.com/BaratiLab/FactFormer` with minimal changes limited to data format compatibility.

Factformer 1D. We also evaluate the 1D version of Factformer, as introduced in Buitrago et al. (2025), using the same configuration of four attention layers, hidden dimension 64, and 4 attention heads (total projection dimension 512). Unlike the 2D case, the 1D variant processes inputs over a single spatial dimension. As such, only one MLP (MLP_x) is used to compute queries and keys, and no spatial averaging is applied. A single linear attention operation is performed per layer along the 1D spatial axis. Values are projected from the input using a linear layer. This model is implemented within the same codebase as the 2D version to maintain consistency, with minor modifications to handle 1D inputs.

U-Net Neural Operator (U-Net). Our U-Net implementation follows the typical encoder-decoder architecture with skip connections (Gupta & Brandstetter, 2023). It consists of four downsampling convolutional blocks, a bottleneck convolutional block, and four upsampling blocks with residual connections linking corresponding encoder and decoder layers. We use a first hidden dimension of 32, and apply channel multipliers of $[1, 2, 4, 8]$ in the encoder path. No time embeddings are used. This setup is adapted to process spatiotemporal data by flattening the spatial dimensions and independently applying the network to each time step. Our implementation is based on the publicly available codebase from PDEBench `https://github.com/pdebench/PDEBench` (Takamoto et al., 2022). For reference, the U-Net Neural Operator variant used in prior work by Buitrago et al. (2025) employs a similar architecture but with channel multipliers of $[1, 2, 2, 2]$.

F Data Generation

F.1 1D Burgers' Equation

We consider the one-dimensional viscous Burgers' equation, expressed as:

$$\partial_t u + u\,\partial_x u = \nu\,\partial_{xx} u, \quad (t, x) \in [0, T] \times [0, L],$$

where ν is the viscosity coefficient. For our experiments, we utilize the publicly available 1D Burgers' dataset from PDEBench (Takamoto et al., 2022), with viscosity $\nu = 0.001$. The original dataset contains 10,000 spatiotemporal samples, each defined on a spatial grid of $1,024$ points and 201 time steps over the interval $[0, 2.01]$ seconds. We restrict our usage to the first 140 time steps and uniformly downsample them to obtain 20 steps spanning up to 1.4 seconds. This truncation is motivated by the observation that, after this point, the dissipative effect of the diffusion term $\partial_{xx} u$ suppresses high-frequency dynamics, causing the solution to evolve slowly (Buitrago et al., 2025). For training, we use the first 2,048 samples from the dataset and reserve the last 1,000 samples for testing.

F.2 1D Kuramoto–Sivashinsky Equation

The Kuramoto–Sivashinsky (KS) equation is given by:

$$\partial_t u + u\,\partial_x u + \partial_{xx} u + \nu\,\partial_{xxxx} u = 0, \quad (t, x) \in [0, T] \times [0, L],$$

with periodic boundary conditions. We follow the exact same setting as provided by Buitrago et al. (2025), using their public repository at `https://github.com/r-buitrago/LPSDA`, which builds upon the implementation of Brandstetter et al. (2022). The spatial domain $[0, 64]$ is discretized into 512 points, and the temporal domain $[0, 2.5]$ is divided into 26 equispaced time steps with fixed $\Delta t = 0.1$.

Initial conditions are generated as random superpositions of sine waves:

$$u_0(x) = \sum_{i=0}^{20} A_i \sin\left(\frac{2\pi k_i}{L} x + \phi_i\right),$$

where for each trajectory, the amplitudes A_i are sampled from a continuous uniform distribution on $[-0.5, 0.5]$, the frequencies k_i are drawn from a discrete uniform distribution over $\{1, 2, \ldots, 8\}$, and the phases ϕ_i are sampled uniformly from $[0, 2\pi]$.

We consider three different viscosities: $\nu = 0.075$, 0.1, and 0.125. For each viscosity, we generate $2,048$ training samples and 256 validation samples.

F.3 2D Navier Stokes Equations

The `TorusLi` Dataset. We consider the two-dimensional incompressible Navier–Stokes equations on the unit torus $\mathbb{T}^2 = [0, 1]^2$ in vorticity form. The evolution of the scalar vorticity field $\omega(x, y, t)$ is governed by:

$$\partial_t \omega + \mathbf{u} \cdot \nabla \omega = \nu \Delta \omega + f,$$

where $\mathbf{u} = (u, v)$ is the velocity field satisfying the incompressibility condition $\nabla \cdot \mathbf{u} = 0$, $\nu > 0$ is the kinematic viscosity, and f is a fixed external forcing function.

We directly reuse the dataset released by Li et al. (2021), referred to as `TorusLi`, which was originally developed to benchmark the FNO. The simulations were generated using a pseudospectral Crank–Nicolson second-order time-stepping scheme on a high-resolution 256×256 grid, and subsequently downsampled to 64×64. All trajectories use a constant viscosity of $\nu = 10^{-5}$ (corresponding to a Reynolds number $\mathrm{Re} = 2000$), and share the same external forcing:

$$f(x, y) = 0.1 \left[\sin(2\pi(x + y)) + \cos(2\pi(x + y))\right].$$

Initial vorticity fields ω_0 are sampled from a Gaussian random field:

$$\omega_0 \sim \mathcal{N}\left(0, 7^{3/2}(-\Delta + 49I)^{-2.5}\right),$$

with periodic boundary conditions. The numerical solver computes the velocity field by solving a Poisson equation in Fourier space and evaluates nonlinear terms in physical space with dealiasing applied. The nonlinear term is handled explicitly in the Crank–Nicolson scheme.

The dataset consists of solutions recorded every $t = 1$ time unit, with a total of 20 time steps per trajectory, corresponding to a final time horizon of $T = 20$. This makes the task relatively long-range compared to other PDE benchmarks. The spatial resolution is fixed at 64×64 for all experiments in this paper.

The `TorusVis` and `TorusVisForce` Datasets. We utilize two additional datasets, `TorusVis` and `TorusVisForce`, introduced by Tran et al. (2023), to evaluate model generalization under varying physical regimes. Both datasets are generated using the same Crank–Nicolson pseudospectral solver used in `TorusLi`, maintaining consistency in numerical methodology.

These datasets extend the Navier–Stokes setting by incorporating variability in viscosity and external forcing. Specifically, each trajectory uses a randomly sampled viscosity ν between 10^{-5} and 10^{-4}. The external forcing function is defined as:

$$f(t, x, y) = 0.1 \sum_{p=1}^{2} \sum_{i=0}^{1} \sum_{j=0}^{1} \left[\alpha_{pij} \sin\left(2\pi p(ix + jy) + \delta t\right) + \beta_{pij} \cos\left(2\pi p(ix + jy) + \delta t\right)\right],$$

where $\alpha_{pij}, \beta_{pij} \sim \mathcal{U}[0, 1]$ are sampled independently for each trajectory. The parameter δ controls the temporal variation in the forcing: for `TorusVis`, $\delta = 0$, resulting in time-invariant forcing; for `TorusVisForce`, $\delta = 0.2$, producing a time-varying force.

As with `TorusLi`, the spatial resolution is fixed at 64×64, and trajectories consist of 20 time steps sampled every $t = 1$ time unit, yielding a total time horizon of $T = 20$.

F.4 2D Richtmeyer–Meshkov (CE-RM) Problem

We consider the 2D **Richtmeyer–Meshkov (CE-RM)** benchmark for the compressible Euler equations ($\gamma = 1.4$) on the unit square $[0, 1]^2$ with periodic boundary conditions. The initial state contains a high-pressure circular region and a heavy fluid on one side of a perturbed interface (Herde et al., 2024). Specifically, the initial pressure and density are given by

$$p_0(x, y) = \begin{cases} 20, & \sqrt{x^2 + y^2} < 0.1, \\ 1, & \text{otherwise}, \end{cases} \tag{10}$$

$$\rho_0(x, y) = \begin{cases} 2, & |x| < I(x, y, \omega), \\ 1, & \text{otherwise}, \end{cases} \tag{11}$$

with initial velocities $v_x = v_y = 0$. The interface $I(x, y, \omega)$ is perturbed by a random Fourier series:

$$I(x, y, \omega) = 0.25 + \epsilon \sum_{j=1}^{10} a_j(\omega) \sin\big(2\pi((x, y) \cdot (1, 0) + b_j(\omega))\big), \tag{12}$$

where a_j and b_j are independent uniform random amplitudes and phases (with $\sum_j a_j = 1$). We integrate the Euler equations up to time $T = 2$, saving 21 equally spaced snapshots in time.

The publicly released **CE-RM dataset** (Herde et al., 2024) contains 1260 trajectories on a 128×128 grid, which we spatially downsample by a factor of two to 64×64. Each snapshot includes five fields (density ρ, velocity v_x, v_y, pressure p, and a passive tracer). In our setup, the passive tracer is assumed available as an input at every time step, while the model predicts only the conserved variables $[\rho, v_x, v_y, p]$. We follow the dataset split of Herde et al. (2024), using the first 1000 trajectories for training and the last 200 for validation. We further apply temporal downsampling by a factor of 2: we take the solution at times indexed $t = 0, 2, 4, 6$ (the first four snapshots) as input and predict the next 7 steps at indices $8, 10, \ldots, 20$. In other words, given the fields at time steps t_0, t_2, t_4, t_6, the model predicts the fields at $t_8, \ldots, t_{20}$.

F.5 2D Gravitational Rayleigh–Taylor (GCE-RT) Problem

The **GCE-RT** problem adds gravitational forcing to the compressible Euler equations on $[0, 1]^2$ with periodic boundaries. We use the two-dimensional Rayleigh–Taylor setup from astrophysics: a $\gamma = 2$ polytropic equilibrium on a model neutron star is perturbed at a random interface (Herde et al., 2024). In cylindrical symmetry ($r = \sqrt{x^2 + y^2}$), the unperturbed pressure and gravitational potential are

$$p(r) = K_0 \left(\frac{\rho_0 \sin(\alpha r)}{\alpha r} \right)^2, \tag{13}$$

$$\phi(r) = -2K_0 \frac{\rho_0 \sin(\alpha r)}{\alpha r}, \tag{14}$$

with $K_0 = p_0/\rho_0^2$ and $\alpha = \sqrt{4\pi G/(2K_0)}$ (with $G = 1$). The initial density profile is

$$\rho(r) = \sqrt{\frac{K_0}{\tilde{K}(r)}} \frac{\rho_0 \sin(\alpha r)}{\alpha r}, \tag{15}$$

where $\tilde{K}(r) = K_0$ for $r < r_{RT}$ and $\tilde{K}(r) = (1 - A/(1 + A))^2 K_0$ for $r \geq r_{RT}$. The Rayleigh–Taylor interface radius is given by

$$r_{RT}(x, y) = 0.25\,(1 + a\cos(\text{atan2}(y, x) + b))\,, \tag{16}$$

with random amplitude $a \in [-1, 1]$ and phase $b \in [-\pi, \pi]$. We also perturb the central density ρ_0, pressure p_0, and Atwood number A via

$$\rho_0 = 1 + 0.2c, \tag{17}$$

$$p_0 = 1 + 0.2d, \tag{18}$$

$$A = 0.05(1 + 0.2e), \tag{19}$$

with $c, d, e \sim \mathcal{U}[-1, 1]$. Initial velocity is set to zero. We evolve this setup to $T = 5$ and save 11 snapshots (every $\Delta t = 0.5$).

The **GCE-RT dataset** (Herde et al., 2024) likewise contains 1260 trajectories on a 128×128 grid, which we spatially downsample to 64×64. Each snapshot includes six fields (ρ, v_x, v_y, p, a tracer, and the gravitational potential ϕ). In our setup, the passive tracer and gravitational potential are assumed available as inputs at every time step, while the model predicts only $[\rho, v_x, v_y, p]$. We follow the dataset split of Herde et al. (2024), using the first 1000 trajectories for training and the last 200 for validation. Since the raw data has 11 time frames, we take indices 0–3 as input and predict indices 4–10 (i.e. given the first 4 snapshots we predict the next 7).

G Pseudocode of 2D SS-NO

```
class SS-NO(nn.Module):
    '''
    Notation:
        B: batch size
        T: temporal length
        X, Y: spatial dimensions
        C: input channels
        H: hidden dimension
    '''

    def __init__(self,
                 lifting_layer: nn.Module,
                 projection_layer: nn.Module,
                 memory_pre_forward_x: nn.Module,
                 memory_pre_backward_x: nn.Module,
                 memory_pre_forward_y: nn.Module,
                 memory_pre_backward_y: nn.Module,
                 memory_t: nn.Module,
                 memory_post_forward_x: nn.Module,
                 memory_post_backward_x: nn.Module,
                 memory_post_forward_y: nn.Module,
                 memory_post_backward_y: nn.Module):
        super().__init__()
        self.p = lifting_layer
        self.q = projection_layer
        self.memory_pre_forward_x = memory_pre_forward_x
        self.memory_pre_backward_x = memory_pre_backward_x
        self.memory_pre_forward_y = memory_pre_forward_y
        self.memory_pre_backward_y = memory_pre_backward_y
        self.memory_t = memory_t
        self.memory_post_forward_x = memory_post_forward_x
        self.memory_post_backward_x = memory_post_backward_x
        self.memory_post_forward_y = memory_post_forward_y
        self.memory_post_backward_y = memory_post_backward_y

    def forward(self, x: Tensor) -> Tensor:
        '''
        Args:
            x: Input sequence of states (B, C, X, Y, T)
        Returns:
            Predicted next state (B, X, Y, 1)
        '''

        if self.training:
            x = x + torch.randn_like(x) * noise

        x = rearrange(x, 'b c x y t -> (b t) x y c')
        x = self.p(x)
        x = rearrange(x, '(b t) x y h -> (b t) h x y', t=T)

        # --- Pre-memory spatial SSM ---
        x = rearrange(x, '(b t) h x y -> (b t y) h x', t=T)
```

```
        x_fwd = self.memory_pre_forward_x(x)[0]
        x_bwd = torch.flip(self.memory_pre_backward_x(torch.flip(x, dims=[-1]))[0], dims=[-1])
        x = x_fwd + x_bwd
        x = rearrange(x, '(b t y) h x -> (b t) x h y', t=T)

        x = rearrange(x, '(b t) x h y -> (b t x) h y', t=T)
        y_fwd = self.memory_pre_forward_y(x)[0]
        y_bwd = torch.flip(self.memory_pre_backward_y(torch.flip(x, dims=[-1]))[0], dims=[-1])
        x = y_fwd + y_bwd
        x = rearrange(x, '(b t x) h y -> (b t) h x y', t=T)

        # --- Temporal SSM ---
        x = rearrange(x, '(b t) h x y -> b x y h t', t=T)
        x = rearrange(x, 'b x y h t -> (b x y) h t')
        x = self.memory_t(x)[0]
        x = rearrange(x, '(b x y) h t -> b x y h t', x=X, y=Y)
        x = rearrange(x, 'b x y h t -> (b t) h x y', t=T)

        # --- Post-memory spatial SSM ---
        x = rearrange(x, '(b t) h x y -> (b t y) h x', t=T)
        x_fwd = self.memory_post_forward_x(x)[0]
        x_bwd = torch.flip(self.memory_post_backward_x(torch.flip(x, dims=[-1]))[0], dims=[-1])
        x = x_fwd + x_bwd
        x = rearrange(x, '(b t y) h x -> (b t) x h y', t=T)

        x = rearrange(x, '(b t) x h y -> (b t x) h y', t=T)
        y_fwd = self.memory_post_forward_y(x)[0]
        y_bwd = torch.flip(self.memory_post_backward_y(torch.flip(x, dims=[-1]))[0], dims=[-1])
        x = y_fwd + y_bwd
        x = rearrange(x, '(b t x) h y -> (b t) h x y', t=T)

        x = self.q(x)
        x = rearrange(x, '(b t) c x y -> b x y t c', t=T)

        return x
```

Listing 3: SS-NO pseudocode

H Limitations

Despite the significant reduction in model size achieved by SS-NO, several limitations remain.

First, the accumulation of gradients in autoregressive training poses a major scalability bottleneck. Increasing the dimensionality of the latent states rapidly inflates memory usage, which in turn restricts the feasibility of adopting more flexible kernels such as Mamba (Gu & Dao, 2024). This limitation arises not from model expressiveness but from the computational cost of backpropagation through long sequences, where gradient accumulation can become prohibitively expensive. However, unlike language applications where extremely long-range dependencies must be modeled explicitly, PDE dynamics are often well-approximated under a Markovian assumption, where only limited temporal memory is required. This suggests that techniques such as truncated backpropagation, memory-efficient gradient checkpointing, or implicit differentiation may provide effective ways to mitigate the scalability bottleneck in the PDE setting.

Second, the formulation of sequence dynamics and state evolution is naturally aligned with data defined on regular grids or meshes, but generalizing this notion to non-grid domains is more challenging. For example, in settings where the domain is a Lie group, the definition of "neighbors" is no longer restricted to a fixed set of directions (e.g., left and right in 1D), but can in principle include any element reachable through group actions. Extending SS-NO to such domains will require new abstractions for neighborhood structure and state propagation. We leave these issues as directions for future investigation.