KOOPMAN NEURAL OPERATOR FOR LEARNING NON-LINEAR PARTIAL DIFFERENTIAL EQUATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

The lacking of analytic solutions of diverse partial differential equations (PDEs) gives birth to series of computational techniques for numerical solutions. In machine learning, numerous latest advances of solver designs are accomplished in developing neural operators, a kind of mesh-free approximators of the infinite-dimensional operators that map between different parameterization spaces of equation solutions. Although neural operators exhibit generalization capacities for learning an entire PDE family simultaneously, they become less accurate and explainable while learning long-term behaviours of non-linear PDE families. In this paper, we propose Koopman neural operator (KNO), a new neural operator, to overcome these challenges. With the same objective of learning an infinite-dimensional mapping between Banach spaces that serves as the solution operator of target PDE family, our approach differs from existing models by formulating a non-linear dynamic system of equation solution. By approximating the Koopman operator, an infinite-dimensional linear operator governing all possible observations of the dynamic system, to act on the flow mapping of dynamic system, we can equivalently learn the solution of an entire non-linear PDE family by solving simple linear prediction problems. In zero-shot prediction and long-term prediction experiments on representative PDEs (e.g., the Navier-Stokes equation), KNO exhibits notable advantages in breaking the tradeoff between accuracy and efficiency (e.g., model size) while previous state-of-the-art models are limited.

1 INTRODUCTION

Partial differential equation solvers are important. In science and engineering, partial differential equation (PDE) is a fundamental tool to characterize various problems (e.g., problems in fluid mechanics, quantum mechanics, and civil engineering) Debnath & Debnath (2005). However, even though significant progress has been achieved on solving PDEs Tanabe (2017), numerous important PDEs, such as the Navier-Stokes equation, still lack analytic solutions Gockenbach (2005). Consequently, the urgent needs of solving complicated PDEs in real applications have given birth to diverse techniques for computationally approximating PDE solutions Mattheij et al. (2005).

Given $\Phi = \Phi\left(D; \mathbb{R}^{d_\phi}\right)$, a Banach space of inputs, and $\Gamma = \Gamma\left(D; \mathbb{R}^{d_\gamma}\right)$, a Banach space of solutions, defined on a bounded open set $D \subset \mathbb{R}^d$, these PDE solvers are expected to approximate a solution operator $\mathcal{F}$ that relates Φ with Γ for a typically time-dependent PDE family

$$\partial_t \gamma\left(x_t\right) = \left(\mathcal{L}_\phi \gamma\right)\left(x_t\right) + \eta\left(x_t\right),\ x_t \in D \times T, \tag{1}$$

$$\gamma\left(x_t\right) = \gamma_B,\ x_t \in \partial D \times T, \tag{2}$$

$$\gamma\left(x_0\right) = \gamma_I,\ x_0 \in D \times \{0\}. \tag{3}$$

In Eq. (1-3), notions γ_B and γ_I denote the boundary and initial conditions, set $T = [0, \infty)$ is the time domain, differential operator $\mathcal{L}_\phi$ is characterized depending on ϕ, fixed function $\eta\left(\cdot\right)$ lives in an appropriate function space determined by $\mathcal{L}_\phi$, and $\gamma\left(\cdot\right)$ is the solution of the PDE family. This formulation gives rise to the solution operator $\mathcal{F} : \left(\phi, \gamma_B, \gamma_I\right) \mapsto \gamma$, which reduces to $\mathcal{F} : \phi \mapsto \gamma$ if boundary and initial conditions are constant. For convenience, we always consider fixed boundary and initial conditions in our subsequent derivations. In application, researchers consider a parametric counterpart $\mathcal{F}_\theta \simeq \mathcal{F}$ parameterized by θ to define optimization problems Li et al. (2020b).

Existing partial differential equation solvers are diverse. To date, diverse types of PDE solvers have been developed, which can be generally classified as following:

- **Classic numerical solvers.** The earlist and commonest PDE solvers, such as finite element Reddy (2019) and finite difference Lipnikov et al. (2014) methods, solve PDEs by discretizing the space according to specific mesh designs. These approaches are granularity-dependent, whose accuracy favors fine-grained meshes while efficiency prefers coarse-grained meshes Tadmor (2012). Therefore, they inevitably face the tradeoff between accuracy and efficiency when the target PDE is complicated Li et al. (2020a).
- **Neural-network-based solvers.** To revolutionize the computational techniques of PDE solving, three types of neural-network-based solvers have been proposed to approximate or enhance the classic ones in a fast manner Raissi et al. (2019); Kochkov et al. (2021):
  - **Mesh-dependent and finite-dimensional operators.** The first type of solvers approximate the solution operator as a parameterized neural network between finite Euclidean spaces after discretizing domains D and T into x and y meshes, i.e., $\mathcal{F}_\theta : \mathbb{R}^x \times \mathbb{R}^y \times \Theta \to \mathbb{R}^x \times \mathbb{R}^y$ Guo et al. (2016); Zhu & Zabaras (2018); Bhatnagar et al. (2019). These solvers are mesh-dependent and require fine-tuning on different values of n, leading to limited generalization capacities Li et al. (2020b).
  - **Neural finite element methods.** The second type of solvers directly parameterize equation solution $\gamma(\cdot)$ as a neural network, which equivalently gives rise to $\mathcal{F}_\theta : D \times T \times \Theta \to \mathbb{R}$ Yu et al. (2018); Raissi et al. (2019); Bar & Sochen (2019); Pan & Duraisamy (2020). Although these solvers are mesh-independent and accurate, they are limited to learn a certain instance of the PDE rather than the entire family Li et al. (2020b). Therefore, similar to the classic numerical ones, these solvers requires new network design and training whenever the instance is changed. Meanwhile, they are not applicable to the cases where the underlying PDE remains unknown.
  - **Neural operators.** The third type of solvers are developed to learn a mesh-dependent and infinite-dimensional solution operator with neural networks, i.e, $\mathcal{F}_\theta : \Phi \times \Theta \to \Gamma$ Lu et al. (2019); Bhattacharya et al. (2020); Nelsen & Stuart (2021); Li et al. (2020b;a); Kovachki et al. (2021); Li et al. (2022). These solvers overcome the dependence on meshes by learning networ parameters in a manner applicable to different discretizations. Because these solvers learn the solution operator directly, they only need to be trained once for a target PDE family. Generating equation solution $\gamma(\cdot)$ of different instances of the PDE family only requires a forward pass of networks, which is computationally favorable Li et al. (2020b;a). Although neural operators are initially not competitive with other neural-network-based solvers because evaluating kernal integral operators is costly, the latest approach, named as Fourier neural operator Li et al. (2020a), resolves this limitation by fast Fourier transform.

Compared with classic numerical solvers, neural-network-based solvers, especially neural operators, are more efficient in dealing with science and engineering questions where PDEs are complicated Li et al. (2020b;a). Therefore, our research primarily focus on neural operator designs.

Existing partial differential equation solvers are not perfect. Despite substantial progress achieved by neural operators in theoretical foundations (e.g., Kovachki et al. (2021)), approximator designs (e.g., Li et al. (2020a; 2022)), and applications (e.g., Guibas et al. (2021)), there still remain numerous challenges in existing neural operator solvers, among which, a critical one lies in the limited capacity of existing models to learn the long-term dynamics of complicated PDEs.

Let us rethink about the iterative update strategy of neural operators Li et al. (2020b)

$$\widehat{\gamma}(x_{t+\varepsilon}) = \sigma\left(W\widehat{\gamma}(x_t) + \int_{D\times\{t\}} \kappa_\theta\left(x_t, y_t, \phi(x_t), \phi(y_t)\right)\widehat{\gamma}(y_t)\,\mathrm{d}y_t\right),\ \forall\ x_t \in D\times\{t\}, \quad (4)$$

where $\varepsilon \in (0,\infty)$ denotes time difference, $\widehat{\gamma} : D\times T \to \mathbb{R}^{d_{\widehat{\gamma}}}$ denotes the neural network representation of equation solution γ generated by specific embedding, mapping $\sigma : \mathbb{R}\to\mathbb{R}$ denotes an element-wise activation function, notion $W : \mathbb{R}^{d_{\widehat{\gamma}}} \to \mathbb{R}^{d_{\widehat{\gamma}}}$ denotes a linear transformation, and $\kappa_\theta : \mathbb{R}^{2(d+d_\phi)} \to \mathbb{R}^{d_{\widehat{\gamma}}}$ is a neural network parameterized by θ Li et al. (2020b). The integral term related to κ_θ defines the kernel integral operator mentioned above, whose computing efficiency is

improved by the well-known Fourier neural operator Li et al. (2020a), i.e., Eq. (4) can be reformulated as $\widehat{\gamma}\left(x_{t+\varepsilon}\right)=\sigma\left[W\widehat{\gamma}\left(x_t\right)+\mathcal{F}^{-1}\left(\mathcal{F}\left(\kappa_\theta\right)\cdot\mathcal{F}\left(\widehat{\gamma}\left(x_t\right)\right)\right)\right]$ for any $x_t\in D\times\{t\}$ (notion $\mathcal{F}\left(\cdot\right)$ is the Fourier transform that can be realized by fast Fourier transform in application).

At the first glance, Eq. (4) is similar to a dynamic system perspective where we study the flow mapping $\zeta:\mathbb{R}^{d_\gamma}\times T\to\mathbb{R}^{d_\gamma}$ of an infinite-dimensional non-linear dynamic system $\gamma_t=\gamma\left(D\times\{t\}\right)$ (notion $\gamma\left(D\times\{t\}\right)$ represents that function γ acts on all elements in set $D\times\{t\}$)

$$\gamma_{t+\varepsilon}=\gamma_t+\int_t^{t+\varepsilon}\zeta\left(\gamma_\tau,\tau\right)\mathsf{d}\tau,\ \forall t\in T. \tag{5}$$

For most non-linear PDE families, the concerned flow mapping ζ is even more intricate than the equation solution γ itself. Similar to other dynamic system prediction tasks, challenges frequently arise as time difference ε enlarges. The infinite-dimensional non-linear flow mapping ζ makes the accurate prediction of long-term dynamics (i.e., $\varepsilon\to\infty$) of γ_t nearly impossible, limiting the prediction accuracy of existing models. To maintain accuracy, existing models are required to enlarge model size or complexity, inevitably facing the trade-off between accuracy and efficiency. Certainly, a model can constrain its strategy to only predict short-term dynamics (i.e., $\varepsilon=1$). However, learning long-term dynamics of PDEs serves as the cornerstone of diverse important applications, such as meteorological forecast Pathak et al. (2022), epidemic prevention Cao et al. (2020), and economic system modelling Aminian et al. (2006). Therefore, overcoming the limitation in learning long-term PDE behaviours is necessary for optimizing PDE solvers, which is the core objective of our research.

Our framework and contribution. In this paper, we attempt to overcome the challenge of predicting long-term PDE dynamics by proposing a new neural operator named as Koopman neural operator (KNO). Our framework and contributions are summarized as following

- **Long-term behaviour of the PDE family as an non-linear dynamic system of equation solution.** Besides learning the solution operator of an entire target PDE family, we formalize a non-linear dynamic system of equation solution described by Eq. (5) in the meanwhile. This characterization supports to optimize the iterative update strategy of neural operators in Eq. (4) using dynamic system theory.
- **Equivalent linear prediction of non-linear dynamics via Koopman operator.** To capture intricate long-term dynamics, our model is designed to learn the Koopman operator, an infinite-dimensional linear operator governing all observations of a dynamic system, to act on the flow mapping ζ of the dynamic system of equation solution. By doing so, we can transform the original task into an equivalent but simpler linear prediction problem.
- **Balance between accuracy and efficiency in zero-shot and long-term prediction.** We compare KNO with existing state-of-the-art models (e.g. the Fourier neural operator Li et al. (2020a)) in zero-shot prediction (i.e., testing on an untrained discretization granularity or an untrained prediction length) and long-term prediction (i.e., with a large ε) experiments with representative PDEs (e.g., the Navier-Stokes equation and the Bateman–Burgers equation). While previous methods suffer from the tradeoff between accuracy and efficiency (e.g., model size), KNO is shown to achive higher accuracy with a smaller model size.

2 Framework of Koopman neural operator

Non-linear dynamic system of equation solution. We begin with formalizing the non-linear dynamic system of equation solution. As defined in Eq. (5), the dynamic system is given as

$$\partial_t\gamma_t=\zeta\left(\gamma_t,t\right),\ \forall\gamma_t\in\mathbb{R}^{d_\gamma}\times T, \tag{6}$$

which can be either non-autonomous (i.e., the flow mapping θ associated with $\zeta\left(\cdot,\cdot\right)$ is time-dependent) or autonomous (i.e., the flow mapping θ is time-independent such that $\partial_t\zeta\left(\cdot,t\right)\equiv 0$) in different PDE families. In modern dynamic system theory Brunton et al. (2022), Eq. (6) generally leads to the cocycle property of the flow mapping θ

$$\theta_t^{t'}=\theta_{t+\tau}^{t'}\circ\theta_t^{t+\tau},\forall t\le t+\tau\le t'\in T \tag{7}$$

where notion $\circ$ denotes the composition of mappings. This cocycle property defines how γ, the equation solution of target PDE family, evolves across adjoining time intervals.

Time-dependent Koopman operator. The non-linear, and potentially non-autonomous, dynamic system in Eq. (6) makes long-term prediction a daunting challenge. In practice, researchers always expect to deal with a linear dynamic system (i.e., $\partial_t \gamma_t = \mathcal{A}\gamma_t$ where $\mathcal{A}$ is a linear operator), whose dynamics is sufficientlly simple and can be accurately learned. Therefore, a natural question is whether we can transform the original system in Eq. (6) to a linear one to simplify the learning task. According to modern dynamic system theory, such an objective can be realized by formulating the Koopman operator $\mathcal{K}$, an infinite-dimensional linear operator governing all possible observations of a dynamic system, to act on the flow mapping ζ and linearizing the original system in an appropriate space Brunton et al. (2022). Let us take the case where system γ_t is autonomous (i.e., $\theta_t^{t'}$ can be simplified as $\theta^{t'}$) as a simple illustration, the family of Koopman operators $\mathcal{K}^\varepsilon : \mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right) \rightarrow \mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right)$, parameterized by time difference ε, is defined based on a set of observation function (or named as measurement function) $\mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right) = \{\mathbf{g} | \mathbf{g} : \mathbb{R}^{d_\gamma} \times T \rightarrow \mathbb{C}\}$ Brunton et al. (2022)

$$\mathcal{K}^\varepsilon \mathbf{g}\left(\gamma_t\right) = \mathbf{g}\left(\theta^\varepsilon\left(\gamma_t\right)\right) = \mathbf{g}\left(\gamma_{t+\varepsilon}\right), \ \forall \gamma_t \in \mathbb{R}^{d_\gamma} \times T. \tag{8}$$

Given with an appropriate space defined by $\mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right)$, we can linearize the dynamics of γ_t via Eq. (8). This idea has seen notable success in fluid dynamics Rowley et al. (2009), robotics Abraham & Murphey (2019), plasma physics Taylor et al. (2018), and neuroscience Brunton et al. (2016).

Different from existing machine-learning-based Koopman operator models that either are limited to autonomous dynamic systems (e.g., the case described by Eq. (8)) Takeishi et al. (2017); Azencot et al. (2020); Otto & Rowley (2019); Alford-Lago et al. (2022) or require a priori knowledge about the eigenvalue spectrum (e.g, the numbers of real and complex eigenvalues) of Koopman operator for non-autonomous dynamic systems Lusch et al. (2018), our approach is rooted in a more general perspective where we consider a time-dependent Koopman operator applicable to both non-autonomous and autonomous dynamic systems Macesic et al. (2018)

$$\mathcal{K}_t^{t+\varepsilon} \mathbf{g}\left(\gamma_t\right) = \mathbf{g}\left(\theta_t^{t+\varepsilon}\left(\gamma_t\right)\right) = \mathbf{g}\left(\gamma_{t+\varepsilon}\right), \ \forall t \leq t+\varepsilon \in T. \tag{9}$$

As shown in Eq. (9)), this Koopman operator governs a time-dependent linear evolution flow of $\mathbf{g}\left(\gamma_t\right)$ in a space defined by $\mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right)$

$$\partial_t \mathbf{g}\left(\gamma_t\right) = \lim_{\varepsilon \rightarrow 0} \frac{\mathcal{K}_t^{t+\varepsilon} \mathbf{g}\left(\gamma_t\right) - \mathbf{g}\left(\gamma_t\right)}{\varepsilon}. \tag{10}$$

In mathematics, the adjoint of the Koopman operator defined by Eq. (9) is the Perron-Frobenius operator of dynamic systems Lasota & Mackey (1985) while the adjoint of the associated Lie operator (see **Appendix A**) is the Liouville operator of Hamiltonian dynamics Gaspard et al. (1995); Gaspard (2005). These properties relate our approach with well-known theories about linear representation of dynamic systems in statistical physics and quantum mechanics Lasota & Mackey (1998).

Computational approximation of Koopman operator. After formalizing the time-dependent Koopman operator, we suggest an efficient computation approach to represent it.

Inspired by the Hankel-DMD Arbabi & Mezic (2017), sHankel-DMD Črnjarić-Žic et al. (2020), and HAVOK Brunton et al. (2017) algorithms, we consider the Krylov sequence Saad (2011) of the observable defined by a unit time step $\varepsilon \in [0, \infty]$

$$\mathcal{R}_n = \left[\mathbf{g}\left(\gamma_0\right), \mathbf{g}\left(\gamma_\varepsilon\right), \mathbf{g}\left(\gamma_{2\varepsilon}\right), \ldots, \mathbf{g}\left(\gamma_{n\varepsilon}\right)\right], \tag{11}$$

$$= \left[\mathbf{g}\left(\gamma_0\right), \mathcal{K}_0^\varepsilon \mathbf{g}\left(\gamma_0\right), \mathcal{K}_\varepsilon^{2\varepsilon} \mathcal{K}_0^\varepsilon \mathbf{g}\left(\gamma_0\right), \ldots, \mathcal{K}_{(n-1)\varepsilon}^{n\varepsilon} \mathcal{K}_{(n-2)\varepsilon}^{(n-1)\varepsilon} \cdots \mathcal{K}_0^\varepsilon \mathbf{g}\left(\gamma_0\right)\right], \tag{12}$$

which can be seen in the Krylov subspace method for computing matrix eigenvalues Saad (2011). Such a sequence can be efficiently sampled by a Hankel matrix representation of the system

$$\mathcal{H}_{m \times n} = \begin{bmatrix} \mathbf{g}\left(\gamma_0\right) & \mathbf{g}\left(\gamma_\varepsilon\right) & \cdots & \mathbf{g}\left(\gamma_{n\varepsilon}\right) \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{g}\left(\gamma_{(m-1)\varepsilon}\right) & \mathbf{g}\left(\gamma_{m\varepsilon}\right) & \cdots & \mathbf{g}\left(\gamma_{(m+n-1)\varepsilon}\right) \end{bmatrix}, \tag{13}$$

whose dimension of delay-embedding is $m \in \mathbb{N}^+$. The columns of $\mathcal{H}_{m \times n}$ approximate functions in the Krylov subspace. Our motivation to consider the Krylov subspace lies in that it has opportunities to span an invariant subspace, $\mathbb{K} \subset \mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right)$, of the Koopman operator

$$\mathbb{K} = \operatorname{span}\left(\mathcal{R}_n\right) \simeq \operatorname{span}\left(\mathcal{H}_{(m,n)}\right) \tag{14}$$

if $n \geq \dim(\mathbb{K}) - 1$ (notion $\dim(\cdot)$ measures the dimensionality). To see these possibilities, we consider the Galerkin projection of the original Koopman operator to $\mathbb{K}$, denoted by $\widehat{\mathcal{K}}_t^{t+\varepsilon} : \mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right) \to \mathbb{K}$ for any $t \in T$. For any function $\mathbf{h}(\cdot) \in \mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right)$, we have

$$\langle \widehat{\mathcal{K}}_t^{t+\varepsilon} \mathbf{h}(\gamma_t), \mathbf{g}(\gamma_{i\varepsilon}) \rangle = \langle \mathcal{K}_t^{t+\varepsilon} \mathbf{h}(\gamma_t), \mathbf{g}(\gamma_{i\varepsilon}) \rangle, \ \forall i = 0, \ldots, m, \tag{15}$$

where $\langle \cdot, \cdot \rangle$ is the inner product. According to Korda & Mezić (2018); Li & Jiang (2022), the Koopman operator restricted to $\mathbb{K}$ approximates the original Koopman operator

$$\lim_{m \to \infty} \int_{\mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right)} \| \widehat{\mathcal{K}}_t^{t+\varepsilon} \mathbf{h}(\gamma_t) - \mathcal{K}_t^{t+\varepsilon} \mathbf{h}(\gamma_t) \|_F \mathsf{d}\mu = 0, \ \forall \mathbf{h}(\cdot) \in \mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right) \tag{16}$$

if the original Koopman operator is bounded and $\mathbb{K}$ happens to be its invariant subspace (or simply $n \to \infty$). Notion μ denotes a measure on $\mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right)$ and $\| \cdot \|_F$ is the Frobenius norm. Therefore, we are expected to approximate the original Koopman operator via a restricted one such that

$$\mathcal{H}_{m \times n}(k+1) = \widehat{\mathcal{K}}_{k\varepsilon}^{(k+1)\varepsilon} \mathcal{H}_{m \times n}(k), \ \forall k = 1, \ldots, n, \tag{17}$$

where $\mathcal{H}_{m \times n}(k)$ denotes the k-th column of $\mathcal{H}_{m \times n}$.

However, the time-dependent property of $\widehat{\mathcal{K}}_t^{t+\varepsilon}$ essentially requires an online optimization if we learn it by a neural network. The expensive online optimization is not favorable for solving PDEs, persuading us to consider an alternative solution. Inspired by ergodic theory Arbabi & Mezic (2017); Cornfeld et al. (2012), we assume that the dynamic system of γ_t is ergodic (i.e., γ_t eventually visits all possible states in $\mathbb{R}^{d_\gamma}$ as $t \to \infty$, thus the proportion of time that γ_t spends on a particular state equals the probability of this state). This assumption makes the time-averaging approximate the true expectation of of an observable as the time approaches to infinity. Under this assumption, we can define an expectation of the Koopman operator controlled by time difference ε

$$\overline{\mathcal{K}}_\varepsilon = \lim_{t \to \infty} \frac{1}{t} \int_{[0,t)} \mathbf{g}(\gamma_\tau)^{-1} \mathbf{g}(\gamma_{\tau+\varepsilon}) \mathsf{d}\tau \simeq \operatorname*{argmin}_{P \in \mathbb{R}} \sum_{k=1}^{n} \| \mathcal{H}_{m \times n}(k+1) - P \mathcal{H}_{m \times n}(k) \|_F. \tag{18}$$

Given a fixed ε, the Koopman operator $\overline{\mathcal{K}}_\varepsilon : \mathcal{G}\left(\mathbb{R}^{d_\gamma} \times T\right) \to \mathbb{K}$ in Eq. (18) can be understood as the time-average of $\widehat{\mathcal{K}}_t^{t+\varepsilon}$ at different t. A neural network representation of $\overline{\mathcal{K}}_\varepsilon$ only requires offline optimization and, therefore, is computationally favorable for solving PDEs.

Neural network architectures of Koopman neural operator. In this section, we realize our framework defined above on neural network architectures, in combination with the objective of solving PDEs. Our neural network model, named as Koopman neural operator (KNO), not only implements the mathematical framework defined above but also utilizes various numerical techniques to optimize its computational efficiency. Below, we introduce the details of architecture designs.

Part 1: Observation. Given an input $\phi_t = \phi(D \times \{t\})$ of the PDE in Eqs. (1-3), we first transform it as $\mathbf{g}(\widehat{\gamma}_t)$ in space $\mathcal{G}\left(\mathbb{R}^{d_{\widehat{\gamma}_t}} \times T\right)$ by an encoder (a single non-linear layer with $\tanh(\cdot)$ activation function) that represent observation function $\mathbf{g}(\cdot)$. Please see **Fig. 1**.

Part 2: Fourier transform. Although our approach focuses on the Koopman operator, it is not necessary to give up the computation acceleration of iterative update strategy in Eq. (4) realized by Fourier transform (see Fourier neural operator Li et al. (2020a) for detailed explanations). In our approach, we also apply the Fourier transform to map $\mathbf{g}(\widehat{\gamma}_t)$ as $\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t) = \mathcal{F} \circ \mathbf{g}(\widehat{\gamma}_t)$ and parameterize the subsequent parts of our network in the Fourier space. Similar to Li et al. (2020a), $\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t)$ is computed by fast Fourier transform, where we truncate the Fourier series at ω, a maximum number of frequency modes. Although this setting is acceptable in computational implementation, it does implies the loss of high-frequency information. To complement the lost information, we design an extra part (**Part 5**) in our model to extract high-frequency information of $\mathbf{g}(\widehat{\gamma}_t)$. See **Fig. 1** for details.

Part 3: Hankel representation and offline Koopman operator. Given $\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t)$ for every $t \in \varepsilon\mathbb{N}^+$, we set a dimension of delay-embedding, $m \in \mathbb{N}$, to define a Hankel matrix $\widehat{\mathcal{H}}_{m \times n}$ of $\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t)$ (note that n equals the number of accessible samples). To ensure that the space $\widehat{\mathbb{K}}$ spanned by the Hankel matrix approximates the invariant sub-space of target Koopman

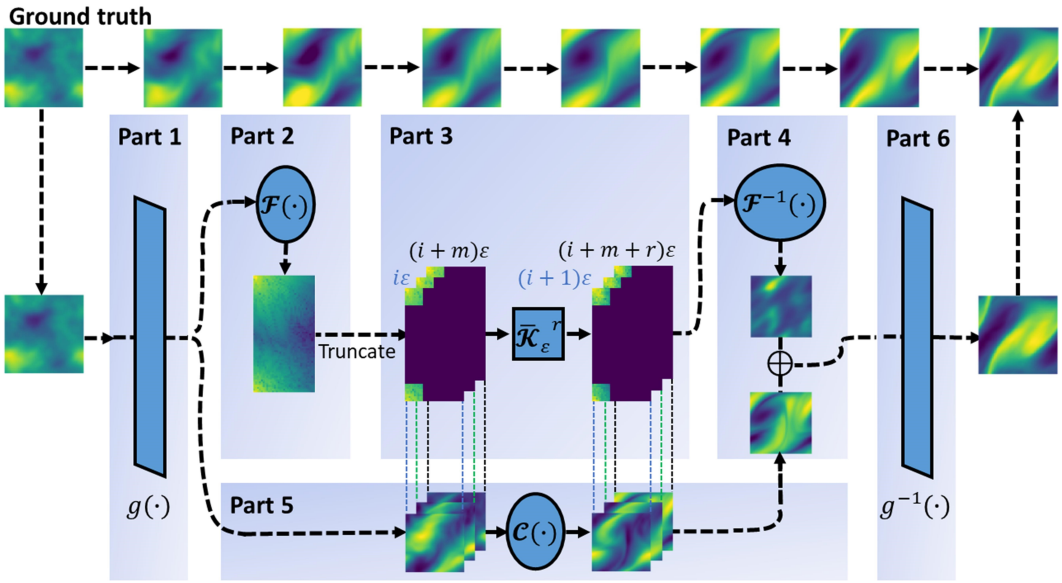

Figure 1: Conceptual illustrations of neural network architectures of KNO. Parameter $r = \frac{t'-t}{\varepsilon}$ is prediction length. Note that the layout of every part is slightly reorganized to offer a clear version.

operator, we train a $o \times o$ linear layer to learn $\overline{\mathcal{K}}_\varepsilon : \mathcal{G}\left(\mathbb{R}^{d_{\widehat{\gamma}}} \times T\right) \to \widehat{\mathbb{K}}$ following Eqs. (18-19). Based on the learned $\overline{\mathcal{K}}_\varepsilon$, we can predict the future state of the latest observable $\mathbf{g}_{\mathcal{F}}\left(\widehat{\gamma}_{(m+n-1)\varepsilon}\right)$ as $\mathbf{g}_{\mathcal{F}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right) = \left[\overline{\mathcal{K}}_\varepsilon \widehat{\mathcal{H}}_{m\times n}\left(n\right)\right]^{\mathsf{T}}(m)$, where notion T denotes the transpose of a matrix. Please see **Fig. 1** for illustrations of **Part 3**.

Part 4: Inverse Fourier transform. Given each predicted state $\mathbf{g}_{\mathcal{F}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$ in **Part 3**, we transform it from the Fourier space to $\mathcal{G}\left(\mathbb{R}^{d_{\widehat{\gamma}_t}} \times T\right)$ by an inverse Fourier transform, i.e, $\mathbf{g}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right) = \mathcal{F}^{-1} \circ \mathbf{g}_{\mathcal{F}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$. See **Fig. 1** for instances.

Part 5: High-frequency information complement. According to the Fourier analysis implemented on feature maps, convolutional layers can amplify high-frequency components Park & Kim (2022). Therefore, we train a convolutional layer $\mathcal{C}$ on the outputs of **Part 1** to extract their high-frequency information, denoted by $\mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_t\right)$, as a complement of **Parts 2-4**. Meanwhile, the convolutional layer also implements an independent forward prediction parallel to **Parts 2-4**, i.e, $\left[\mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(i+1)\varepsilon}\right), \ldots, \mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(i+m+1)\varepsilon}\right)\right]^{\mathsf{T}} = \mathcal{C}\left[\mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{i\varepsilon}\right), \ldots, \mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(i+m)\varepsilon}\right)\right]^{\mathsf{T}}$ for any $i = 1, \ldots, n$. See **Fig. 1** for illustrations.

Part 6: Inverse observation. Given two future states, $\mathbf{g}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$ and $\mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$, of the latest observable independently predicted by **Parts 2-4** and **Part 5**, we unify them by $\mathbf{g}_{\mathcal{U}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right) = \mathbf{g}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right) + \mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$. We train an non-linear decoder (a single non-linear layer with $\tanh\left(\cdot\right)$ activation function) to represent the inverse of observation function $\mathbf{g}^{-1}\left(\cdot\right) \simeq \mathbf{g}_{\mathcal{U}}^{-1}\left(\cdot\right)$ and transform $\mathbf{g}_{\mathcal{U}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$ to $\widehat{\gamma}_{(m+n)\varepsilon}$, the target state of equation solution in space $\mathbb{R}^{d_{\widehat{\gamma}_t}}$. Please see **Fig. 1**.

Based on **Parts 1-6**, we have developed a new iterative update strategy different from Eq. (4). For any $t' > t \in \varepsilon\mathbb{N}$, we have

$$\widehat{\gamma}_{t'} = \left[\mathbf{g}^{-1}\Big(\underbrace{\mathcal{F}^{-1} \circ \overline{\mathcal{K}}_\varepsilon^{\,t'-t} \circ \mathcal{F} \circ \mathbf{g}\left(\widehat{\gamma}_{[t-m\varepsilon,t]}\right)}_{\textbf{Parts 1-4}} + \underbrace{\mathcal{C}^{t'-t} \circ \mathbf{g}\left(\widehat{\gamma}_{[t-m\varepsilon,t]}\right)}_{\textbf{Part 1 and part 5}}\Big)\right]^{\mathsf{T}}(m)\,, \tag{19}$$

where $\widehat{\gamma}_{[t-m\varepsilon,t]}$ is a vector $[\widehat{\gamma}_{t-m\varepsilon}, \ldots, \widehat{\gamma}_t]$ defined by $m \in \mathbb{N}$, the dimension of delay-embedding. In **Fig. 1**, we illustrate the one-unit architecture of KNO. Similar to Fourier neural operator Li et al.

(2020a), a x-unit KNO architecture can be produced by cascading the copy of **Parts 2-5** x times. Based on Eq. (19), the loss function of KNO is defined as

$$\mathcal{L} = \alpha \|\widehat{\gamma}_{t'} - \gamma_{t'}\|_F + \beta \sum_{i=0}^{m} \|\mathbf{g}^{-1} \circ \mathbf{g}\left(\widehat{\gamma}_{t-im\varepsilon}\right) - \gamma_{t-im\varepsilon}\|_F, \tag{20}$$

where $\alpha, \beta \in (0, \infty)$ control the weights of prediction and reconstruction processes in loss function, respectively. Below, we test our KNO model in experiments.

3 EXPERIMENTS

Details of implemented PDEs. We implement our experiments on the 1-dimensional Bateman–Burgers equation Benton & Platzman (1972) and the 2-dimensional Navier-Stokes equation Wang (1991). Please see **Appendix B** for mathematical definitions of these PDEs. The data of these PDEs is provided by Li et al. (2022) to ensure reproducibility.

Details of experiment designs. We conduct five experiments to validate our model:

- **Mesh-independent experiment.** As suggested in previous works Lu et al. (2019); Bhattacharya et al. (2020); Nelsen & Stuart (2021); Li et al. (2020b;a); Kovachki et al. (2021); Li et al. (2022), neural operator models are expected to be mesh-independent becasue they learn the solution operator of an entire PDE family. Therefore, we design an experiment to validate the mesh-independent property of KNO.
- **Long-term prediction experiment.** To validate the long-term prediction capacity of KNO, we design prediction tasks where $t' - t$ varies across different values and verify if KNO robustly maintains accuracy as $t' - t$ in Eq. (19) enlarges.
- **Zero-shot prediction experiment (discretization granularity).** Following the idea Li et al. (2020a), we test the generalization ability of KNO by testing it on untrained discretization granularity (e.g., in a way similar to super-resolution Li et al. (2020a)).
- **Zero-shot prediction experiment (prediction length).** Apart from the generalization on untrained discretization granularity, we also validate the generalization capacity of KNO on untrained prediction length (e.g., train a KNO on a time difference $t' - t = r\varepsilon$ and test it on another time difference $r'\varepsilon > r\varepsilon$ in Eq. (19)).
- **Ablation experiment.** To demonstrate the significance of the learned Koopman operator in KNO designs, we implement an ablation experiment in **Appendix C**.

All experiments are implemented on a single Nvidia Titan V GPU with 12GB memory.

Details of implemented models. Besides KNO, we implement the following models for comparison: Fourier neural operator (FNO) Li et al. (2020a), U-shaped neural operator (UNO) Rahman et al. (2022), convolutional LSTM (ConvLSTM) Wang et al. (2020), and U-Net Ronneberger et al. (2015). Other common neural network models, such as the vanilla residual neural network (ResNet) He et al. (2016) and deep hidden physics model (DHPM) Raissi (2018), are shown as less efficient on complex fluid systems Wang et al. (2020) and, therefore, are not considered in our research. Each model is trained by its default optimizer (e.g., KNO and FNO are trained by the Adam optimizer).

Mesh-independent experiment. Our mesh-independent experiment is implemented on the data of 1-dimensional Bateman–Burgers equation generated under different discretization conditions (i.e., spatial resolution of meshes). The data with highest resolution is generated following the Gaussian initialization introduced in Li et al. (2020a). The data with lower resolution are directly downsampled from the data with higher resolution. We choose FNO as a baseline for comparison. As for other baseline models less efficient than FNO in mesh-independent experiment, such as graph neural operator (GNO) Li et al. (2020b) and multipole graph neural operator (MGNO) Li et al. (2020c) (see results reported by Li et al. (2020a)), we no longer discuss them for convenience. We implement multiple versions of KNO and FNO with different hyper-parameters (e.g., operator size o, frequency mode number f, the iteration number of the Koopman operator $r = \frac{t'-t}{\varepsilon}$ in KNO, and the width of FNO w). Under every condition, these models are trained on 1000 samples and conduct 1-second

forward prediction (i.e., $t' - t = 1$) on 200 samples for performance evaluation. During training, the batch size is fixed as 64. The learning rate is initialized at 0.001 and is halved every 100 epochs. The weights of prediction and reconstruction in the loss function are defined as $\alpha = 5$ and $\beta = 0.5$, respectively. As illustrated in **Fig. 2(a)**, KNO achieves almost constant prediction error in different discretization settings. Compared with FNO models, the prediction error of KNO models maintains more stable across different conditions. Notably, a one-unit KNO architecture only requires about 5×10^3 parameters to outperform a one-unit FNO architecture with more than 2×10^7 parameters, suggesting the potential of KNO to break the trade-off of accuracy and efficiency.

Long-term prediction experiment. Our long-term prediction experiment is implemented on the 2-dimensional Navier-Stokes equation, where we consider the low viscosity cases, i.e., $\nu \in \{10^{-3}, 10^{-4}\}$. The spatial discretization of all systems is implemented with 2^{12} grids. In the case where $\nu = 10^{-3}$, we consider a 40-time-interval prediction task whose samples are the state sequences of vorticity fields in space $\gamma\left((0,1)^2 \times [0,10)\right)$ and prediction targets are the future state sequences in space $\gamma\left((0,1)^2 \times (10,50]\right)$. Models are trained and tested on 1000 and 200 samples, respectively. In a more difficult case with $\nu = 10^{-4}$, we consider a 10-time-interval prediction task from $\gamma\left((0,1)^2 \times [0,10)\right)$ to $\gamma\left((0,1)^2 \times (10,20]\right)$. Models are trained and tested on 8000 and 200 samples, respectively. **Table 1** reports our full results, **Fig. 2(b)** visualizes the performance of experiment with $\nu = 10^{-3}$, and **Fig. 2(d)** visualizes instances of predicted results with $\nu = 10^{-4}$. These results suggest that KNO is optimal in balancing between accuracy and efficiency.

Zero-shot prediction experiment concerning discretization granularity. As suggested by Li et al. (2020a), a mesh-independent neural operator model can be trained only on the data with lower resolution to predict the data with higher resolution (referred to as zero-shot super-resolution Li et al. (2020a)). In our research, we validate this property by implementing a zero-shot experiment. Same as our long-term prediction experiment, there are two data sets of the 2-dimensional Navier-Stokes equation distinguished according to the viscosity $\nu \in \{10^{-3}, 10^{-4}\}$ and one data set of the 1-dimensional Bateman–Burgers equation. The settings and objective of prediction tasks on each

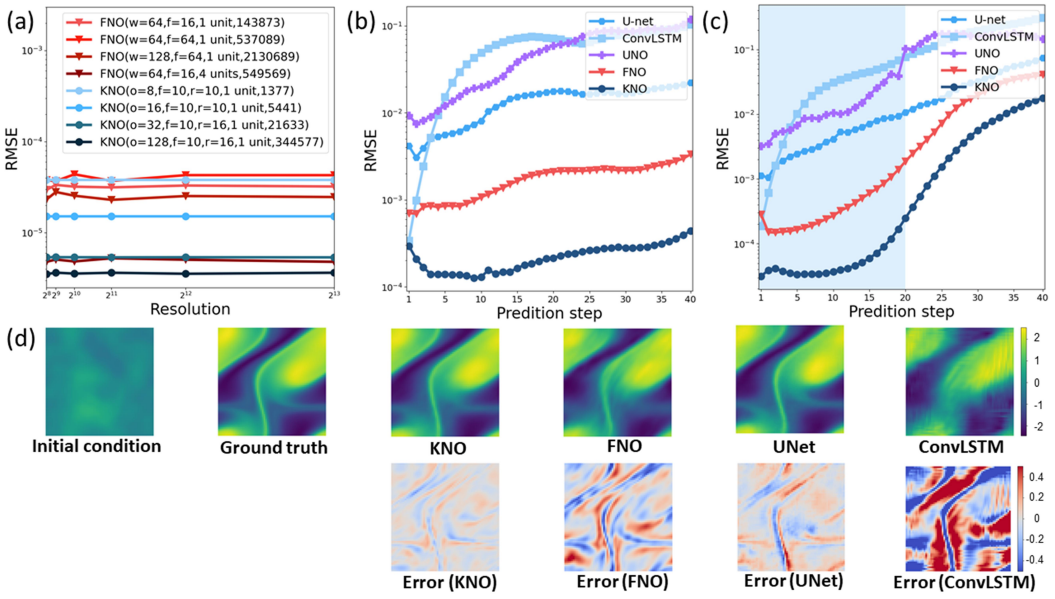

Figure 2: Experiment results of KNO. (a) Results of the mesh-independent experiment. (b) Results of the long-term prediction experiment on the 2-dimensional Navier-Stokes equation with a viscosity coefficient $\nu = 10^{-3}$. (c) Zero-shot prediction experiment concerning prediction length. (d) Visualization of prediction results on the 2-dimensional Navier-Stokes equation with $\nu = 10^{-4}$.

Models	Parameters	$\nu = 10^{-3}$	$\nu = 10^{-4}$
FNO (default settings, one-unit)	233897	1.78×10^{-3}	2.61×10^{-2}
FNO (default settings, two-unit)	464717	$\mathbf{2.12\times10^{-4}}$	$\underline{7.72\times10^{-3}}$
ConvLSTM (default settings)	10001	6.05×10^{-2}	1.69×10^{-1}
U-Net (default settings)	24950491	1.77×10^{-3}	7.90×10^{-2}
KNO ($o = 24$, $f = 10$, $r = 12$, one-unit)	80850	3.00×10^{-4}	9.60×10^{-3}
KNO ($o = 32$, $f = 10$, $r = 12$, one-unit)	206538	$\underline{2.37\times10^{-4}}$	$\mathbf{6.66\times10^{-3}}$

Table 1: Full results of long-term prediction experiment. Accuracy is measured by RMSE. The best model is marked in bold while the second best one is marked by the underline.

Grid number	2^8	2^9	2^{10}
Setting 1	3.820924×10^{-5}	3.820929×10^{-5}	3.820926×10^{-5}
Setting 2	1.521201×10^{-5}	1.521201×10^{-5}	1.521200×10^{-5}
Setting 3	5.403627×10^{-6}	5.403630×10^{-6}	5.403628×10^{-6}
Setting 4	3.553927×10^{-6}	3.553980×10^{-6}	3.553943×10^{-6}
Grid number	2^{11}	2^{12}	2^{13}
Setting 1	3.820927×10^{-5}	3.820931×10^{-5}	3.820932×10^{-5}
Setting 2	1.521202×10^{-5}	1.521202×10^{-5}	1.521200×10^{-5}
Setting 3	5.403630×10^{-6}	5.403632×10^{-6}	5.403615×10^{-6}
Setting 4	3.553957×10^{-6}	3.553942×10^{-6}	3.553938×10^{-6}

Table 2: Zero-shot experiment (discretization granularity) on the 1-dimensional Bateman–Burgers equation. KNO settings $(o, f, r) \in \{(8, 10, 10), (16, 10, 10), (32, 10, 16), (128, 10, 16)\}$ are referred to as Settings 1-4, respectively. Accuracy is measured by RMSE.

	$\nu = 10^{-3}$		$\nu = 10^{-4}$	
	32	64	32	64
FNO (default settings, one-unit)	1.78×10^{-3}	1.79×10^{-3}	1.41×10^{-2}	1.54×10^{-2}
KNO ($o = 24$, $f = 10$, $r = 12$, one-unit)	$\mathbf{3.22\times10^{-4}}$	$\mathbf{3.31\times10^{-4}}$	$\mathbf{9.63\times10^{-3}}$	$\mathbf{9.60\times10^{-3}}$

Table 3: Zero-shot experiment (discretization granularity) on the 2-dimensional Navier-Stokes equation. Accuracy is measured by RMSE.

data set is exactly same as the long-term prediction experiment expect that all models are trained on a lower resolution and tested on a higher resolution. Specifically, models are trained on 2^8 grids and tested on $\{2^9, \ldots, 2^{13}\}$ grids for the 1-dimensional Bateman–Burgers equation while they are trained on 2^{10} grids and tested on 2^{12} grids for the 2-dimensional Navier-Stokes equation. **Tables 2-3** suggest the optimality of KNO in this task.

Zero-shot prediction experiment concerning prediction length. Because KNO is proposed to model the intricate dynamics of PDE solutions, it is natural to wonder if KNO can achieve optimal performance in zero-shot prediction with an untrained prediction length, i.e., train the model on a prediction length $t' - t = r\varepsilon$ and test it on a larger prediction length $r'\varepsilon > r\varepsilon$. This objective can be simply realized by increasing $r = \frac{t'-t}{\varepsilon}$, the iteration number of the Koopman operator, during prediction. To validate the optimality of KNO in such a task, we implement an experiment on the 2-dimensional Navier-Stokes equation with a viscosity coefficient $\nu = 10^{-3}$ and 2^{12} grids. All models are trained on $\gamma\left((0,1)^2 \times [0,10)\right)$ to predict the label in $\gamma\left((0,1)^2 \times (10,30]\right)$ (supervised learning). Apart from the trained supervised learning task, models are also required to predict $\gamma\left((0,1)^2 \times (30,50]\right)$, an untrained target. As illustrated by **Fig. 2(c)**, KNO outperforms other models in both supervised prediction and zero-shot prediction tasks.

4 Conclusion

In summary, we develop the Koopman neural operator, a neural operator with the potential to break the tradeoff between accuracy and efficiency in solving PDEs. See **Appendix D** for its basic code.

REFERENCES

Ian Abraham and Todd D Murphey. Active learning of dynamics for data-driven control using koopman operators. *IEEE Transactions on Robotics*, 35(5):1071–1083, 2019.

Ralph Abraham, Jerrold E Marsden, and Tudor Ratiu. *Manifolds, tensor analysis, and applications*, volume 75. Springer Science & Business Media, 2012.

Daniel J Alford-Lago, Christopher W Curtis, Alexander T Ihler, and Opal Issan. Deep learning enhanced dynamic mode decomposition. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(3):033116, 2022.

Farzan Aminian, E Dante Suarez, Mehran Aminian, and Daniel T Walz. Forecasting economic data with neural networks. *Computational Economics*, 28(1):71–88, 2006.

Hassan Arbabi and Igor Mezic. Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the koopman operator. *SIAM Journal on Applied Dynamical Systems*, 16 (4):2096–2126, 2017.

Omri Azencot, N Benjamin Erichson, Vanessa Lin, and Michael Mahoney. Forecasting sequential data using consistent koopman autoencoders. In *International Conference on Machine Learning*, pp. 475–485. PMLR, 2020.

Leah Bar and Nir Sochen. Unsupervised deep learning algorithm for pde-based forward and inverse problems. *arXiv preprint arXiv:1904.05417*, 2019.

Edward R Benton and George W Platzman. A table of solutions of the one-dimensional burgers equation. *Quarterly of Applied Mathematics*, 30(2):195–212, 1972.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

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

Bingni W Brunton, Lise A Johnson, Jeffrey G Ojemann, and J Nathan Kutz. Extracting spatial–temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. *Journal of neuroscience methods*, 258:1–15, 2016.

Steven L Brunton, Bingni W Brunton, Joshua L Proctor, Eurika Kaiser, and J Nathan Kutz. Chaos as an intermittently forced linear system. *Nature communications*, 8(1):1–9, 2017.

Steven L Brunton, Marko Budisic, Eurika Kaiser, and J Nathan Kutz. Modern koopman theory for dynamical systems. *SIAM Review*, 64(2):229–340, 2022.

Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in neural information processing systems*, 33:17766–17778, 2020.

Carmen Chicone and Yuri Latushkin. *Evolution semigroups in dynamical systems and differential equations*. Number 70. American Mathematical Soc., 1999.

Isaac P Cornfeld, Sergei Vasilevich Fomin, and Yakov Grigor'evĭc Sinai. *Ergodic theory*, volume 245. Springer Science & Business Media, 2012.

Nelida Črnjarić-Žic, Senka Maćešić, and Igor Mezić. Koopman operator spectrum for random dynamical systems. *Journal of Nonlinear Science*, 30(5):2007–2056, 2020.

Lokenath Debnath and Lokenath Debnath. *Nonlinear partial differential equations for scientists and engineers*. Springer, 2005.

Pierre Gaspard. Chaos, scattering and statistical mechanics. *Chaos*, 2005.

Pierre Gaspard, Grégoire Nicolis, Astero Provata, and S13835391995PhRvE Tasaki. Spectral signature of the pitchfork bifurcation: Liouville equation approach. *Physical Review E*, 51(1):74, 1995.

Mark S Gockenbach. *Partial differential equations: analytical and numerical methods*, volume 122. Siam, 2005.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Efficient token mixing for transformers via adaptive fourier neural operators. In *International Conference on Learning Representations*, 2021.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 481–490, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

Milan Korda and Igor Mezić. On convergence of extended dynamic mode decomposition to the koopman operator. *Journal of Nonlinear Science*, 28(2):687–710, 2018.

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

Andrzej Lasota and Michael C Mackey. *Probabilistic properties of deterministic systems*. Cambridge university press, 1985.

Andrzej Lasota and Michael C Mackey. *Chaos, fractals, and noise: stochastic aspects of dynamics*, volume 97. Springer Science & Business Media, 1998.

Mengnan Li and Lijian Jiang. Reduced-order modeling for koopman operators of nonautonomous dynamic systems in multiscale media. *arXiv preprint arXiv:2204.13180*, 2022.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

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

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

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022.

Konstantin Lipnikov, Gianmarco Manzini, and Mikhail Shashkov. Mimetic finite difference method. *Journal of Computational Physics*, 257:1163–1227, 2014.

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

Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):1–10, 2018.

Senka Macesic, Nelida Crnjaric-Zic, and Igor Mezic. Koopman operator family spectrum for nonautonomous systems. *SIAM Journal on Applied Dynamical Systems*, 17(4):2478–2515, 2018.

Robert MM Mattheij, Sjoerd W Rienstra, and JHM Ten Thije Boonkkamp. *Partial differential equations: modeling, analysis, computation*. SIAM, 2005.

Nicholas H Nelsen and Andrew M Stuart. The random feature model for input-output maps between banach spaces. *SIAM Journal on Scientific Computing*, 43(5):A3212–A3243, 2021.

Samuel E Otto and Clarence W Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

Shaowu Pan and Karthik Duraisamy. Physics-informed probabilistic learning of linear embeddings of nonlinear dynamics with guaranteed stability. *SIAM Journal on Applied Dynamical Systems*, 19(1):480–509, 2020.

Namuk Park and Songkuk Kim. How do vision transformers work? *arXiv preprint arXiv:2202.06709*, 2022.

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

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

Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

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

Junuthula Narasimha Reddy. *Introduction to the finite element method*. McGraw-Hill Education, 2019.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Clarence W Rowley, Igor Mezić, Shervin Bagheri, Philipp Schlatter, and Dan S Henningson. Spectral analysis of nonlinear flows. *Journal of fluid mechanics*, 641:115–127, 2009.

Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

Eitan Tadmor. A review of numerical methods for nonlinear partial differential equations. *Bulletin of the American Mathematical Society*, 49(4):507–554, 2012.

Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. *Advances in Neural Information Processing Systems*, 30, 2017.

Hiroki Tanabe. *Functional analytic methods for partial differential equations*. CRC Press, 2017.

Roy Taylor, J Nathan Kutz, Kyle Morgan, and Brian A Nelson. Dynamic mode decomposition for plasma diagnostics and validation. *Review of Scientific Instruments*, 89(5):053501, 2018.

CY Wang. Exact solutions of the steady-state navier-stokes equations. *Annual Review of Fluid Mechanics*, 23(1):159–177, 1991.

Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1457–1466, 2020.

Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

A THE ASSOCIATED LIE OPERATOR OF THE KOOPMAN OPERATOR

Besides the linear system in Eq. (9)) in the main text, we can also consider the generator operator of such a Koopman operator, which is referred to as the Lie operator because it is the Lie derivative of $\mathbf{g}(\cdot)$ along the vector field $\gamma(\cdot)$ Koopman (1931); Abraham et al. (2012); Chicone & Latushkin (1999)

$$\mathcal{L}_t \mathbf{g} = \lim_{t+\varepsilon \to t} \frac{\mathcal{K}_t^{t+\varepsilon} \mathbf{g}(\gamma_t) - \mathbf{g}(\gamma_t)}{t+\varepsilon - t}. \tag{21}$$

The generator operator also defines a linear system of $\mathbf{g}(\gamma_t)$ because

$$\partial_t \mathbf{g}(\gamma_t) = \lim_{\varepsilon \to 0} \frac{\mathbf{g}(\gamma_{t+\varepsilon}) - \mathbf{g}(\gamma_t)}{\varepsilon} = \lim_{t+\varepsilon \to t} \frac{\mathcal{K}_t^{t+\varepsilon} \mathbf{g}(\gamma_t) - \mathbf{g}(\gamma_t)}{\varepsilon} = \mathcal{L}_t \mathbf{g}(\gamma_t). \tag{22}$$

Although our work primarily focuses on the Koopman operator, one can also model the Lie operator in application.

B MATHEMATICAL DETAILS OF IMPLEMENTED PDES

In our experiments, we consider the 1-dimensional Bateman–Burgers equation Benton & Platzman (1972) and the 2-dimensional Navier-Stokes equation Wang (1991). Below, we introduce their mathematical definitions.

Bateman–Burgers equation. The 1-dimensional Bateman–Burgers equation is

$$\partial_t \gamma(x_t) + \partial_x \left(\frac{\gamma^2(x_t)}{2} \right) = \nu \partial_{xx} \gamma(x_t), \ x_t \in (0,1) \times (0,1], \ x_t \in (0,1) \times (0,1], \tag{23}$$

$$\gamma(x_0) = \gamma_I, \ x_0 \in (0,1) \times \{0\}, \tag{24}$$

where γ_I is a periodic initial condition $\gamma_I \in L^2_{\text{periodic}}[(0,1); \mathbb{R}]$, parameter $\nu \in (0, \infty)$ is the viscosity coefficient, which is set as $\nu = 100$ in our experiments. The data set of Eqs. (23-24) is provided by Li et al. (2020a). Please see Li et al. (2020a) for details of data generation.

Navier-Stokes equation. Mathematically, the incompressible 2-dimensional Navier-Stokes equation in a vorticity form is defined as

$$\partial_t \gamma(x_t) + \chi(x_t) \nabla \gamma(x_t) = \nu \Delta \gamma(x_t) + \psi(x_t), \ x_t \in (0,1)^2 \times (0, \infty), \tag{25}$$

$$\nabla \chi(x_t) = 0, \ x_t \in (0,1)^2 \times (0, \infty), \tag{26}$$

$$\gamma(x_0) = \gamma_I, \ x_0 \in (0,1) \times \{0\}, \tag{27}$$

where $\gamma(\cdot)$ measures the vorticity, $\chi(\cdot)$ defines the velocity, $\psi(\cdot)$ denotes a time-independent forcing term. The viscosity coefficient is set as $\nu \in \{10^{-3}, 10^{-4}\}$. Similar to the situation of Bateman–Burgers equation, the data of Eqs. (25-27) is provided by Li et al. (2020a). Please see Li et al. (2020a) for details.

C ABLATION EXPERIMENT RESULTS

Our objective in the ablation experiment is to compare between the performance of KNO models with ($\beta > 0$) and without ($\beta = 0$) the reconstruction term of loss function in Eq. (20). When $\beta > 0$, the prediction is undertaken by a learned Koopman operator (a linear layer of $o \times o$) associated with a convolutional layer in **Parts 2-5** while encoder (**Part 1**) and decoder (**Part 6**) contribute to reconstruction. In such a case, the performance is mainly contributed by the Koopman operator because the convolutional layer only serves as the complement of high-frequency information (as suggested by Li et al. (2020a), a pure convolutional network only achieves poor performance in PDE solving). Once $\beta = 0$, encoder and decoder become to be trained for prediction as well, making the whole network a unified predictor (similar to FNO Li et al. (2020a)). The Koopman operator is validated as important if the prediction performance mainly realized by it ($\beta > 0$) is same as or better than the performance achieved by the whole network as a unified predictor ($\beta = 0$). The ablation experiment is implemented as a 1-second prediction task on the data of 1-dimensional Bateman–Burgers equation with 2^8 grids. As shown in **Table 2**, KNO models generally perform better when $\beta > 0$, suggesting the significance of the Koopman operator.

Operator size o	Mode number f	Iteration number r	α	β	MSE
8	64	10	5.0	0.5	$\mathbf{2.43 \times 10^{-5}}$
8	64	10	5.0	0	2.67×10^{-5}
16	10	16	5.0	0.5	$\mathbf{1.05 \times 10^{-5}}$
16	10	16	5.0	0	1.10×10^{-5}
32	10	16	5.0	0.5	$\mathbf{5.37 \times 10^{-6}}$
32	10	16	5.0	0	6.04×10^{-6}
32	64	16	5.0	0.5	$\mathbf{5.47 \times 10^{-6}}$
32	64	16	5.0	0	5.78×10^{-6}
128	10	16	5.0	0.5	$\mathbf{3.53 \times 10^{-6}}$
128	10	16	5.0	0	4.34×10^{-6}

Table 4: Results of ablation experiment.

D CODE

The basic implementation of the Koopman neural operator is demonstrated here. The full version of our code will be released once the double-blind review finishes.

```
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F


torch.manual_seed(0)

class encoder(nn.Module):
    def __init__(self, t_len, op_size):
        super(encoder, self).__init__()
        self.layer = nn.Linear(t_len, op_size)
    def forward(self, x):
        x = self.layer(x)
        x = torch.tanh(x)
        return x

class decoder(nn.Module):
    def __init__(self, t_len, op_size):
        super(decoder, self).__init__()
        self.layer = nn.Linear(op_size, t_len)
    def forward(self, x):
        x = torch.tanh(x)
        x = self.layer(x)
        return x

class Koopman_Operator1D(nn.Module):
    def __init__(self, op_size, modes_x = 16):
        super(Koopman_Operator1D, self).__init__()
        self.op_size = op_size
        self.scale = (1 / (op_size * op_size))
        self.modes_x = modes_x
        self.koopman_matrix = nn.Parameter(self.scale * torch.rand
            (op_size, op_size, self.modes_x, dtype=torch.cfloat))
    # Complex multiplication
    def time_marching(self, input, weights):
        # (batch, t, x), (t, t+1, x) -> (batch, t+1, x)
        return torch.einsum("btx,tfx->bfx", input, weights)
    def forward(self, x):
```

```
        batchsize = x.shape[0]
        # Fourier Transform
        x_ft = torch.fft.rfft(x)
        # Koopman Operator Time Marching
        out_ft = torch.zeros(x_ft.shape, dtype=torch.cfloat,
            device = x.device)
        out_ft[:, :, :self.modes_x] = self.time_marching(x_ft[:,
            :, :self.modes_x], self.koopman_matrix)
        #Inverse Fourier Transform
        x = torch.fft.irfft(out_ft, n=x.size(-1))
        return x

class KNO1d(nn.Module):
    def __init__(self, op_size, modes_x = 16, decompose = 4, t_len
        = 1):
        super(KNO1d, self).__init__()
        # Parameter
        self.op_size = op_size
        self.decompose = decompose
        # Layer Structure
        self.enc = encoder(t_len, op_size)
        self.dec = decoder(t_len, op_size)
        self.koopman_layer = Koopman_Operator1D(self.op_size,
            modes_x = modes_x)
        self.w0 = nn.Conv1d(op_size, op_size, 1)
    def forward(self, x):
        # Reconstruct
        x_reconstruct = self.enc(x)
        x_reconstruct = torch.tanh(x_reconstruct)
        x_reconstruct = self.dec(x_reconstruct)
        # Predict
        x = self.enc(x) # Encoder
        x = x.permute(0, 2, 1)
        x_w = x
        for i in range(self.decompose):
            x1 = self.koopman_layer(x) # Koopman Operator
            x = torch.tanh(x + x1)
            # x = x + x1
        x = self.w0(x_w) + x
        x = x.permute(0, 2, 1)
        x = self.dec(x) # Decoder
        return x, x_reconstruct

class Koopman_Operator2D(nn.Module):
    def __init__(self, op_size, modes):
        super(Koopman_Operator2D, self).__init__()
        self.op_size = op_size
        self.scale = (1 / (op_size * op_size))
        self.modes_x = modes
        self.modes_y = modes
        self.koopman_matrix = nn.Parameter(self.scale * torch.rand
            (op_size, op_size, self.modes_x, self.modes_y, dtype=
            torch.cfloat))

    # Complex multiplication
    def time_marching(self, input, weights):
        # (batch, t, x,y ), (t, t+1, x,y) -> (batch, t+1, x,y)
        return torch.einsum("btxy,tfxy->bfxy", input, weights)
```

```
    def forward(self, x):
        batchsize = x.shape[0]
        # Fourier Transform
        x_ft = torch.fft.rfft2(x)
        # Koopman Operator Time Marching
        out_ft = torch.zeros(x_ft.shape, dtype=torch.cfloat,
            device = x.device)
        out_ft[:, :, :self.modes_x, :self.modes_y] = self.
            time_marching(x_ft[:, :, :self.modes_x, :self.modes_y
            ], self.koopman_matrix)
        out_ft[:, :, -self.modes_x:, :self.modes_y] = self.
            time_marching(x_ft[:, :, -self.modes_x:, :self.modes_y
            ], self.koopman_matrix)
        #Inverse Fourier Transform
        x = torch.fft.irfft2(out_ft, s=(x.size(-2), x.size(-1)))
        return x

class KNO2d(nn.Module):
    def __init__(self, op_size, modes = 10, decompose = 6, t_len =
        10):
        super(KNO2d, self).__init__()
        # Parameter
        self.op_size = op_size
        self.decompose = decompose
        self.modes = modes
        # Layer Structure
        self.enc = encoder(t_len, op_size)
        self.dec = decoder(t_len, op_size)
        self.koopman_layer = Koopman_Operator2D(self.op_size, self
            .modes)
        self.w0 = nn.Conv2d(op_size, op_size, 1)
    def forward(self, x):
        # Reconstruct
        x_reconstruct = self.enc(x)
        x_reconstruct = torch.tanh(x_reconstruct)
        x_reconstruct = self.dec(x_reconstruct)
        # Predict
        x = self.enc(x) # Encoder
        x = x.permute(0, 3, 1, 2)
        x_w = x
        for i in range(self.decompose):
            x1 = self.koopman_layer(x) # Koopman Operator
            x = torch.tanh(x + x1)
        x = self.w0(x_w) + x
        x = x.permute(0, 2, 3, 1)
        x = self.dec(x) # Decoder
        return x, x_reconstruct
```