Neural Stochastic PDEs: Resolution-Invariant Learning of Continuous Spatiotemporal Dynamics

Cristopher Salvi
Imperial College London &
The Alan Turing Institute
c.salvi@imperial.ac.uk

Maud Lemercier
University of Warwick
maud.lemercier@warwick.ac.uk

Andris Gerasimovičs
University of Bath
ag2616@bath.ac.uk

Abstract

Stochastic partial differential equations (SPDEs) are the mathematical tool of choice for modelling spatiotemporal PDE-dynamics under the influence of randomness. Based on the notion of mild solution of an SPDE, we introduce a novel neural architecture to learn solution operators of PDEs with (possibly stochastic) forcing from partially observed data. The proposed *Neural SPDE* model provides an extension to two popular classes of physics-inspired architectures. On the one hand, it extends Neural CDEs and variants – continuous-time analogues of RNNs – in that it is capable of processing incoming sequential information arriving at arbitrary spatial resolutions. On the other hand, it extends Neural Operators – generalizations of neural networks to model mappings between spaces of functions – in that it can parameterize solution operators of SPDEs depending simultaneously on the initial condition and a realization of the driving noise. By performing operations in the spectral domain, we show how a Neural SPDE can be evaluated in two ways, either by calling an ODE solver (emulating a spectral Galerkin scheme), or by solving a fixed point problem. Experiments on various semilinear SPDEs, including the stochastic Navier-Stokes equations, demonstrate how the Neural SPDE model is capable of learning complex spatiotemporal dynamics in a resolution-invariant way, with better accuracy and lighter training data requirements compared to alternative models, and up to 3 orders of magnitude faster than traditional solvers.

1 Introduction

Stochastic partial differential equations (SPDEs) are the mathematical formalism used to model many physical, biological and economic systems subject to the influence of randomness, be it intrinsic (e.g. quantifying uncertainty) or extrinsic (e.g. modelling environmental random perturbations). Notable examples of SPDEs include the *Kardar–Parisi–Zhang (KPZ) equation* for modelling random interface growth such as the propagation of a forest fire from a burnt region to an unburnt region [12], the *Ginzburg-Landau equation* describing phase transitions of ferromagnets and superconductors near critical temperature [34], or the *stochastic Navier-Stokes equations* modelling the dynamics of a turbulent fluid flow under the presence of local random fluctuations [31]. For an introduction to the theory of SPDEs see Hairer [11]; a comprehensive textbook is Holden et al. [15].

Classical numerical approaches for solving SPDEs include finite difference methods and spectral Galerkin methods [27] among others. To ensure accuracy and stability of numerical solutions of complex SPDEs, computations must be carried out at high resolution using fine discretization grids, rendering the resulting schemes computationally intractable. This limitation motivates the study of data-driven methods that can learn solutions to differential equations from partially observed data.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Related work There has been an increased interest in recent years to combine neural networks and differential equations into a hybrid approach [5, 17, 37].

Neural controlled differential equations (Neural CDEs), as popularised by [3, 18, 32], are continuous-time analogues to recurrent neural networks (RNN, GRU, LSTM etc.). The input to a Neural CDE model is a multivariate time series interpolated into a continuous path $X : [0, T] \to \mathbb{R}^{d_\xi}$; the model consists of a matrix-valued feedforward neural network $f_\theta : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h \times d_\xi}$ parameterizing the vector field of the following dynamical system (and satisfying some minimal Lipschitz regularity to ensure existence and uniqueness of solutions)

$$z_0 = \ell_\theta(u_0), \quad z_t = z_0 + \int_0^t f_\theta(z_s) dX_s, \quad u_t = \pi_\theta(z_t), \tag{1}$$

where $\ell_\theta : \mathbb{R}^{d_u} \to \mathbb{R}^{d_h}$ and $\pi_\theta : \mathbb{R}^{d_h} \to \mathbb{R}^{d_u}$ are feedforward neural networks. The output response $u : [0, T] \to \mathbb{R}^{d_u}$ is then fed to a (possibly pathwise) loss function (mean squared, cross entropy etc.) and trained via stochastic gradient descent in the usual way. In practice, the term "dX_t" means that the solution z_t of the equation (e.g. attention required from a doctor) can change in response to a change of an external stream of information X_t (e.g. heart rate of a patient).

Depending on the level of roughness of the control path X, the integral in eq. (1) can be interpreted in different ways. In [18], X is assumed differentiable and is obtained in practice via cubic splines interpolation of the original time series. In this way, the term "dX_s" can be interpreted as "$\dot{X}_s ds$" so that eq. (1) becomes an ODE of the form $\dot{z}_t = f_\theta(z_t)\dot{X}_t$ that can be evaluated numerically via a call to an ODE solver of choice (Euler, Runge-Kutta, implicit, adaptive stepsize schemes etc.). More generally, if X is of bounded variation then the integral above can be seen as a classical Riemann–Stieltjes or Young integral [38]. Neural SDEs [19, 20, 22, 26] are a special subclass of Neural CDEs where the control is a sample path from a d_ξ-dimensional Brownian motion (which is not of bounded variation), and eq. (1) is understood via stochastic integration (Itô, Stratonovich etc.). Neural RDEs [32] allow to relax even further the regularity assumptions on X by treating the integral using rough integration [10, 30]. In practice, Neural RDEs are particularly well suited for long time series. This is due to the fact that the model can be evaluated via a numerical scheme from stochastic analysis (called the *log-ODE method* [32]) over intervals much larger than what would be expected given the sampling rate of the time series. However, the space complexity of the numerical solver increases exponentially in the number of channels d_ξ, thus model complexity becomes intractable for high dimensional time series. Despite offering many advantages for modelling temporal dynamics, these models are not designed to process signals varying both in space and in time such as physical fields described by SPDEs. In particular, although these models are time-resolution invariant, they are not space-resolution invariant, and are not well suited to capture nonlinear interactions between the various space-time points typically observed in SPDE-dynamics. Similar to CDEs, the solution u to an SPDE is characterized by an initial condition u_0 and a driving noise X. However, in the case of CDEs, (u_0, X_t, u_t) are vectors, while in the case of SPDEs they are functions.

Neural Operators [21, 23, 24, 28] are generalizations of neural networks capable of modelling mappings between spaces of functions and offer an attractive option for learning with spatiotemporal data [21]. Among all kinds of Neural Operators, *Fourier Neural Operators* (FNOs) [25] stand out because of their easier parametrization while demonstrating similar learning performance compared to other Neural Operator models. However, Neural Operators generally fail to incorporate the effect that an external (possibly random) spatiotemporal signal might have on the system they describe. In the case of SPDEs, the external signal is indeed random (e.g. sample from a Wiener process) and its presence leads to new phenomena, both at the mathematical and the physical level, often describing more complex and realistic dynamics than the ones arising from deterministic PDEs.

Contributions To overcome the above limitations faced by Neural CDEs and Neural Operators, we introduce the *neural stochastic partial differential equation* (Neural SPDE) model, capable of learning solution operators of SPDEs from partially observed data by processing, continuously in time and space, incoming sequential information arriving at an arbitrary resolution. We propose two separate algorithms to evaluate our model: the first reduces the Neural SPDE to a system of ODEs in Fourier space, which can then be solved numerically by means of any ODE solver of choice (emulating a spectral Galerkin scheme); the second rewrites the Neural SPDE as a fixed point problem, which is solved via classical root-finding schemes. For both choices of evaluation, the Neural SPDE model inherits memory-efficient backpropagation capabilities provided by existing

adjoint-based and implicit-differentiation-based methods respectively. Finally, we perform extensive experiments on various semilinear SPDEs, including the stochastic Ginzburg-Landau, Korteweg-De Vries, Navier-Stokes equations. The empirical results illustrate several useful aspects of our model: 1) it is space and time resolution-invariant, meaning that even if trained on a lower resolution it can be directly evaluated on a higher resolution; 2) it requires a lower amount of training data to achieve similar or better performance compared to alternative models; 3) its evaluation is up to 3 orders of magnitude faster than traditional numerical solvers.

The outline of the paper is as follows: in Sec. 2 we provide a brief introduction to SPDEs which will help us to define our Neural SPDE model in Sec. 3, followed by numerical experiments in Sec. 4. In Appendix A, we provide an overview of the computational aspects of SPDEs used to design the model and solve SPDEs numerically. Additional experiments can be found in Appendix B.

2 Background on SPDEs

Let $T > 0$ and $d, d_u, d_\xi \in \mathbb{N}$. Let $\mathcal{D} \subset \mathbb{R}^d$ be a bounded domain. Let $\mathcal{H}_u = \{f : \mathcal{D} \to \mathbb{R}^{d_u}\}$ and $\mathcal{H}_\xi = \{f : \mathcal{D} \to \mathbb{R}^{d_\xi}\}$ be two Hilbert spaces of functions from $\mathcal{D}$ to $\mathbb{R}^{d_u}$ and $\mathbb{R}^{d_\xi}$ respectively. We consider a large class of SPDEs of the following type

$$du_t = (\mathcal{L}u_t + F(u_t))\, dt + G(u_t)dW_t, \tag{2}$$

where W_t is either an infinite dimensional Q-Wiener process [27, Def. 10.6] or a cylindrical Wiener process [11, Def. 3.54] with values in $\mathcal{H}_\xi$, $F : \mathcal{H}_u \to \mathcal{H}_u$ and $G : \mathcal{H}_u \to L(\mathcal{H}_\xi, \mathcal{H}_u)$ are two continuous operators, $L(\mathcal{H}_\xi, \mathcal{H}_u)$ is the space of bounded linear operators from $\mathcal{H}_\xi$ to $\mathcal{H}_u$, and $\mathcal{L}$ is a linear differential operator generating a *semigroup*[1] $e^{t\mathcal{L}} : \mathcal{H}_u \to \mathcal{H}_u$. For further details on Wiener processes see Appendix A.2, and for a primer on semigroup theory see Hairer [11, Section 4]. A function $u : [0, T] \to \mathcal{H}_u$ is said to be a *mild solution* of the SPDE (2) if for any $t \in [0, T]$ it satisfies

$$u_t = e^{t\mathcal{L}}u_0 + \int_0^t e^{(t-s)\mathcal{L}}F(u_s)ds + \int_0^t e^{(t-s)\mathcal{L}}G(u_s)dW_s,$$

where the second integral is a stochastic integral interpreted in the Itô sense [11, Def. 3.57]. Thus, an SPDE can be informally thought of as an SDE with values in the functional space $\mathcal{H}_u$ and driven by an infinite dimensional Brownian motion W. Assuming global Lipschitz regularity on F and G, a mild solution u to (2) exists and is unique [11, Thm. 6.4], at least for short times.

We follow Friz and Hairer [9] and consider a regularization $W^\epsilon = \varphi^\epsilon * W$ of the driving noise W with a mollifier[2], where $*$ means convolution. As done in Kidger et al. [18] for Neural CDEs, we can rewrite the mild solution of the mollified version of eq. (2) as the following randomly forced PDE

$$u_t = e^{t\mathcal{L}}u_0 + \int_0^t e^{(t-s)\mathcal{L}}H_\xi(u_s)ds, \quad H_\xi(u_t) := F(u_t) + G(u_t)\xi_t, \tag{3}$$

where $\xi = \dot{W}^\epsilon$, $\mathcal{H}_u = L^2(\mathcal{D}, \mathbb{R}^{d_u})$ and $\mathcal{H}_\xi = L^2(\mathcal{D}, \mathbb{R}^{d_\xi})$. We will refer to ξ as *white noise* if W is a cylindrical Wiener process and as *coloured noise* if W is a Q-Wiener process.

In view of machine learning applications, one should think of W as a continuous space-time embedding of an underlying spatiotemporal data stream. In this paper we are only going to consider W to be a sample path from a Wiener process, but we emphasise that the Neural SPDE model extends, in principle, beyond the scope of SPDEs and could be used for example to process videos in computer vision applications, which we leave as future work. Next we introduce the Neural SPDE model.

[1] A strongly continuous semigroup S on $\mathcal{H}_u$ is a family of bounded linear operators $S = \{S_t : \mathcal{H}_u \to \mathcal{H}_u\}_{t\geq 0}$ with the properties that: 1) $S_0 = \text{Id}$, the identity operator on $\mathcal{H}_u$, 2) $S_t \circ S_s = S_{t+s}$, for any $s, t \geq 0$, and 3) the function $t \mapsto S_t u$ is continuous from $[0, T]$ to $\mathcal{H}_u$, for any $u \in \mathcal{H}_u$.

[2] A mollifier φ is a smooth function on $\mathbb{R}^{d+1}$ that is: 1) compactly supported, 2) $\int_{\mathbb{R}^{d+1}} \varphi(x)dx = 1$, and 3) $\lim_{\epsilon\to 0} \varphi^\epsilon(x) = \lim_{\epsilon\to 0} \epsilon^{-(d+1)}\varphi(x/\epsilon) = \delta(x)$, where δ is the Diract delta function and the limit must be understood in the space of Schwartz distributions.

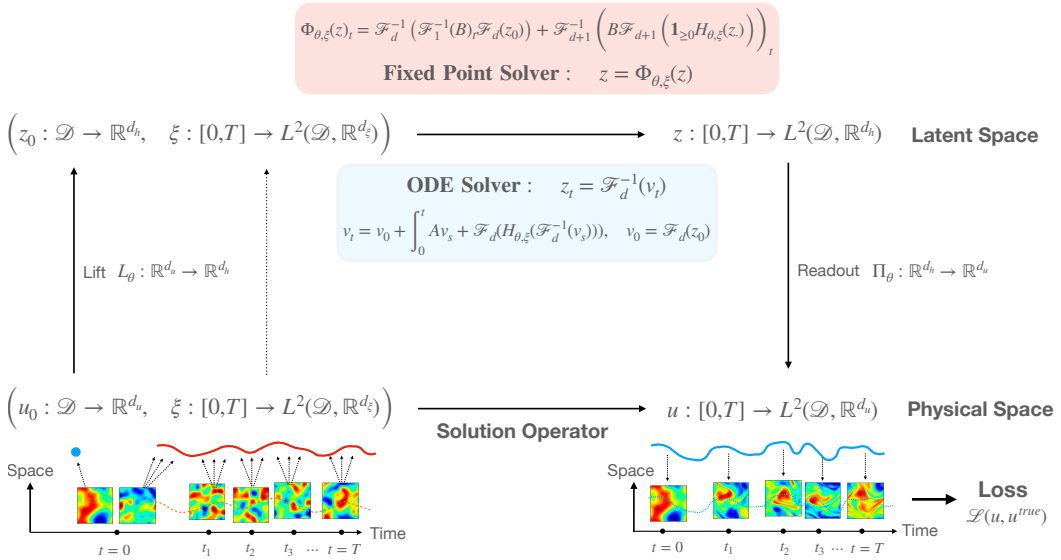

Figure 1: A signal ξ and an initial condition u_0 are observed on a, possibly irregular, spatiotemporal grid. u_0 is lifted to z_0 living in a latent space. The solution of an SPDE in latent space driven by the signal ξ and with initial condition z_0 is obtained using either a fixed point solver or an ODE solver. The solution z is then projected back to the physical space by a linear readout. The output signal u is then fed to a pathwise loss function.

3 Neural SPDEs

For a large class of differential operators $\mathcal{L}$, the action of the semigroup $e^{t\mathcal{L}}$ can be written as an integral against a kernel function $\mathcal{K}_t : \mathcal{D} \times \mathcal{D} \to \mathbb{R}^{d_u \times d_u}$ such that

$$(e^{t\mathcal{L}}h)(x) = \int_{\mathcal{D}} \mathcal{K}_t(x,y)h(y)\mu_t(dy),$$

for any $h \in \mathcal{H}_u$, any $x \in \mathcal{D}$ and $t \in [0,T]$, and where μ_t is a Borel measure on $\mathcal{D}$. As in Kovachki et al. [21], here we take μ_t to be the Lebesgue measure on $\mathbb{R}^d$ but other choices can be made, for example to incorporate prior information. We assume that $\mathcal{K}$ is stationary so that eq. (3) can be rewritten in terms of the spatial convolution $*$

$$u_t = \mathcal{K}_t * u_0 + \int_0^t \mathcal{K}_{t-s} * H_\xi(u_s)ds.$$

For a large class of SPDEs of the form (2), both F and G are local operators acting on a function $h \in \mathcal{H}_u$. In other words, the evaluations $F(h)(x)$ and $G(h)(x)$ at any point $x \in \mathcal{D}$ only depend $h(x)$, and not on the evaluation $h(y)$ at some other point $y \in \mathcal{D}$ in the neighbourhood of x.

3.1 The model

Let $\mathcal{H}_h = L^2(\mathcal{D}, \mathbb{R}^{d_h})$ for some latent space dimension $d_h > d_u$. Let

$$L_\theta : \mathbb{R}^{d_u} \to \mathbb{R}^{d_h}, \quad F_\theta : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}, \quad G_\theta : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h \times d_\xi}, \quad \Pi_\theta : \mathbb{R}^{d_h} \to \mathbb{R}^{d_u}$$

be four feedforward neural networks. For any differentiable control $\xi : [0,T] \to \mathcal{H}_\xi$, define the map $H_{\theta,\xi} : \mathcal{H}_u \to \mathcal{H}_u$ so that for any $h \in \mathcal{H}_u$, $x \in \mathcal{D}$, $t \in [0,T]$

$$H_{\theta,\xi}(h)(x) = F_\theta(h(x)) + G_\theta(h(x))\xi_t,$$

A Neural SPDE is defined as follows

$$z_0(x) = L_\theta(u_0(x)), \quad z_t = \mathcal{K}_t * z_0 + \int_0^t \mathcal{K}_{t-s} * H_{\theta,\xi}(z_s)ds, \quad u_t(x) = \Pi_\theta(z_t(x)). \tag{4}$$

We note that globally Lipschitz conditions can be imposed by using ReLU or tanh activation functions in the neural networks F_θ and G_θ. In secs. 3.2 and 3.3 we propose two distinct algorithms to evaluate the Neural SPDE model (4) which are based on two different parameterization of the kernel $\mathcal{K}$.

3.2 Evaluating the model by solving a system of ODEs

Levereraging the *convolution theorem*, we can rewrite the integral in eq. (4) as follows

$$z_t = \mathcal{F}^{-1}\Big(\mathcal{F}(\mathcal{K}_t)\mathcal{F}(z_0) + \int_0^t \mathcal{F}(\mathcal{K}_{t-s})\mathcal{F}(H_{\theta,\xi}(z_s))ds\Big),$$

where $\mathcal{F}, \mathcal{F}^{-1}$ are the *d-dimensional Fourier transform* (FT) and its inverse (see Def. A.1). If one further assumes that $\mathcal{L}$ is a polynomial differential operator, it can be shown that there exists a map $A : \mathbb{C}^d \to \mathbb{C}^{d_h \times d_h}$ such that $\mathcal{F}(\mathcal{K}_t)(y) = e^{tA(y)}$ (see appendix B.1 for a derivation). It follows that

$$z_t = \mathcal{F}^{-1}\Big(e^{tA}\mathcal{F}(z_0) + \int_0^t e^{(t-s)A}\mathcal{F}(H_{\theta,\xi}(z_s))ds\Big) = \mathcal{F}^{-1}(v_t),$$

where $v_t : \mathbb{C}^d \to \mathbb{C}^{d_h}$ is the solution of the following ODE

$$v_t = v_0 + \int_0^t Av_s + \mathcal{F}(H_{\theta,\xi}(\mathcal{F}^{-1}(v_s))).$$

Hence, z_t can be obtained by applying the inverse FT to the output of an ODE solver on $[0, t]$ with initial condition $\mathcal{F}(z_0)$, vector field $\Psi_{\theta,\xi} := A + \mathcal{F} \circ H_{\theta,\xi} \circ \mathcal{F}^{-1}$, i.e.

$$z_t \approx \mathcal{F}^{-1}\big(\text{ODESolve}(\mathcal{F}(z_0), \Psi_{\theta,\xi}, [0, t])\big)$$

This approach can naturally be seen as a "neural version" of the classical *spectral Galerkin method* for SPDEs as described in Appendix A.3.

We note that this numerical evaluation of the Neural SPDE model (4) allows to inherit memory-efficient adjoint-based backpropagation capabilities as in the evaluation of a Neural CDE. For further details on adjoint-based backpropagation we refer the reader to Chen et al. [5], Kidger [17].

3.3 Evaluating the model by solving a fixed point problem

In our second approach of model evaluation, we make use of three different versions of the FT: the time-only FT $\mathcal{F}_1$ and its inverse $\mathcal{F}_1^{-1}$, the space-only FT $\mathcal{F}_d$ and its inverse $\mathcal{F}_d^{-1}$, and the space-time FT $\mathcal{F}_{d+1}$ and its inverse $\mathcal{F}_{d+1}^{-1}$ (see Def. A.1 for details). Denoting by $\star$ the space-time convolution, the integral in eq. (4) can be rewritten as

$$z_t = \mathcal{K}_t * z_0 + (\mathcal{K} \star \mathbb{1}_{\geq 0} H_{\theta,\xi}(z.))_t\,,$$

where $\mathbb{1}_{\geq 0}$ is the indicator function restricting the temporal domain to the positive real line. Using again the convolution theorem we obtain

$$z_t = \mathcal{F}_d^{-1}\left(\mathcal{F}_d(\mathcal{K}_t)\mathcal{F}_d(z_0)\right) + \mathcal{F}_{d+1}^{-1}\left(\mathcal{F}_{d+1}\left(\mathcal{K}\right)\mathcal{F}_{d+1}\left(\mathbb{1}_{\geq 0}H_{\theta,\xi}(z.)\right)\right)_t,$$

where all multiplications are matrix-vector multiplications. Using the trick introduced in [25], one can parameterize $\mathcal{F}_{d+1}(\mathcal{K})(y)$ directly in Fourier space as a complex tensor B, so that the solution of eq. (4) can be obtained by solving the fixed point problem $z = \Phi_{\theta,\xi}(z)$ with

$$\Phi_{\theta,\xi}(z)_t := \mathcal{F}_d^{-1}\left(\mathcal{F}_1^{-1}(B)_t\mathcal{F}_d(z_0)\right) + \mathcal{F}_{d+1}^{-1}\left(B\mathcal{F}_{d+1}\big(\mathbb{1}_{\geq 0}H_{\theta,\xi}(z.)\big)\right)_t,$$

and where we used the fact that $\mathcal{F}_d(\mathcal{K}_t) = \mathcal{F}_1^{-1}(\mathcal{F}_{d+1}(\mathcal{K}))_t$. This can be solved numerically using classical root-finding schemes (e.g. by Picard's iteration)

$$z \approx \text{FixedPointSolve}(z_0, \Phi_{\theta,\xi}).$$

Analogously to adjoint-based backpropagation for the evaluation approach mentioned in Sec. 3.2, there is a mechanism that leverages the *implicit function theorem* allowing to backpropagate through the operations of a fixed point solver in a memory-efficient way. See Bai et al. [2] for further details.

We note that the FTs are numerically approximated using the *discrete Fourier transform* (DFT) and selecting a maximum number of frequency modes [3] (see Appendix A.1 for details).

[3] The DFT approximates the Fourier series expansion truncated at a maximum number of modes. This allows to specify the shape of the two complex tensors A ($k^1_{\text{max}} \times \ldots \times k^d_{\text{max}} \times d_h \times d_h$) in Sec. 3.2 and B ($k^1_{\text{max}} \times \ldots \times k^{d+1}_{\text{max}} \times d_h \times d_h$) in Sec. 3.3. The k^i_{max} are treated as hyperparameters of the model.

3.4 Space-time resolution-invariance

As depicted in Fig. 1, the input to a Neural SPDE corresponds to a (possibly irregularly sampled) time-indexed sequence of (possibly partially-observed) spatial observations recorded on a space-time grid. The data is then interpolated into a continuous spatiotemporal signal ξ and initial condition u_0. By construction, a Neural SPDE operates in continuous time and space on the tuple of functions (u_0, ξ) and produces a spatiotemporal response u, which is also continuous in space and time; the function u can then be evaluated at an arbitrary space-time resolution, possibly different from the one used during training. In the next section we will demonstrate empirically that even if trained on a coarser resolution, a Neural SPDE can be evaluated on a finer resolution without sacrificing performance, a property known as *zero-shot super-resolution*.

3.5 Comparison of the two evaluation methods

Number of parameters For a fixed dimension d_h of the latent space, the majority of trainable parameters in the ODE parameterization lies in the complex tensor A, which consists of $k^1_{\text{max}} ... k^d_{\text{max}} d_h^2$ parameters, where each k^i_{max} is the maximum number of selected frequencies in the Fourier domain. Regarding the Fixed Point parameterization, the bulk of the parameters is in the complex tensor B, which consists of $k^1_{\text{max}} ... k^{d+1}_{\text{max}} d_h^2$, where the additional frequency is due to the fact that we are taking the FFT in space-time rather than just in space as done in the ODE Solver approach. Hence, the latter would in principle have the advantage of using a lower number of parameters than the former; however, to achieve similar performance, we found that the dimensionality of the latent space has to be roughly 20 times higher, which offsets the aforementioned advantage.

Time complexities The time complexity of the ODE Solver approach is $\mathcal{O}(N N_x \log(N_x))$, where N is the number of time steps taken by the ODE solver and N_x is the number of points on the spatial grid, while the complexity of the Fixed Point approach is $\mathcal{O}(I N_x N_t (\log(N_x) + \log(N_t)))$ where I is the number of Picard iterations and N_t is the number of points on the temporal grid. In our experiments we choose $N_t \approx N_x$ and $I N_t \approx N$, making the two complexities comparable.

Speed of computation We found that the ODE approach is approximately 10 times slower than the Fixed Point approach. We believe this is largely an implementation issue of the `torchdiffeq` library, while the FFT is a highly optimised transform in Pytorch.

3.6 Considerations about convergence

We follow Friz and Hairer [9] and consider a regularization $W^\epsilon = \varphi^\epsilon * W$ of the driving noise W with a compactly supported smooth mollifier φ^ϵ. It is a classical result (Wong-Zakai [35]) from rough path theory [10, 30] that, for the case of SDEs, the sequence of random ODEs driven by the mollification of Brownian motion converges in probability to a limiting process that does not depend on the choice of mollifier and agrees with the Stratonovich solution of the SDE. Furthermore, the solution map $(u_0, W) \mapsto u$ is continuous in an appropriate rough path topology. This result nicely extends to the setting of SPDEs driven by a finite dimensional noise [9, Thm. 1.3]: if u^ϵ denotes the random PDE solutions driven by $\dot{W}^\epsilon dt$ (instead of $\circ dW_t$), then u^ϵ converges in probability to a limiting process corresponding to the Stratonovich solution of the SPDE. In our setting though, the driving noise is infinite dimensional and the resulting integral cannot be interpreted in the Stratonovich sense because otherwise the corresponding Itô-Stratonovich correction would be infinite. Nonetheless, Hairer and Pardoux [14, Thm. 1.1] show that, in the case of the heat operator and under appropriate renormalization and drift correction, the random PDE solution u^ϵ converges in probability to the Itô solution of the SPDE, and that the solution map is continuous in an appropriate regularity structures topology. We note that extending this result to a generic differential operators would require a similarly rigorous proof, which goes beyond the scope of this article and that we leave as future work.

4 Experiments

In this section, we run experiments on three semilinear SPDEs: the stochastic Ginzburg-Landau equation in 4.1, the stochastic Korteweg-De Vries equation in 4.2, and the stochastic Navier-Stokes equations in 4.3. We note that although the assumption of globally Lipschitz vector fields might be

violated for the following SPDEs, well-posedness (i.e. existence of global solutions) can be shown using equation-specific arguments. We consider three supervised operator-learning settings:

- $u_0 \mapsto u$, assuming the noise ξ is not observed;
- $\xi \mapsto u$, assuming the noise ξ is observed, but the initial condition u_0 is fixed across samples;
- $(u_0, \xi) \mapsto u$, assuming the noise ξ is observed and u_0 changes across samples.

We note that learning the operator $u_0 \mapsto u$ of an SPDE without observing the driving noise ξ unavoidably yields poor results for all considered models as only partial information about the system is provided as input. However, we find it informative to include the performances obtained in this setting, as this provides a sanity check that emphasizes the importance of the noise in all the experiments we consider in this paper. Moreover, the ability to process the initial condition u_0 on its own (in absence of noise) testifies that Neural SPDEs can also be used to learn deterministic PDEs. We provide an example on the deterministic Navier-Stokes equations in Appendix B.5.

Neural CDE, Neural RDE, FNO and DeepONet [28, 29] will be the main benchmark models. In addition, we also propose an additional baseline Neural CDE-FNO, which is a hybrid model consisting of a Neural CDE where the drift is modelled by an FNO and the diffusion by a feedforward neural network. The motivation for using a FNO to represent the drift comes from the universal approximation properties of FNOs studied in Kovachki et al. [21, Thm. 4].

An interesting line of work to tackle SPDE-learning is provided in [7, 16]. The authors construct a set of features from the pair (u_0, ξ) following the definition of a model from the theory of regularity structures [13]. They then perform linear [7] and nonlinear [16] regression from these features to the solution of the SPDE at a single time point. Therefore, these models would have to be retrained for any new prediction. In addition, both [7, 16] assume knowledge of the differential operator $\mathcal{L}$ governing the dynamics, while Neural SPDE learns a representation of $\mathcal{L}$ via the parametrization of the associated kernel. For these reasons these recent models are not included in our benchmark.

For all the experiments, the loss function is the relative pathwise L^2 error. The hyper-parameters for all the models are selected by grid-search (see Appendix B.2 for further experimental details). Experiments are run on a Tesla P100 NVIDIA GPU. The code for the experiments is provided in the supplementary material. Additional experiments may be found in Appendix B.

4.1 Stochastic Ginzburg-Landau equation

We start with the stochastic Ginzburg-Landau equation, a reaction diffusion equation in 1D given by

$$\partial_t u - \Delta u = 3u - u^3 + \xi, \qquad u(t,0) = u(t,1), \quad u(0,x) = u_0(x), \quad (t,x) \in [0,T] \times [0,1].$$

This equation is also known as the Allen-Cahn equation in 1-dimension and is used for modeling various physical phenomena like superconductivity [34]. Here ξ denotes space-time white noise with sample paths generated using classical sampling schemes for Wiener processes detailed in A.2.

Table 1: **Ginzburg-Landau**. Relative L2 error on the test set. x indicates that the model is not applicable.

Model	$N = 1\,000$			$N = 10\,000$		
	$u_0 \mapsto u$	$\xi \mapsto u$	$(u_0, \xi) \mapsto u$	$u_0 \mapsto u$	$\xi \mapsto u$	$(u_0, \xi) \mapsto u$
NCDE	x	0.112	0.127	x	0.056	0.072
NRDE	x	0.129	0.150	x	0.070	0.083
NCDE-FNO	x	0.071	0.066	x	0.066	0.069
DeepONet	0.130	0.126	x	0.126	0.061	x
FNO	0.128	0.032	x	0.126	0.027	x
NSPDE (Ours)	0.128	**0.009**	**0.012**	0.126	**0.006**	**0.006**

We consider two data-regimes: a *low data regime* where the total number of training observations is $N = 1\,000$, and a *large data regime* where $N = 10\,000$. In both cases, the response paths are generated by solving the SPDE along each sample path of the noise ξ using a finite difference

scheme described in Appendix A.3 using 128 evenly distanced points in space and time and step size $\Delta t = 10^{-3}$. Following the same setup as in Chevyrev et al. [7, eq. (3.6)], we solve the SPDE until $T = 0.05$ resulting in 50 time points . We choose as initial condition $u_0(x) = x(1-x) + \kappa\eta(x)$, with $\eta(x) = a_0 + \sum_{k=-10}^{k=10} a_k/(1+|k|^2)\sin(k\pi x)$ where $a_k \sim \mathcal{N}(0,1)$. We take $\kappa = 0$ and $\kappa = 0.1$ to generate a dataset where the initial data is either fixed or varies across samples. We provide extra experiments on this SPDE for larger time horizons T and multiplicative forcing in Appendix B.3. We report the results in Table 1. The Neural SPDE model (NSPDE) yields the lowest relative error for all tasks, reaching one order of magnitude improvement on the main task $(u_0, \xi) \mapsto u$ in the large data regime compared to all the applicable benchmark models (NCDE, NRDE, NCDE-FNO). In all settings, even with a limited amount of training samples ($N = 1\,000$), NSPDE achieves $\sim 1\%$ error rate, and marginally improves to $< 1\%$ error when $N = 10\,000$.

4.2 Stochastic Korteweg–De Vries equation

Next, we consider the stochastic Korteweg–De Vries (KdV) equation, a higher order SPDE given by

$$\partial_t u + \gamma \partial_x^3 u = 6u\partial_x u + \xi, \qquad u(t,0) = u(t,1), \quad u(0,x) = u_0(x), \quad (t,x) \in [0,T] \times [0,1]\,.$$

This equation is used to describe the propagation of nonlinear waves at the surface of a fluid subject to random perturbations (another wave equation is studied in Appendix B.4). We refer the reader to Wazwaz [36] for an overview on the KdV equation and its relations to solitary waves. The stochastic forcing is given by $\xi = \dot{W}$ for W being a partial sum approximation of a Q-Wiener process as per Example 10.8 in Lord et al. [27] with $\lambda_j \sim j^{-5+\varepsilon}$ and $\phi_j(x) = \sin(j\pi x)$ (see eq. (6) in Appendix A.2). Taking small $\varepsilon > 0$ guarantees that W_t is twice differentiable in space for every $t \geq 0$. To generate the datasets, we solve the SPDE with $\gamma = 0.1$ until $T = 0.5$.

Table 2: **Stochastic KdV**. Relative L2 error on the test set. The symbol x indicates that the model is not applicable.

(a) $N = 1\,000$ and $T = 0.5$.

Model	$u_0 \mapsto u$	$\xi \mapsto u$	$(u_0, \xi) \mapsto u$
NCDE	x	0.464	0.466
NRDE	x	0.497	0.503
NCDE-FNO	x	0.126	0.259
DeepONet	0.874	0.235	x
FNO	0.835	0.079	x
NSPDE (Ours)	0.832	**0.004**	**0.008**

(b) $N = 1\,000$ and $T = 1$.

Model	$u_0 \mapsto u$	$\xi \mapsto u$	$(u_0, \xi) \mapsto u$
FNO	0.913	0.112	x
NSPDE (Ours)	0.904	**0.009**	0.012

(c) Subsampling ($N = 1\,000$ and $T = 0.5$).

Subsampling rates		$\xi \mapsto u$	$(u_0, \xi) \mapsto u$
Time : 0%	Space : 0%	0.004	0.008
Time : 10%	Space : 0%	0.076	0.059
Time : 0%	Space : 50%	0.005	0.008

The stochastic forcing is simulated using 128 evenly distanced points in space and a time step $\Delta t_{\text{ref}} = 10^{-3}$. We then approximate realizations of the solution of the KdV equation using a time step $\Delta t = 10^{-2}$ until $T = 0.5$. Here, the initial condition is given by $u_0(x) = \sin(2\pi x) + \kappa\eta(x)$, where η is defined as in Sec. 4.1. Similarly to Sec. 4.1 we either take $\kappa = 0$ or $\kappa = 1$ to generate datasets where the initial condition is either fixed or varies across samples. Each dataset consists of $N = 1\,000$ training observations. As reported in Table 2a, Neural SPDEs outperforms the second best model FNO by a full order of magnitude in the task $\xi \mapsto u$ and the second best model NCDE-FNO by almost two orders of magnitude in the task $(u_0, \xi) \mapsto u$. We also perform the same tasks for a larger time horizon $T = 1$ and report the results of a comparison against FNO in Table 2b.

Partial observations Neural SPDEs are able to process signals that are irregularly sampled both in space and in time by interpolating between observations. Yet, the ability of a model to process irregular data does not guarantee its robustness when some observations are dropped. Robustness can only be guaranteed if the signal is regular enough so that replacing dropped observations by interpolation results in a new signal that is close, in some suitable norm, to the original signal. To illustrate this point, we run two additional experiments where we drop uniformly at random 1) 10% of the data in time and 2) 50% of the data in space. As it can be observed in Table 2c, the performance of Neural SPDE remains roughly unchanged when data is dropped in space but decreases when data

is dropped in time, which is to be expected since the driving signal is a Q-Wiener process, which is rough in time, but smoother in space. We also note that to ensure a good approximation of the FT by the FFT, the interpolation must translate the irregular data to a (possibly finer) regular grid.

4.3 Stochastic Navier-Stokes equations in 2D

Finally, we consider the vorticity form of the Navier-Stokes equations for an incompressible flow

$$\partial_t w - \nu \Delta w = -u \cdot \nabla w + f + \sigma \xi, \qquad w(0,x) = w_0(x), \quad (t,x) \in [0,T] \times [0,1]^2, \tag{5}$$

where u is the unique divergence free ($\nabla \cdot u = 0$) velocity field such that $w = \nabla \times u$. These equations describe the motion of an incompressible fluid with viscosity ν subject to external forces [34]. The deterministic forcing f, defined as in Li et al. [25], is a function of space only. The stochastic forcing ξ is given by $\xi = \dot{W}$ for W being a Q-Wiener process which is colored in space and rescaled by $\sigma = 0.05$ (see Appendix A.2). The initial condition is generated according to $w_0 \sim \mathcal{N}(0, 3^{3/2}(-\Delta + 49I)^{-3})$ with periodic boundary conditions. The viscosity is set to $\nu = 10^{-4}$.

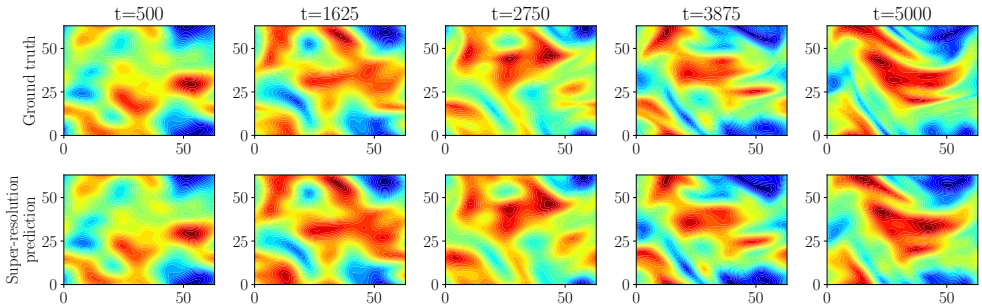

Figure 2: **Top panel:** Solution of the vorticity equation for one realisation of the stochastic forcing between the 500th and the $5\,000^{\text{th}}$ time steps. **Bottom panel:** Predictions with the Neural SPDE model given the initial condition at the 500th time step and the forcing between the 500th and the $5\,000^{\text{th}}$ time steps. The model is trained on a 16×16 mesh and evaluated on a 64×64 mesh.

For each realization of the Q-Wiener process (sampled according to the scheme in Appendix A.2) we solve eq. (5) with a pseudo-spectral solver described in Appendix A.3, where time is advanced with a Crank–Nicolson update. We solve the SPDE on a 64×64 mesh in space and use a time step of size 10^{-3}. For the tasks $u_0 \mapsto u$ and $\xi \mapsto u$, we generate the datasets by solving the SPDE up to time $T = 1$ and downsample the trajectories by a factor of 10 in time (resulting in 100 time steps) and 4 in space (resulting in a 16×16 spatial resolution). The number of training samples is $N = 1\,000$. To generate the training set for the task $(u_0, \xi) \mapsto u$, we generate 10 long trajectories of $15\,000$ steps each up to time $T = 15$. We partition each trajectory into consecutive sub-trajectories of 500 time-steps using a rolling window. This yields a total of $2\,000$ input-output pairs. We split the data into shorter sequences of 500 time steps so that one batch fits in memory of the used GPU.

Table 3: **Stochastic Navier-Stokes**. Relative L2 error on the test set. The symbol x indicates that the model is not applicable, and - indicates that the model does not fit in memory.

Model	$u_0 \mapsto u$	$\xi \mapsto u$	$(u_0, \xi) \mapsto u$
NCDE	x	0.366	0.843
NRDE	x	-	-
NCDE-FNO	x	0.326	0.178
DeepONet	0.432	0.348	x
FNO	0.188	0.039	x
NSPDE (Ours)	0.155	**0.034**	**0.049**

As shown in Table 3, Neural SPDEs marginally outperforms FNO on the task $\xi \mapsto u$, but with a significantly larger gap on the task $(u_0, \xi) \mapsto u$ from the second best model NCDE-FNO. Fig. 2 indicates that our model is capable of zero-shot super-resolution in space-time, achieving good performance even when evaluated on a larger time horizon and on an upsampled spatial grid. Finally, we report in Table 4 some run time statistics indicating that NSPDEs can be up to 3 orders of magnitude faster than traditional numerical solvers.

5 Conclusion

We introduced Neural SPDEs, a model capable of learning solution operators of PDEs with (possibly stochastic) forcing from partially observed data. Our model provides an extension to two classes of physics-inspired models. It extends Neural CDEs in that it is resolution-invariant both in space and in time, and it extends Neural Operators as it can be used to learn solution operators of SPDEs depending simultaneously on the initial condition and driving noise. We performed extensive experiments illustrating how the model achieves superior performance while requiring a lower amount of training data compared to other models, and its evaluation is up to 3 orders of magnitude faster than traditional numerical solvers.

Table 4: Ratio of the inference time of the trained NSPDE over the runtime of the numerical solver. We use the same spatiotemporal discretization for both NSPDE and the numerical solver.

Dataset	Speedup
Ginzburg-Landau	59×
Korteweg-De Vries	80×
Navier-Stokes	300×

Limitations and future work Similarly to other neural operator models, parameterising the kernel in Fourier space is by no means the only available option; other parameterisations could mitigate some of the disadvantages of the FFT (irregular grids, aliasing effect ...), see for example [23]. A question we leave to future work is how to construct a discrepancy between probability measures supported on spatiotemporal signals, generalizing for example the signature kernel MMD in [33]. Neural SPDEs paired with such discrepancy would allow the design of new generative models for spatiotemporal signals. Another research direction will be to assess whether Neural SPDEs can be used in computer vision to process videos at arbitrary resolution.

Acknowledgments and Disclosure of Funding

This project was supported by G-Research and by DataSig under the grant EP/S026347/1.

References

[1] Sh A Alimov, RR Ashurov, and AK Pulatov. Multiple fourier series and fourier integrals. In *Commutative Harmonic Analysis IV*, pages 1–95. Springer, 1992.

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32:690–701, 2019.

[3] Alexis Bellot and Mihaela Van Der Schaar. Policy analysis using synthetic controls in continuous-time. In *International Conference on Machine Learning*, pages 759–768. PMLR, 2021.

[4] William L Briggs and Van Emden Henson. *The DFT: an owner's manual for the discrete Fourier transform*. SIAM, 1995.

[5] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6572–6583, 2018.

[6] 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.

[7] Ilya Chevyrev, Andris Gerasimovics, and Hendrik Weber. Feature engineering with regularity structures. *arXiv preprint arXiv:2108.05879*, 2021.

[8] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[9] Peter K Friz and Martin Hairer. *A course on rough paths*. Springer, 2020.

[10] Massimiliano Gubinelli. Controlling rough paths. *Journal of Functional Analysis*, 216(1): 86–140, 2004.

[11] Martin Hairer. An introduction to stochastic pdes. *arXiv preprint arXiv:0907.4178*, 2009.

[12] Martin Hairer. Solving the kpz equation. *Annals of mathematics*, pages 559–664, 2013.

[13] Martin Hairer. A theory of regularity structures. *Inventiones mathematicae*, 198(2):269–504, 2014.

[14] Martin Hairer and Étienne Pardoux. A wong-zakai theorem for stochastic pdes. *Journal of the Mathematical Society of Japan*, 67(4):1551–1604, 2015.

[15] Helge Holden, Bernt Øksendal, Jan Ubøe, and Tusheng Zhang. Stochastic partial differential equations. In *Stochastic partial differential equations*, pages 141–191. Springer, 1996.

[16] Peiyan Hu, Qi Meng, Bingguang Chen, Shiqi Gong, Yue Wang, Wei Chen, Rongchan Zhu, Zhi-Ming Ma, and Tie-Yan Liu. Neural operator with regularity structure for modeling dynamics driven by spdes. *arXiv preprint arXiv:2204.06255*, 2022.

[17] Patrick Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

[18] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *arXiv preprint arXiv:2005.08926*, 2020.

[19] Patrick Kidger, James Foster, Xuechen Li, and Terry Lyons. Efficient and accurate gradients for neural sdes. *arXiv preprint arXiv:2105.13493*, 2021.

[20] Patrick Kidger, James Foster, Xuechen Li, Harald Oberhauser, and Terry Lyons. Neural sdes as infinite-dimensional gans. *arXiv preprint arXiv:2102.03657*, 2021.

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

[22] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pages 3870–3882. PMLR, 2020.

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

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

[25] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, et al. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2020.

[26] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

[27] Gabriel J Lord, Catherine E Powell, and Tony Shardlow. *An introduction to computational stochastic PDEs*, volume 50. Cambridge University Press, 2014.

[28] 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.

[29] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

[30] Terry J Lyons. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 14(2):215–310, 1998.

[31] Remigijus Mikulevicius and Boris L Rozovskii. Stochastic navier–stokes equations for turbulent flows. *SIAM Journal on Mathematical Analysis*, 35(5):1250–1310, 2004.

[32] James Morrill, Cristopher Salvi, Patrick Kidger, and James Foster. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pages 7829–7838. PMLR, 2021.

[33] Cristopher Salvi, Thomas Cass, James Foster, Terry Lyons, and Weixin Yang. The signature kernel is the solution of a goursat pde. *SIAM Journal on Mathematics of Data Science*, 3(3): 873–899, 2021.

[34] Roger Temam. *Infinite-dimensional dynamical systems in mechanics and physics*, volume 68. Springer Science & Business Media, 2012.

[35] Krystyna Twardowska. Wong-zakai approximations for stochastic differential equations. *Acta Applicandae Mathematica*, 43(3):317–359, 1996.

[36] Abdul-Majid Wazwaz. Solitary waves theory. In *Partial Differential Equations and Solitary Waves Theory*, pages 479–502. Springer, 2009.

[37] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 1(5):1–11, 2017.

[38] William Henry Young. Vi. on the general theory integration. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 204(372-386):221–252, 1905.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See the paragraph entitled "Contributions" in Sec. 1 for a clear statement of the contributions, and Sec. 2 for further details on the paper's scope.
   (b) Did you describe the limitations of your work? [Yes] See Sec. 2 and Sec. 5.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Sec. 2.
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix B.2.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Error bars are not reported because it would be too computationally expensive for the baseline models NCDE, NRDE and NCDE-FNO.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The type of GPU is specified in Sec. 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Citations can be found in the descriptions of the experiments in Sec. 4.
   (b) Did you mention the license of the assets? [Yes] See the code provided as supplementary material.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The code, data and models are included in the supplemental material to reproduce the experiments.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The assets are licensed under the MIT License.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

This appendix is organized as follows. In Appendix A we provide a summary of the computational aspects of SPDEs used for data simulation and model definition, emphasizing the important role of the Fourier Transform (A.1) for simulating noise realizations of Wiener processes (A.2) and building numerical solvers for SPDEs (A.3). In Appendix B we provide additional considerations about our Neural SPDE model and further experimental details (B.2) and additional experiments on the stochastic Ginzburg-Landau (B.3) and wave (B.4) equations, and on the deterministic Navier-Stokes PDE (B.5).

A Computational aspects of SPDEs

We start this section with the definition of the *Fourier Transform* (FT). We then define the *Discrete Fourier Transform* (DFT) as an approximation to the FT of a function observed at finitely many locations. Next, we discuss the role played by the FT to sample realizations of Wiener processes, necessary to build spectral solvers for SPDEs. The interested reader is referred to Briggs and Henson [4] and Lord et al. [27] for further details.

A.1 The Fourier Transform

Let V be a vector space over the complex numbers (e.g. $\mathbb{C}^{d_h}$ or $\mathbb{C}^{d_h \times d_h}$). Let $r \in \mathbb{N}$ and let $\mathcal{C} \subset \mathbb{R}^r$ be a compact subset of $\mathbb{R}^r$. In the paper we used either $r = d$ and $\mathcal{C} = \mathcal{D}$ or $r = d + 1$ and $\mathcal{C} = [0, T] \times \mathcal{D}$.

Definition A.1 (r**-dimensional Fourier Transform).** The r-dimensional FT $\mathcal{F}_r : L^2(\mathbb{R}^r, V) \to L^2(\mathbb{R}^r, V)$ and its inverse $\mathcal{F}_r^{-1} : L^2(\mathbb{R}^r, V) \to L^2(\mathbb{R}^r, V)$ are defined as follows

$$\mathcal{F}_r(f)(y) = \int_{\mathbb{R}^r} e^{-2\pi i \langle x, y \rangle} f(x) dx, \quad \mathcal{F}_r^{-1}(g)(x) = \int_{\mathbb{R}^r} e^{2\pi i \langle x, y \rangle} g(y) dy$$

for any $f, g \in L^2(\mathbb{R}^r, V)$, where $i = \sqrt{-1}$ is the imaginary unit and $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product on $\mathbb{R}^r$.

In practice, we do not observe a function on $\mathbb{R}^r$ but on a subset $\mathcal{C} \subset \mathbb{R}^r$. Furthermore, functions are observed at finitely many locations in $\mathcal{C}$, and another transform—the discrete Fourier transform (DFT)—is used for numerical computations.

In the sequel we denote by Π_N the set of periodic sequences indexed on $\mathbb{Z}_r$ with period vector $(N_1, \dots, N_r)$.

Definition A.2 (r**-dimensional Discrete Fourier Transform).** The r-dimensional DFT $\mathcal{D}_r : \Pi_N \to \Pi_N$ and its inverse $\mathcal{D}_r^{-1} : \Pi_N \to \Pi_N$ are defined as follows,

$$\mathcal{D}_r(u)_n = \sum_{k \in \mathbb{Z}^r \cap \mathcal{R}_N} u_k e^{-2\pi i \langle n, N^{-1} k \rangle}, \quad \mathcal{D}_r^{-1}(v)_k = \frac{1}{|\det N|} \sum_{n \in \mathbb{Z}^r \cap \mathcal{R}_N} v_n e^{2\pi i \langle n, N^{-1} k \rangle}$$

with $N = \mathrm{diag}(N_1, \dots, N_r) \in \mathbb{N}^{r \times r}$, and $\mathcal{R}_N$ the rectangular domain $\mathcal{R}_N = \{x \in \mathbb{R}^r \mid 0 \leq x_i < N_i,\ i = 1, \dots, r\}$.

The DFT of a sequence can be computed exactly and efficiently using the *fast Fourier transform* (FFT) algorithm [8] which reduces the complexity from $\mathcal{O}(M^2)$ to $\mathcal{O}(M \log M)$ where $M = N_1 N_2 \dots N_r$. Most importantly, the FFT algorithm is implemented in machine learning libraries such as PyTorch, which provide support for GPU acceleration and automatic differentiation capabilities.

Note that if we have a finite sequence, we may still define its DFT by implicitly extending the sequence periodically. In particular, when a compactly supported function is sampled on its interval of support, and the samples are used as input for a DFT, it is as if the periodic extension of the function had been sampled. More precisely, consider an input sequence which corresponds to the evaluation of a function f on a regular grid of $\mathcal{C} = \mathcal{R}_N$. For simplicity, suppose that $N_i = N_1$ for all $i = 1, \dots, r$ and consider the grid points $x_n = nL/N_1$ for $n \in \mathbb{Z}^r \cap \mathcal{R}_N$. Taking the DFT of the sequence of general term $u_n = f(x_n)$ we obtain for all $n \in \mathbb{Z}^r$,

$$\mathcal{D}_r(u)_n = \sum_{k \in \mathbb{Z}^r \cap \mathcal{R}_N} u_k e^{-2\pi i \langle n, k/N_1 \rangle} = \sum_{k \in \mathbb{Z}^r \cap \mathcal{R}_N} f(x_k) e^{-2\pi i \langle y_n, x_k \rangle},$$

where y_n are the reciprocal frequency points given by $y_n = n/L$ for $n \in \mathbb{Z}^r \cap \mathcal{R}_N$. The DFT of a compactly supported (or approximately compactly supported) function f sampled on the regular grid of points x_k approximates the FT of f at the frequency points y_n (up to a constant multiplicative factor).

The FT is closely related to the notions of *Fourier coefficients* and *Fourier Series* defined hereafter.

Definition A.3 (r**-dimensional Fourier series).** Let f be a piecewise smooth function $f : \mathbb{R}^r \to V$ which is periodic in x_i with period $L_i \in \mathbb{R}_+$ for all $i = 1, \dots, r$. The r-dimensional Fourier series of f is a representation of the form,

$$f(x) \sim \sum_{n \in \mathbb{Z}^r} c_n(f) e^{2\pi i \langle L^{-1} n, x \rangle},$$

where $L = \mathrm{diag}(L_1, \dots, L_r) \in \mathbb{R}^{r \times r}$ and $c_n(f)$ are complex coefficients, called *Fourier coefficients*, given by

$$c_n(f) = \frac{1}{|\mathrm{det} L|} \int_{\mathcal{R}_L} e^{-2\pi i \langle L^{-1} n, x \rangle} f(x) dx, \;\; n \in \mathbb{Z}^r$$

where $\mathcal{R}_L \subset \mathbb{R}^r$ denotes the rectangular domain of sides $L_1, \dots, L_r$.

We note that in the definition above, the sign $\sim$ means that the series is a formal series and no statement is made about the convergence of the series (the forms of convergence are studied in Alimov et al. [1]). If f is compactly supported on $\mathcal{R}_L$, we may still define its Fourier coefficients, and in this case $\mathcal{F}_r(f)(y_n) = |\mathrm{det} L| c_n(f)$ at the frequency points $y_n = L^{-1} n$.

Numerical consideration

Consider a function f which has compact support (or is periodic) which is observed at M locations in its support (or its unitary cell $\mathcal{R}_L$). When using the DFT to approximate M points of the spectrum $\mathcal{F}_r(f)(y_k)$ (or M coefficients $c_k(f)$), a so-called *aliasing* error usually occurs: due to the periodicity of the DFT, the kth coefficient of the DFT includes the contributions not only of the kth frequency mode, but also from higher modes of the underlying function f. In general the accuracy of the highest frequency modes is more impacted by this error, and aliasing occurs specifically when we compute nonlinear terms in the physical space. For example, in the main paper we approximate the evaluation on a discretization spatiotemporal grid $D \times \mathcal{T}$ of $\mathcal{F}_{d+1}^{-1}(\mathcal{F}_{d+1}(\mathcal{K}) \mathcal{F}_{d+1}(f))$ by $\mathcal{D}_{d+1}^{-1}(B \mathcal{D}_{d+1}(f|_{D \times \mathcal{T}}))$ where $f = \mathbb{1}_{\geq 0} H_{\theta,\xi}(z)$ and $H_{\theta,\xi}$ is nonlinear. One possibility to mitigate aliasing is to set to zero the DFT terms (arising in nonlinearities) corresponding to the highest frequency modes before we apply the inverse DFT to go back to the physical space. This is precisely what we do when we parametrize only $k_{\text{max}}^1 \times \ldots \times k_{\text{max}}^{d+1} \times d_h \times d_h$ entries of the complex tensor B, and set the others to zero, hence resolving potential aliasing errors. We note that specific rules have been proposed (notably in the literature on pseudo-spectral solvers) to deal with specific nonlinearities. However, in the context of Neural SPDE we learn the nonlinearities, hence the number of frequency modes that we retain is treated as an hyperparameter.

A.2 Stochastic simulation of Wiener processes

After defining Wiener processes we outline the sampling procedure that we used to simulate the datasets in the main paper. For more details on computational aspects of SPDEs the reader is referred to Lord et al. [27].

Throughout this section, H will denote a separable Hilbert space (e.g. $H = L^2(\mathcal{D})$) with a complete orthonormal basis $\{\phi_k\}_{k \in \mathbb{N}}$. Let $(\Omega, \mathcal{F}, \mathcal{F}_t, \mathbb{P})$ be a filtered probability space.

A.2.1 Q-Wiener process

Consider an operator $\mathcal{Q} : H \to H$ such that there exists a bounded sequence of nonnegative real numbers $\{\lambda_k\}_{k \in \mathbb{N}}$ such that $Q\phi_k = \lambda_k \phi_k$ for all $k \in \mathbb{N}$ (this is implied by Q being a trace class, non-negative, symmetric operator, for example).

Definition A.4 (Q**-Wiener process).** Let Q be a trace class non negative, symmetric operator on H. A H-valued stochastic process $\{W(t) : t \geq 0\}$ is called a Q-Wiener process if

1. $W(0) = 0$ almost surely;
2. $W(t;\omega)$ is a continuous sample trajectory $\mathbb{R}^+ \mapsto H$, for each $\omega \in \Omega$;
3. $W(t)$ is $\mathcal{F}_t$-adapted and has independent increments $W(t) - W(s)$ for $s < t$;
4. $W(t) - W(s) \sim \mathcal{N}(0, (t-s)Q)$ for all $0 \le s \le t$.

In analogy to the Karhunen Loéve expansion, it can be shown that $W(t)$ is a Q-Wiener process if and only if for all $t \ge 0$,

$$W(t) = \sum_{j=1}^{\infty} \sqrt{\lambda_j}\phi_j\beta_j(t) \tag{6}$$

where $\beta_j(t)$ are i.i.d. Brownian motions, and the series converges in $L^2(\Omega, H)$. Moreover the series is $\mathbb{P}$-a.s. uniformly convergent on $[0,T]$ for arbitrary $T > 0$. (i.e. converges in $L^2(\Omega, \mathcal{C}([0,T], H))$).

In the Navier-Stokes example, we drive the SPDE by samples ξ from a Q-Wiener process in two dimensions. Here we follow Lord et al. [27, Example 10.12] and explain how the sampling procedure works in this case. Let $D = (0, L_1) \times (0, L_2)$ and consider an $L^2(D)$-valued Q-Wiener process $W(t)$. If the eigenfunctions of Q are given by,

$$\phi_k(x) = \frac{1}{\sqrt{L_1 L_2}} e^{2i\pi(k_1 x_1/L_1 + k_2 x_2/L_2)}$$

numerical approximation of sample paths from $W(t)$ are easy to obtain through a DFT. Denote by λ_k the eigenvalues of Q (e.g. $\lambda_k = e^{-\alpha|k|^2}$ for some parameter $\alpha > 0$) and let $\mathcal{J}$ be the index set defined by,

$$\mathcal{J} := \{(j_1, j_2) \in \mathbb{Z}^2 \ : -J_1/2 + 1 \le j_1 \le J_1/2, \ -J_2/2 + 1 \le j_2 \le J_2/2\}$$

The goal is to sample from the truncated expansion of $W(t)$,

$$W^J(t) = \sum_{j \in \mathcal{J}} \sqrt{\lambda_j}\phi_j\beta_j(t),$$

at the collection of sample points,

$$x_k = (L_1 k_1/J_1, L_2 k_2/J_2)^T, \qquad 0 \le k_1 \le J_1 - 1, \ 0 \le k_2 \le J_2 - 1.$$

Consider the random variable $Z(t_n, x)$ defined by,

$$Z(t_n, x) = \sqrt{\Delta t} \sum_{j \in \mathcal{J}} \sqrt{\lambda_j}\phi_j(x)\xi_j^n, \qquad \xi_j^n \sim \mathbb{C}\mathcal{N}(0, 2),$$

meaning that $\xi_j^n = a + ib$ with $a, b \overset{\text{i.i.d}}{\sim} \mathcal{N}(0,1)$ such that $Z(t_n, x_k)$ is a complex random variable with independent real and imaginary part with the same distribution as two independent copies of the increment $W^J(t_n + \Delta t, x_k) - W^J(t_n, x_k)$. Furthermore, $Z(t_n, x_k)$ can be expressed in the form,

$$Z(t_n, x_k) = \frac{1}{J_1 J_2} \sum_{j_1=-J_1/2+1}^{J_1/2} \sum_{j_2=-J_2/2+1}^{J_2/2} \widetilde{Z}_{j_1,j_2} e^{2i\pi\left(j_1 \frac{k_1}{J_1} + j_2 \frac{k_2}{J_2}\right)} \tag{7}$$

where $\widetilde{Z}_{j_1,j_2} = \sqrt{\Delta t \lambda_{j_1,j_2}} J_1 J_2 \xi_{j_1,j_2}^n$ We recognize that the matrix with entries given by eq. (7) is the 2D inverse DFT of the $J_1 \times J_2$ matrix with entries $\widetilde{Z}_{j_1,j_2}$. Therefore, we can sample two independent copies of

$$W^J(t_n + \Delta t, x_k) - W^J(t_n, x_k), \ \ 0 \le k_1 \le J_1 - 1, \ 0 \le k_2 \le J_2 - 1$$

by computing a single 2D inverse DFT.

A.2.2 Cylindrical Wiener process

If the operator $Q = I$ is the identity, then Q is not of trace class on H so that the series in eq. (6) does not converge in $L^2(\Omega, H)$. This motivates the definition of cylindrical Wiener processes.

Definition A.5 (Cylindrical Wiener process). Let H be a separable Hilbert space. A cylindrical Wiener process (a.k.a space-time white noise) is a H-valued stochastic process $\{W(t) : t \geq 0\}$ defined by

$$W(t) = \sum_{j=1}^{\infty} \phi_j \beta_j(t) \tag{8}$$

where $\{\phi_j\}$ is any orthonormal basis of H and $\beta_j(t)$ are i.i.d. Brownian motions.

In all examples except Navier-Stokes, we drive the SPDE by samples ξ from a cylindrical Wiener process in one dimension. Let $D = (0, L)$ and consider an $L^2(D)$-valued cylindrical Wiener process $W(t)$. As explained in Lord et al. [27, Example 10.31], if we take the basis

$$\phi_k(x) = \sqrt{2/L} \sin(k\pi x/L)$$

numerical approximation of sample paths from $W(t)$ are easy to obtain. The goal is to sample from the truncated expansion,

$$W^J(t) = \sum_{j=1}^{J} \phi_j \beta_j(t), \tag{9}$$

at the collection of sample points $x_k = kL/J$ for $k = 1, \ldots, J$. Observing that a trigonometric identity yields,

$$\mathrm{Cov}\left(W^J(t, x_i), W^J(t, x_k)\right) = (tL/J)\delta_{ik}, \qquad i, k = 1, \ldots, J$$

the increments $W^J(t_n + \Delta t, x_k) - W^J(t_n, x_k) \sim \mathcal{N}(0, \Delta t L/J)$ for all $k = 1, \ldots, k$.

A.3 Numerical solvers

In this section we present an overview of the numerical solvers for SPDEs we used to generate the data for all the experiments. The stochastic Ginzburg-Landau (Sec. 4.1 and appendix B.3), stochastic wave (Appendix B.4) equations have been solved using the finite difference method, while the stochastic Korteweg–De Vries (Sec. 4.2) and Navier Stokes (Sec. 4.3) equations have been solved using the spectral Galerkin method. We use the same setup as in Sec. 2. In particular, we focus on stochastic semilinear evolution equations of the form

$$du_t = (\mathcal{L}u_t + F(u_t))\, dt + G(u_t) dW_t \tag{10}$$

where W_t is either a Q-Wiener process or a cylindrical Wiener process and $\mathcal{L}$ is a linear differential operator generating a semigroup $e^{t\mathcal{L}}$. We consider nonlinearities F, G regular enough (see Lord et al. [27, Assumption 10.23]) to guarantee existence and uniqueness of mild solutions of eq. (10) [27, Thm. 10.26].

A.3.1 Finite difference method

We illustrate this numerical method for the reaction-diffusion equation

$$du_t = \left(\epsilon \partial^2_{xx} u + F(u_t)\right) dt + \sigma dW_t, \quad u(0, x) = u_0(x),$$

with homogeneous Dirichlet boundary conditions and where $\epsilon, \sigma > 0$ are constants. We assume for simplicity that u_0, u_t, W_t are real-valued and $\mathcal{D} = (0, a)$. The generalization to higher dimensions is straightforward.

Consider the grid points $x_j = jh$, where $h = \frac{a}{J}$ and $j = 0, ..., J$, for some spatial resolution $J \in \mathbb{N}$. Let $u_J(t)$ be the *finite difference approximation* of $[u(t, x_1), ..., u(t, x_{J-1})]$ (similarly for $W_J(t)$) resulting from the solution of the following SDE

$$du_J(t) = [-\epsilon M u_J(t) + \hat{f}(u_J(t))]dt + \sigma dW_J(t)$$

where $\hat{f}(u_J) = [f(u_1), ..., f(u_{J-1})]^T$ and M is the $(J-1)\times(J-1)$ matrix approximating Laplacian (with free boundary conditions) which is given by

$$M = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}$$

One could modify M for specific boundary conditions. For instance in the case of periodic boundary one should modify $M_{1,J-1} = M_{J-1,1} = -1$ (see Lord et al. [27, Chapter 3.4] for Dirichlet and Neuman boundary condition modifications of M). To discretize in time, we may apply numerical methods for SDEs (see for example Lord et al. [27, Chapter 8]). Choosing the standard Euler-Marayama scheme with time step Δt yields an approximation $u_{J,n}$ to $u_J(t_n)$ at $t_n = n\Delta t$ defined by

$$u_{J,n+1} = (I + \Delta t \epsilon M)^{-1}\left(u_{J,n} + \hat{f}(u_{J,n})\Delta t + \sigma(W_J(t_{n+1}) - W_J(t_n))\right)$$

The increments $(W_J(t_{n+1}) - W_J(t_n))$ are generated using techniques discussed in Appendix A.2.

A.3.2 Spectral Galerkin method

Consider again a separable Hilbert space H. Assume that the differential operator $\mathcal{L}$ in eq. (10) has a complete set of orthonormal eigenfunctions $\{\phi_j\}_{j\in\mathbb{N}}$ and eigenvalues $\lambda_j < 0$, ordered so that $\lambda_{j+1} < \lambda_j$. Then, we can define the semigroup $e^{t\mathcal{L}}$ as follows

$$e^{t\mathcal{L}}h = \sum_{j=1}^{\infty} e^{\lambda_j t}\langle h, \phi_j\rangle\phi_j, \quad h \in H.$$

Define the *Galerkin subspace* $V_J = \text{Span}\{\phi_1, ..., \phi_J\}$ and the orthonormal projections $P_J : H \to V_J$ as follows

$$P_J h = \sum_{i=1}^{J}\langle u, \phi_j\rangle\phi_j, \quad h \in H.$$

Then, the following defines *spectral Galerkin approximation* of eq. (10)

$$du_J(t) = (\mathcal{L}_J u_J(t) + P_J F(u_J(t)))dt + P_J G(u_J(t))dW_J(t), \quad u_J(0) = P_J u_0$$

where $u_J := P_J u$ and $\mathcal{L}_J := P_J\mathcal{L}$ and $W_J = P_J W$ is as in (9). Using a Euluer-Marayama discretization as above, we obtain the following discretization

$$u_{J,n+1} = (I + \Delta t\mathcal{L}_J)^{-1}(u_{J,n} + \Delta t P_J F(u_{J,n}) + P_J G(u_{J,n})\Delta W_{J,n}).$$

This approach is particularly convenient for problems with additive noise where the eigenfunctions of $\mathcal{L}$ and Q (the covariance of the Q-Wiener process W) are equal, which is the case for all the experiments in this paper generated with this method. The eigenfunctions of the Laplacian with periodic boundary conditions correspond to the Fourier basis exponentials; therefore, one can define the projection P_J in terms of the DFT.

B Further experiments

In this section with discuss additional details about the NSPDE model, its training procedure an of the baseline models, including how the relevant hyperparameters have been selected for each model.

B.1 Derivation of the ODE parameterisation

If one assumes that $\mathcal{L}$ is a polynomial differential operator of degree N of the form

$$\mathcal{L} = \sum_{n=0}^{N}\sum_{\substack{n_1,...,n_d \\ n_1+...+n_d=n}} C_{n_1,...,n_d}\frac{\partial^n}{\partial x_1^{n_1}...\partial x_d^{n_d}},$$

where $C_{n_1,\dots,n_d} \in \mathbb{C}^{d_h \times d_h}$ are complex matrices, then the FT of the kernel associated to $\mathcal{L}$ satisfies

$$\mathcal{F}(\mathcal{K}_t)(y) = e^{tP(iy)} \in \mathbb{C}^{d_h \times d_h},$$

for any $y \in \mathbb{C}^d$, where e is the matrix exponential and P is the following matrix-valued polynomial

$$P(y) = \sum_{n=0}^{N} \sum_{\substack{n_1,\dots,n_d \\ n_1+\dots+n_d=n}} (2\pi)^n y_1^{k_1} \dots y_d^{k_d} C_{n_1,\dots,n_d}.$$

Therefore, there exists a map $A : \mathbb{C}^d \to \mathbb{C}^{d_h \times d_h}$ such that $\mathcal{F}(\mathcal{K}_t)(y) = e^{tA(y)}$. It follows that

$$z_t = \mathcal{F}^{-1}\Big(e^{tA}\mathcal{F}(z_0) + \int_0^t e^{(t-s)A}\mathcal{F}(H_{\theta,\xi}(z_s))ds\Big) = \mathcal{F}^{-1}(v_t),$$

where $v_t : \mathbb{C}^d \to \mathbb{C}^{d_h}$ is the solution of the following ODE

$$v_t = v_0 + \int_0^t Av_s + \mathcal{F}(H_{\theta,\xi}(\mathcal{F}^{-1}(v_s))).$$

as shown in section 3.2.

B.2 Additional experimental details

For all experiments the dataset is split into a training, validation and test sets with relative sizes $70\%/15\%/15\%$. For all models, a grid search on the hyperparameters is performed using the training and validation sets. We use the Adam optimizer and a scheduler which reads the validation loss and reduces the learning rate if no improvement is seen for a *patience* number of epochs. Additionally, an early stopping method is used to halt the training of the model if no improvement is seen after a *patience* number of epochs. The hyperparameters included in the grid search are stated below and examples of hyperparameter selection results are provided in tables 5 to 9.

NSPDE The hyperparameters included in the grid search are the number of frequency modes used to parametrize the kernel in Fourier space $B = \mathcal{F}_{d+1}(\mathcal{K})$ and the number of forward iterations used to solve the fixed point problem.

FNO The hyperparameters included in the grid search are the number of frequency modes used to parametrize the kernel and the number of layers M. Note that the numbers of frequency modes in the grid search differ from the ones used for the NSPDE model by a factor 2 to ensure that the effective number of retained modes is the same. For both the NSPDE model and FNO, we kept the number of hidden channels fixed to $d_h = 32$ as this systematically yielded better performances than previously included values and enabled to perform the grid search in a reasonable time.

DeepONet The *Deep Operator Network* (DeepONet) [28] is another popular class of neural network models for learning operators on function spaces. The DeepONet architecture is based on the universal approximation theorem of Chen and Chen [6]. It consists of two sub-networks referred to as the *branch* and the *trunk* networks. The trunk acts on the coordinates $(t, x) \in [0, T] \times \mathcal{D}$, while the branch acts on the evaluation of the initial condition u_0 on a discretized grid D. Therefore, the DeepONet is not a space resolution-invariant architecture. The output of the network is expressed as

$$\text{DeepONet}(u_0)(t, x) = \sum_{k=1}^{p} b_k(u_0)\tau_k(t, x) + b_0,$$

where the b_k and the τ_k are the outputs of the branch and trunk network respectively. The trunk network is usually a feedforward neural network, and one can chose the architecture of the branch network depending on the structure of the input domain. We follow Lu et al. [28] and use feedforward neural networks for both the trunk and the branch networks. We perform a grid search on the depth and width of the trunk and branch feedforward neural networks.

NRDE/NCDE/NCDE-FNO The hyperparameters included in the grid search are the number of hidden channels and the type of solver as implemented by torchdiffeq [5]. We note that we used a depth-2 NRDE model (depth-2 already results in $d_\xi = 8\,385$ for forcings observed at 128 spatial points and higher depths models could not fit in memory) and recall that NCDE is a depth-1 NRDE.

Table 5: Grid search NCDE (KdV)

d_h	# parameters	solver	validation loss
8	136464	rk4	0.511
16	272672	rk4	0.510
32	545088	rk4	0.505
8	136464	euler	0.560
16	272672	euler	0.561
32	545088	euler	0.556

Table 6: Grid search NCDE-FNO (KdV)

d_h	# parameters	solver	validation loss
8	5761	rk4	0.140
16	15617	rk4	0.142
32	48769	rk4	0.145
8	5761	euler	0.310
16	15617	euler	0.314
32	48769	euler	0.321

Table 7: Grid search DeepONet (KdV)

Branch & trunk width	Branch depth	Trunk depth	# parameters	validation loss
256	3	4	1935616	0.258
128	3	4	885888	0.269
128	3	3	869376	0.279
128	3	2	852864	0.284
512	3	4	4526592	0.294
512	2	2	3738624	0.295
256	4	4	2001408	0.295
512	4	4	4789248	0.302
256	2	2	1738240	0.304
256	3	3	1869824	0.307
128	4	3	885888	0.311
128	4	2	869376	0.311
256	3	2	1804032	0.313
128	2	2	836352	0.315
256	4	2	1869824	0.316
128	4	4	902400	0.318
256	4	3	1935616	0.319
512	4	3	4526592	0.320
512	4	2	4263936	0.334
512	3	2	4001280	0.350
128	2	4	869376	0.350
512	3	3	4263936	0.359
512	2	3	4001280	0.366
256	2	3	1804032	0.370
128	2	3	852864	0.382
256	2	4	1869824	0.392
512	2	4	4263936	0.395

Table 8: Grid search FNO (KdV)

d_h	depth	modes 1	modes 2	# parameters	validation loss
32	1	16	16	532993	0.163
32	1	16	50	1647105	0.117
32	1	32	16	1057281	0.163
32	1	32	50	3285505	0.117
32	2	16	16	1058337	0.163
32	2	16	50	3286561	0.120
32	2	32	16	2106913	0.163
32	2	32	50	6563361	0.118
32	3	16	16	1583681	0.163
32	3	16	50	4926017	0.120
32	3	32	16	3156545	0.163
32	3	32	50	9841217	0.120
32	4	16	16	2109025	0.163
32	4	16	50	6565473	0.122
32	4	32	16	4206177	0.163
32	4	32	50	13119073	0.120

Table 9: Grid search NSPDE (KdV)

d_h	Picard's iterations	modes 1	modes 2	# parameters	validation loss
32	1	32	32	1055233	0.023
32	1	32	100	3283457	0.011
32	1	64	32	2103809	0.030
32	1	64	100	6560257	0.009
32	2	32	32	1055233	0.018
32	2	32	100	3283457	0.012
32	2	64	32	2103809	0.015
32	2	64	100	6560257	0.010
32	3	32	32	1055233	0.016
32	3	32	100	3283457	0.013
32	3	64	32	2103809	0.022
32	3	64	100	6560257	0.011
32	4	32	32	1055233	0.019
32	4	32	100	3283457	0.012
32	4	64	32	2103809	0.021
32	4	64	100	6560257	0.016

B.3 Stochastic Ginzburg-Landau equation

Recall that the stochastic Ginzburg-Landau equations are of the form,

$$\begin{aligned} \partial_t u - \Delta u &= 3u - u^3 + G(u)\xi, \\ u(0,x) &= u_0(x), \quad (t,x) \in [0,T] \times [0,1] \end{aligned} \tag{11}$$

subject to either Periodic or Dirichlet boundary conditions. Periodic boundary conditions are given by $u(t,0) = u(t,1)$ for all $t \geq 0$ and Dirichlet boundary conditions are given by $u(t,0) = u(t,1) = 0$ for all $t \geq 0$. Initial condition we take as in Sec. 4.1 $u_0(x) = x(1-x) + \kappa\eta(x)$ with $\kappa = 0$ or $\kappa = 0.1$ depending on a task. In both Periodic and Dirichlet case we can take $\eta(x)$ as in Sec. 4.1 though in Dirichlet case one must take $a_0 = 0$ to ensure u_0 being zero at the boundary.

We first reproduce an experiment from Sec. 4.1 on the additive stochastic Ginzburg-Landau equation but with Dirichlet boundary conditions instead of the periodic. We compare it to the benchmark of FNO model which was the most successful among all the benchmarks of Sec. 4. From Table 10 we

see that even though Neural SPDE model depends on the spectral methods the errors did not increase compared to the periodic equation in Sec. 4.1 (see Table 1). Our algorithm still outperforms FNO whose relative $L2$ error increased slightly. The fact that Neural SPDE can be applied to non-periodic equations could be perhaps explained by interpolation (L_θ) and projection (Π_θ) neural networks that could correct for non-periodicity of the data.

Table 10: **Additive stochastic Ginzburg-Landau equation with homogeneous Dirichlet boundary conditions**. The experimental setup is the same as in the main paper. We report the relative L2 error on the test set. The symbol x indicates that the model is not applicable. N is fixed to $1\,000$.

Model	$u_0 \mapsto u$	$\xi \mapsto u$	$(u_0, \xi) \mapsto u$
FNO	0.132	0.023	x
NSPDE (Ours)	0.135	0.008	0.010

We now take a look at the specific hyperparameter: number of forward iterations in the fixed point solver. We also call this a number of Picard iterations P. Theoretically as P increases Fixed Point Solver should converge to the true solution (see [11]). This suggests that higher P should improve the performance of the Neural SPDE algorithms. In practise we observed in both additive Ginsburg Landau equation from Sec. 4.1 and in KdV equation from Sec. 4.2 that $P = 1$ could already be enough. This could be explained either by dominance of the linear part of the equation or by overfitting in these cases. Thus we present an experiment on the multiplicative stochastic Ginzbug-Landau equation over a longer (compared to Sec. 4.1) time interval. In the Table 11 we compare NSPDE with $P \in \{1, 2, 3, 4\}$ and again include FNO benchmark (which performed best in the previous experiments). We see that NSPDE with even $P = 1$ outperforms FNO. Relative $L2$ error for $T = 0.05$ increases for both NSPDE and FNO due to more complicated multiplicative noise. In Table 11 we present for each P the best result over other hyperparameters obtained by cross validation. One could clearly see an improvement in error as we increase the number of Picard iterations P (with an exception of the case $T = 0.05$ where $P = 3$ outperformed $P = 4$). This improvement becomes more apparent as the time frame T increases. Heuristically (and qualitatively) this is due to the fact that for the short times solution of the SPDE is relatively close to its linearised version and that nonlinearity of the equation starts to play a bigger role for larger T.

Table 11: **Multiplicative stochastic Ginzburg-Landau equation**. We report the relative L2 error on the test for FNO and NSPDE (Ours) for different number of Picard iterations on the task $\xi \to u$.

Time horizon	FNO	NSPDE ($P = 1$)	NSPDE ($P = 2$)	NSPDE ($P = 3$)	NSPDE ($P = 4$)
$T = 0.05$	0.040	0.023	0.018	**0.016**	0.017
$T = 0.10$	0.068	0.042	0.041	**0.040**	**0.040**
$T = 0.25$	0.105	0.079	0.077	0.073	**0.072**

B.4 The stochastic wave equation

In this section we consider the following nonlinear wave equation with multiplicative stochastic forcing,

$$\begin{aligned} \partial_t^2 u - \Delta u &= \cos(\pi u) + u^2 + u\xi, \\ u(t,0) &= u(t,1), \\ u(0,x) &= u_0(x), \\ \partial_t u(0,x) &= v_0(x), \quad (t,x) \in [0,T] \times [0,1]. \end{aligned} \tag{12}$$

The nonlinear stochastic wave equation arises in relativistic quantum mechanics and is also used in simulations of nonlinear waves that are subject to either noisy observations or random forcing. We refer a reader to Temam [34] for an overview on the nonlinear wave equation. The above equation can put in a form of eq. (2) by rewriting it as a system for $(u, v) = (u, \partial_t u)$. To generate training

datasets, we solve the SPDE using a finite difference method with 128 evenly distanced points in space and a time step size $\Delta t = 10^{-3}$. As in Chevyrev et al. [7, eq. (3.5)], we solve the SPDE until $T = 0.5$. We then downsample the temporal resolution by a factor 5, resulting in 100 time points. Here, the initial condition is given by $u_0(x) = \sin(2\pi x) + \kappa\eta(x)$, where η is defined in Sec. 4.1 and for simplicity initial velocity v_0 is taken deterministic $v_0(x) = x(1-x)$. Similarly to Sec. 4.1 we either take $\kappa = 0$ or $\kappa = 1$ to generate datasets where the initial condition is either fixed or varies across samples. Each dataset consists of $N = 1\,000$ training observations.

Table 12: **Stochastic Wave equation**. We report the relative L2 error on the test set. The symbol x indicates that the model is not applicable. N is fixed to $1\,000$.

Model	$u_0 \mapsto u$	$\xi \mapsto u$	$(u_0, \xi) \mapsto u$
NCDE	x	0.142	0.432
NRDE	x	0.146	0.445
NCDE-FNO	x	0.029	0.037
DeepONet	0.190	0.143	x
FNO	0.151	0.026	x
NSPDE (Ours)	0.150	**0.023**	**0.026**

B.5 Deterministic Navier-Stokes PDE

In this final experiment, we demonstrate that our Neural SPDE model can also be used in the setting of PDEs without any stochastic term. We do so by studying the example from [25] on deterministic Navier-Stokes. More precisely, we consider the 2D Navier-Stokes equation for a viscous, incompressible fluid in vorticity form:

$$\partial_t w(t,x) - \nu\Delta w(t,x) = f(x) - u(t,x)\cdot\nabla w(t,x), \qquad t\in[0,T], x\in[0,1]^2 \tag{13}$$

$$\nabla\cdot u(t,x) = 0, \qquad t\in[0,T], x\in[0,1]^2 \tag{14}$$

$$w(x,0) = w_0(x), \qquad x\in[0,1]^2 \tag{15}$$

where $u : [0,T]\times[0,1]^2 \to \mathbb{R}^2$ is the velocity field, $w = \nabla\times u$ is the vorticity with $w_0 : [0,1]^2 \to \mathbb{R}$ being the initial vorticity. Here f is a deterministic forcing term which we take as in [25]. We follow the experimental setup from [25] and use the dataset (available under an MIT license) where $\nu = 10^{-5}$, $N = 1000$ and $T = 20$. We achieve similar performances as FNO with a L2 error of 0.17. A comparison between a true and predicted trajectory is depicted in Fig. 3.

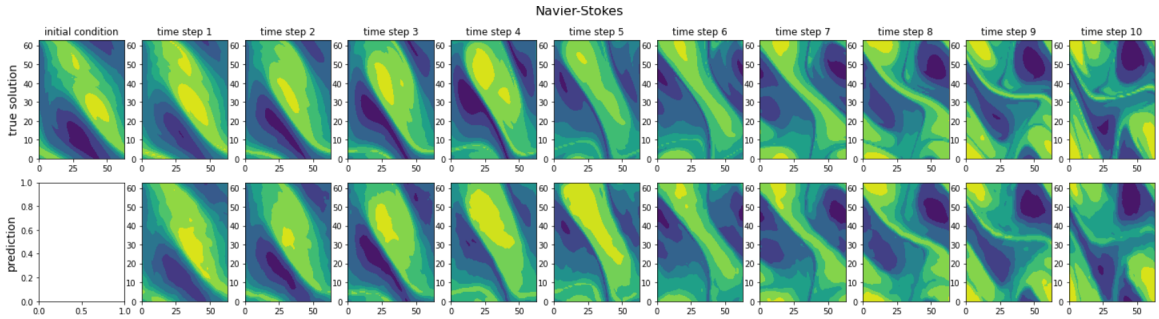

Figure 3: **Top panel:** Initial vorticity and ground truth vorticity at later time steps on a 64×64 mesh. **Bottom panel:** Predictions of the Neural SPDE model.