A MODEL-CONSTRAINED TANGENT SLOPE LEARNING APPROACH FOR DYNAMICAL SYSTEMS

Abstract. Real-time accurate solutions of large-scale complex dynamical systems are in critical need for control, optimization, uncertainty quantification, and decision-making in practical engineering and science applications, especially digital twin applications. This paper contributes in this direction a model-constrained tangent slope learning (`mcTangent`) approach. At the heart of `mcTangent` is the synergy of several desirable strategies: i) a tangent slope learning to take advantage of the neural network speed and the time-accurate nature of the method of lines; ii) a model-constrained approach to encode the neural network tangent slope with the underlying governing equations; iii) sequential learning strategies to promote long-time stability and accuracy; and iv) data randomization approach to implicitly enforce the smoothness of the neural network tangent slope and its likeliness to the truth tangent slope up second order derivatives in order to further enhance the stability and accuracy of `mcTangent` solutions. Rigorous results are provided to analyze and justify the proposed approach. Several numerical results for transport equation, viscous Burgers equation, and Navier-Stokes equation are presented to study and demonstrate the robustness and long-time accuracy of the proposed `mcTangent` learning approach.

Key words. dynamical systems; model-constrained learning; sequential learning; the method of lines; data randomization; tangent slope; accuracy and stability; regularization;

1. Introduction. Dynamical systems are pervasive in engineering and science applications. They are typically time-dependent systems of ordinary differential equations (ODEs) or partial differential equations (PDEs). The latter is not different from the former from the method of lines viewpoint in which a PDE reduces to a system of ODEs after a spatial discretization. For practical settings, simulating a dynamical system could be challenging due to a large number of degrees of freedom, and hence the number of ODEs, interdependent on each other in a highly nonlinear manner. For multi-scale or stiff systems of ODEs, explicit time discretization schemes, though straightforward, are not efficient to due time stepsize limitation to ensure stability. Implicit schemes, on the other hand, are stable but computationally expensive as a large linear system of equations needs to be solved at each time step. Though currently infeasible, real-time accurate approximate solutions for the practical complex dynamical system are highly desirable for control, optimization, uncertainty quantification, and decision-making.

Towards achieving real-time solutions for dynamical systems, various pure data-driven deep learning attempts have been made. Autoencoder architecture has been explored to simulate fluid flows [19]. Autoencoder with physics-informed regularization to improve accuracy has been proposed to predict the future sea surface temperature given past series of measurements [10]. In [45], a graph network-based model is trained to approximate the forward map and inference model, and then used to speed up control algorithms. As an effort to combine traditional and machine learning approaches, the authors in [31] introduce a deep Koopman model—an auto-encoder architecture of convolution neural network—to predict the dynamics of airflow over a cylinder. Comprehensive overviews of machine learning methods for forecasting dynamical systems can be found in [25] and [2]. The work in [11] presents a review and aspects of using machine learning techniques to simulate turbulent flows.

Instead of replacing traditional computational approaches with pure data-driven machine learning models, which is debatable and an active research direction, one can use machine learning methods to speed up only computationally demanding modules. This could maintain desirable physics constraints as in traditional approaches while gaining computational time. Indeed, a convolution neural network (CNN) can be

trained to learn the numerical error between high-resolution and low-resolution simulations [34]. Combining the CNN prediction with low-resolution simulations can then achieve similar high-resolution accuracy while being faster and at that the same time not compromising the physics. In a different effort, neural networks are trained to replace components/terms severely affected by a low-resolution grid [21]. The predictions from neural networks are then unrolled over multiple time steps to improve long-time inference performance. A recurrent neural network can also be used to enhance the effectiveness of geometric multigrid methods for simulating Navier-Stokes equations [29].

Completely replacing traditional methods while respecting governing equations, we argue, is highly desirable for machine learning methods because fast but nonphysical solutions are undesirable. A popular deep-learning approach aiming to accomplish this goal is the physics-informed neural network (PINN) [42]. Similar to least squares finite element methods, PINN trains deep learning solution constrained by the PDE residual through a regularization [42, 38, 40, 41, 57, 51]). PINN can learn solutions that attempt to make the PDE residual small. However, the PINN approach directly approximates the PDE solution in infinite dimensional spaces. While universal approximation results (see, e.g., [9, 13, 28, 18]) could ensure any desired accuracy with a sufficiently large number of neurons, practical network architectures are moderate in both depth and width, and hence the number of weights and biases, the accuracy of PINN can be limited. Moreover, PINN requires a retrain for new scenarios such as new boundary conditions, or new initial conditions, or new values of the underlying parameters. A physics-informed recurrent neural network has also been studied in [15]. In order to produce physically consistent and better prediction results, energy flow and density-depth constraint laws are integrated into the loss function.

Instead of learning the infinite-dimensional solution as in PINN, learning discretized solutions of dynamical systems is equally popular. The work in [59] uses a neural network to approximate the derivative of the system state in reduced projected subspace. The neural network is then combined with forward Euler and Runge-Kutta time discretization schemes to achieve high-accuracy solutions. Alternatively, a feed forward neural network can be used to directly learn the map from the solution at the current time step to the solution in the next time step [33]. The stability and accuracy of long-time prediction are reinforced by introducing a Jacobian regularization into the loss function. Realizing several drawbacks of the direct learning approach, the authors in [54] propose to learn the tangent slope with Runge-Kutta schemes. Once trained, the learned tangent slope can be used with any time discretization schemes and any time step size. In [53], the authors propose to learn a correction neural network that lifts low-resolution solutions to high-resolution accuracy, and the training procedure includes low-resolution differentiable codes. Similarly, differential molecular dynamics simulations [47] have been implemented in `Jax` [4]. Alternatively, the authors in [14] develop a differentiable simulations package that wraps a numerical simulator as a gradient kernel for end-to-end back-propagation used in optimization algorithms. Similar to [47], a differentiable physic simulations package equipped with the adjoint method for backpropagation is developed in [12], which enables the embedding of physical forward model into the training process.

In this paper, aiming at simulating dynamical systems in real-time, we propose a model-constrained tangent slope deep learning (`mcTangent`) approach that has several appealing features over existing methods. First, it operates on finite dimensional systems and is thus in principle easier to train. However, it is spatial discretization-dependent for systems governed by PDEs. Second, it learns the underlying tangent

slope and thus is semi-discrete in nature. Once trained, it can be deployed with any time discretization schemes with any time step size. The next three features are the main advances beyond the work in [54]. Third, it aims to fulfill the governing equations by constraining a fully discrete system in the loss function during training. Fourth, it is equipped with sequential learning strategies and thus promotes stability and accuracy in simulating the underlying dynamical systems far beyond the training time horizon. Fifth, our approach imposes regularizations on the smoothness of the neural network tangent and its derivatives implicitly via data randomization. This provides extra stability and accuracy for `mcTangent` solutions.

The paper is organized as follows. Section 2 introduces an abstract dynamical system and a model-constrained tangent slope learning (`mcTangent`) approach. Both sequential machine learning and sequential model-constrained strategies will be discussed in detail in subsection 2.2 and subsection 2.3. Data randomization approach then follows with an in-depth semi-heuristic argument to reveal its implicit regularization nature in subsection 2.4. In particular, data randomization induces smoothness regularization for the underlying neural network via the standard machine learning loss. The beauty of the model-constrained loss term is that it not only enforces the likeliness of the neural network and the truth tangent slopes but also implicitly constrains their likeliness up to second-order derivatives via data randomization. subsection 2.5 provides a rigorous estimation for prediction error using `mcTangent` approach. Several numerical results using the proposed `mcTangent` approach for transport equation, viscous Burger's equation, and Navier-Stokes equation are presented in section 3. We also provide detailed information on parameter tuning, randomness setting, and the cost for both training and testing. Section 4 concludes the paper with future work.

2. Model-constrained tangent slope deep learning solutions for dynamical systems.

2.1. Motivation. For the concreteness and simplicity of the exposition, let us consider an abstract dynamical system governed by the following time-dependent scalar PDE equation of the form

$$\frac{\partial u}{\partial t} = \mathcal{G}\left(u, \nabla u, \ldots\right) \qquad \text{in } \Omega \subset \mathbb{R}^d, \tag{2.1}$$

where $t \in [0, T]$, $u\left(\boldsymbol{x}\right) \in \mathbb{R}$ for any $\boldsymbol{x} \in \mathbb{R}^d$, and $d \in \{1, 2, 3\}$. We also assume (2.1) is equipped with appropriate initial conditions and boundary conditions to ensure its well-posedness.

In this paper, we are interested in *parametrized PDEs*. For downstream tasks such as design, control, optimization, inference, and uncertainty quantification, these PDEs need to be solved many times. As such, we wish to approximate solutions of (2.1) in real time for any parameters (e.g. initial conditions or boundary conditions, or some parameter). Training a PINN together with parameters (either by themselves or their neural networks weights and biases as another set of optimization variables) [8, 39, 26, 27] may not be efficient as a new solution (corresponding to new parameters) requires a retrain. We note that attempts using pure data-driven deep learning to learn the parameter-to-solution map have been explored (see, e.g., [22, 55, 35, 50, 36, 22, 49, 16]). On the other hand, standard numerical methods such as finite difference, finite volume, and finite elements [48, 23, 17] discretize (2.1) both in time and space. One of the most popular approaches is perhaps the method of lines (see, e.g., [46]) in which one performs spatial discretization first to obtain a system of (possibly nonlinear)

ordinary differential equations of the form

$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{G}(\boldsymbol{u}), \tag{2.2}$$

where $\boldsymbol{u}$ and $\boldsymbol{G}$ are vector representations of finite dimensional approximations of u and $\mathcal{G}$, respectively. Now, either an explicit or implicit (or their combination) can be deployed to discretize the temporal derivative. For the former, the most expensive operation is the evaluation of tangent slope[1] $\boldsymbol{G}(\boldsymbol{u})$. For the latter, evaluating both $\boldsymbol{G}(\boldsymbol{u})$ and its (possibly approximate) Jacobian for each time step play a vital role. Implementing the Jacobian, even with the adjoint method [52], is a significant part of the programming effort. Automatic differentiation can mitigate this programming burden at the expense of more memory bandwidth. In summary, computing $\boldsymbol{G}(\boldsymbol{u})$ and its Jacobian is a major part, both in implementation and computational time, of existing numerical methods.

To overcome the time burden of estimating the tangent slope and its Jacobian, we present a model-constrained tangent slope deep learning approach (`mcTangent`) inspired by the semi-discrete nature of the method of lines. In particular, we first learn the tangent slope $\boldsymbol{G}(\boldsymbol{u})$ using neural network and then use a time discretization to solve for approximations of $\boldsymbol{u}$. Our approach thus aims to approximate only the spatial discretization and leaves the temporal discretization for traditional time integrators. At the heart of our approach is the incorporation of the governing equations into the neural network tangent by constraining the learning task to respect a temporal discretization of (2.2). Again, unlike PINN and its siblings which learn the infinite-dimensional solution u, our approach learns the tangent slope of the finite-dimensional approximation $\boldsymbol{u}$. Furthermore, we constrain the physics on the discrete level. Clearly, our approach is discretization-dependent while PINN requires neither spatial discretization nor temporal discretization.

2.2. Model-constrained neural network approach with sequential data learning. In this section, we construct a model-constrained neural network $\Psi(\boldsymbol{u})$ to learn $\boldsymbol{G}(\boldsymbol{u})$. This is done in tandem with a time discretization of (2.2). For clarity, we limit our presentation to forward Euler method

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \Delta t\, \boldsymbol{G}\left(\boldsymbol{u}^k\right), \tag{2.3}$$

as it is straightforward to extend the approach to any time discretization scheme, and we provide a brief discussion at the end of the section. The task at hand is to train $\Psi(\boldsymbol{u})$ on a certain spatial mesh $\mathcal{T}$ corresponding to a spatial discretization. To begin, let us denote the numerical solutions of (2.3) at $N_t + 1$ time steps on a finer mesh $\mathcal{T}^f$ as

$$\left\{\boldsymbol{u}^0, \boldsymbol{u}^1, \ldots, \boldsymbol{u}^{N_t}\right\}.$$

which are then down-sampled on $\mathcal{T}$ for training $\Psi(\boldsymbol{u})$. Doing so has proved to yield more accurate predictions than training directly on the solutions on $\mathcal{T}$ [34, 21, 58]. This is not surprising as the down-sampled training data on $\mathcal{T}$ is more accurate than the solution on $\mathcal{T}$.

The next idea that we like to incorporate into our approach is sequential training. The key is to feed the predictions back to the neural network model to enable a

[1] We call the right hand side $\boldsymbol{G}(\boldsymbol{u})$ as the tangent slope as it is a generalization of the tangent slope field in scalar ordinary differential equation.

better long-time predictive capability. Using this idea [56] deploys a mixture of graph neural network and 3D-U-Net neural network to model fluid flows. Similarly, in [58] sequential learning is used to train a network to obtain the optimal finite difference coefficients from the high-resolution training data. In the context of atmosphere modeling, [5] introduces a stable and highly accurate long-time prediction loss function with sequential training. Following [34, 21, 56, 58], we partition the training data in $N_t - S$ overlapping subsets

$$\mathcal{U} := \left\{ \left(\boldsymbol{u}^0, \boldsymbol{u}^1, \ldots, \boldsymbol{u}^{S+1}\right), \left(\boldsymbol{u}^1, \boldsymbol{u}^2, \ldots, \boldsymbol{u}^{S+2}\right), \ldots, \left(\boldsymbol{u}^{N_t-S-1}, \boldsymbol{u}^{N_t-S}, \ldots, \boldsymbol{u}^{N_t}\right) \right\}.$$

For convenience in the exposition, we enumerate these $N_t - S$ subsets as

$$\mathcal{U} := \{ \left(\boldsymbol{u}^{0,0}, \boldsymbol{u}^{0,1}, \ldots, \boldsymbol{u}^{0,S+1}\right), \left(\boldsymbol{u}^{1,0}, \boldsymbol{u}^{1,1}, \ldots, \boldsymbol{u}^{1,S+1}\right), \\ \ldots, \left(\boldsymbol{u}^{N_t-S-1,0}, \boldsymbol{u}^{N_t-S-1,1}, \ldots, \boldsymbol{u}^{N_t-S-1,S+1}\right) \},$$

where the second superscript denotes the local index in each subset. To distinguish from the sequential model-constrained learning in subsection 2.3, let us call the machine learning approach based on these overlapping subsets as *sequential data learning*.

We next discuss how we use each subset in our model-constrained approach. Consider the kth subset $\left(\boldsymbol{u}^{k,0}, \boldsymbol{u}^{k,1}, \ldots, \boldsymbol{u}^{k,S+1}\right)$, for $k = 0, \ldots, N_t - S - 1$. Starting from $\tilde{\boldsymbol{u}}^{k,0} = \boldsymbol{u}^{k,0}$, we can write the sequence of approximate solutions $\left\{\tilde{\boldsymbol{u}}^{k,i}\right\}_{i=1}^{S+1}$ for (2.2) using forward Euler time discretization with the neural network tangent $\Psi(\boldsymbol{u})$ as

$$\tilde{\boldsymbol{u}}^{k,i+1} = \tilde{\boldsymbol{u}}^{k,i} + \Delta t \Psi\left(\tilde{\boldsymbol{u}}^{k,i}\right), \quad i = 0, \ldots, S. \tag{2.4}$$

On the other hand, if we feed $\tilde{\boldsymbol{u}}^{k,i}$ through the forward Euler discretization (2.3) we obtain

$$\bar{\boldsymbol{u}}^{k,i+1} = \tilde{\boldsymbol{u}}^{k,i} + \Delta t\, \boldsymbol{G}\left(\tilde{\boldsymbol{u}}^{k,i}\right), \quad i = 0, \ldots, S. \tag{2.5}$$

As can be seen $\tilde{\boldsymbol{u}}^{k,i+1} \neq \bar{\boldsymbol{u}}^{k,i+1}$, though we wish they are the same. *If they were, the approximate solutions using neural network tangent would respect the governing discretized equation exactly.* Obviously, this is not feasible in general. Thus, we resort to requiring $\tilde{\boldsymbol{u}}^{k,i+1}$ as close as possible to $\bar{\boldsymbol{u}}^{k,i+1}$. One way to accomplish this is to consider the following loss function for the kth batch:

$$\mathcal{J}_k := \frac{1}{S+1} \sum_{i=1}^{S+1} \left(\left\| \boldsymbol{u}^{k,i} - \tilde{\boldsymbol{u}}^{k,i} \right\|_2^2 + \alpha \left\| \bar{\boldsymbol{u}}^{k,i} - \tilde{\boldsymbol{u}}^{k,i} \right\|_2^2 \right), \tag{2.6}$$

where α is a model-constrained penalty (or regularization) parameter, which controls the magnitude of the model-constrained loss (relative to the machine learning loss and). Parameter tuning in subsection 3.4 shows that a single value $\alpha = 10^5$ works well for all numerical examples in section 3. The first term of the loss (2.6)—the ML Loss in Figure 1—ensures the data consistency, while the second term—the MC Loss in Figure 1—is to force approximate solutions of (2.2) using neural network tangent $\Psi(\boldsymbol{u})$ to best fit the underlying space-time discretization (2.3). The schematic of the `mcTangent` architecture with sequential data learning for the kth data subset and $S = 1$ is illustrated in Figure 1. We note that, unlike SINDy [6], which discovers the dynamic systems from a dictionary of common differential operators, `mcTangent` aims to approximate high dimensional complex nonlinear tangent slope $\boldsymbol{G}$ operator using neural network.

Remark 2.1. Note that it is not essential that $\bar{\boldsymbol{u}}^i$ must be obtained by the forward Euler scheme (2.3). In fact, our approach is flexible in the sense that any one-step explicit scheme, denoted as $\mathcal{F}$, (including explicit Runge-Kutta) is admissible. In such a case, our neural network can be considered as learning the forward Euler approximation of the ground-truth scheme.

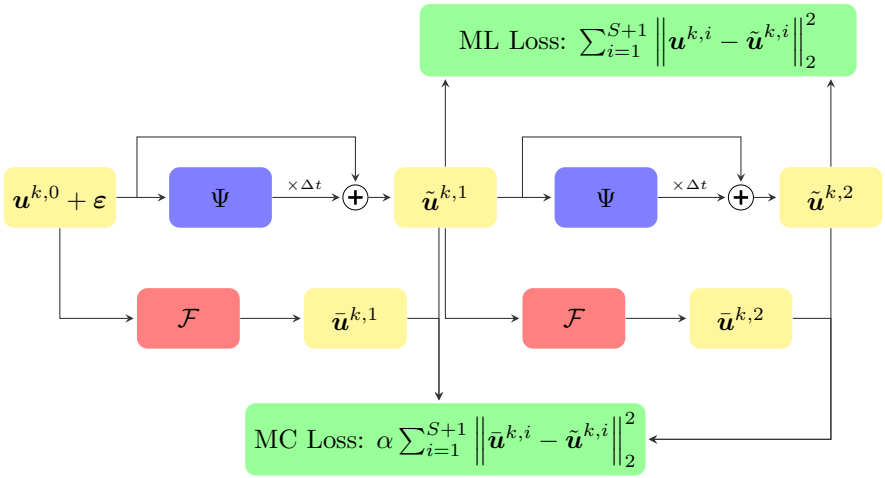

Fig. 1: The schematic of the `mcTangent` approach with sequential data learning with $S = 1$. For the data randomization approach in Section 2.4, the random noise vector, $\boldsymbol{\varepsilon}$, is added to the first input of the neural network.

Taking all the batches into account yields the total loss function

$$\mathcal{J} := \frac{1}{(N_t - S)(S+1)} \sum_{k=0}^{N_t-S-1} \sum_{i=1}^{S+1} \left(\left\| \boldsymbol{u}^{k,i} - \tilde{\boldsymbol{u}}^{k,i} \right\|_2^2 + \alpha \left\| \bar{\boldsymbol{u}}^{k,i} - \tilde{\boldsymbol{u}}^{k,i} \right\|_2^2 \right). \tag{2.7}$$

To gain insight into our `mcTangent` approach, we consider a linear problem in which $\boldsymbol{G}(\boldsymbol{u}) = \boldsymbol{G}\boldsymbol{u}$, and a one-layer linear neural network $\Psi\left(\boldsymbol{u}^{k,0}\right) = \boldsymbol{W}\boldsymbol{u}^{k,0} + \mathbf{b}$. Under a mild condition, our approach should exactly recover the underlying tangent slope, i.e. $\Psi(\boldsymbol{u}) = \boldsymbol{G}\boldsymbol{u}$. Indeed, let $S = 0$ so that the loss function (2.7) becomes

$$\begin{aligned} \mathcal{J} =& \frac{1+\alpha}{N_t} \sum_{k=0}^{N_t-1} \left\| \boldsymbol{u}^{k,1} - \tilde{\boldsymbol{u}}^{k,1} \right\|_2^2 = \frac{1+\alpha}{N_t} \left\| U^1 - \tilde{U}^1 \right\|_F^2 \\ =& \frac{(1+\alpha)\,\Delta t^2}{N_t} \left\| \boldsymbol{G}U^0 - \left(\boldsymbol{W}U^0 + \mathbf{b}\mathbb{1}^T \right) \right\|_F^2 \end{aligned} \tag{2.8}$$

where U^{t_i} and $\tilde{U}^{t_i}$ are matrices with true and predictive solutions as columns, respectively, and $\mathbb{1}$ is the unit column vector.

LEMMA 2.2. *The optimal solution* $(\boldsymbol{W}^*, \mathbf{b}^*)$ *for the training problem*

$$\min_{\boldsymbol{W},\mathbf{b}} \mathcal{J}$$

is given by

$$\boldsymbol{W}^* = \boldsymbol{G}\overline{UU}^\dagger, \quad \mathbf{b}^* = \boldsymbol{G}\left(\boldsymbol{I} - \overline{UU}^\dagger \right) \overline{\boldsymbol{u}}, \tag{2.9}$$

where $\overline{\boldsymbol{u}} := \frac{1}{N_t} U^{t_0} \mathbb{1}$ *is the column-average of matrix* U^{t_0}, $\overline{U} := U^{t_0} - \overline{\boldsymbol{u}} \mathbb{1}^T$, *and* † *denotes the pseudo-inverse. Consequently, the optimal network reads*

$$\Psi(\boldsymbol{u}) = \boldsymbol{G}\overline{U}\overline{U}^\dagger \boldsymbol{u} + \boldsymbol{G}\left(\boldsymbol{I} - \overline{U}\overline{U}^\dagger\right)\overline{\boldsymbol{u}}.$$

Remark 2.3. Lemma 2.2 tells us that the optimal network exactly recovers the true forward map $\boldsymbol{G}$ if $\overline{U}$ is a full row rank matrix. (In that case, $\overline{U}\overline{U}^\dagger = \mathbf{I}$.) This holds, for example, when the number of independent data samples is equal to the discretized dimension. We would like to point out that the MC loss term is the same as the ML loss term (up to a constant), and thus does not provide any extra information in this simple case. When $S > 0$, at the time of writing this paper, we are not able to find a closed form solution as in Lemma 2.2. We leave it as future work.

Remark 2.4. Although we learn the tangent slope using the Forward Euler scheme, it is straightforward to use any explicit scheme, such as Adams–Bashforth and Runge-Kutta methods to accomplish our goal. For example (ignoring extra subscripts for sequential data learning for simplicity), using the two-step Adams-Bashforth scheme, `mcTangent` solutions read

$$\tilde{\boldsymbol{u}}^{i+1} = \tilde{\boldsymbol{u}}^i + \frac{3}{2}\Delta t\, \Psi\left(\tilde{\boldsymbol{u}}^i\right) - \frac{1}{2}\Delta t\, \Psi\left(\tilde{\boldsymbol{u}}^{i-1}\right),$$

as oppose to the solutions using the truth tangent slope

$$\boldsymbol{u}^{i+1} = \boldsymbol{u}^i + \frac{3}{2}\Delta t\, \boldsymbol{G}\left(\boldsymbol{u}^i\right) - \frac{1}{2}\Delta t\, \boldsymbol{G}\left(\boldsymbol{u}^{i-1}\right).$$

Similarly, `mcTangent` solutions based on the second-order Runge-Kutta scheme reads

$$\begin{aligned}
\tilde{k}_1 &= \Delta t\, \Psi\left(\tilde{\boldsymbol{u}}^i\right) \\
\tilde{k}_2 &= \Delta t\, \Psi\left(\tilde{\boldsymbol{u}}^i + k_1\right) \\
\tilde{\boldsymbol{u}}^{i+1} &= \tilde{\boldsymbol{u}}^i + \frac{\tilde{k}_1 + \tilde{k}_2}{2},
\end{aligned}$$

as oppose to the solutions using the truth tangent slope

$$\begin{aligned}
k_1 &= \Delta t\, \boldsymbol{G}\left(\boldsymbol{u}^i\right) \\
k_2 &= \Delta t\, \boldsymbol{G}\left(\boldsymbol{u}^i + k_1\right) \\
\boldsymbol{u}^{i+1} &= \boldsymbol{u}^i + \frac{k_1 + k_2}{2}.
\end{aligned}$$

Clearly, we have to modify the lost function accordingly, but the idea is the same as forward Euler approach that we have presented above.

Remark 2.5. Note that we have used forward Euler time discretization for both (2.4) and (2.5) for simplicity, but this is not necessary. We recommend to use time discretizations with the same order of accuracy for both as accuracy gain in incompatible discretizations may not be well paid-off by additional computational demand. For example, if we use low-order accuracy for (2.4) but higher-order accuracy for (2.5), `mcTangent` solution could be more accurate with smaller constant in the order of accuracy (still low-order) since it tries to match more accurate solution from (2.5). However, the training cost could increase significantly due to several evaluations (and hence differentiations for back-propagation) of the truth tangent slope $\boldsymbol{G}$ in (2.5). Clearly high-order accurate approaches could tax the training time significantly.

2.3. Model-constrained neural network approach both sequential data and sequential model learnings. In subsection 2.2, we present a sequential data learning approach for the proposed model-constrained neural network $\Psi(\boldsymbol{u})$ to learn the tangent slope while being constrained to provide the best possible approximate solutions for (2.3) for each time step. In order to improve the long-time predictive capability and accuracy, this section constructs, in addition to *sequential data learning*, a *sequential model learning* strategy for training the neural network $\Psi(\boldsymbol{u})$ is proposed. Sequential model learning is designed to promote the neural network solutions to respect the underlying discretization scheme for multiple time steps concurrently. In particular, starting from $\tilde{\boldsymbol{u}}^{k,i}$ we can carry out R steps forward in time using the underlying discretization (2.3) as

$$\bar{\boldsymbol{u}}^{k,i,r} = \bar{\boldsymbol{u}}^{k,i,r-1} + \Delta t\, \boldsymbol{G}\left(\bar{\boldsymbol{u}}^{k,i,r-1}\right), \quad r = 1, \ldots, R,$$

and using the neural network approximation (2.4) as

$$\tilde{\boldsymbol{u}}^{k,i,r} = \tilde{\boldsymbol{u}}^{k,i,r-1} + \Delta t\, \Psi\left(\tilde{\boldsymbol{u}}^{k,i,r-1}\right), \quad r = 1, \ldots, R,$$

where $\bar{\boldsymbol{u}}^{k,i,0} = \tilde{\boldsymbol{u}}^{k,i,0} = \tilde{\boldsymbol{u}}^{k,i}$. Here the third superscript r has been introduced to keep track of R sequential forward steps starting from $\tilde{\boldsymbol{u}}^{k,i}$ for both exact and neural network tangent slopes. In order to ensure that these corresponding R sequential predictions closely match each other, we consider the following loss function

(2.10)
$$\mathcal{J} := \frac{1}{(N_t - S)(S+1)} \sum_{k=0}^{N_t-S-1} \sum_{i=1}^{S+1} \left(\left\| \boldsymbol{u}^{k,i} - \tilde{\boldsymbol{u}}^{k,i} \right\|_2^2 + \frac{\alpha}{R} \sum_{r=1}^{R} \left\| \bar{\boldsymbol{u}}^{k,i,r} - \tilde{\boldsymbol{u}}^{k,i,r} \right\|_2^2 \right).$$

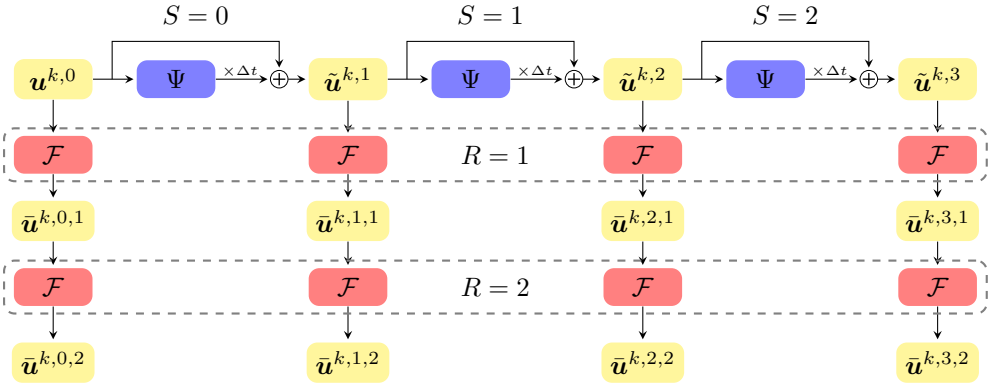

Fig. 2: The schematic of the `mcTangent` approach with both sequential data and model learnings with $S = 2, R = 2$.

The schematic of the `mcTangent` architecture with both sequential data and model learnings for the kth data subset and $S = 2, R = 2$ is depicted in Figure 2. Clearly, when $R = 1$ we recover (2.7) from (2.10). In other words, (2.10) is a generalization of (2.7). Intuitively, larger values for R, S increase the predictive capacity of `mcTangent` solutions, and as an example this will be demonstrated for the Burgers equation in 3.2. However, it is computationally expensive to use large values for both S and

R. In the numerical results in section 3, we study two combinations: $S \geq 1, R = 1$ and $S = 1, R \geq 1$. In order to have a deeper understanding of the role of the loss function (2.10) in training the neural network tangent and its predictive capability, we shall provide an in-depth heuristic argument in subsection 2.4 and a rigorous error estimation for `mcTangent` predictions in subsection 2.5.

2.4. Data randomization. It has been observed [44] that adding a small amount of noise to training data not only increases the generalization on unseen data but also reduces accumulated errors in predictions. In fact, clean noise-free data does not represent the accumulated error in the predictive state that is fed back to the network to produce subsequent predictions. Moreover, noisy data encourages neural network predictions to be more robust to noise-corrupted inputs and errors. In order to investigate the significance of different noise additions (adding noise to the training inputs, weights of the neural network, and output labels) on the model generalization, [1] demonstrates that the reasonable noise level in the outputs does not influence the trained network. Randomizing training data, on the one hand, prevents the neural network from overfitting data, and on the other hand, can make the network insensitive to noise in data in the validation phase.

It is well-known that randomization induces a regularization of the gradient of the loss function with respect to the inputs [43]. Consequently, the neural network, if a proper noise level is used, is regularized to be a smooth function of the input data. The smoothness reduces the sensitivity to the variation in the input [30] and can enhance the stability of long-time predictions [37]. The work in [3] showed that adding noise to data is equivalent to introducing a Tikhonov regularization to the loss function (where the regularization parameter is the noise variance) and thus improving the model generalization. However, the analysis is only valid in the context of infinite training data set, as pointed out in [1].

Inspired by the aforementioned work, we randomize the input data for the model-constrained network. We shall show that randomization induces regularizations not only to promote the smoothness of the network but also to enhance the similarity of the derivatives of the network $\Psi(\boldsymbol{u})$ and the true tangent slope $\boldsymbol{G}(\boldsymbol{u})$. As shall be seen, the numerical results in section 3 reveal that randomization improves significantly the long-term stability and accuracy.

In this paper, we randomize the input $\boldsymbol{u}$ of the network as

$$\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{\varepsilon}, \tag{2.11}$$

where $\boldsymbol{\varepsilon}$ is a normal random vector $\boldsymbol{\varepsilon} \sim \mathcal{N}\left(\mathbf{0}, \delta^2 \mathbf{I}\right)$. Note that the following heuristic arguments also hold for any random vector with independent components, each of which is a random variable with zero mean and variances δ^2. Let $\mathbb{E}[\cdot]$ denote the expectation with respect to $\boldsymbol{\varepsilon}$. Following [1], for a generic loss function $\mathcal{L}(\boldsymbol{u})$ we have

$$\begin{aligned} \mathbb{E}\left[\mathcal{L}(\boldsymbol{v})\right] =& \mathcal{L}(\boldsymbol{u}) + \mathbb{E}\left[\left.\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}} \boldsymbol{\varepsilon}\right] + \frac{1}{2}\mathbb{E}\left[\boldsymbol{\varepsilon}^T \left.\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{u}^2}\right|_{\boldsymbol{u}} \boldsymbol{\varepsilon}\right] + o\left(\|\boldsymbol{\varepsilon}\|^2\right) \\ \approx& \mathcal{L}(\boldsymbol{u}) + \frac{1}{2}\mathbb{E}\left[\boldsymbol{\varepsilon}^T \left.\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{u}^2}\right|_{\boldsymbol{u}} \boldsymbol{\varepsilon}\right], \end{aligned} \tag{2.12}$$

where we have used sufficient small noise $\boldsymbol{\varepsilon}$ (relatively to $\boldsymbol{u}$) so that the high-order term $o\left(\|\boldsymbol{\varepsilon}\|^2\right)$, using the standard "small o" notation, is assumed to be negligible. We consider $S = 0$ and $R = 1$. (For $S > 0$ and/or $R > 1$, the sequential inputs to

the network contain the error which may not satisfy the condition for (2.12) to hold.) In this case, the loss function (2.7) becomes

$$\mathcal{J} = \frac{1}{N_t}\sum_{k=0}^{N_t-1} \underbrace{\left\|\boldsymbol{u}^{k,1} - \tilde{\boldsymbol{u}}^{k,1}\right\|_2^2}_{\mathcal{L}_{\mathrm{ML}}(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon})} + \alpha \underbrace{\left\|\bar{\boldsymbol{u}}^{k,1} - \tilde{\boldsymbol{u}}^{k,1}\right\|_2^2}_{\mathcal{L}_{\mathrm{MC}}(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon})} \tag{2.13}$$

We now study the randomized ML loss term $\mathcal{L}_{\mathrm{ML}}\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right)$ and the randomized MC loss term $\mathcal{L}_{\mathrm{MC}}\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right)$ to gain insights into the role of randomization.

The machine learning loss term reads

$$\mathcal{L}_{\mathrm{ML}}\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right) = \left\|\boldsymbol{u}^{k,1} - \left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}+\Delta t\Psi\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right)\right)\right\|_2^2$$

which is a function of true input $\boldsymbol{u}^{k,0}$ plus a random noise vector $\boldsymbol{\varepsilon}$. *It is important to note that we do not randomize the true data* $\boldsymbol{u}^{k,1}$ *against which we compare the machine prediction* $\tilde{\boldsymbol{u}}^{k,1}$. Replacing $\mathcal{L}$ by $\mathcal{L}_{\mathrm{ML}}$ in (2.12) yields

$$\mathbb{E}\left[\mathcal{L}_{\mathrm{ML}}\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right)\right] \approx \underbrace{\left\|\boldsymbol{u}^{k,1} - \left(\boldsymbol{u}^{k,0}+\Delta t\Psi\left(\boldsymbol{u}^{k,0}\right)\right)\right\|_2^2}_{\mathcal{L}_{\mathrm{ML}}(\boldsymbol{u}^{k,0})} + \delta^2\left[\mathcal{P}_1\left(\boldsymbol{u}^{k,0}\right) + \mathcal{P}_2\left(\boldsymbol{u}^{k,0}\right)\right], \tag{2.14}$$

where

$$\mathcal{P}_1\left(\boldsymbol{u}^{k,0}\right) = \mathbf{Tr}\left[\left(\mathbf{I}+\Delta t\left.\frac{\partial\Psi}{\partial\boldsymbol{u}}\right|_{\boldsymbol{u}^{k,0}}\right)^T\left(\mathbf{I}+\Delta t\left.\frac{\partial\Psi}{\partial\boldsymbol{u}}\right|_{\boldsymbol{u}^{k,0}}\right)\right], \tag{2.15}$$

with $\mathbf{Tr}\left[\cdot\right]$ as the trace operator, and

$$\mathcal{P}_2\left(\boldsymbol{u}^{k,0}\right) = \mathbf{Tr}\left[\Delta t\left.\frac{\partial^2\Psi}{\partial\boldsymbol{u}^2}\right|_{\boldsymbol{u}^{k,0}} \odot \left[\left(\boldsymbol{u}^{k,0}+\Delta t\Psi\left(\boldsymbol{u}^{k,0}\right)\right) - \boldsymbol{u}^{k,1}\right]\right], \tag{2.16}$$

where $\odot$ denotes the dot product of the third order tensor $\Delta t\left.\frac{\partial^2\Psi}{\partial\boldsymbol{u}^2}\right|_{\boldsymbol{u}^{k,0}}$ and the vector $\left[\left(\boldsymbol{u}^{k,0}+\Delta t\Psi\left(\boldsymbol{u}^{k,0}\right)\right) - \boldsymbol{u}^{k,1}\right]$.

From (2.14), three observations are in order. First, on average, the randomized ML loss term is approximately the original ML loss term plus two additional terms $\mathcal{P}_1$ and $\mathcal{P}_2$ scaled by the variance δ^2 of the noise. Second, the first term $\mathcal{P}_1$ is positive and thus is a regularization. It enforces the boundedness of the gradient (and hence the smoothness) of the neural network. Third, the second term $\mathcal{P}_2$ can be either positive or negative. However, when the time step Δt is small and/or the ML misfit term $\left[\left(\boldsymbol{u}^{k,0}+\Delta t\Psi\left(\boldsymbol{u}^{k,0}\right)\right) - \boldsymbol{u}^{k,1}\right]$ is small (e.g. with sufficient training), the contribution of the second term is expected to be dominated by the first and thus is negligible. When neither of these two conditions is satisfied, if the training enforces small "curvature" of the neural network (i.e. small $\left.\frac{\partial^2\Psi}{\partial\boldsymbol{u}^2}\right|_{\boldsymbol{u}^{k,0}}$) then the second term is also negligible. When this happens, training with randomization provides extra smoothness to the network.

Next, from (2.4) and (2.5), the randomized MC loss term can be written as

$$\mathcal{L}_{\mathrm{MC}}\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right) = \left\|\bar{\boldsymbol{u}}^{k,1} - \tilde{\boldsymbol{u}}^{k,1}\right\|_2^2 = \Delta t^2\left\|\boldsymbol{G}\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right) - \Psi\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right)\right\|_2^2.$$

Applying (2.12) with $\mathcal{L}_{\text{MC}}$ in place of $\mathcal{L}$ gives

$$\mathbb{E}\left[\mathcal{L}_{\text{MC}}\left(\boldsymbol{u}^{k,0}+\boldsymbol{\varepsilon}\right)\right] \approx \underbrace{\Delta t^2 \left\|\boldsymbol{G}\left(\boldsymbol{u}^{k,0}\right)-\Psi\left(\boldsymbol{u}^{k,0}\right)\right\|_2^2}_{\mathcal{L}_{\text{MC}}(\boldsymbol{u}^{k,0})} + \delta^2\left[\mathcal{Q}_1\left(\boldsymbol{u}^{k,0}\right)+\mathcal{Q}_2\left(\boldsymbol{u}^{k,0}\right)\right], \tag{2.17}$$

where

$$\mathcal{Q}_1\left(\boldsymbol{u}^{k,0}\right) = \Delta t^2 \mathbf{Tr}\left[\left(\left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}^{k,0}} - \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}^{k,0}}\right)^T \left(\left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}^{k,0}} - \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}^{k,0}}\right)\right], \tag{2.18}$$

and

$$\mathcal{Q}_2\left(\boldsymbol{u}^{k,0}\right) = \mathbf{Tr}\left[\Delta t\left(\left.\frac{\partial^2 \boldsymbol{G}}{\partial \boldsymbol{u}^2}\right|_{\boldsymbol{u}^{k,0}} - \left.\frac{\partial^2 \Psi}{\partial \boldsymbol{u}^2}\right|_{\boldsymbol{u}^{k,0}}\right) \odot \Delta t\left(\Psi\left(\boldsymbol{u}^{k,0}\right)-\boldsymbol{G}\left(\boldsymbol{u}^{k,0}\right)\right)\right]. \tag{2.19}$$

As can be seen, the randomized MC loss term is approximately a sum of the original ML loss term and two additional terms. The first term $\mathcal{Q}_1$ is non-negative and behaves like a regularization to enforce the likeliness of the derivatives with respect to $\boldsymbol{u}$ of the neural network tangent $\Psi(\boldsymbol{u})$ and the true tangent $\boldsymbol{G}(\boldsymbol{u})$. The second term, though could be either negative or positive, can be negligible with sufficient training so that the MC misfit $\Delta t\left(\Psi\left(\boldsymbol{u}^{k,0}\right)-\boldsymbol{G}\left(\boldsymbol{u}^{k,0}\right)\right)$ is relatively small. Another possibility for the insignificance of the second term is when the difference in the "curvature" of the neural network tangent and the true tangent is sufficiently small. In that case, training with randomization promotes the closeness of not only $\Psi(\boldsymbol{u})$ and $\boldsymbol{G}(\boldsymbol{u})$ but their first and second derivatives with respect to $\boldsymbol{u}$: *confirming the significant advantages obtained from data randomization.* Next, combining (2.13), (2.14), and (2.17) yields the following result.

THEOREM 2.6. *Let the input of the neural network be randomized as in (2.11). Then*

$$\begin{aligned}\mathbb{E}[\mathcal{J}] = &\frac{1}{N_t}\sum_{k=0}^{N_t-1}\left(\mathcal{L}_{ML}\left(\boldsymbol{u}^{k,0}\right)+\alpha\mathcal{L}_{MC}\left(\boldsymbol{u}^{k,0}\right)\right)\\ &+\frac{\delta^2}{N_t}\sum_{k=0}^{N_t-1}\left[\mathcal{P}_1\left(\boldsymbol{u}^{k,0}\right)+\mathcal{P}_2\left(\boldsymbol{u}^{k,0}\right)+\alpha\left(\mathcal{Q}_1\left(\boldsymbol{u}^{k,0}\right)+\mathcal{Q}_2\left(\boldsymbol{u}^{k,0}\right)\right)\right]+o\left(\|\boldsymbol{\varepsilon}\|^2\right).\end{aligned} \tag{2.20}$$

The first sum in (2.20) is the original loss (without randomization) and the second sum consists of additional terms induced by data randomization. These additional terms play a vital role in stimulating the stability and accuracy of the neural network. Indeed, as discussed above, randomizing the machine learning loss term encourages the smoothness of the neural network tangent by penalizing its first and second derivatives implicitly. Note that explicitly penalizing the first derivative of a neural network as in [33] is possible, but this could be computationally expensive and challenging. Doing so for both the first and second derivatives is not recommended. The above heuristic analysis of data randomization also reveals the *power of the model-constrained term* in training neural network: it promotes the agreement of the neural network tangent and the true tangent up to second order that is otherwise not realizable using the standard data-driven approach with only machine learning loss term.

2.5. Estimation of prediction errors. In this section, we show how data randomization helps improve the stability and accuracy of long-time predictions. We are interested in predicting solutions of the system (2.2) starting from an initial condition $\boldsymbol{u}^0$ that is not in the training set. To that end, it is natural to compare the `mcTangent` solutions $\tilde{\boldsymbol{u}}^i$ in (2.4) with the solutions $\boldsymbol{u}^i$ obtained from the discretized system (2.3). Let us define the neural prediction error as

$$\boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^i\right) = \boldsymbol{u}^{i+1} - \left[\tilde{\boldsymbol{u}}^i + \Delta t \Psi\left(\tilde{\boldsymbol{u}}^i\right)\right], \quad \varepsilon^{i+1} = \left\|\boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^i\right)\right\|_2. \tag{2.21}$$

From (2.3), (2.4), and (2.21) we have

$$\begin{aligned} \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^i\right) =& \left(\boldsymbol{u}^i + \Delta t \boldsymbol{G}\left(\boldsymbol{u}^i\right)\right) - \left(\tilde{\boldsymbol{u}}^i + \Delta t \Psi\left(\tilde{\boldsymbol{u}}^i\right)\right) \\ =& \Delta t \boldsymbol{G}\left(\tilde{\boldsymbol{u}}^i + \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^{i-1}\right)\right) - \Delta t \Psi\left(\tilde{\boldsymbol{u}}^i\right) + \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^{i-1}\right) \end{aligned} \tag{2.22}$$

Applying the Taylor expansion for the first term gives

$$\Delta t \boldsymbol{G}\left(\tilde{\boldsymbol{u}}^i + \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^{i-1}\right)\right) = \Delta t \boldsymbol{G}\left(\tilde{\boldsymbol{u}}^i\right) + \Delta t \left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i} \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^{i-1}\right) + o\left(\varepsilon^i\right) \tag{2.23}$$

Substituting back to (2.22), we have

$$\begin{aligned} &\boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^i\right) = \Delta t\left[\boldsymbol{G}\left(\tilde{\boldsymbol{u}}^i\right) - \Psi\left(\tilde{\boldsymbol{u}}^i\right)\right] + \Delta t \left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i} \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^{i-1}\right) + \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^{i-1}\right) \\ &+ o\left(\varepsilon^i\right) = \Delta t\left[\boldsymbol{G}\left(\tilde{\boldsymbol{u}}^i\right) - \Psi\left(\tilde{\boldsymbol{u}}^i\right)\right] + \Delta t \left[\left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i} - \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right] \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^{i-1}\right) \\ &+ \left[\mathbf{I} + \Delta t \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right] \boldsymbol{e}_{\mathrm{ML}}\left(\tilde{\boldsymbol{u}}^{i-1}\right) + o\left(\varepsilon^i\right) \end{aligned} \tag{2.24}$$

Applying triangle inequality and Cauchy–Schwarz inequality for (2.24) and using (2.21) yields

$$\begin{aligned} \varepsilon^{i+1} \leq& \Delta t \left\|\boldsymbol{G}\left(\tilde{\boldsymbol{u}}^i\right) - \Psi\left(\tilde{\boldsymbol{u}}^i\right)\right\|_2 \\ &+ \Delta t \left\|\left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i} - \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right\|_2 \varepsilon^i + \left\|1 + \Delta t \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right\|_2 \varepsilon^i + o\left(\varepsilon^i\right), \quad i \geq 0. \end{aligned} \tag{2.25}$$

We observe in (2.25) that the first term on the right-hand side is the model-constrained loss term being as small as possible at the training data. On the other hand, $\Delta t \left\| \left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i} - \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right\|_2$ and $\left\|1 + \Delta t \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right\|_2$ are regularized to be bounded and/or small by data randomization (see subsection 2.4). A heuristic argument reveals that the prediction error is under control at all times. Indeed, suppose $\Delta t \left\|\boldsymbol{G}\left(\tilde{\boldsymbol{u}}^i\right) - \Psi\left(\tilde{\boldsymbol{u}}^i\right)\right\|_2$, $\Delta t \left\| \left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i} - \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right\|_2$, and $\left\|1 + \Delta t \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right\|_2$ are bounded. Since $\varepsilon^0 = 0$, ε^1 is bounded, and by induction ε^i is also bounded for $i \geq 0$. A rigorous version of this argument is given in Theorem 2.7.

THEOREM 2.7. *Assume that the second derivative of $\boldsymbol{G}(\boldsymbol{u})$ with respect to $\boldsymbol{u}$ is uniformly bounded. Let*

$$f^{i+1} := \Delta t \left\|\boldsymbol{G}\left(\tilde{\boldsymbol{u}}^i\right) - \Psi\left(\tilde{\boldsymbol{u}}^i\right)\right\|_2,$$

and

$$g^{i+1} := \Delta t \left\|\left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i} - \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right\|_2 + \left\|1 + \Delta t \left.\frac{\partial \Psi}{\partial \boldsymbol{u}}\right|_{\tilde{\boldsymbol{u}}^i}\right\|_2 + c^i,$$

where $c^i = \mathcal{O}\left(\varepsilon^i\right)$. *Then, the prediction error* ε^n *at time* t_n *satisfies*

$$\varepsilon^n \leq \sum_{k=1}^{n} \left(\Pi_{i=k+1}^{n} g^i\right) f^k.$$

Proof. The proof is a simple application of a discrete Gronwall lemma on (2.25).∎

Remark 2.8. Note that the boundedness of the second derivative of $\boldsymbol{G}(\boldsymbol{u})$ with respect to $\boldsymbol{u}$ is valid for problem (2.2) with a smooth tangent slope. The boundedness of f^i and g^i is not too restricted if the prediction scenarios are close to the training data. Indeed, as argued in subsection 2.4, data randomization enforces the small values for f^i and g^i at the training points. Now, due to the smoothness of $\Psi(\boldsymbol{u})$ and $\boldsymbol{G}(\boldsymbol{u})$ and their closeness in both values and derivatives (again by randomization), the continuity guarantees the small values for f^i and g^i during the prediction.

Remark 2.9. Theorem 2.7 allows us to bound the error between the neural network prediction with the exact solution of the original PDEs (2.1) provided that an error estimation of the solution of the discretized equation (2.3) is given. Indeed, suppose the error in the discretized solution $\boldsymbol{u}^n$ and the exact solution $\boldsymbol{u}(t_n)$ at time t_n is bounded by $\mathcal{O}(\Delta t + h^p)$, where h is the mesh size and p is the order of accuracy of the underlying spatial discretization. Then by a simple application of triangle inequality we have

$$\tilde{\boldsymbol{u}}^n - \boldsymbol{u}(t_n) = \mathcal{O}\left(\Delta t + h^p + \sum_{k=1}^{n} \left(\Pi_{i=k+1}^{n} g^i\right) f^k\right),$$

which shows that in order to get the optimal accuracy and computational effort we ideally need to balance not only the temporal and spatial discretization errors but also the error in the neural network. Clearly, balancing the former two is not that difficult from a numerical analysis point of view, but balancing also the network error is challenging as it depends on the actual training process and randomization.

3. Numerical results. In this section, we present several numerical results using the proposed model-constrained tangent slope neural network (`mcTangent`) approach for transport equation (subsection 3.1), viscous Burger's equation (subsection 3.2), and Navier-Stokes equation (subsection 3.3). As shall be shown, `mcTangent` solutions are—thanks to the model-constrained term and data randomization—stable and capable of producing accurate approximations far beyond the training time horizons. In subsection 3.4 we provide detailed information on parameter tuning, randomness setting, and the cost for both training and testing.

Five hyperparameters of interest are the number of training samples, noise level δ, sequential machine learning steps S, sequential model-constrained learning steps R, and regularization parameter α. For convenience, we shall conventionally write them in a group. For example the $(d600, 2\%, 1, 1, 0)$ setting means we consider 600 training data samples, 2% noise, S = 1, R = 1, and $\alpha = 0$. In order to ensure the fairness between simulations and the comparison among approaches, we use fixed random keys for training and testing data generation, for adding noise, and for neural network parameter initialization We implement our approach and perform all computations in `JAX` [4]. We would like to point out that all computations (training, testing, and predicting) are done with single precision accuracy.

3.1. One-dimensional (1D) wave/transport equation. The 1D wave equation considered in this section is given by

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0,$$

with the wave speed $c = 1$, the spatial domain $x \in (0, 1)$, and time horizon $t \in (0, T)$. The equation is equipped with an initial condition $u(x, 0) = u_0(x)$ and periodic boundary condition. We are interested in real-time approximate solutions of the wave equation for any initial condition $u_0(x)$.

Data generation. In this problem, the initial condition samples are drawn from

$$u_0(x) = \sum_{i=1}^{5} a_i \sin(2\pi x\, i) + \sum_{i=1}^{5} b_i \cos(2\pi x\, i),$$

where a_i, b_i are distributed by the standard normal distribution with zero mean and unit variance, i.e., $a_i, b_i \sim \mathcal{N}(0, 1)$. We solve the wave equation with the forward Euler scheme for the temporal derivative and the first order upwind finite difference scheme for spatial derivative. The time horizon is chosen as $T = 5 \times 10^{-2}$. A fine space-time mesh with $n_x = 10000$ points in space and $n_t = 2000$ points in time is deployed to achieve highly accurate solutions. The training data samples are obtained by extracting the high resolution solutions on a coarser uniform space-time mesh with $n_x = 200$ and $n_t = 100$. In this simple problem, we generate a fixed training data set of 200 initial conditions. Note that we aim to predict long-time solutions, $t \in (0, 3)$, from the short-time training data in the interval $t \in \left(0, 5 \times 10^{-2}\right)$.

Neural network architecture. Because of the linear nature of the problem and the first order upwind finite difference scheme, a linear neural work is sufficient to approximate the resulting tangent slope. The linear neural network is defined as

$$\Psi\left(\boldsymbol{u}^i\right) = \boldsymbol{W}\boldsymbol{u}^i + \mathbf{b},$$

where the weights $\boldsymbol{W} \in \mathcal{R}^{n_x \times n_x}$, and the bias $\mathbf{b} \in \mathcal{R}^{n_x}$. To train, we use `ADAM` [20] optimizer with default parameters and the learning rate of 10^{-3}. We determine the best combination of weights and biases (and hence the final trained network) as the one that provides the lowest accumulated mean square error for 500 time steps for the test sample. Specifically, during the training process, at each epoch, we solve for the predictions from the test initial condition with the current-epoch learned network. The accumulated mean-square error between predictive solutions and ground truth solutions is calculated at the 500th time step to determine the "optimal" network.

Long-time predictions. Shown in Figure 3 is the mean-square error between true (high resolution) solutions and predicted ones obtained by various neural networks, each of which is trained with both randomized and noise-free training data.

For pure data-driven machine learning networks ($\alpha = 0$, and thus no model-constrained term), we observe that noise-free data trained networks outperform those trained with noisy data ones. This is not surprising as for this linear problem, as predicted by Lemma 2.2, one can obtain linear networks that accurately learn the tangent slope with sufficiently rich data. Therefore, the predictions by the learned linear networks are almost the same as the ground truth solutions. On the contrary, training with noisy data causes the neural network to predict solutions with a small amount of error such that it adapts to (possibly overfits) the amount of noise in the ground-truth solutions. Figure 4 presents the weight matrix and bias vector for two

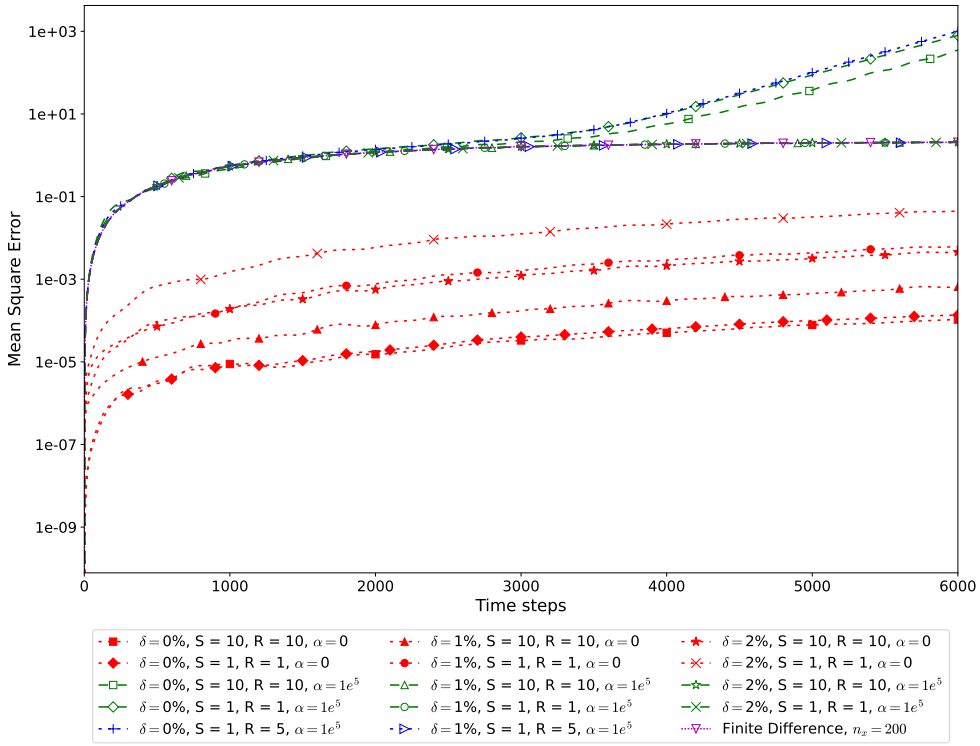

Fig. 3: **Wave/transport equation**. Comparison between different neural network approaches with/without randomization.

cases $(d200, 0\%, 10, 1, 0)$ and $(d200, 0\%, 1, 1, 0)$ with noise-free data. It can be seen that both networks are almost identical and both have only an upper diagonal with a large magnitude. We also note that the bias vector is relatively small and thus we ignore this bias vector in the subsequent comparisons. We present the test predicted solutions for the setting $(d200, 0\%, 1, 1, 0)$ in Figure 5. As the network fits the tangent slope for high-resolution data, accurate results are preserved far beyond the training time horizon, while finite difference results on the same coarse grid show a severe diffusion/dissipation effect. Furthermore, settings with a large number of sequential steps such as $(d200, 0\%, 10, 1, 0)$, $(d200, 1\%, 10, 1, 0)$ and $(d200, 2\%, 10, 1, 0)$ yield more accurate neural networks than their counterparts with $S = 1$. The reason is that long sequential training reduces the prediction error.

For model-constrained neural networks, we use $\alpha = 10^5$ as the regularization parameter for all cases. We tested with different values for α and almost the same results are obtained for larger values, while smaller values make neural networks perform similarly to the pure data-driven machine learning networks. It can be seen in Figure 3 that training with randomized data returns neural networks, regardless of S, R values, as good as the coarse finite difference approximation with $n_x = 200$. This is expected as we constrain the training with a coarse finite difference model. The trained weight matrices and bias vectors for these neural networks corresponding to three settings $\left(d200, 1\%, 10, 1, 10^5\right)$, $\left(d200, 1\%, 1, 1, 10^5\right)$ and $\left(d200, 1\%, 1, 5, 10^5\right)$ are shown in Figure 6. Again, the bias vectors do not have a significant role in the predictions. Note

that, unlike those from purely data-driven in Figure 4 which have arbitrary structure, the model-constrained weight matrices, after ignoring small elements, have the same structure as the first-order upwind scheme matrix. Among these neural networks, the long sequential model-constrained network with $R = 5$ is closest to the first-order upwind scheme matrix. It is not surprising as the neural network is constrained to satisfy the first-order upwind scheme in multiple time steps. On the other hand, the neural network trained with noise-free data shows instability starting from the 2000th time step in long-term predictions. This instability is due to the lack of regularizations as compared to the randomized cases for which regularizations are explicit via the model-constrained term and implicit via randomization (see subsection 2.4).

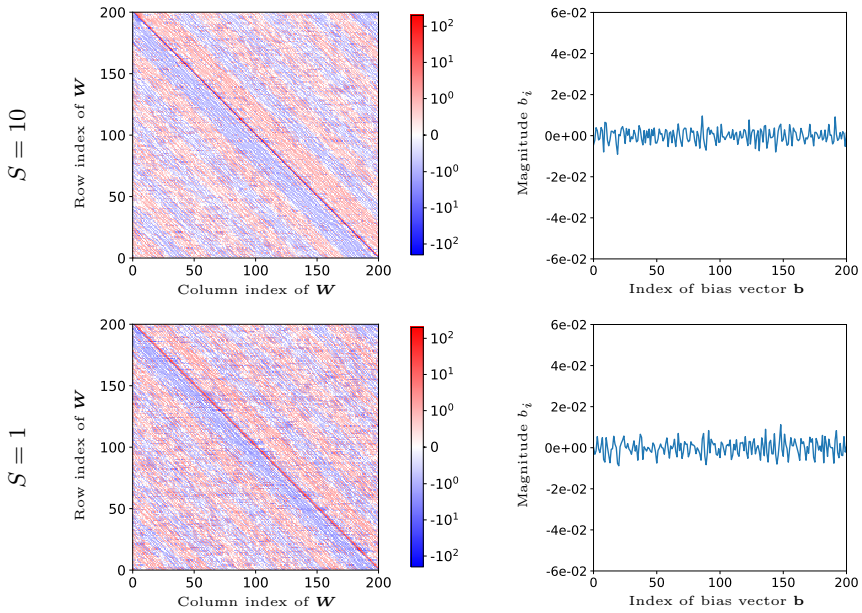

Fig. 4: **Wave/transport equation**. Pure data-driven trained linear neural network parameters: weight matrix heat maps (*left column*) and bias vector magnitudes (*right column*) with $\alpha = 0, \delta = 0\%$.

Implicit time integration with learned network. One of the advantages of our proposed tangent slope learning approach is that once trained the learned tangent slope can be used with any time discretization method. To demonstrate this, we use the learned neural network tangent for the setting $(d200, 0\%, 1, 1, 0)$ with both backward and forward Euler schemes using a time stepsize $\Delta t' = \frac{50}{3}\Delta t$ which is much larger than the training stepsize. It can be seen in Figure 7 that the forward Euler solutions blow up for both learned and true tangent slopes, which is obvious as the time stepsize is much larger than the stable time stepsize. Both approaches are stable with implicit integration and the results are comparable (though the learned tangent slope was trained with a smaller time step size).

Direct learning versus `mcTangent` slope learning. We now compare our tangent slope learning and direct learning. Here, by direct learning we mean learning the map from $\boldsymbol{u}^i$ to $\boldsymbol{u}^{i+1}$ for two consecutive time steps. Clearly, unlike the former, the latter is tailored, and thus limited, to a particular space-time discretization. To be fair,

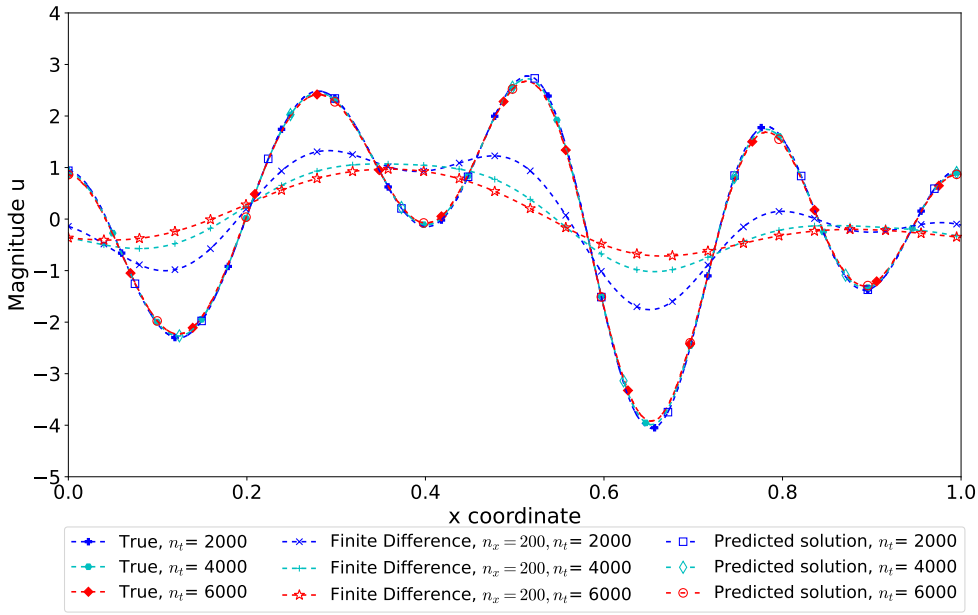

Fig. 5: **Wave/transport equation**. The predicted solutions at time steps $n_t = 2000, 4000, 6000$ by learned neural network corresponding to $(d200, S = 1, R = 1, \alpha = 0, \delta = 0\%)$, finite difference solutions on coarse grid with $n_x = 200$, and the high resolution solutions (True).

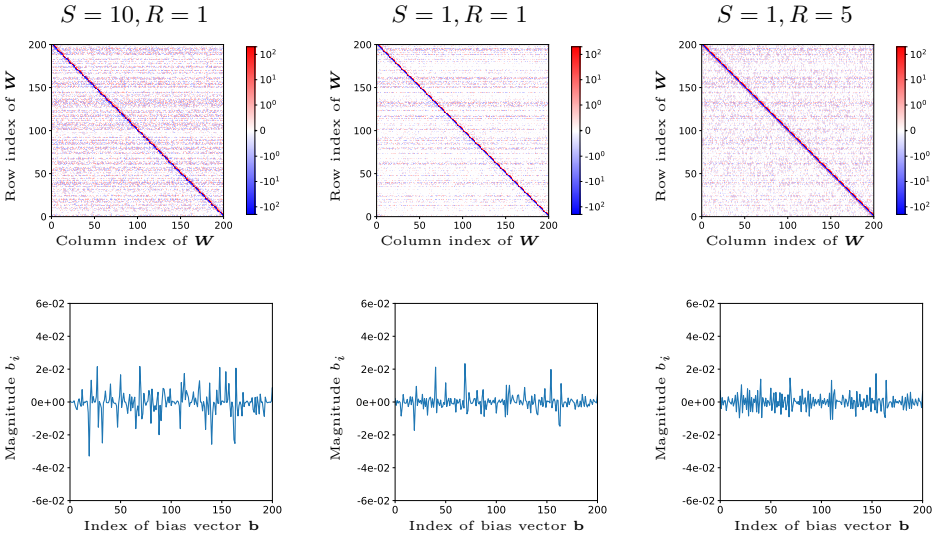

Fig. 6: **Wave/transport equation**. Trained model-constrained linear neural network parameters: weight matrix heat map (*top row*) and bias vector magnitude (*bottom row*) with $\alpha = 1e^5, \delta = 1\%$.

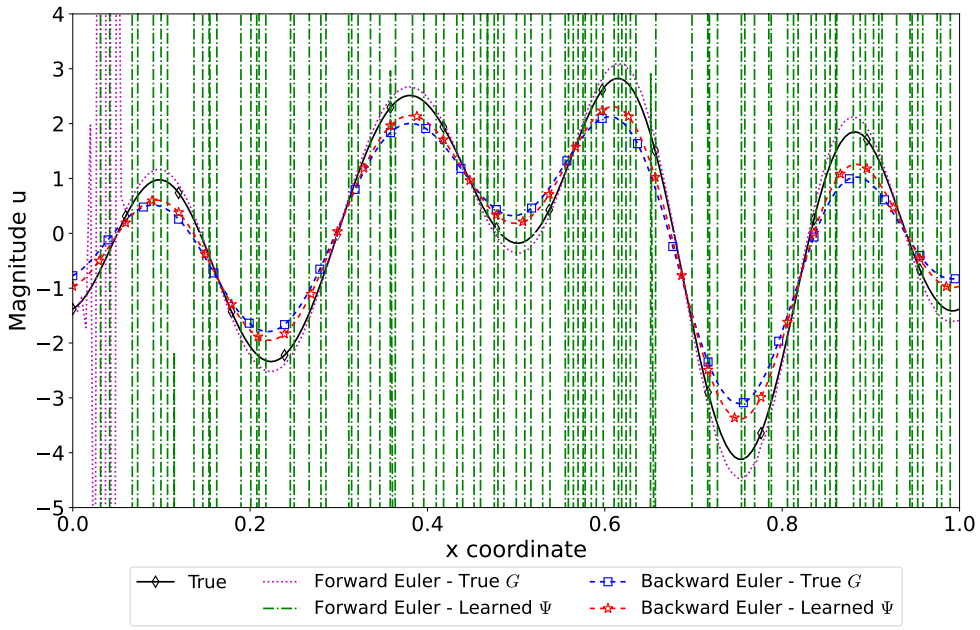

Fig. 7: **Wave/transport equation**. Predicted solutions at $t = 0.1$ obtained by forward Euler scheme and backward Euler scheme with the true tangent slope G and learned neural network counterparts with time stepsize $\Delta t' = \frac{50}{3}\Delta t = \frac{25}{3} \times 10^{-3}$.

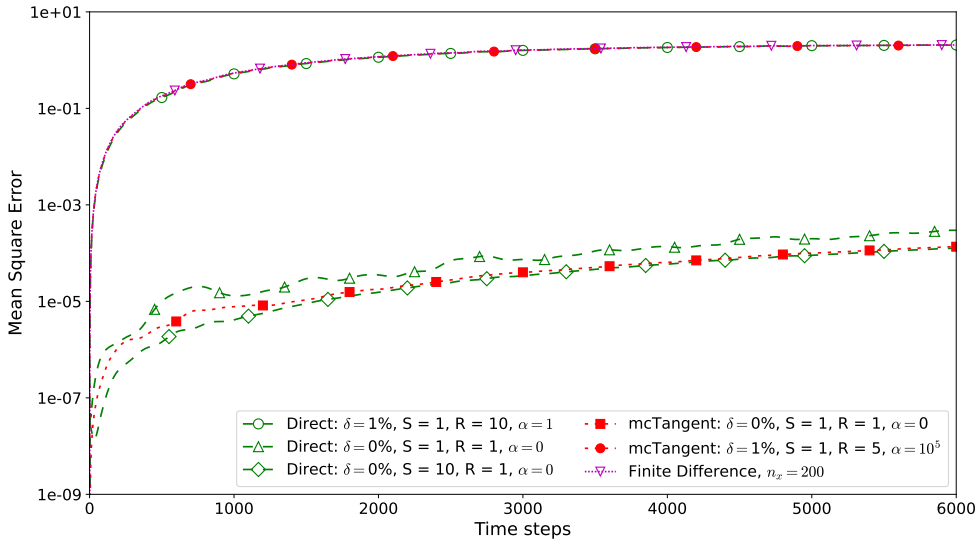

Fig. 8: **Wave/transport equation**. Comparison between direct neural networks (Direct) and tangent slope neural networks (`mcTangent`).

we also use the linear network with zero bias for the direct learning approach. Figure 8

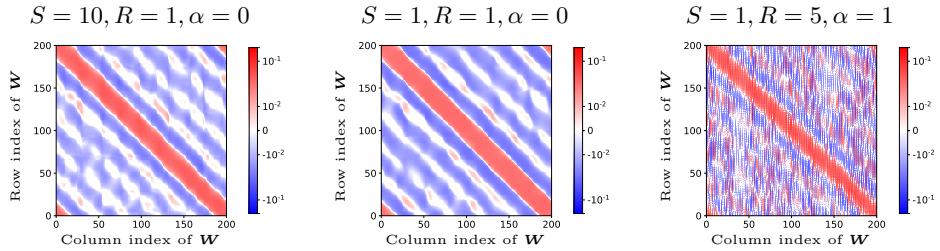

Fig. 9: **Wave/transport equation**. Heat map of weight matrix of direct linear neural networks.

presents the mean-square error of predictions obtained by direct neural networks and tangent slope networks, both with and without model-constrained terms. As can be seen, both direct and tangent slope neural networks are comparable in terms of accuracy. However, the learned weight matrices of direct neural networks do not have the pattern of the underlying space-time discretization matrices, and this can be observed from Figure 9. That is, while our tangent slope approach preserves the structure of spatial discretization, the direct approach, which seems to be natural, does not.

3.2. 2D Burger's equation. We consider the following viscous 2D Burger's equations

$$\begin{aligned}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} &= \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \\
\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} &= \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right),
\end{aligned}$$

where $x, y \in [0, 1]$ and $t \in (0, T]$. The boundary condition is periodic and the initial velocity is given by $v(x, y, 0) = v_0(x, y) = 1$ and $u(x, y, 0) = u_0(x, y)$. We take the viscosity coefficient to be $\nu = 10^{-2}$. We aim to predict x-velocity u in the time interval $t \in (0, 1.5)$ given an initial velocity $u_0(x, y)$ at $t = 0$.

Data generation. We draw periodic samples of $\boldsymbol{u}$ using the truncated Karhunen-Loève expansion

$$u_0(x, y) = \exp\left(\sum_{i=1}^{15} \sqrt{\lambda_i}\, \omega_i(x, y)\, z_i\right),$$

where $\mathbf{z} = \{z_i\}_{i=1}^{15} \sim \mathcal{N}(0, \mathbf{I})$, and (λ, ω) are eigenpairs obtained by the eigendecomposition of the covariance operator $7^{\frac{3}{2}}(-\Delta + 49\mathbf{I})^{-2.5}$, where Δ is the Laplacian operator, with periodic boundary conditions. Training data corresponding to each initial velocity is generated from a 128×128 high-resolution spatial mesh and 1000 time steps for the time horizon $T = 0.1$ using finite difference method. These high resolution solutions are down-sampled on a coarser mesh of 100 time steps ($\Delta t = 10^{-3}$) and 32×32 spatial mesh. These down-sampled solutions are treated as true solutions for the training process. Meanwhile, we draw 10 test initial velocity samples independently, and the corresponding test data set of 10 samples is created in the same manner. However, the time horizon $T = 1.5$ for test samples is chosen—much larger

than the trained time horizon—with time stepsize $\Delta t = 10^{-3}$. This helps us test the accuracy and stability of neural network solutions beyond the training regime.

Neural network architecture. We use a shallow network of one layer with 5000 neurons for all cases to approximate the tangent slope of Burger's equations. Note that we have compared the one-layer network with two- and three-layer networks with different numbers of neurons ranging from 100 to 5000. These deeper networks perform poorly with small data sets and are improved with large data sets in which the shallow one has comparable performance. Note that one-layer neural network approximation capabilities are rigorously justified by past universal approximation theories (see, e.g. [9, 13, 28, 18]) and our current work [7]. Thus we shall use a one-layer neural network for all numerical results. In addition, ReLU [32] is used as the activation function. `ADAM` optimizer is used with the learning rate of 10^{-4} and the training batch size is 40 samples. For this example, reasonably optimal weights/biases are the ones giving the lowest accumulated mean square error after 1500 time steps for 10 test data. We take $\alpha = 10^5$ for the regularization parameter as this gives the best results from our numerical experiments (not shown here).

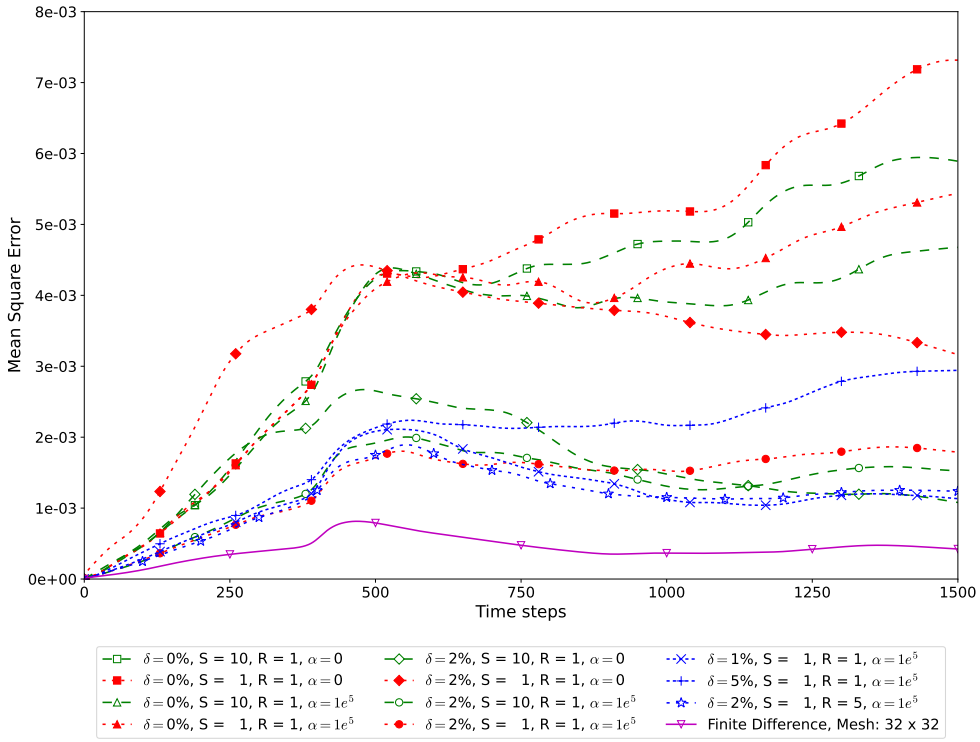

Fig. 10: **Burger's equations**. Comparison of mean square error among different neural networks trained with 200 data with/without noise. Recall that $\alpha = 0$ corresponds to the pure data-driven neural network training without model-constrained terms. Forward solver denotes the numerical solution on 32×32 spatial mesh.

Comparison of different learned neural networks. Figure 10 presents the comparison of mean square error obtained by different learned neural networks with the data set of 200 samples. It can be seen that, in general, the model-constrained

neural networks are far better than their pure data-driven counterparts (i.e. with $\alpha = 0$). Additionally, long sequential machine learning trainings with $S = 10$ provide slightly better accuracy than $S = 1$, except for the noisy data with pure data-driven network in which the improvement is significant.

For model-constrained neural networks, long sequential training results with $S = 10$ in two settings $\left(d200, 0\%, 10, 1, 10^5\right)$ and $\left(d200, 2\%, 10, 1, 10^5\right)$ show an marginal improvement compared to short sequential training with $S = 1$ in two settings $\left(d200, 0\%, 1, 1, 10^5\right)$ and $\left(d200, 2\%, 1, 1, 10^5\right)$. Therefore, $S = 1$ is sufficient and we use it for the rest of numerical results with model-constrained neural networks. Figure 10 shows that using 5% noise causes the neural network corresponding to $\left(d200, 5\%, 1, 1, 10^5\right)$ to perform poorly, while 1% noise gives almost the same accuracy as 2% noise. It is noticeable that the long sequential model-constrained training with $R = 5$ $(d200, 2\%, 1, 5, 10^5)$ yields higher accuracy than the others. However, large R is more computationally expensive since many passes through the back-propagation computational graph are needed.

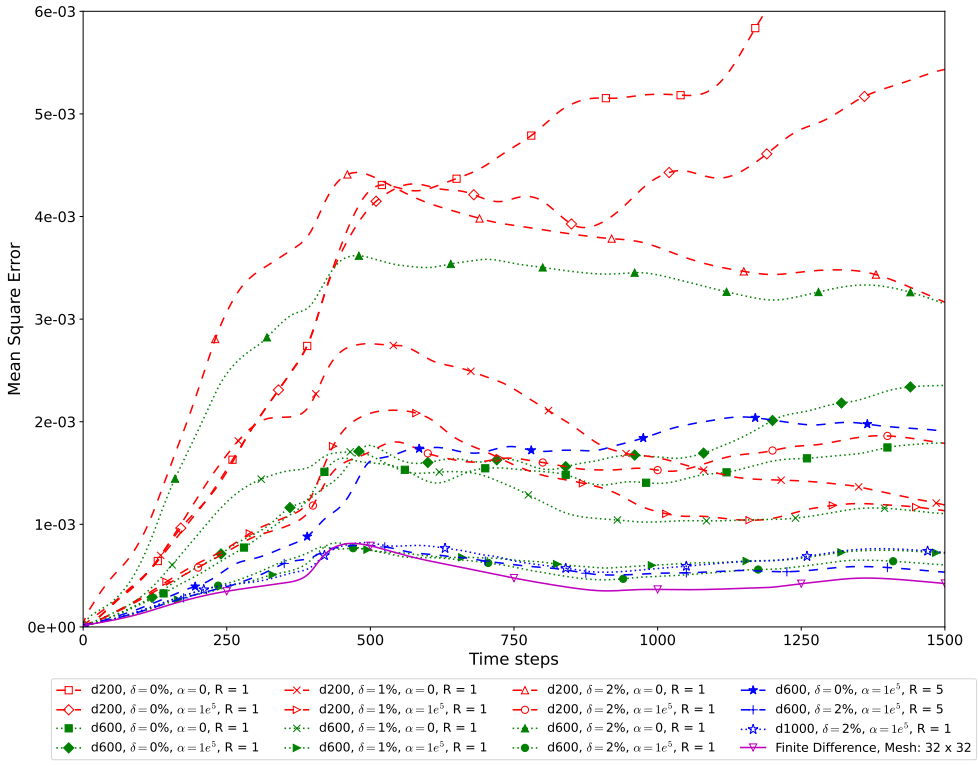

Fig. 11: **Burger's equations**. The mean square error versus the number of time steps for various learned neural networks using 200, 600, and 1000 data samples with $S = 1$.

Long-time predictions with small and large training data sets. As discussed above, since long sequential machine learning training does not provide significant improvement, we consider $S = 1$ for numerical results using large data sets in Figure 11. As can be seen, compared to 200 data samples, training with 600 data

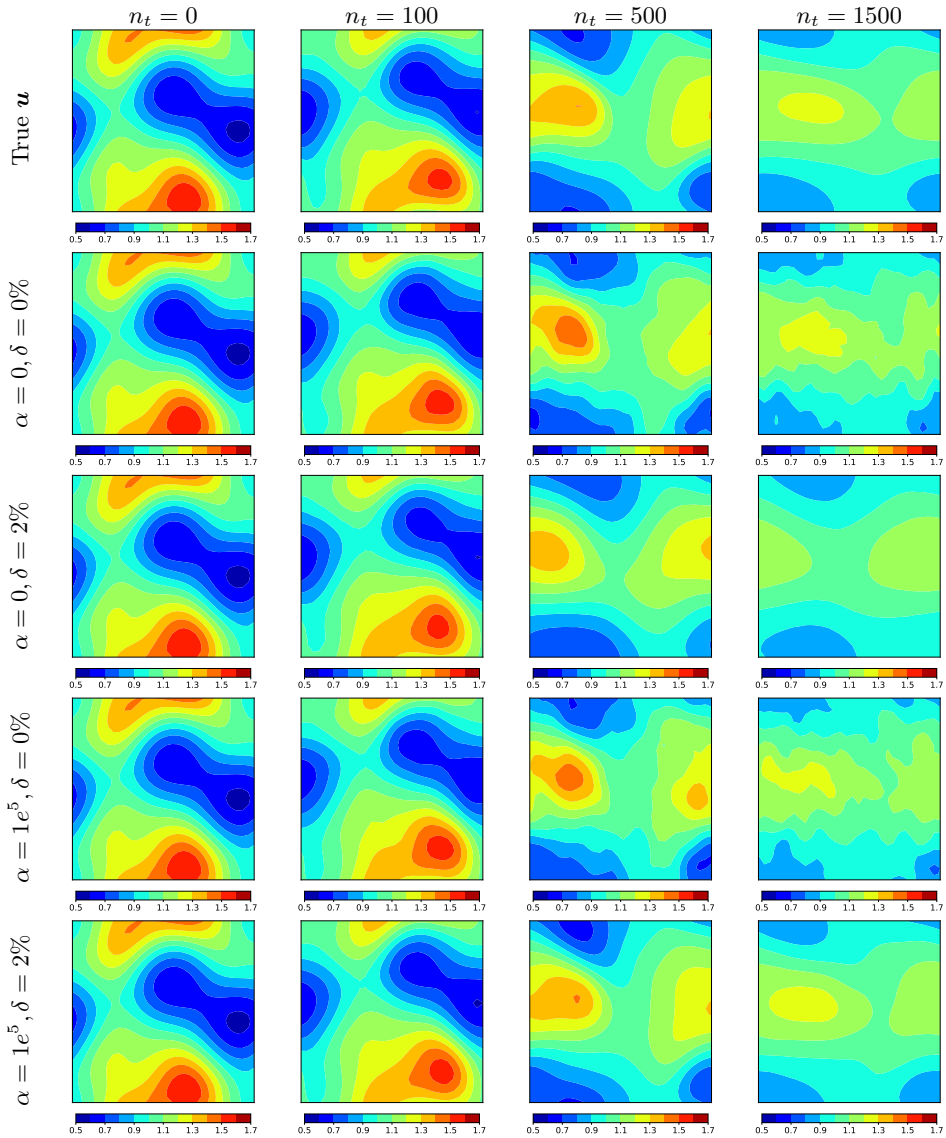

Fig. 12: **Burger's equations**. Predicted solutions at different time steps (n_t) obtained by various learned neural network tangents with 600 training data samples and $S = 1$ and $\Delta t = 10^{-3}$. *Top row*: True high-resolution solution; *Second row*: pure data-driven network without data randomization; *Third row*: pure data-driven network with noisy data; *Fourth row*: model-constrained network without data randomization; *Fifth row*: model-constrained network with noisy data.

samples provides more accurate predictions. Moreover, model-constrained neural networks with randomized data are the most accurate among others (model-constrained with noise-free data and pure machine learning with/without randomized data). We can also observe that using more than 600 data samples does not provide significant

improvements but is more expensive. Unlike the case with 200 data samples, long and short sequential model-constrained trainings with $R = 5$ and $R = 1$, respectively, provide similar results for 600 data samples. This is expected as richer data reduces the significance of the model-constrained term.

As shown in Figure 12, predicted solutions obtained by the model-constrained approach (the fifth row) with data randomization are in good agreement with the ground-truth counterparts. On the contrary, the pure data-driven approach with data randomization (the third row) shows poor long-time predictions. We also observe that both pure data-driven learning solutions and model-constrained solutions (the second and fourth rows, respectively) without randomization are unstable for long-time predictions. It is not surprising since both do not have sufficient regularizations compared to the randomized cases in which extra regularizations are implicitly performed (see subsection 2.4). Moreover, regularizations induced by data randomization shown in subsection 2.4 stabilize the network predictions and this can be clearly seen by comparing the third and the second rows for the pure data-driven learning approach, and by comparing the fourth and the fifth rows for the model-constrained learning approach.

Figure 13 plots the contours of the learned and the true tangent slopes. Clearly, the learned model-constrained tangent slope with data randomization provides the best agreement with the true tangent slope. This is not surprising as both the governing equations (explicit via model-constrained term) and sufficient regularizations (implicit via data randomization) are incorporated.

Predictive flexibility in time for `mcTangent` approach As discussed above, one appealing feature of tangent slope learning is that once trained it can be used to solve for approximate solutions with smaller or larger time stepsizes, despite the fact that it is trained based on a particular spatial discretization. On the contrary, direct learning is attached to a space-time discretization. Figure 14 shows the model-constrained tangent slope learning solutions and contours of the corresponding learned tangent slope at various times for the setting $\left(d600, 2\%, 1, 1, 10^5\right)$. Here we use half of the training time stepsize $\Delta t'' = \frac{1}{2}\Delta t = 5 \times 10^{-4}$. It can be seen that these predictions are indistinguishable from ones (the fifth row in Figure 12 for prediction solutions and the fifth column in Figure 13 for predicted tangent slopes) obtained by using the training time stepsize $\Delta t = 10^{-3}$ with the same learned network.

Implicit time integration with learned network. Another appealing feature of tangent slope learning is that once trained it can be deployed with any time discretization schemes. We use the learned network from the setting $\left(d600, 2\%, 1, 1, 10^5\right)$ together with the backward Euler method with a larger time stepsize $\Delta t' = 12.5\Delta t = 1.25 \times 10^{-2}$, where $\Delta t = 10^{-3}$ is the training stepsize. Shown in Figure 15 are predicted solutions at $t = \{0, 0.1, 0.5, 1.5\}$ corresponding to $0, 100, 500, 1500$th time steps. We observe that solutions using the forward Euler scheme, regardless of using the true tangent slope or learned one (second and third rows, respectively), are unstable as the time stepsize $\Delta t'$ is too big for stability. On the contrary, using the backward Euler scheme, `mcTangent` solutions are comparable to the true counterparts (fourth and fifth rows, respectively). Clearly, due to large time stepsize, both are more diffusive compared to the true solutions with small time stepsize Δt in the first row.

Direct learning versus `mcTangent` slope learning. Recall that by direct learning we mean learning the map from $\boldsymbol{u}^i$ to $\boldsymbol{u}^{i+1}$ for two consecutive time steps. We investigate the difficulty and complexity of direct learning. Specifically, we use a data set with 600 samples with/without data randomization to learn the neural network with one layer of 5000 neurons that maps velocities from one step to the

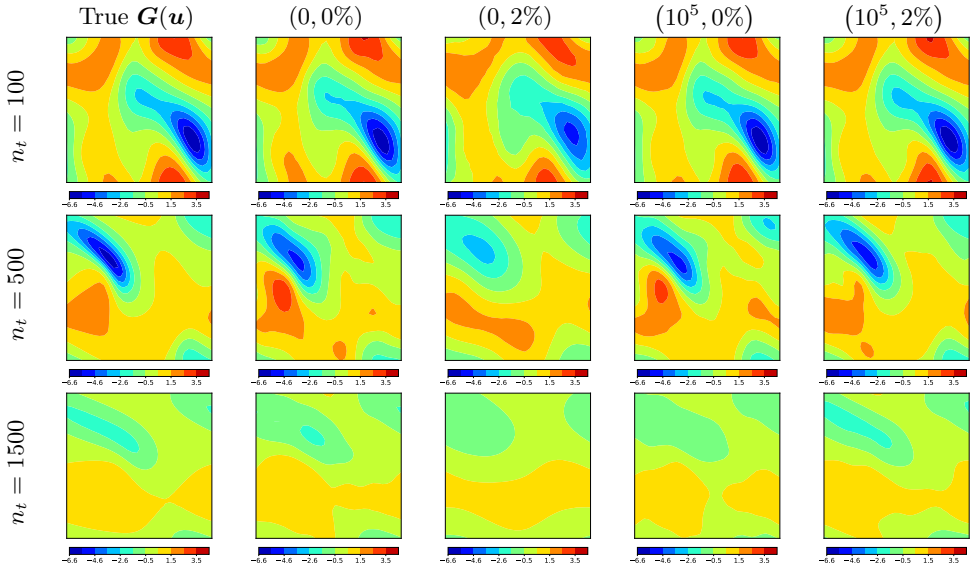

Fig. 13: **Burger's equations**. Contours of True and various learned tangent slopes. Contours are plotted at different time steps n_t for four different combinations of regularization parameter α and noise level δ. For all cases, we use 600 data samples, $S = 1$, and $\Delta t = 10^{-3}$. *First column*: True tangent slope ; *Second column*: pure data-driven tangent slope without data randomization $(0, 0\%)$; *Third column*: pure data-driven tangent slope with data randomization $(0, 2\%)$; *Fourth column*: model-constrained tangent slope without data randomization $\left(10^5, 0\%\right)$; *Fifth column*: model-constrained tangent slope with data randomization $\left(10^5, 2\%\right)$.

next. As shown in Figure 16, the direct learning approach (with the best combination of hyperparameters) for the setting $(d600, 2\%, 1, 3, 2)$ is less accurate for both short-time and long-time predictions compared to the tangent learning counterpart with even a smaller data set of 200 samples with the setting $\left(d200, 2\%, 1, 1, 10^5\right)$. Interestingly, unlike the tangent learning approach, the direct learning approach, both pure data-driven and model-constrained approaches, trained with randomized data is less accurate compared to noise-free data in short sequential training $S = 1$. On the other hand, data randomization does not have visible benefits on long sequential training $S = 10$. Specifically, both $(d600, 0\%, 10, 1, 0)$ and $(d600, 2\%, 10, 1, 0)$ settings behave similarly.

Also seen in Figure 16, among pure data-driven networks ($\alpha = 0$) with direct learning, long sequential machine learning training with $S = 10$ is the most accurate. Model-constrained network with direct learning for the setting $(d600, 2\%, 1, 1, 10)$ is much more accurate compared to the pure data-driven network with direct learning for the same setting. Moreover, sequential model-constrained networks for $R = 2, 3$ corresponding to two settings $(d600, 2\%, 1, 2, 2)$ and $(d600, 2\%, 1, 3, 2)$ are comparable to much longer sequential machine learning network with $S = 10$ for the setting $(d600, 2\%, 10, 1, 0)$. In the presented results, it is important to point out that for direct learning, care must be taken in choosing a good regularization parameter α. For example, $\alpha = 2$ is good for $R = 2, 3$, but $\alpha = 10$ is good for $R = 1$. On the

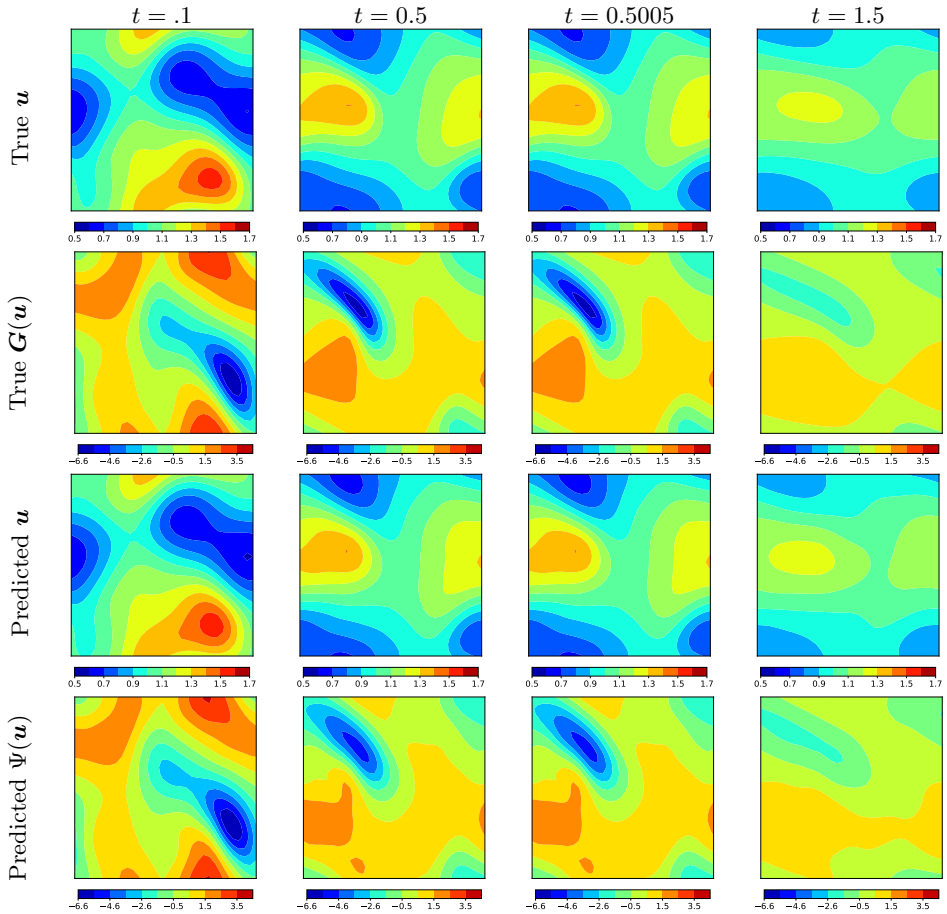

Fig. 14: **Burger's equations**. Predicted solutions and tangent slope using by `mcTangent` neural networks with $\left(d600, 2\%, 1, 1, 10^5\right)$, and time step $\Delta t'' = \frac{1}{2}\Delta t = 5 \times 10^{-4}$. *First row*: True high-resolution solutions; *Second row*: contours of True tangent slope, $\boldsymbol{G}(\boldsymbol{u})$; *Third row*: Predicted `mcTangent` solutions, $\boldsymbol{u}$; *Fourth row*: contours of `mcTangent` tangent slope, $\Psi(\boldsymbol{u})$.

contrary, tangent learning is more robust. In particular, a single $\alpha = 10^5$ works well for all settings. Solutions predicted by direct and tangent learnings (both with model-constrained terms) for $(d600, 2\%, 1, 3, 2)$ and $\left(d200, 2\%, 1, 1, 10^5\right)$, respectively, are shown in Figure 17. As can be observed, tangent learning solutions with even smaller data set $\left(d200, 2\%, 1, 1, 10^5\right)$ are much more accurate than the direct learning with $(d600, 2\%, 1, 3, 2)$. This is due to the fact that direct learning tries to learn a mixed space-time discretization, which is more difficult than learning only the spatial discretization in tangent learning.

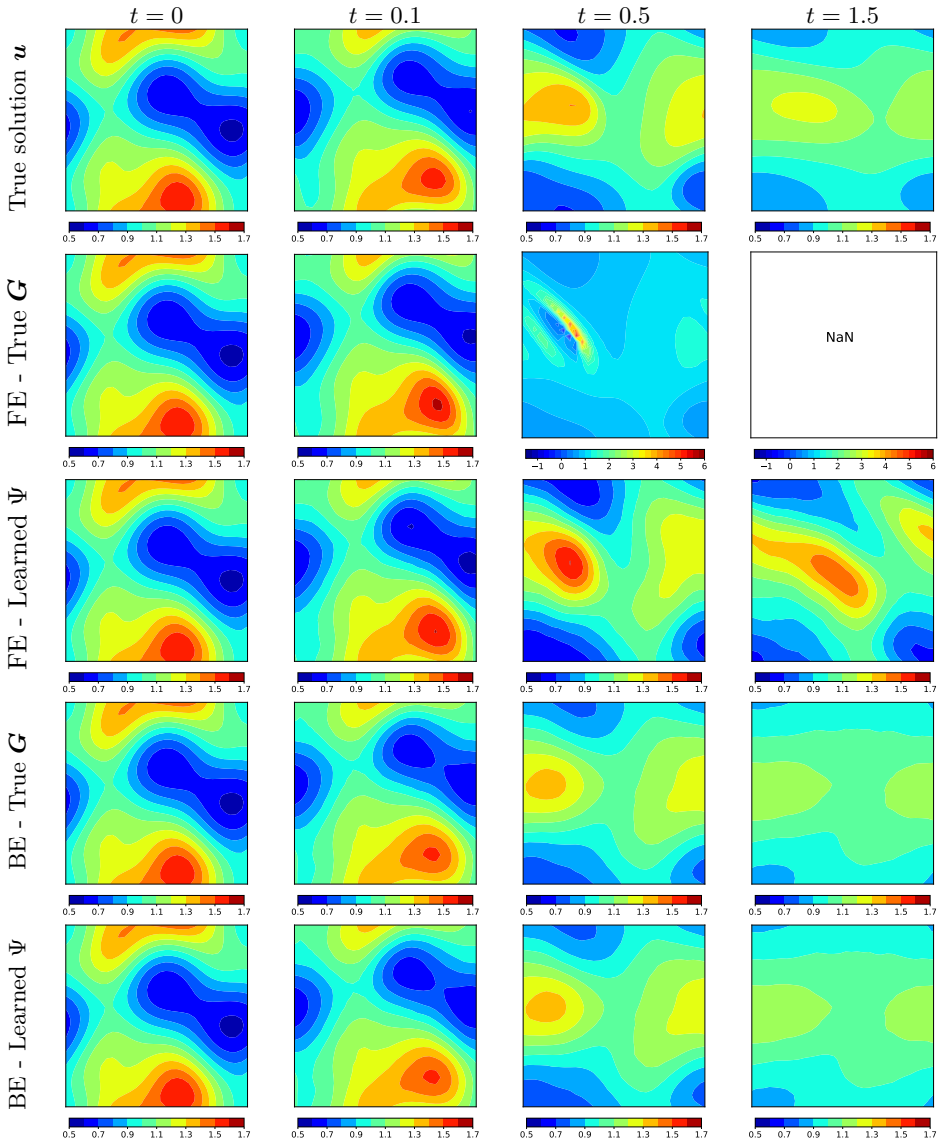

Fig. 15: **Burger's equations**. Predicted solutions at different times obtained by Forward Euler (FE) scheme and Backward Euler (BE) scheme using large stepsize $\Delta t' = 12.5\Delta t = 1.25 \times 10^{-2}$ with the true tangent slope $\boldsymbol{G}$ and the learned neural network Ψ for the setting $\left(d600, 2\%, 1, 1, 10^5\right)$.

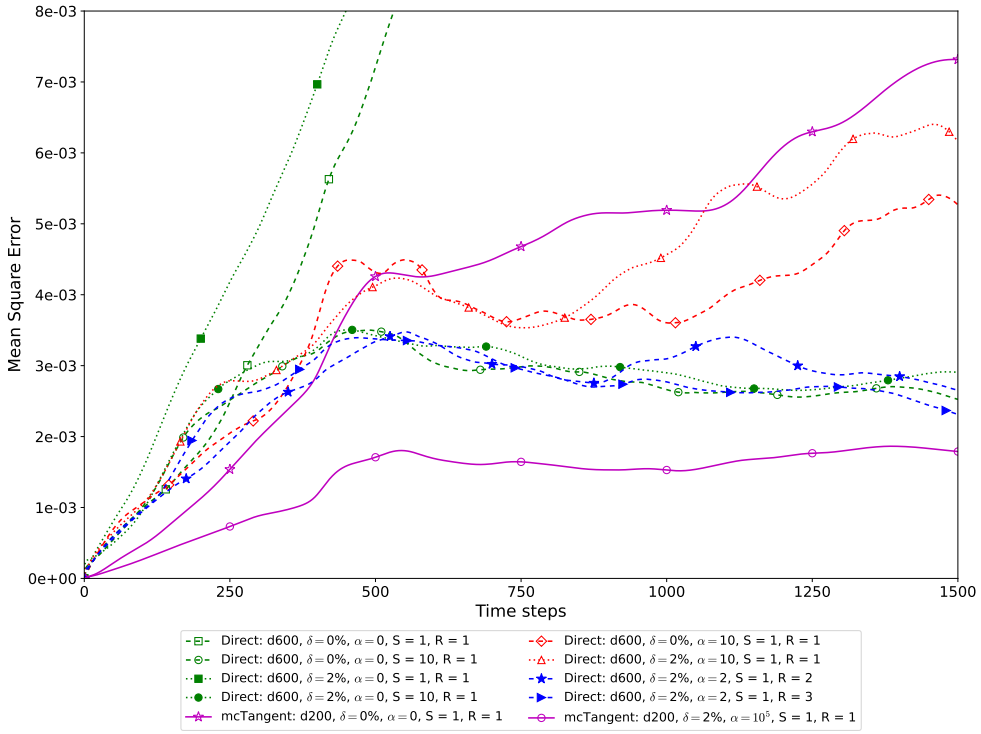

Fig. 16: **Burger's equations**. The mean square error versus time steps obtained by the direct learning approach (Direct) using 600 training samples and the tangent learning approach (`mcTangent`) using 200 training samples.

3.3. Navier-Stokes equation. The vorticity form of the 2D Navier-Stokes equation for viscous and incompressible fluid [24] can be written as

$$\begin{aligned} \partial_t u(x,t) + v(x,t)\cdot\nabla u(x,t) &= \nu\Delta u(x,t) + f(x), && x\in(0,1)^2, t\in(0,T] \\ \nabla\cdot v(x,t) &= 0, && x\in(0,1)^2, t\in(0,T] \\ u(x,0) &= u_0(x), && x\in(0,1)^2 \end{aligned}$$

where $v(x,t)$ is the velocity field, $u=\nabla\times v$ is the vorticity, u_0 is the initial vorticity, $f(x) = 0.1\left(\sin\left(2\pi\left(x_1+x_2\right)\right)+\cos\left(2\pi\left(x_1+x_2\right)\right)\right)$ is the forcing function and $\nu = 10^{-3}$ is the viscosity coefficient. Our goal is to solve for the vorticity $u(x,y,t)$ given the initial condition u_0 at $t=0$ by a trained tangent network Ψ .

Data generation. Data pair $(\boldsymbol{u},\boldsymbol{y})$ is generated by a similar procedure outlined for Burger's equation problem in subsection 3.2. In particular, we draw samples of $\boldsymbol{u}_0$ using the truncated Karhunen-Loève expansion

$$\boldsymbol{u}_0 = \sum_{i=1}^{15}\sqrt{\lambda_i}\,\omega_{\mathbf{i}}(x)\,z_i,$$

where $z_i\sim\mathcal{N}(0,1), i=1,\ldots,15$, and (λ,ω) is eigenpairs obtained by the eigen-decomposition of the covariance operator $7^{\frac{3}{2}}\left(-\Delta+49\mathbf{I}\right)^{-2.5}$ with periodic boundary

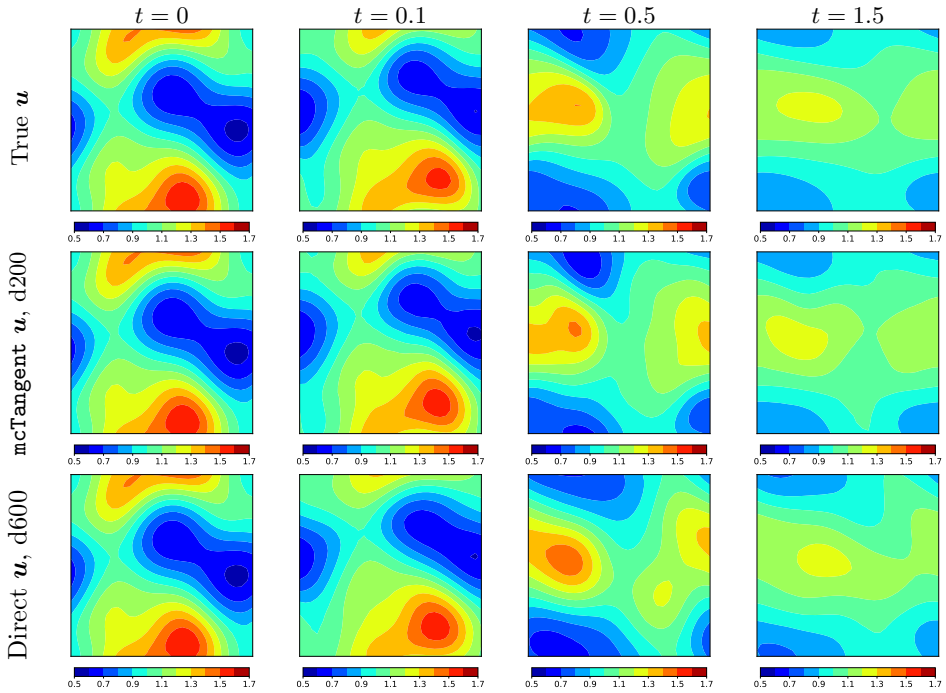

Fig. 17: **Burger's equations**. Solutions at different times, *First row*: True high-resolution solutions; *Second row*: learned tangent slope neural network solutions with $\left(d200, 2\%, 1, 1, 10^5\right)$; *Third row*: learned direct neural network solutions with $(d600, 2\%, 1, 3, 2)$.

conditions. Next, given initial vorticity $\boldsymbol{u}_0$, we solve the Navier-Stokes equation by the stream-function formulation with a pseudospectral method [24]. High resolution solutions are obtained on a uniform 128×128 spatial mesh and uniform 1000 time steps in $(0, 2)$. The high-resolution solutions are then down-sampled on a coarser mesh 32×32 in space and 200 uniform time steps, and they are used as the training data. To verify the accuracy of the learned neural network, we draw 10 test samples independently. It turns out that the Navier-stokes equation is much more challenging than Burger's equation, thus we use 200 time steps for each training data as opposed to 100 for the Burger equation. Similar to the above, to challenge the learned network we use 1500 time steps for testing, and thus the testing time horizon is far beyond the training time horizon.

Neural network architecture. With the same observation for the Burger equation in subsection 3.2, we use a shallow network of one layer with 5000 neurons using ReLU activation function. `ADAM` optimizer with default parameters is used with the learning rate of 2×10^{-4}, while the training batch size is 2 samples. The chosen "optimal" network is the one having the lowest accumulated mean square error after 1500 time steps for 10 testing samples. Following the wave and Burger examples, we pick a relatively large value for the model-constrained regularization parameter $\alpha = 10^5$.

Long-time predictions. Figure 18 shows the mean-square error of predictions

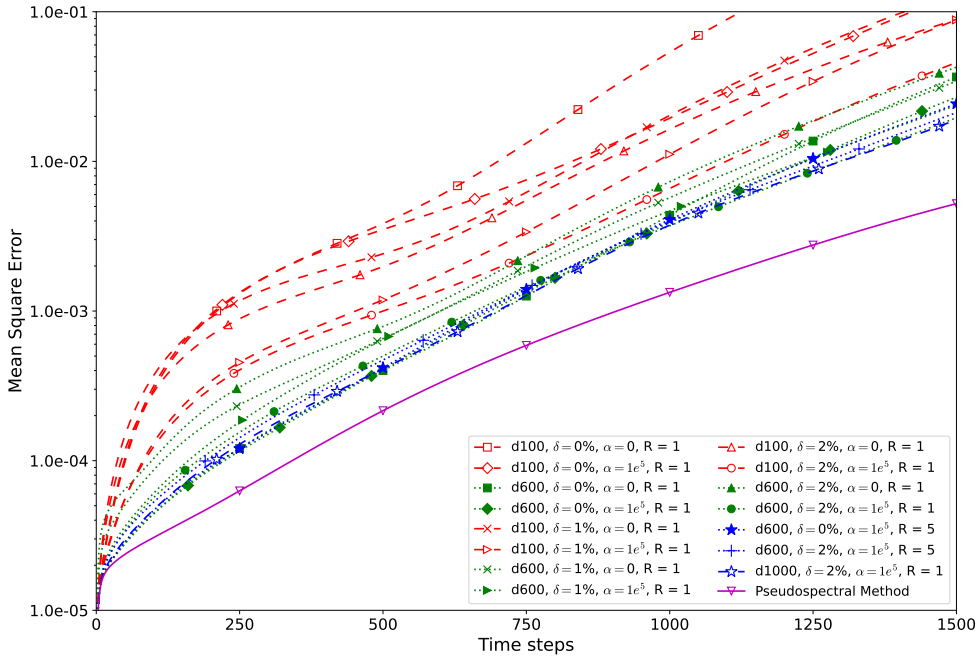

Fig. 18: **Navier-Stokes equation**. The mean-square error versus time steps obtained by the various learned neural network with $S = 1$.

and ground truth solutions as a function of time steps. It can be seen that training with a large data set with 600 samples provides much more accurate solutions than with small data set with 100 samples. On the one hand, among learned neural networks trained with 100 data samples, the model-constrained network with data randomization for $\left(d100, 2\%, 1, 1, 10^5\right)$ setting is far closer to the true solution than the other networks. This implies that the model-constrained approach has a significant contribution to producing accurate predictions in the context of small data. In the case of richer data set with 600 samples, networks with two settings $(d600, 0\%, 1, 1, 0)$ and $\left(d600, 0\%, 1, 1, 10^5\right)$ trained with noise-free data show a good performance in the short time predictions, while the long-time predictions deteriorate. Noticeably, between these two networks, the model-constrained one has more accurate predictions starting from the 500th time step. In the meantime, with the same data set with 600 samples, pure data-driven neural networks trained with higher noise level data give a higher error, for example, $(d600, 2\%, 1, 1, 0)$ neural network predictions are less accurate than those obtained from $(d600, 1\%, 1, 1, 0)$. In contrast, model-constrained network with 2% noise level $\left(d600, 2\%, 1, 1, 10^5\right)$ is superior to 1% noise level $\left(d600, 1\%, 1, 1, 10^5\right)$. Another point is that as we increase the sequential model-constrained value to $R = 5$, we obtain good predictions in both short-time and long-time intervals. Two model-constrained networks with $\left(d600, 0\%, 1, 5, 10^5\right)$ and $\left(d600, 2\%, 1, 5, 10^5\right)$ are comparable to the network with much larger data set $\left(d1000, 0\%, 1, 1, 10^5\right)$ without randomization. However, the noisy data network $\left(d600, 2\%, 1, 5, 10^5\right)$ outperforms the noise-free one $\left(d600, 0\%, 1, 5, 10^5\right)$ in the long-time predictions. In summary, model-constrained network with data randomization outperforms all other networks. Given a test initial

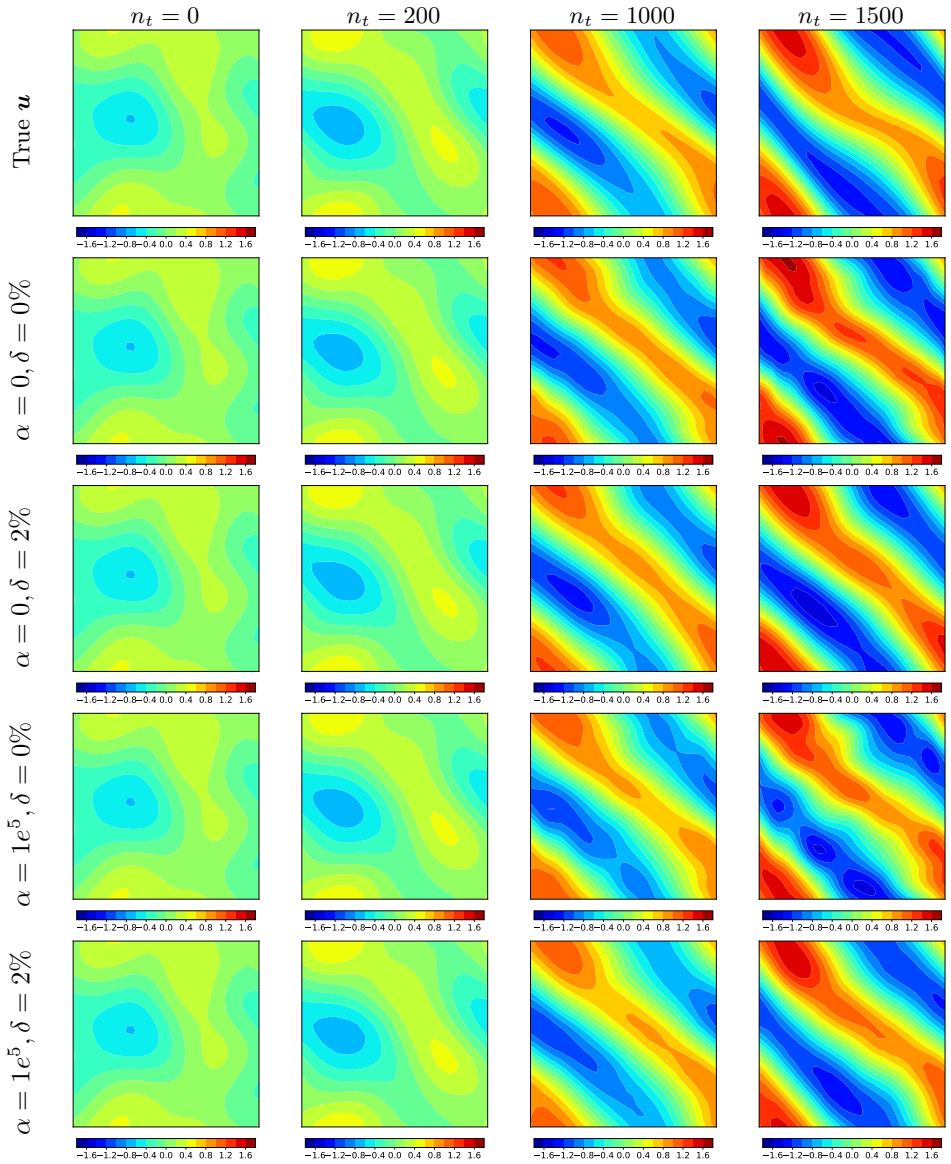

Fig. 19: **Navier-Stokes equation**. Predicted solutions at different time steps obtained by various trained networks with 600 data samples. *First row*: ground truth; *Second row*: pure data-driven network solutions with noise-free data; *Third row*: pure data-driven network solutions with randomized data; *Fourth row*: model-constrained network solutions with noise-free data; *Fifth row*: model-constrained network solutions with randomized data.

vorticity, the plots of predicted solutions obtained by different learned networks are shown in Figure 19. As can be seen, the model-constrained network with the setting $\left(d600, 2\%, 1, 1, 10^5\right)$ provides the most accurate solutions as opposed to others trained

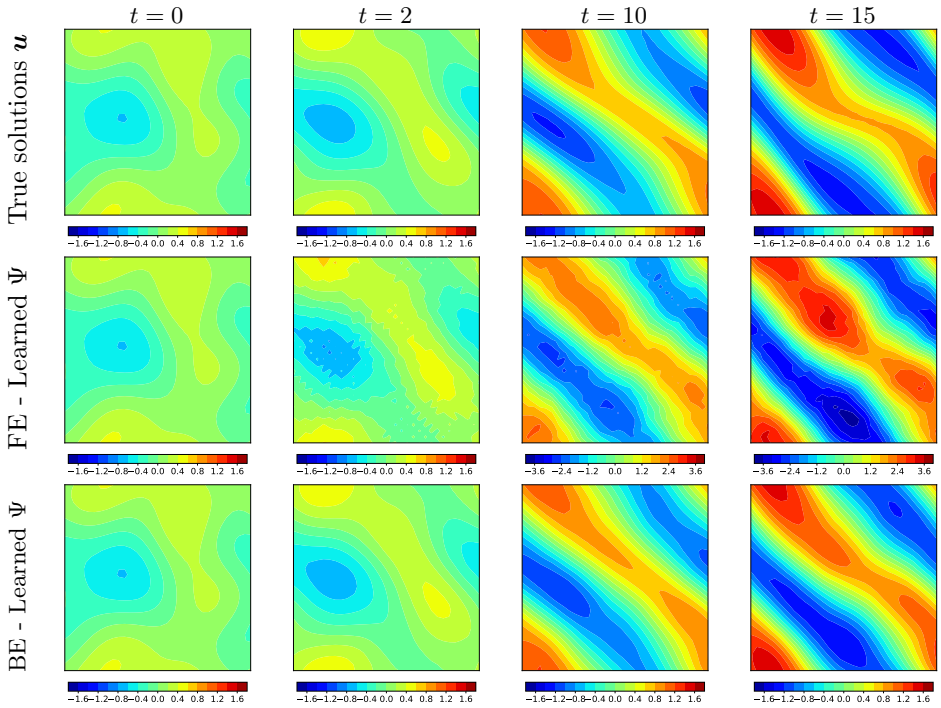

Fig. 20: **Navier-Stokes equation**. Comparison of various neural network solutions. *Top row*: True solutions with spectral method (Crank–Nicolson time integration scheme) with the true tangent slope; *Second row*: Forward Euler (FE) scheme (a different scale bar in the third and fourth columns); *Third row*: Backward Euler (BE) scheme with the learned neural network for 20 times larger time stepsize $\Delta t' = 20\Delta t = 0.2$.

from the same data set.

Implicit time integration with learned network. We used the learned network for backward Euler scheme with 20 times larger time stepsize, $\Delta t' = 20\Delta t = 0.2$, compared to training stepsize $\Delta t = 0.01$. As shown in Figure 20, forward Euler scheme with the learned network shows severe instability, while the backward Euler scheme with the learned network solutions is in good agreement with the spectral solution with Crank–Nicolson scheme with a much smaller time stepsize.

3.4. Information on parameter tuning, randomness, and cost.

3.4.1. Parameter tuning. The purpose of this section is to determine a good set of hyperparameters including the learning rate, batch size, the number of layers, and the number of neurons on each layer. To that end, we set the random seed to 0 in order to have a fair initialization for all networks. For initialization, weights are drawn randomly from zero-mean Gaussian distribution with a variance of 0.01, while biases are set to zero. For all cases, we take $S = 1, R = 1$, and noise-free data set with 600 samples. We carry out the tuning process manually for only Burgers and Navier-Stokes examples as the transport example admits an analytical solution. We pick the learning rate in $\left\{10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}, 10^{-3}\right\}$, batch size in

$\{2, 10, 40, 100\}$, the number of layers in $\{1, 2, 3\}$, the number of neurons per layers in $\{50, 200, 1000, 5000, 10000\}$, and the model-constrained regularization parameter α in $\{10, 10^3, 10^5, 10^7\}$. We pick the combination of parameters that provides the best testing accuracy (see also subsection 3.2 and subsection 3.3 for the discussion on testing accuracy) in each numerical problem. The chosen parameter set is then used for training different values of S, R and noise level δ.

3.4.2. Robustness with random initializations and data randomization. In this section, we study the effect of weights/biases random initialization and data randomization on the performance of the chosen neural network architectures in subsection 3.4.1. We provide the study for Burger's equations in subsection 3.2 since the result for the Navier-Stokes equation in subsection 3.3 would be similar. For random initialization of weights/biases, we initialize the neural network with 32 different random seeds ranging from 0 to 31. For each random seed, we use the same set of hyper-parameters found in subsection 3.4.1. As shall be shown, our model-constrained approach is robust in random initialization, that is, all random seeds work equally well. Thanks to this robustness, we simply initialize weights/biases with random seed 0 and study the effect of 32 different noise random seeds ranging from 0 to 31 for data randomization. As an example, we compare the mean and variance of the mean square error between the pure data-driven machine learning case $[d600, 2\%, 0, 1, 0]$ and the corresponding model-constrained case $\left[d600, 2\%, 10^5, 1, 1\right]$. The mean and variance results in Figure 21 and Figure 22 show that `mcTangent` networks are not only accurate but also more reliable with a smaller variance compared to the pure data-driven counterparts. Consequently—again thanks to the model-constrained term—the performance of `mcTangent` networks are robust to both weights/biases random initialization and data randomization.

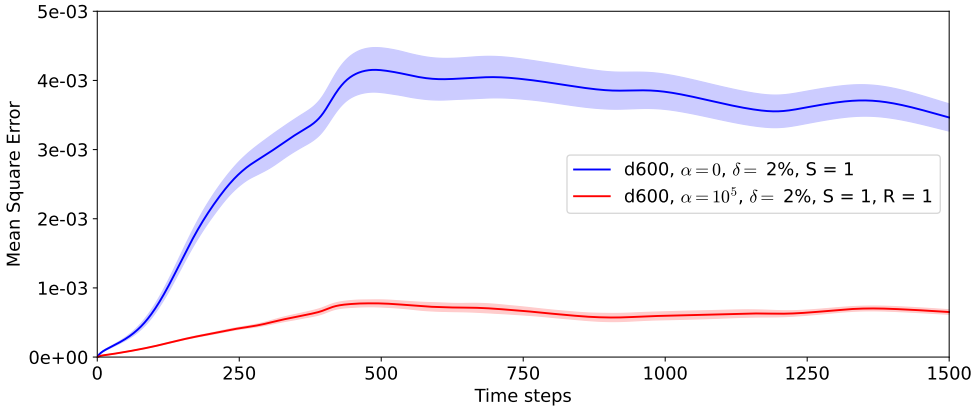

Fig. 21: The mean and variance of mean square error of predictions for `mcTangent` and pure data-driven machine learning approaches, obtained by 32 different neural networks corresponding to 32 different weights/biases random initializations.

3.4.3. Training and testing cost. Table 1 presents the training computational cost for Burgers' problem using different values of S and R. The `mcTangent` neural network is learned with 200 training samples with 2% additive noise, $\alpha = 10^5$, learning rate 10^{-4}, and batch size 40. It can be seen that in the purely data-driven approach,

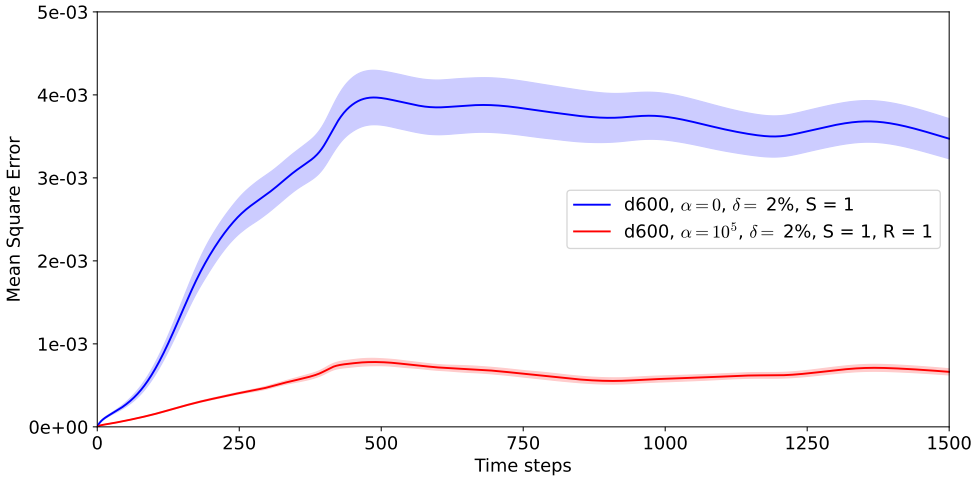

Fig. 22: The mean and variance of mean square error of predictions for `mcTangent` and pure data-driven machine learning approaches, obtained by training a neural network (with weights/biases being initialized with random seed 0) with 32 random realizations of data corresponding to 32 different random seeds.

i.e. $S = 1, R = 0$, the computational time per epoch is small, but the number of required epochs for convergence is larger. It is not surprising that adding larger R and S leads to a significant increase in computational cost per epoch. However, in this problem, since the overall convergence rate (measured in the terms of the number of epochs) is faster, the total amount of time for training model-constrained neural networks are at most three times larger than the pure machine learning method. To be more specific, $S = 1, R = 0$ network requires 11.67 hours compared to 22.22 hours for $S = 1, R = 1$ network. On the other hand, 35 hours and 34 hours are needed to train the cases $S = 10, R = 1$ and $S = 1, R = 5$, respectively. We note that all model-constrained networks corresponding to $S = 1, R = 1$, $S = 10, R = 1$ and $S = 1, R = 5$ provide comparable accuracy levels which are significantly better than that obtained by the pure data-driven machine learning approach corresponding to $S = 1, R = 0$, and this is shown in Figure 10.

To verify the computational benefits in the prediction stage, we compare the computational time between the ground truth solution using the truth tangent slope $\boldsymbol{G}(\tilde{\boldsymbol{u}})$ and `mcTangent` tangent slope $\Psi(\tilde{\boldsymbol{u}})$ in Table 2. It can be seen that the `mcTangent` tangent slope is much faster (more than 10 times faster) than the truth tangent slope for the 2D Navier-Stokes problem. For the 2D Burgers' problem, the `mcTangent` tangent slope evaluation is negligibly faster than the truth. That is, even with small-scale 2D problems with fast finite difference evaluations, the neural network is still faster. It is important to point out that the computational cost for `mcTangent` neural network remains unchanged, 2×10^{-4} seconds, for either Burgers or Navier-Stokes equations. We expect the computational gain is much more notable for 3D complex problems where the evaluation of the truth tangent slope is much more demanding. The gain is even more significant for implicit methods as in these cases not only the evaluation of the tangent slope but also the evaluation of its Jacobian is needed. This poses great challenges for traditional numerical methods, but for the `mcTangent` approach, the

evaluations of a feed-forward network and its Jacobian are trivial and fast.

Table 1: Training cost for Burgers' equations using different values of S and R. `mcTangent` neural network is learned with 200 samples with 2% additive noise, $\alpha = 10^5$, learning rate 10^{-4}, and batch size 40.

	1 Epoch (seconds)	Number of Epoch	Training time (hours)
$S = 1, R = 0$	0.07	6.0×10^5	11.67
$S = 1, R = 1$	0.20	4.0×10^5	22.22
$S = 10, R = 1$	0.86	1.5×10^5	35.83
$S = 1, R = 5$	0.62	2.5×10^5	34.44

Table 2: Computational cost of ground truth tangent slope $\boldsymbol{G}(\tilde{\boldsymbol{u}}^i)$ (mesh grid: 32×32), and trained neural network $\Psi\left(\tilde{\boldsymbol{u}}^i\right)$

	$G(\boldsymbol{u})$ (seconds)	$\Psi(\boldsymbol{u})$ (seconds)
Burgers' equation	2.1×10^{-4}	2×10^{-4}
Navier-Stokes equation	7.0×10^{-3}	2×10^{-4}

4. Conclusions. We have presented a model-constrained tangent slope learning (`mcTangent`) approach to simulate dynamical systems in real-time. At the heart of `mcTangent` is a careful craft synergizing several desirable strategies: i) a tangent slope learning to take advantage of the neural network speed and time-accurate nature of the method of lines; ii) a model-constrained approach to encode the neural network tangent slope with the underlying physics; iii) sequential learning strategies to promote long-time stability and accuracy; and iv) data randomization approach to implicitly regularize the smoothness of the neural network tangent and its likeliness to the truth tangent up second order derivatives in order to further enhance the stability and accuracy of `mcTangent` solutions. Rigorous results are provided to analyze and justify the proposed approach. Several numerical results for transport equation, viscous Burgers equation, and Navier-Stokes equation are presented to study and demonstrate the capability of the proposed `mcTangent` learning approach. Further theoretical analysis of `mcTangent` with both sequential learning strategies is ongoing to provide a deeper understanding of the approach. Strategies to improve the accuracy and to strongly encode the underlying governing equations are also part of future work.

Disclosure statement. No potential conflict of interest was reported by the author(s).

REFERENCES

[1] Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural computation*, 8(3):643–674, 1996.

[2] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. Deep learning for time series forecasting: Tutorial and literature survey. *ACM Computing Surveys (CSUR)*, 2018.

[3] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] Noah D Brenowitz and Christopher S Bretherton. Prognostic validation of a neural network unified physics parameterization. *Geophysical Research Letters*, 45(12):6289–6298, 2018.

[6] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[7] Tan Bui-Thanh. A unified and constructive framework for the universality of neural networks, 2021.

[8] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Opt. Express*, 28(8):11618–11633, Apr 2020.

[9] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[10] Emmanuel De Bézenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, 2019.

[11] Karthik Duraisamy. Perspectives on machine learning-augmented reynolds-averaged and large eddy simulation models of turbulence. *Physical Review Fluids*, 6(5):050504, 2021.

[12] Philipp Holl, Vladlen Koltun, Kiwon Um, and Nils Thuerey. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *NeurIPS Workshop*, volume 2, 2020.

[13] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[14] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *ICLR*, 2020.

[15] Xiaowei Jia, Jared Willard, Anuj Karpatne, Jordan Read, Jacob Zwart, Michael Steinbach, and Vipin Kumar. Physics guided rnns for modeling dynamical systems: A case study in simulating lake temperature profiles. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 558–566. SIAM, 2019.

[16] Jiaqi Jiang, Mingkun Chen, and Jonathan A. Fan. Deep neural networks for the evaluation and design of photonic devices. *Nature Reviews Materials*, Dec 2020.

[17] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

[18] Jesse Johnson. Deep, skinny neural networks are not universal approximators. In *International Conference on Learning Representations*, 2019.

[19] Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer graphics forum*, volume 38, pages 59–70. Wiley Online Library, 2019.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] 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.

[22] Keisuke Kojima, Bingnan Wang, Ulugbek Kamilov, Toshiaki Koike-Akino, and Kieran Parsons. Acceleration of fdtd-based inverse design using a neural network approach. In *Advanced Photonics 2017 (IPR, NOMA, Sensors, Networks, SPPCom, PS)*, page ITu1A.4. Optical Society of America, 2017.

[23] Randall J LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[24] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020.

[25] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

[26] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[27] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G. Johnson. Physics-informed neural networks with hard constraints for inverse design, 2021.

[28] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power

of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[29] Nils Margenberg, Dirk Hartmann, Christian Lessig, and Thomas Richter. A neural network multigrid solver for the navier-stokes equations. *Journal of Computational Physics*, 460:110983, 2022.

[30] Kiyotoshi Matsuoka. Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):436–440, 1992.

[31] Jeremy Morton, Antony Jameson, Mykel J Kochenderfer, and Freddie Witherden. Deep dynamical modeling and control of unsteady fluid flows. *Advances in Neural Information Processing Systems*, 31, 2018.

[32] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[33] Shaowu Pan and Karthik Duraisamy. Long-time predictive modeling of nonlinear dynamical systems using neural networks. *Complexity*, 2018, 2018.

[34] Jaideep Pathak, Mustafa Mustafa, Karthik Kashinath, Emmanuel Motheau, Thorsten Kurth, and Marcus Day. Using machine learning to augment coarse-grid computational fluid dynamics simulations. *arXiv preprint arXiv:2010.00072*, 2020.

[35] Raphaël Pestourie, Youssef Mroueh, Thanh V. Nguyen, Payel Das, and Steven G. Johnson. Active learning of deep surrogates for pdes: application to metasurface design. *npj Computational Materials*, 6(1):164, Oct 2020.

[36] John Peurifoy, Yichen Shen, Li Jing, Yi Yang, Fidel Cano-Renteria, Brendan G. DeLacy, John D. Joannopoulos, Max Tegmark, and Marin Soljačić. Nanophotonic particle simulation and inverse design using artificial neural networks. *Science Advances*, 4(6), 2018.

[37] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

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

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

[40] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125 – 141, 2018.

[41] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, 348:683 – 693, 2017.

[42] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[43] Russell Reed, Seho Oh, RJ Marks, et al. Regularization using jittered training data. In *International Joint Conference on Neural Networks*, volume 3, pages 147–152, 1992.

[44] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[45] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR, 2018.

[46] William E Schiesser. *The numerical method of lines: integration of partial differential equations*. Elsevier, 2012.

[47] Samuel S. Schoenholz and Ekin D. Cubuk. Jax m.d. a framework for differentiable physics. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

[48] Gordon D Smith, Gordon D Smith, and Gordon Dennis Smith Smith. *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, 1985.

[49] Sunae So, Trevon Badloe, Jaebum Noh, Jorge Bravo-Abad, and Junsuk Rho. Deep learning enabled inverse design in nanophotonics. *Nanophotonics*, 9(5):474, February 2020.

[50] Mohammad H. Tahersima, Keisuke Kojima, Toshiaki Koike-Akino, Devesh Jha, Bingnan Wang, Chungwei Lin, and Kieran Parsons. Deep neural network inverse design of integrated

photonic power splitters. *Scientific Reports*, 9(1):1368, Feb 2019.

[51] Rohit K. Tripathy and Ilias Bilionis. Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, 375:565 – 588, 2018.

[52] Jeroen Tromp, Carl Tape, and Qinya Liu. Seismic tomography, adjoint methods, time reversal and banana-doughnut kernels. *Geophysical Journal International*, 160(1):195–216, 2005.

[53] Kiwon Um, Robert Brand, Yun Raymond Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in Neural Information Processing Systems*, 33:6111–6122, 2020.

[54] Yi-Jen Wang and Chin-Teng Lin. Runge-kutta neural network for identification of dynamical systems in high accuracy. *IEEE transactions on neural networks*, 9 2:294–307, 1998.

[55] Daniel A. White, William J. Arrighi, Jun Kudo, and Seth E. Watts. Multiscale topology optimization using neural network surrogate models. *Computer Methods in Applied Mechanics and Engineering*, 346:1118–1135, 2019.

[56] Tailin Wu, Qinchen Wang, Yinan Zhang, Rex Ying, Kaidi Cao, Rok Sosič, Ridwan Jalali, Hassan Hamam, Marko Maucec, and Jure Leskovec. Learning large-scale subsurface simulations with a hybrid graph network simulator. *arXiv preprint arXiv:2206.07680*, 2022.

[57] Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 2019.

[58] Jiawei Zhuang, Dmitrii Kochkov, Yohai Bar-Sinai, Michael P Brenner, and Stephan Hoyer. Learned discretizations for passive scalar advection in a two-dimensional turbulent flow. *Physical Review Fluids*, 6(6):064605, 2021.

[59] Qinyu Zhuang, Juan Manuel Lorenzi, Hans-Joachim Bungartz, and Dirk Hartmann. Model order reduction based on runge–kutta neural networks. *Data-Centric Engineering*, 2, 2021.